COSC 123
Computer Creativity

Course Introduction

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Course Objectives

1) To be creative with programming and write fun, interesting programs

2) To master fundamental programming skills of data variables, decisions, iteration, and methods

3) To learn to create stories using the Alice programming language

4) To learn the Java language, the basics of object-oriented programming, and how to create larger programs

5) To learn about graphics, events, and exceptions in Java

How to Pass This Course

The most important things to do to pass this course:

- ◆Attend and participate in class
  - ⇨Read notes *before* class as preparation.
- ◆Attend the labs and do all lab assignments
  - ⇨They are for marks, and they are good practice and exam questions.

To get an “A” in this course do all the above plus:

- ◆Spend more time practicing programming including questions in the notes and the free-form labs.

My Expectations

My goal is for you to ***SHOW UP TO CLASS AND LABS*** and spend the effort to learn the material.

Although this class may be “easy” for some, you will not pass this class without effort and ***attendance***.

The course will be very straightforward – If you do the work, you will do well.

You should practice programming outside of class/labs.

Your mark is 80% perspiration and 20% inspiration.

The Lab Assignments

In each lab we will work on computers on a lab assignment.

Lab assignments are worth ***20%*** of your overall grade.

Most assignments are due approximately one week after the lab.

- ◆No late assignments will be accepted.
- ◆An assignment may be handed in any time before the due date.
- ◆Some lab assignments are larger and allow you to create your own programs.

Lab assignments are done in pairs (pair programming).

The lab assignments are critical to learning the material and are designed to prepare you for the exams!

Pair Programming

All lab assignments and projects will be done using the ***pair-programming approach***.

- ◆Students will select a partner at the start of class that will be their partner for the duration of the course.
- ◆Students may ask the professor for help in finding a suitable partner.
- ◆Accommodation is made for students whose partner leaves the course before its completion.
- ◆Both students in the pair receive the same mark.

Pair programming has been shown to increase learning and satisfaction while programming.

Class Quizzes and Questions

To encourage attendance and effort, ***20%*** of your overall grade is allocated to answering questions in class.

There are two types of questions:

- ◆**10%** - for electronic questions answered using clickers
  - ⇨There will be at least 90 questions each worth 1 mark. **You need at least 70 to get the full 10%.**
  - ⇨**You must be present with your clicker to get your answers counted.**
  - ⇨The marks are pro-rated. Example: if you get 50 right you would get 50/70 = 7%.
- ◆**10%** - for programming and written questions
  - ⇨There will be at least 40 programming and written questions.
  - ⇨**You need at least 40 points.** You get 2 points for showing a correct answer in class on or before the day it is covered, and 1 point for providing an answer within 3 days of that class.
  - ⇨**You should plan and work ahead as not all questions will be given sufficient time to complete during class time.**

Why are you here?
Reasons Why People Take This Course

A) I want an easy credit.

B) I want an easy Science credit (Arts Majors).

C) I want to learn how to be creative using programming.

D) COSC 122 was okay, and I am interested in more.

E) Alice and 3D worlds look pretty cool.

What do you expect?
What Grade are You Expecting to Get?

A) A

B) B

C) C

D) D

E) F

Programming Experience
What is Your Programming Experience?

A) None (or I forgot everything I have seen before)

B) I remember some of the programming in COSC 122 or have programmed on my own before.

C) I have taken COSC 111/121 or equivalent.

D) I have taken Computer Science courses beyond the 1st year.

E) I program all the time. I plan on being the next Bill Gates or Steve Jobs.

Why this Course is Important

This course will make programming fun and relevant.

- ◆Our economy, health, and entertainment is dependent on software written by programmers.
- ◆We will learn to be creative programmers, so that we may create great software to be used by others.

Important results:

- ◆***Storyboarding*** – We will use Alice to tell stories with programs.
- ◆***Algorithmic Thinking*** – We will learn how to solve problems by specifying precise sequences of actions.
- ◆***Collaboration*** – We will program in teams of two to build interpersonal skills and increase our knowledge.
- ◆***Java Language*** – We will learn the Java programming language that can be used in many areas including future computer science courses.

The Essence of the Course

If you walk out of this course with nothing else you should:

Become a creative programmer with the ability to problem solve, perform critical thinking, and communicate precisely.

This course is not about learning a particular language or even programming itself.

Introduction to Alice

Alice is a computer environment in which you create virtual worlds containing three-dimensional characters and objects that move and interact.

Alice is an integrated development environment (IDE) – a program used to create and run another program.

Versions for Windows and Mac OS are available from the Alice website: **http://www.alice.org**.

Let's try a couple of demos!

COSC 123
Computer Creativity

Introduction to Alice

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) Learn the basic idea of programming and its key concepts

2) Experiment with the Alice environment and create worlds.

3) Learn about objects, classes, and methods.

4) Set and modify the properties of an object.

5) Create new objects including composite objects.

6) Learn how to animate many objects simultaneously.

Programming

What is programming?

- ***Programming*** is the process of constructing programs in order to instruct a computer on how to solve problems. It is the act of writing out the steps of an *algorithm*.
- A ***program*** is a sequence of simple computer instructions in some *language* which tell the computer the necessary steps to solve a problem or complete a task.
- A ***language*** is the structure and syntax used to communicate to the computer the tasks it is required to perform.

Why do we program?

Electronic devices require instructions to perform their function. Programming is our way of communicating those instructions.

Programs are written to do many things:

- Allow computes to communicate on the Internet
- Control airplanes, factories, cars, and electronics
- Send email, make a YouTube video, send a Twitter message
- Run businesses, handle inventory, trade stocks
- Any millions of others… Your ideas?

The ability to program in a digital society makes you a content ***producer*** rather than just a ***consumer***.

- Producers have the ability to impact others by creating and distributing their creations.

Programming and Creativity

Programming creates digital content. ***Creativity*** is at two levels:

- 1) Programming allows us to express our visions electronically for others to use.
- 2) The act of programming to realize the vision requires creativity and problem solving.

All programs that you use (Internet, email, Microsoft Office, YouTube, Google) are the result of programmer creativity.

- They had the vision to determine what they wanted to build and how that product can impact society.
- They had the ability to realize that vision by creating the necessary programs.

Programming Languages

Often the fun and creativity that programming allows gets lost in the details of the programming language.

The programming language is the format that we express our vision and approach to the computer. Each language has its own features, benefits, issues, and syntax.

The challenge is that to communicate with the computer we need to learn the language and associated tools and rules.

- Learning the language and tools takes practice and patience.

Analogy:

- Writers need to be fluent in the language they write.
- Artists need to know the basic techniques for painting/drawing.

Programming Languages Alice and Java

The two programming languages that we use are very different.

Alice is a graphical language designed to teach programming. All Alice programming is done graphically (very little typing). Alice programs are 3D stories and animations.

Java is a general purpose language used in industry and other programming courses. Java allows you to create anything and runs on most computes and cell phones.

Artistic comparison: Alice is like paint by numbers whereas Java is an open canvas for oil painting.

- ◆Issue: "With great power comes great responsibility."

Programming Concepts

There are some basic techniques common to all languages. **Learning the concepts is more important than the language.**

Key programming concepts:

- ◆***data variables*** – storing and using data in named locations
- ◆***expressions*** – computations on data to produce new results
- ◆***execution order*** – instructions are given in the correct order
- ◆***decisions*** – perform different actions based on a condition
- ◆***iteration*** – repeat a sequence of steps multiple times
- ◆***methods*** – groups of instructions with a particular purpose
- ◆***data structures*** – organizations to hold many data items
- ◆***code organization*** – larger programs need to structure the code so it can be easily created and modified (object-oriented)

Programming - Art or Science?

Is programming an art or a science?

- ◆It is a science because algorithms and data structures can be analyzed for performance and chosen with respect to their relevance to a particular problem.
- ◆It is an art or craft because skills of programmers vary widely, even with similar training, and the "best" solution to the problem is often open to debate.

In computer science, we teach you the "science" component.

- ◆We want you to understand the choices you make and the reasons for them.
- ◆However, students will all have different natural abilities and talents with respect to programming.
- ◆If it is easy or natural for you, great! If not, then fall back on the science and the techniques we teach to help you!

Programming: Art or Science?

Question: What do you think programming is most like?

A) Art (creativity)

B) Science (experimentation)

C) Engineering (construction)

D) All of the above

E) Other or none of the above

Programming Practice

Like arts/sports, programming is a skill that requires practice.

- ◆A musician practices scales to learn the basics and does the same song many times to master the techniques. Each song has its own skills and techniques used.
- ◆A programmer practices by creating programs to perform tasks. The programs require understanding of the language and tools, and the solutions require composing techniques.

Key point: Like an artist, you must commit to practicing the craft. Programming skill comes from practice not memorization.

The labs are designed to give you some practice, but mastery will require more. Practicing is your studying for this course!

The 5 Basic Steps of Software Development

A programmer does ***NOT*** begin creating without a plan.

Developing a program should follow five basic steps:

- ◆1) **Specification** - Determine the scope of your problem and **what** you want your program to do.
- ◆2) **Design** - Determine the structures and algorithms necessary (**how**) to solve your problem at a high-level of abstraction.
- ◆3) **Implementation** - Start writing the code on the computer.
- ◆4) **Testing, Execution, and Debugging** - Test your program for various cases and fix any problems.
- ◆5) **Maintenance** - Over time, modify your program as necessary to handle new data or more complicated problems.

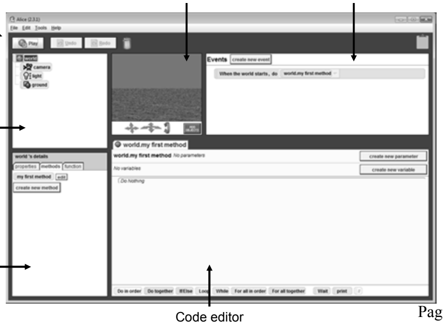
Programming with Alice

Alice is a 3D programming language designed to teach programming.

Alice allows you to compose stories which contain objects and scenery that interact.

Programming a story involves creating objects and scenery (the data), moving and interacting the objects (the instructions and methods), and everything in the story occurs according to a script (set of instructions). The script may involve decisions, loops, and events.

Alice Environment



Start Program

World View

Events Editor

Object Tree

Details Pane

Code editor

Objects

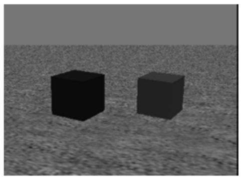
All ***objects*** in the Alice world are listed in the ***object tree***.

Objects are elements in the world that have a name.

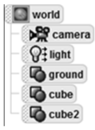
Two standard objects are the camera and the light source. Most worlds have a ground surface which is also an object.

The camera controls set the point of view.

World View



Object Tree



This world has five objects: camera, light, ground, cube, and cube2.

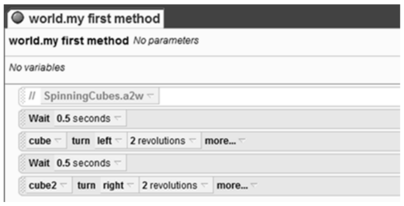
Camera Controls

Methods

A ***method*** is a set of statements that can be ***called***.

- Methods perform actions and are associated with objects. The methods define an object's behavior (what it can do).

The ***world*** object has a method called ***my first method***. This method is called when the animation starts.

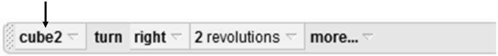


```
world.my first method
world.my first method No parameters
No variables
// SpinningCubes.a2w
Wait 0.5 seconds
cube turn left 2 revolutions more...
Wait 0.5 seconds
cube2 turn right 2 revolutions more...
```

Methods (2)

Calling a method is the act of running a method of an object.

Methods can accept ***parameters*** which provide input data for the method to use.



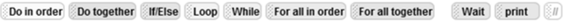
`cube2 turn right 2 revolutions more...`

Object

Method

Parameters

A method consists of a sequence of statements. Statements may be calls to other methods or statements to perform decisions, loops, or calculations. Statements in Alice:



Built-in Methods

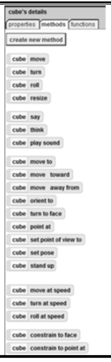
Built-in methods exist for almost all objects in Alice.

Other methods can be developed and added.

Some useful methods are:

- say
- think
- sound

Add a method call to your code by clicking on the object, selecting the methods tab, then dragging method into the code area.



cube's details

properties | methods | functions

create new method

cube move
cube turn
cube roll
cube resize

cube say
cube think
cube play sound

cube move to
cube move toward
cube move away from
cube orient to
cube turn to face
cube point at
cube set point of view to
cube set pose
cube stand up

cube move at speed
cube turn at speed
cube roll at speed

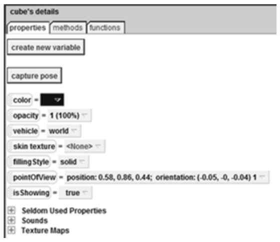
cube constrain to face
cube constrain to point at

Properties

Properties describe an object's state at any point in time.

- ◆e.g. color, texture

The value of the property can be changed directly or during animation using a method call.



color = ■
opacity = 1 (100%)
vehicle = world
skin texture = <None>
fillingStyle = solid
pointOfView = position: 0.58, 0.86, 0.44; orientation: (-0.05, -0, -0.04) 1
isShowing = true

Classes

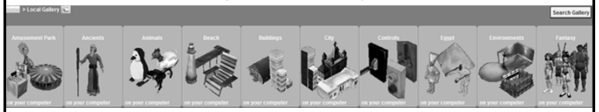
A ***class*** is a template for an object.

A class determines an object's methods and properties.

In Alice, classes are organized into galleries.

There are built-in (local) galleries and galleries on the Web.

Exploring the Local Gallery of Classes



Terminology Summary

An ***object*** is an instance of a class that has its own properties and methods. Properties and methods define what the object is and what it can do.

A ***class*** is a generic template (blueprint) for creating an object. All objects of a class have the same methods and properties (although the property values can be different).

A ***property*** is an attribute or feature of an object.

A ***method*** is a set of statements that performs an action.

A ***parameter*** is data passed into a method for it to use.

Objects

Question: Which of the following is not an object?

A) camera

B) world

C) wait

D) cube

Objects and Methods

Question: True or false: It is possible to have a method with no parameters.

A) true

B) false

Classes

Question: True or false: The two cube objects have the same class.

A) true

B) false

Classes and Objects

Question: True or false: Two objects that have the same class have the same methods.

A) true

B) false

Classes and Objects (2)

Question: True or false: Two objects that have the same class may have different values for their properties.

A) true

B) false

Demonstration Exercise Classes, Objects, Methods, Properties

Start Alice and open up `SpinningCubes.a2w`.

- ◆Save a version of the file in your own directory on F:.

Items to try:

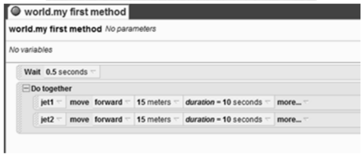
- ◆Play the animation. Then close the animation.
- ◆Try moving the camera using the camera controls.

Change the program to have these steps in order:

- ◆1) Make `cube` turn left once and `cube2` turn right 5 times.
- ◆2) Make `cube` go up 5 meters after it spins.
- ◆3) Change the color of `cube2` to yellow. (`properties` tab)
- ◆4) Call `resize` method on `cube` to make its ½ its size.
- ◆5) Add any object from the gallery to the world and make it move up 5 meters.

Do Together Statement

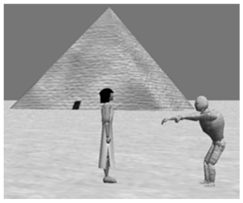
The ***Do Together*** statement allows several things to be done simultaneously.

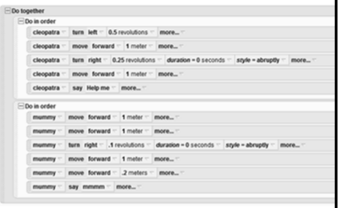


```
world.my first method
world.my first method No parameters
No variables
  Wait 0.5 seconds
  Do together
    jet1 move forward 15 meters duration = 10 seconds more...
    jet2 move forward 15 meters duration = 10 seconds more...
```

Do In Order Statement

The ***Do In Order*** statement forces the statements it contains to be executed in order.



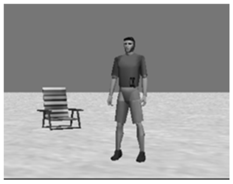


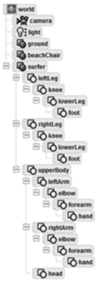
```
Do together
  Do in order
    cleopatra turn left 0.5 revolutions more...
    cleopatra move forward 1 meter more...
    cleopatra turn right 0.25 revolutions duration = 0 seconds style = abruptly more...
    cleopatra move forward 1 meter more...
    cleopatra say Help me more...
  Do in order
    mummy move forward 1 meter more...
    mummy move forward 1 meter more...
    mummy turn right .1 revolutions duration = 0 seconds style = abruptly more...
    mummy move forward 1 meter more...
    mummy move forward .2 meters more...
    mummy say mmmm more...
```

Composite Objects

A ***composite object*** is an object that contains other objects.

It is possible to control the whole object or any of its parts.

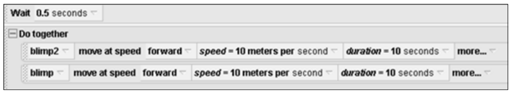




```
world
  camera
  light
  ground
  beachChair
  surfer
    leftLeg
      knee
        lowerLeg
          foot
    rightLeg
      knee
        lowerLeg
          foot
    upperBody
      leftArm
        elbow
          forearm
            hand
      rightArm
        elbow
          forearm
            hand
      head
```

Do Together and Do In Order

Question: What will this code do?



```
Wait 0.5 seconds
⊟ Do together
    blimp2 move at speed forward speed = 10 meters per second duration = 10 seconds more...
    blimp move at speed forward speed = 10 meters per second duration = 10 seconds more...
```

A) Move blimp then blimp2.

B) Move blimp2 then blimp.

C) Move blimp and blimp2 at the same time.

Demonstration Exercise
Do Together and Do In Order

Start Alice and open up `SpinningCubes.a2w`.

Change the program to have these steps in order:

- ◆1) Make `cube` turn left once and `cube2` turn right once at the same time.
- ◆2) Make `cube` turn move up 5 meters and `cube2` move left 5 meters.

Demonstration Exercise
Composite Object

Start Alice and open up `SurferWave.a2w`.

New ideas:

- ◆The `surfer` is a composite object.
- ◆To capture a pose, move the object into a certain position then under `properties` click `capture pose button`. Then to make the person go into that pose again use `set pose` method.

Change the program to have these steps in order:

- ◆1) Make the surfer say "`Hello`" while waving.
- ◆2) Make the surfer's arm go back to normal after he is done waving.
- ◆3) Using capture pose and set pose, make a pose with the arms spread out from the body parallel to the ground (looks like a T). Then put character in that pose and put him back again.
  - ⇨Make sure to capture original standing pose.

Conclusion

Object-oriented programming uses:

- ◆***Objects*** – are instances of a class that have their own properties and methods.
- ◆***Classes*** – are generic templates (blueprints) for creating objects.
- ◆***Methods*** – contains statements that perform an action.
- ◆***Parameters*** – are data passed into a method.
- ◆***Properties*** – are attributes/features of objects.

Object-oriented programming involves defining objects and manipulating their properties and methods to perform useful actions.

Summary of Alice

In Alice:

- ◆`Galleries` contain ***classes*** of objects.
- ◆An ***object*** is created from a class when it is put into the world.
- ◆Calling ***methods*** on objects make the objects do things.
- ◆A ***property*** is a feature of an object such as its color.
- ◆***Composite objects*** contain other objects.
- ◆***Do Together*** makes actions occur simultaneously.
- ◆***Do In Order*** makes actions happen sequentially.

COSC 123
Computer Creativity
Methods

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) Create our own methods for objects.

2) Declare and manipulate variables.

3) Generate and use random numbers.

4) Create methods with parameters and understand how parameters work.

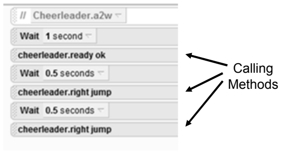
☆

Methods

A ***method*** is a sequence of statements that performs an action for an object.

Creating a method avoids repeating statements and allows for better code organization.

Methods allow you to add new actions for an object.

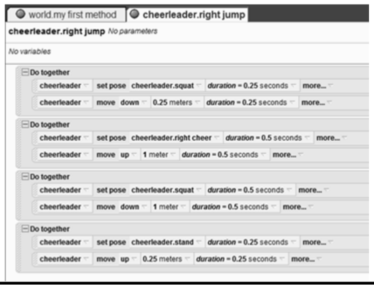


```
// Cheerleader.a2w
Wait 1 second
cheerleader.ready ok
Wait 0.5 seconds
cheerleader.right jump
Wait 0.5 seconds
cheerleader.right jump
```

Calling Methods

Cheerleader Method Example

The **`right jump`** method is a sequence of statements that makes the cheerleader jump. Many of these statements are calls to other (built-in) methods such as `set pose` and `move`.



```
cheerleader.right jump No parameters
No variables
Do together
    cheerleader set pose cheerleader.squat duration = 0.25 seconds more...
    cheerleader move down 0.25 meters duration = 0.25 seconds more...
Do together
    cheerleader set pose cheerleader.right cheer duration = 0.5 seconds more...
    cheerleader move up 1 meter duration = 0.5 seconds more...
Do together
    cheerleader set pose cheerleader.squat duration = 0.5 seconds more...
    cheerleader move down 1 meter duration = 0.5 seconds more...
Do together
    cheerleader set pose cheerleader.stand duration = 0.25 seconds more...
    cheerleader move up 0.25 meters duration = 0.25 seconds more...
```

Creating Methods

To create a method:

- ◆1) Click on the object in the object tree to select it.
- ◆2) Click on the **`create new method`** button.
- ◆3) Give the method a name.
- ◆4) Add statements to the method to make it perform the actions desired.

Note: Usually methods are associated with a class but in Alice methods are associated with objects.

Why do we Create Methods?

Two main reasons to create methods:

- ◆1) To organize code into blocks that have specific purpose
- ◆2) To avoid duplication by reusing code

A method is a block of statements that does something useful.

- ◆The block of code is separated from other statements which makes it easier to read and modify.
- ◆The block of code can be called many times if the method needs to be done multiple times.

What is the alternative? Copy and paste and duplicate code. You will realize over time that this is actually the harder way to do things.

Variables

A ***variable*** is a name that represents a spot in memory that can store a value. The variable must be ***declared***, which defines the variable's name and the type of data it will hold.

An object's properties are managed using variables (called ***instance variables***).

Object properties can be used by every method of the object.

Variables in methods:

- Variables declared (created) in a method are ***local*** – available only in that method.
- Parameters are variables that are passed into a method. The method can use the variables while it is executing.

Data Types in Alice

A variable can hold one of the following data types:

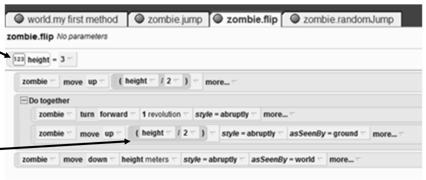
- a number (integer or floating point)
- a Boolean (true or false)
- a character string
- a reference to any other type of object

The ***data type*** of a variable defines how much memory is needed to store that variable value.

A variable has only one data type (can only store one type of data at any time).

Expressions

An ***expression*** consists of operands (variables, numbers) manipulated with operators (such as +,-,/,*).



```
zombie.flip No parameters
123 height = 3
zombie move up ( height / 2 ) more...
Do together
    zombie turn forward 1 revolution style = abruptly more...
    zombie move up ( height / 2 ) style = abruptly asSeenBy = ground more...
zombie move down height meters style = abruptly asSeenBy = world more...
```

Create variable `height` with value = 3.

Use `height` in methods and calculations.

Methods

Question: True or false: It is possible to create a method with no statements.

A) True

B) False

Methods in Alice

Question: True or false: In Alice, two objects of the same class always have the same methods.

A) True

B) False

Variables

Question: True or false: A variable declared inside one method can be accessed in another method.

A) True

B) False

Instance Variables

Question: True or false: An instance variable (object property) can be accessed by any method of that object.

A) True

B) False

Variables and Data Types

Question: True or false: In Alice, a single variable can store numbers and strings.

A) True

B) False

Parameters

A ***parameter*** is data passed into a method for it to use.

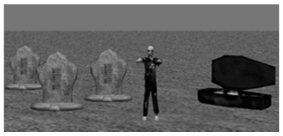
Methods may have zero parameters or as many as they want.

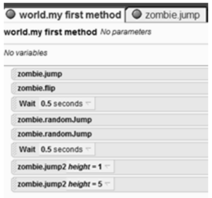
Each parameter has a data type.

To call a method with parameters you must pass in the necessary values (called arguments) for the method to use.

Using parameters makes a method more powerful and useful.

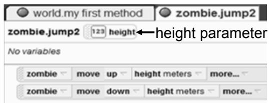
Example Method with Parameters





```
world.my first method | zombie.jump
world.my first method  No parameters
No variables
zombie.jump
zombie.flip
Wait 0.5 seconds
zombie.randomJump
zombie.randomJump
Wait 0.5 seconds
zombie.jump2 height = 1
zombie.jump2 height = 5
```

Calling the `jump2` method with different inputs.



```
world.my first method | zombie.jump2
zombie.jump2  123 height
No variables
zombie  move up  height meters  more...
zombie  move down  height meters  more...
```

height parameter

Random Numbers

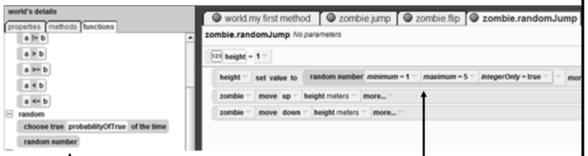
A random number is a number generated in a particular range.

Function **`random number`** is a world function. You provide the minimum and maximum number, and the function returns a number in that range.

- ◆Note: Make sure to specify if you want an integer or float.

Using random numbers allows your story and object behavior to change each time.

Applying Random Numbers



```
world's details
properties | methods | functions
a != b
a > b
a >= b
a < b
a <= b
random
choose true probabilityOfTrue of the time
random number
```

```
world.my first method | zombie.jump | zombie.flip | zombie.randomJump
zombie.randomJump  No parameters
123 height = 1
height  set value to  random number minimum = 1  maximum = 5  integerOnly = true  mor
zombie  move up  height meters  more...
zombie  move down  height meters  more...
```

`random number` function is under world functions.

Calling the `random number` function asking for a number between 1 and 5 that must be an integer.

Demonstration Exercise
Methods

Use `Cheerleader.a2w`. Tasks:

- Add a `left jump` method to the cheerleader that causes her to jump with her left arm and leg raised (use `left cheer` pose).
- Modify both `left jump` and `right jump` methods to use a new variable called `height` that controls the jump distance. Set `height` to a random number between 1 and 3 meters.
- Create a second cheerleader object by copying the first one.

Story:

- At the same time both cheerleaders should:
  - ⇨ readyOk
  - ⇨ rightJump
  - ⇨ leftJump
- Note that the cheerleaders will jump different heights, but they should be synchronized in their movements.

Demonstration Exercise
Parameters and Expressions

Use `Jet.a2w`. Tasks:

- Modify the `circle` method to accept a `time` parameter that is used to determine the duration (time to complete a circle).
- Create new `circle2` method that calculates time (not a parameter) to make sure that the jet travels the same distance regardless of speed.
- Create a second jet by copying the first one.

Story:

- Have `jet1` call `circle` three times. Each time the speed should be random between 10 and 100. The time parameter should be: first call 1 second, next 2 seconds, last 3 seconds.
- Have `jet1` call `circle2` twice. Once with speed 50 then 200.
- Make `jet1` and `jet2 circle` at the same time with speed 50 and time 1 second.

Conclusion

Methods can be added to objects to define additional behaviors.

- Creating methods organizes code and allows us to use the same code multiple times.

Variables defined in a method are local variables (only used in that method). Parameters are always local variables.

Object properties are instance variables that can be used by any method of the object.

Expressions use variables to calculate new values.

Objectives

Key terms: method, parameter, expressions, variable, value

Alice skills:

- Call a method.
- Create a method.
- Create and use variables.
- Generate random numbers.
- Create method parameters.
- Rename objects.
- Copying objects.

COSC 123
Computer Creativity

Decisions and Loops

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) The **`If/Else`** statement is used to make decisions.

2) A decision requires a condition that consists of relational operators and Boolean functions.

3) A set of statements can be executed multiple times using **`While`** and **`Loop`** statements.

☆

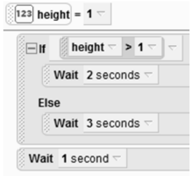
The `If/Else` Statement

Decisions are used to allow the program to perform different actions in certain conditions.

To make a decision we must do two things:

- 1) Determine the ***condition*** in which to make the decision.
- 2) Tell the computer what to do if the condition is true or false.

Example:



```
123 height = 1
If  height > 1          ← Condition to check:  True or False?
    Wait 2 seconds      ← Do this if condition is true.
Else
    Wait 3 seconds      ← Do this if condition is false.
Wait 1 second           ← Do statement after if in either case.
```

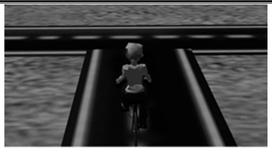
Condition to check: True or False?

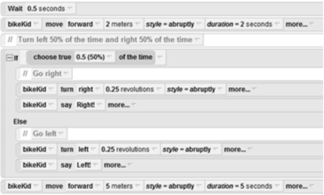
Do this if condition is **true**.

Do this if condition is **false**.

Do statement after if in **either case**.

Example `If/Else` Statement Left or Right?





```
Wait 0.5 seconds
bikeKid move forward 2 meters style = abruptly duration = 2 seconds more...
// Turn left 50% of the time and right 50% of the time
If choose true 0.5 (50%) of the time
    // Go right
    bikeKid turn right 0.25 revolutions style = abruptly more...
    bikeKid say Right! more...
Else
    // Go left
    bikeKid turn left 0.25 revolutions style = abruptly more...
    bikeKid say Left! more...
bikeKid move forward 5 meters style = abruptly duration = 5 seconds more...
```

Condition →

Go right statements

Go left statements

Demonstration Exercise Decisions

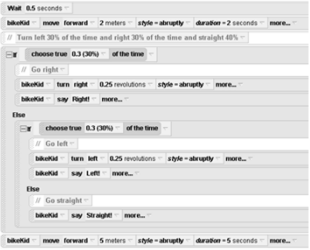
Use **`intersection.a2w`**.

Tasks:

- Play the animation.
- Modify so that the biker turns left 90% of the time.
- Modify so that the bike turn is smoother by moving forward and turning right at the same time.
- Modify so that the biker plays says "Hello" regardless of which direction turned.

Nested `If/Else` Statements More Than Two Possibilities

An **`If/Else`** statement could contain another ***nested*** **`If/Else`** statement.



```
Wait 0.5 seconds
bikeKid move forward 2 meters style = abruptly duration = 2 seconds more...
// Turn left 30% of the time and right 30% of the time and straight 40%
If choose true 0.3 (30%) of the time
    // Go right
    bikeKid turn right 0.25 revolutions style = abruptly more...
    bikeKid say Right! more...
Else
    If choose true 0.3 (30%) of the time
        // Go left
        bikeKid turn left 0.25 revolutions style = abruptly more...
        bikeKid say Left! more...
    Else
        // Go straight
        bikeKid say Straight! more...
bikeKid move forward 5 meters style = abruptly duration = 5 seconds more...
```

Nested if statement

Demonstration Exercise Nested Decisions

Use **`intersection2.a2w`**.

Tasks:

- Play the animation.
- Modify so that if the bike goes left he says "whoo hoo" while at the same time spinning around once.
- Modify so that there is a 50% of turning back around if the decision was to go straight.
- Add comments to say what each block of code in your if/else statements does.

Relational Operators

Relational operators are used to compare numeric data:

◆>	- Greater than
◆>=	- Greater than or equal
◆<	- Less than
◆<=	- Less than or equal
◆==	- Equal
◆!=	- Not equal

The Logical Operators

Operators:

`both` **X** `and` **Y**	- true if X and Y are true, false otherwise
`not` **X**	- true if X is false
`either` **X** `or` **Y**	- true if either X or Y or both are true

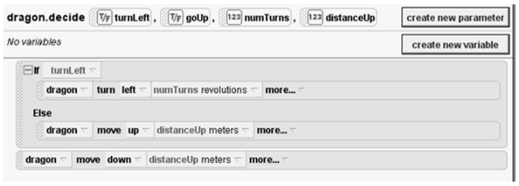
Examples:

- not *(This is COSC 123)*
- both *(This is COSC 123)* AND *(My name is Joe Smith)*
- either *(This is COSC 123)* OR *(My name is Joe Smith)*

Decisions

Question: What is the result of this code if:

- turnLeft = true ; goUp = false ; numTurns = 5 ; distanceUp = 7



```
dragon.decide  T/F turnLeft , T/F goUp , 123 numTurns , 123 distanceUp
No variables
  If  turnLeft
    dragon  turn  left  numTurns revolutions  more...
  Else
    dragon  move  up  distanceUp meters  more...
  dragon  move  down  distanceUp meters  more...
```

A) The dragon goes up 7 meters then down 7 meters.

B) The dragon turns around 5 times.

C) The dragon turns around 5 times and then down 7 meters.

Relational Operators

Question: True or false: a is true and b is false.

What is **`both a and b`**?

A) true

B) false

Relational Operators (2)

Question: True or false: a is true and b is false.

What is **`either a or b`**?

A) true

B) false

Decision Exercises

Exercise #1: Turning Boat - Create a water world with a boat.

- Make the boat turn one half turn to the right if a random number between 1 and 100 is even otherwise one half turn to the left if the random number is odd.
- All your code can be in the my first method.

Exercise #2: Turning Zamboni - Create a world with a zamboni.

- Create a method called `turn` for the zamboni that has a parameter called **num**.
- In the `turn` method, decide if **num** is between 50 and 100 inclusive. If it is, turn the zamboni around.
- Test your code with four method calls in my first method with the values 75, 50, 25, 150. Go forward 10 meters before each call.

Repetition using Loops

A ***loop*** allows the programmer to repeat statements.

There are two loops in Alice:

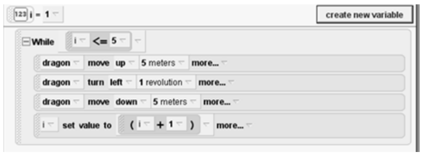
- the `While` statement
- the `Loop` statement

The `While` Statement

A `While` statement executes the statements it contains as long as its condition remains true.

- The condition is checked at the start of the loop and at the start of every loop ***iteration***.
- An infinite loop is a loop whose condition never becomes false, and the loop never ends.

Example:



```
123 i = 1                                   create new variable
While  i <= 5
    dragon  move  up  5 meters  more...
    dragon  turn  left  1 revolution  more...
    dragon  move  down  5 meters  more...
    i  set value to  ( i + 1 )  more...
```

Demonstration Exercise While Loop

Use `Collision.a2w`.

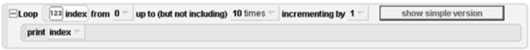
Tasks:

- Play the animation.
- Modify so that the vehicles and camera move twice as fast.
- Modify so that when trucks collide the cement truck says ouch!.
- Modify so that the cement truck's wheel breaks loose and continues to roll down the street.

The `Loop` Statement

The `Loop` statement allows you to control the exact number of repetitions.

The `Loop` statement uses a condition that tests the value of an integer counter variable and stops when its value reaches a specified end value.



```
Loop  123 index  from 0  up to (but not including) 10 times  incrementing by 1   show simple version
    print index
```

Demonstration Exercise Decisions

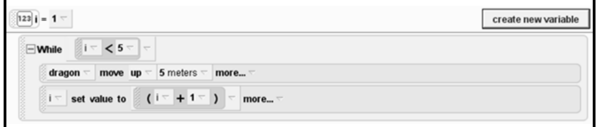
Use `SpeedingCar.a2w`.

Tasks:

- Play the animation.
- Modify so that the it calculates the total distance the car travels during its trip. Print out the result when the car comes to a stop.
- Modify so that the distance is updated while the car is moving. Display the result as a 3D text object in the window.
- Modify so that the speed is also continuously updated.

`While` Statement

Question: How many times does this `While` statement execute?

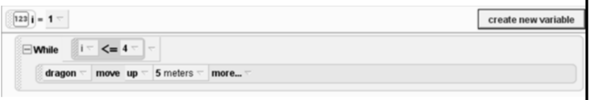


```
i = 1
While i < 5
    dragon move up 5 meters more...
    i set value to ( i + 1 ) more...
```

A) 0
B) 3
C) 4
D) 5
E) 6

`While` Statement (2)

Question: How many times does this `While` statement execute?

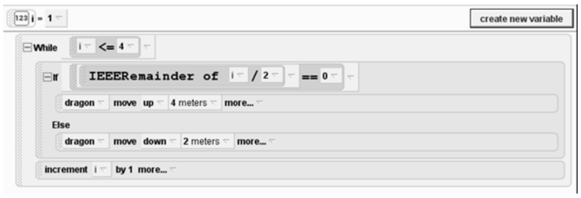


```
i = 1
While i <= 4
    dragon move up 5 meters more...
```

A) 0
B) 3
C) 4
D) 5
E) infinite

`While` Statement and Decisions

Question: How far off the ground is the dragon?

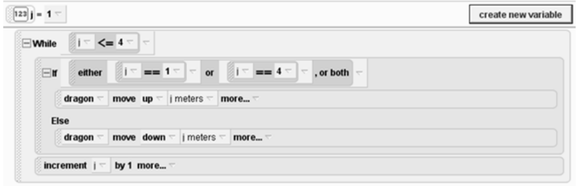


```
i = 1
While i <= 4
    If IEEERemainder of i / 2 == 0
        dragon move up 4 meters more...
    Else
        dragon move down 2 meters more...
    increment i by 1 more...
```

A) -2 **B)** 0 **C)** 2 **D)** 4 **E)** 8

`While` Statement and Decisions (2)

Question: How far off the ground is the dragon?

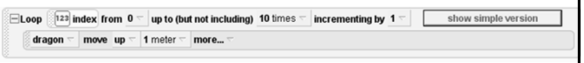


```
i = 1
While i <= 4
    If either i == 1 or i == 4 , or both
        dragon move up i meters more...
    Else
        dragon move down i meters more...
    increment i by 1 more...
```

A) -10 **B)** 0 **C)** 2 **D)** 4 **E)** 10

`Loop` Statement

Question: How many times does this loop execute?

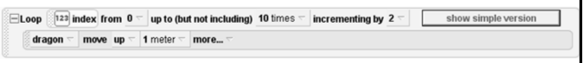


```
Loop index from 0 up to (but not including) 10 times incrementing by 1
    dragon move up 1 meter more...
```

A) 0
B) 9
C) 10
D) 11
E) infinite

`Loop` Statement (2)

Question: How many times does this loop execute?



```
Loop index from 0 up to (but not including) 10 times incrementing by 2
    dragon move up 1 meter more...
```

A) 0
B) 4
C) 5
D) 10
E) infinite

Exercises
Decisions and Loops

Exercise #1: Counting - Create a world with a 3D text object that counts from 1 to 10. Then counts down from 10 to 1.

Exercise #2: Jumping - Create a world where four characters perform jumps in unison five times. Each time give one of the characters a 30% chance to replace a jump with a full turn.

Exercise #3: Bouncing Ball - Create a world where a ball rolls off a table, bounces on the ground, and comes to rest. Decrease the height of the bounce by half each time. Move the ball away from the table slightly each bounce. Stop when the bounce height is small.

Conclusion

Decisions using the `If/Else` statement allow for controlling the flow of a program and decide when to execute certain statements.

Repetition of a block of statements can be done using:

- ◆`While` statement that executes statements until its condition is false
- ◆`Loop` statement that executes statements a specific number of times
- ◆An infinite loop is a loop whose condition never becomes false (the loop never ends).

Decisions using `If/Else` statements and repetition using `While/Loop` statements can be nested.

Objectives

Key terms: decision, loop, condition

Alice skills:

- ◆Making decisions with `If/Else`.
- ◆Conditions: relational operators and Boolean (logical) operators.
- ◆Nested `If/Else` decisions.
- ◆Repetition using the `While` statement.
- ◆Repetition using the `Loop` statement.
- ◆Nested repetition statements.
- ◆Using 3D Text boxes.

COSC 123
Computer Creativity

Events

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) Explore the different types of events in the Alice world.

2) Handle events for the keyboard and mouse.

☆

Event Processing

Most software has a graphical user interface (GUI) that contains buttons, text fields, lists, and menus.

An ***event*** is generated every time the user interacts with a user interface component by clicking a mouse or typing.

The code that responds to events is called ***event handling***.

There is a default event: **`When the world starts`**. This event is executed immediately when the world is played.

Reference
Alice Events

World Events

- `When the world starts`
  - ⇨ Occurs once when world first plays
- `While the world is running`
  - ⇨ Executes continually while world runs

Mouse Events

- `When the mouse is clicked on something`
  - ⇨ Occurs if button is pressed and released
- `While the mouse is pressed on something`
  - ⇨ Occurs while mouse button held on object
- `Let the mouse move the camera`
  - ⇨ Dragging the mouse moves the camera
- `Let the mouse orient the camera`
  - ⇨ Dragging mouse moves camera direction
- `Let the mouse move objects`
  - ⇨ Dragging mouse can move objects

Keyboard Events

- `When a key is typed`
  - ⇨ Occurs when key is pressed and released
- `While a key is pressed`
  - ⇨ Occurs while key is held
- `Let the arrow keys move the object`
  - ⇨ Arrows keys move an object

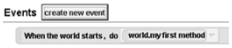
Condition Events

- `When a variable changes`
  - ⇨ Occurs when variable changes
- `While something is true`
  - ⇨ Occurs while expression is true
- `When something becomes true`
  - ⇨ Occurs once when true

World Events

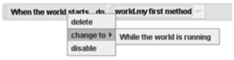
World events occur when a world starts running or as it continues to run.

`When the world starts` event happens once at start of world and usually calls **`my first method`**.



Events | create new event

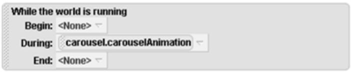
When the world starts, do | world.my first method



When the world starts, do | world.my first method

`While the world running` event has three parts:

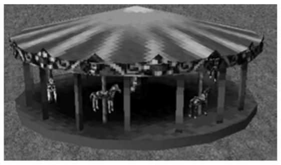
- ◆**`Begin`** and **`End`** are executed only once.
- ◆**`During`** section is executed repeatedly.

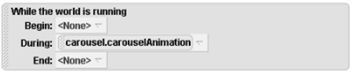


While the world is running
Begin: <None>
During: carousel.carouselAnimation
End: <None>

World Events
Repeating Actions While World Runs

The statements in the **`During`** section repeat for as long as the world is running. Note that this is background animation, and you can have other code to do other things at the same time.





While the world is running
Begin: <None>
During: carousel.carouselAnimation
End: <None>

Demonstration Exercise World Events

Use `carousel.a2w`.

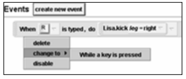
Tasks:

- ◆Make the carousel pause 1 second before starting to turn.
- ◆Create three planes that fly past the carousel one at a time.

Keyboard Events

Keyboard events allow us to respond to keyboard presses.

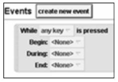
The **`When a key is typed`** fires when a key is pressed *and released*.



Events | create new event

When R is typed, do Lisa.kick *leg* = right

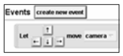
`While a key is pressed` events are fired when the user presses *and holds* any of the standard keyboard keys (digits, letters, space bar or enter key).



Events | create new event

While *any key* is pressed
Begin: <None>
During: <None>
End: <None>

The **`Let the arrow keys move an object`** event allows the user to move an object forward, backward, and turn right and left using the arrow keys.

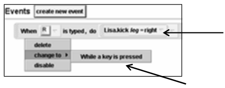


Events | create new event

Let ↑ ← ↓ → move camera

Keyboard Events Examples Dancer

We will make a dancer, Lisa, move when we press certain keys.

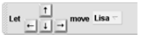


Events | create new event

When R is typed, do Lisa.kick *leg* = right

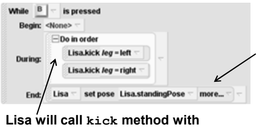
Lisa will call `kick` method (with right leg) when `R` is typed.

`While a key is pressed` is not in event menu, so must change a `When is typed` event.



Let ↑ ← ↓ → move Lisa

Allows Lisa to be moved with arrow keys.



While B is pressed
Begin: <None>
During: Do in order
  Lisa.kick *leg* = left
  Lisa.kick *leg* = right
End: Lisa set pose Lisa.standingPose more...

Capture a standing pose and `set pose` when user releases `B` so Lisa is not in an awkward position.

Lisa will call `kick` method with both legs in order while `B` is held.

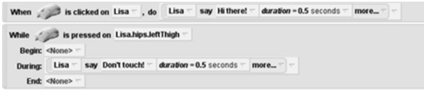
Mouse Events

Mouse events allow the user to interact with the mouse.

1) The **`When mouse is clicked on something event`** executes code when a user clicks on an object.

2) The **`While mouse is pressed on something event`** executes code in during section multiple times while the mouse button is held on an object. Begin/end sections are done once.

`When mouse is clicked on something event`



When is clicked on Lisa , do Lisa say Hi there! *duration* = 0.5 seconds more...

While is pressed on Lisa.hips.leftThigh
Begin: <None>
During: Lisa say Don't touch! *duration* = 0.5 seconds more...
End: <None>

`While mouse is pressed on something event`

Mouse Events Moving the Camera and World Objects

3) Using **`Let the mouse move the camera`** event allows the user to move the camera forward, backward, right, and left.

Events | create new event

Let move the camera

4) The **`Let the mouse orient the camera`** event allows rotating the camera.

Events | create new event

Let orient the camera

5) The **`Let the mouse move an object`** event allows the user to move any object or a list of objects that you specify.

Events | create new event

Let move Any Object

Demonstration Exercise Keyboard and Mouse Events

Use `Rockette.a2w`.

Tasks:

- ◆Make the dancer do left leg kicks and a head turn with the "`l`" key is typed.
- ◆Make the dancer do a left knee raise when the "`h`" key is pressed and a right knee raise when the "`g`" key is pressed.
- ◆Make it so that when the user holds "`b`" key the dancer repeatedly kicks her left leg and then her right leg.
- ◆Allow the user to move the dancer around with the arrow keys.
- ◆Make the dancer say "`Hi there!`" when you click on her.
- ◆Make the dancer say "`Don't touch!`" when you press (and hold) the mouse button over her legs.
- ◆Make the mouse able to move and orient the camera.

Condition Events

Condition events occur when the program state changes such as when the value of a variable changes.

Three types:

1) The **`When a variable changes event`** executes when the value of a variable that you provide changes.

2) The **`While something is true event`** occurs as long as a Boolean expression is true. (May be many times)

3) The **`When something becomes true event`** occurs once when a Boolean expression becomes true when it was previously false.

Events and Event Handling

Question: What happens when an event occurs and there is no event handler for it?

A) An event handler is created for it automatically.

B) The event is ignored and discarded.

C) An error occurs.

World Events

Question: True or false: A world always needs a **`When the world starts`** event.

A) true

B) false

World Events

Question: Which one of these statements is true?

A) The **`When the world starts`** event may be done multiple times.

B) The **`begin`** section of the **`While the world is running`** event is done every time the event occurs.

C) The **`during`** section of the **`While the world is running`** event may be performed multiple times.

D) The **`end`** section of the **`While the world is running`** event may be performed multiple times.

Keyboard Events
`When a key is typed` event

Question: The user is holding down the "**`b`**" key. How many times does the **`When a key is typed`** event occur?

A) 0

B) 1

C) 2

D) many times (depends how long the key is held for)

Keyboard Events
`While a key is pressed` event

Question: The user is holding down the " **`b`**" key. How many times does the **`While a key is pressed`** event occur?

A) 0

B) 1

C) 2

D) many times (depends how long the key is held for)

Mouse Events
`While mouse is pressed` event

Question: The user is holding down the mouse button on an object. How many times does the **`While mouse is pressed`** event occur?

A) 0

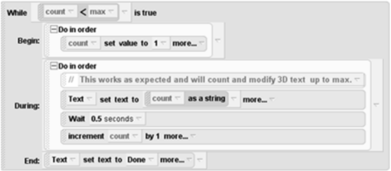
B) 1

C) 2

D) many times (depends how long the mouse button is held for)

Condition Events
`While something is true` event

Question: How many times does the during part of the event execute if count is originally 0 and max is 10?



```
While   count < max   is true
Begin:  Do in order
            count set value to 1 more...
During: Do in order
            // This works as expected and will count and modify 3D text  up to max.
            Text set text to count as a string more...
            Wait 0.5 seconds
            increment count by 1 more...
End:    Text set text to Done more...
```

A) 0

B) 1

C) 9

D) 10

Exercises
Events

Exercise #1: Shooting Tank - Create a world with a tank.

- ◆Move the tank around with the arrow keys.
- ◆Use `while a key is pressed` for the `L` and `R` keys to rotate the turret left and right.
- ◆Create a bullet (rectangle) and another object. When you press `space` shoot the bullet (forward 10 m). If it hits an object, have it say "Ouch!". Make sure you can shoot multiple times.

Exercise #2: Score a goal - Create a world with a net.

- ◆The left/right arrow keys move a net around a circle.
- ◆Have a ball randomly shoot out from the center of the circle in a random direction.
- ◆The goal is to catch the ball in the net. Keep score.
  - ⇨Hint: Use a dummy object for the puck's starting spot. Make the puck invisible (opacity 0) then move it back to starting spot.

Conclusion

Events are generated under various circumstances such as user interaction with the keyboard and mouse.

Event handlers allow a program to respond to events.

Some events fire (execute) only once while other events fire repeatedly as long as the action is occurring.

- ◆In Alice, events have **`While`** in name if execute multiple times and **`When`** if execute only once.

Event types in Alice:

- ◆***World events*** – apply to whole world (starting, running)
- ◆***Keyboard events*** – handle key presses
- ◆***Mouse events*** – handle mouse clicks ; can be used to move objects and camera
- ◆***Condition events*** – detect variable changes

Objectives

Key terms:

- ◆event, event handling

Alice skills:

- ◆Creating event handlers for four types of events: world, keyboard, mouse, condition.
- ◆Grouping objects in object tree.
- ◆Dummy objects for use with camera or object movement.

COSC 123
Computer Creativity

Lists and Arrays

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) Create and use lists and arrays as examples of data structures.

2) Use the `For all together` and `For all in order` statements with lists.

Managing Multiple Objects

Programs need to be able to deal with lots of data without becoming tedious to write.

A ***data structure*** holds and manages a group of objects using one variable name.

Alice has two data structures: ***lists*** and ***arrays***.

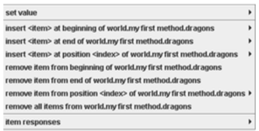
Lists

A ***list*** is a linear data structure whose size can grow.

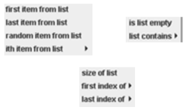
A list has methods that allow you to:

- ◆retrieve an item at a particular index in the list
- ◆add an item to the front, end, or middle of the list
- ◆remove an item from the list
- ◆determine the size (number of items in the list)

List methods

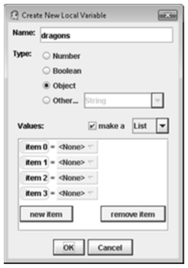

set value
insert <item> at beginning of world.my first method.dragons
insert <item> at end of world.my first method.dragons
insert <item> at position <index> of world.my first method.dragons
remove item from beginning of world.my first method.dragons
remove item from end of world.my first method.dragons
remove item from position <index> of world.my first method.dragons
remove all items from world.my first method.dragons
item responses

List functions



Creating a List

A list is created in Alice by using the `create new variable` button then check `make a List` box. Then, you can add items to your list by clicking on the `new item` button.



Create New Local Variable

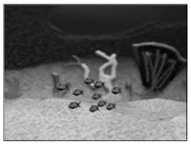
Name: dragons

Type: ○ Number ○ Boolean ● Object ○ Other... String

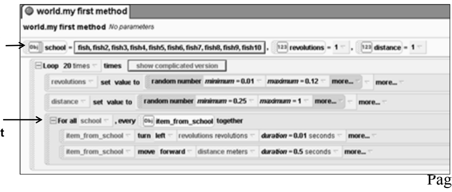
Values: ☑ make a List

item 0 = <None>
item 1 = <None>
item 2 = <None>
item 3 = <None>

List Example



- ➢ In this example there are ten objects in a list.
- ➢`For all together` moves all objects in the list at the same time.



```
world.my first method
world.my first method No parameters
Obj school = fish, fish2, fish3, fish4, fish5, fish6, fish7, fish8, fish9, fish10 , 123 revolutions = 1 , 123 distance = 1
  Loop 20 times times  show complicated version
    revolutions set value to random number minimum = 0.01 maximum = 0.12 more... more...
    distance set value to random number minimum = 0.25 maximum = 1 more... more...
    For all school , every Obj item_from_school together
      item_from_school turn left revolutions revolutions duration = 0.01 seconds more...
      item_from_school move forward distance meters duration = 0.5 seconds more...
```

List object school

For all together statement

Lists
For all together and in order

The `For all together` statement performs the operations on all objects at the same time.

The `For all in order` statement performs the operations one at a time in the order the objects are in the list.

Demonstration Exercise
Lists

Use `SchoolOfFish.a2w`.

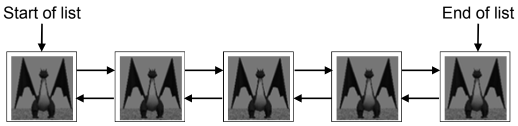
Tasks:

- ◆Add a new fish to the school.
- ◆Make a fish disappear from the school with 50% probability every loop iteration.
  - ⇨Start off by making first fish in list disappear then try to make any fish in the list disappear.
  - ⇨Note: Make an object disappear by setting its opacity to 0%.
- ◆Add a shark that swims into the school and eats a random fish in the list every time it swims into the school.

List Structure

A list is a flexible linear data structure as you can add entries any where in the list and the list can grow to any size.

This flexibility is achieved by ***linking*** each entry together. The downside to this flexibility is that it is less efficient to get to any particular element in the list as you must ***traverse*** the list.

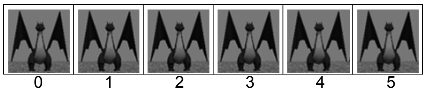


Arrays

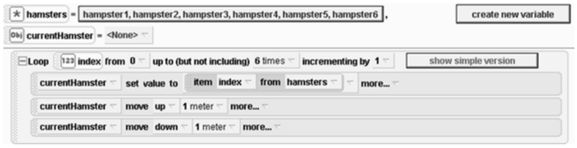
An ***array*** is a fixed size linear data structure that stores multiple items. The size of the array is determined when it is created.

An element in an array can be accessed directly by its index without visiting other items. The `For all together` and `For all in order` statements cannot be applied to arrays. Access each element in an array using an `Loop` instead.

Array with 6 elements:



Arrays Example



```
hamsters = hampster1, hampster2, hampster3, hampster4, hampster5, hampster6 ,
currentHamster = <None>
Loop index from 0 up to (but not including) 6 times incrementing by 1
    currentHamster set value to item index from hamsters more...
    currentHamster move up 1 meter more...
    currentHamster move down 1 meter more...
```

Demonstration Exercise
Arrays

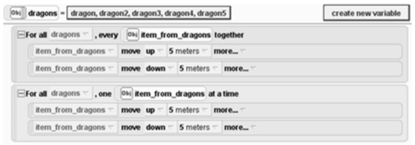
Use `hamster.a2w`.

Tasks:

- ◆Add two more hamsters to the array.
- ◆Make all hamsters jump up in order.
- ◆Make the hamsters jump up in reverse order.
- ◆Make every second hamster jump.
- ◆Make a random hamster jump. Make it so a mouse click on a hamster will whack it. Keep score.

Lists
For all together and For all in order

Question: What will this code do?



```
Obj dragons = dragon, dragon2, dragon3, dragon4, dragon5
For all dragons , every Obj item_from_dragons together
    item_from_dragons move up 5 meters more...
    item_from_dragons move down 5 meters more...
For all dragons , one Obj item_from_dragons at a time
    item_from_dragons move up 5 meters more...
    item_from_dragons move down 5 meters more...
```

A) Move each dragon up/down separately then move each dragon up/down together.

B) Move each dragon up/down together then move each dragon up/down separately.

C) Move each dragon up/down together only.

D) Move each dragon up/down separately only.

List Index

Question: Given the list below at what index is "A"?

D F G X A B C

A) 4
B) 5
C) 6
D) 7

List Insert

Question: Given the list below what is the result if insert Z at index 2?

D F G X A B C

A) Z D F G X A B C
B) D Z F G X A B C
C) D F Z G X A B C
D) D F G Z X A B C

Arrays versus List

Question: Your program must store 10000 objects. You will never have more than 10000 objects, and you need the ability to get a particular object quickly.

What should you use an array or a list?

A) Array
B) List

Exercises
Lists and Arrays

Exercise #1: Soldier March

- ◆Create a group of 12 soldiers in three rows of four.
- ◆Make them all move forward, turn left half a turn, then move forward again at the same time.
- ◆Make each row of soldiers do the actions above in turn.

Exercise #2: Pick an object - Create a world with five objects.

- ◆ Allow the user to click on the objects.
- ◆The order the objects are clicked are the order they should be put into an array.
- ◆After the objects are put in an array, print out the contents of the array.
- ◆Bonus: Instead of printing, line the objects up in the order they were picked.

Conclusion

Lists and ***arrays*** are data structures that hold and manage a group of objects using one variable name.

The elements stored in lists and arrays can be accessed using a numeric index, which starts at 0.

When performing list operations, list elements are shifted to make room for new items or to close the gap after a removal.

An array is generally more efficient than a list, but it has a fixed size.

Objectives

Key terms: data structure, list, array

Alice skills:

- Create and use lists and arrays.
- Use list methods and `For all together` and `For all in order`.
- Importing and exporting objects.

COSC 123
Computer Creativity

Introduction to Java

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) Introduce Java, a general-purpose programming language, and compare it with Alice

2) Examine the Eclipse development environment for developing Java programs

3) Execute our first Java program and analyze its basic contents

4) Learn how to read input, write to the screen, declare and use variables, and perform basic calculations in Java

Introduction to Java

Java is a general-purpose, object-oriented language developed in 1991 by a group led by James Gosling and Patrick Naughton of Sun Microsystems.

Major advantages of Java:

- ◆Can run on almost any type of machine.
- ◆Popular language for web and system development.
- ◆Good teaching language because many issues such as memory management are hidden.

Java is an ***interpreted***, rather than compiled, language. This makes it portable but also affects performance for some applications.

The Java Virtual Machine (JVM)

The ***Java Virtual Machine (JVM)*** is a program that executes a Java program on an individual machine.

After the Java compiler compiles your program:

- ◆your program is in Java byte form which is a set of instructions for the JVM to execute (not the same as machine code)

When you run your program:

- ◆the JVM is started by the operating system
- ◆the JVM loads your program and begins executing it
- ◆each byte in your compiled Java program is either an instruction or data used by the JVM
- ◆the JVM translates instructions in your program to the appropriate machine code for the machine it is running on

The JVM is effectively a ***virtual machine*** in your computer.

Java and Alice

Java and Alice perform the same operations using different syntax.

Operation	Alice	Java
Assignment	Set value	=
Arithmetic	+, -, *, /	+, -, *, /
Remainder	IEEERemainder	%
Relational	<, <=, >, >=, ==, !=	<, <=, >, >=, ==, !=
Logical	Not, both a and b, either a or b or both	! (not), a && b (and), a || b (or)
Decisions	If/else	If/else
Repetition	Loop, While	for, while

Eclipse

It is possible to write Java programs using any text editor and compile them using the Java compiler.

An ***integrated development environment*** makes it easier to write code, find errors, and run your programs.

We will use the ***Eclipse*** environment in this course.

- ◆Eclipse is a generic, extensible development environment that can be used for Java and other languages.
- ◆Eclipse makes coding easier with automatic error checking, code completion, and source debugging.
- ◆Eclipse will **NOT** make it easier to figure out **WHAT** to write, but it will make **HOW** to write it easier.

Eclipse Initial Setup
Creating a Workspace and a Project

A ***workspace*** is the place where Eclipse will store all of your projects.

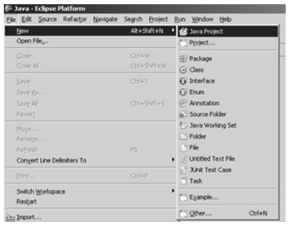
- ◆ You will be prompted for your workspace on start up if you have not selected one.

Create a new workspace on **`F:`** with a directory name **`workspace`**.

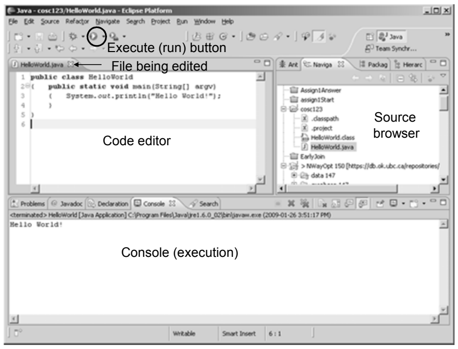
A ***project*** is a group of program files for some purpose. We will create a sample project called **`cosc123`**. You will also create projects for each assignment.

- ◆ Give the project a name and click finish. Ignore all options for now.

Create a New Project using
`File->New->Java Project`



Eclipse Main Screen



```
public class HelloWorld
{   public static void main(String[] argv)
    {   System.out.println("Hello World!");
    }
}
```

```
<terminated> HelloWorld [Java Application] C:\Program Files\Java\jre1.6.0_02\bin\javaw.exe (2009-01-26 3:51:17 PM)
Hello World!
```

Eclipse
Perspectives and Views

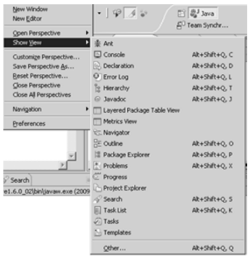
A ***perspective*** is an organization of views to accomplish a certain task (debugging, coding, etc.).

- ◆ The two perspectives we will use are Java and Debugging.
- ◆ Eclipse remembers how you place the views in each perspective.

A ***view*** is a window on the screen associated with a task.

- ◆ The major views are:
  - ⇨ Navigator – shows files in project
  - ⇨ Console – shows program output
  - ⇨ Problems – shows errors in code
- ◆ You may open, close, and organize views in each perspective.

Selecting Eclipse views using
`Window->Show View`



Eclipse
Creating a Program File

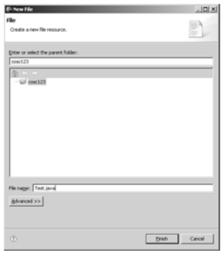
To create a program code file, select **`File->New->File`** or **`File->New->Class`** and provide a folder and file name.

The other choice is to right click on a folder in the navigation view and select **`New->File`**.

Type the file name (should end with **`.java`**) and click **`Finish`**.

To edit this file, double click on it, and it will open in the editor.

Creating a new file using
`File->New->File`

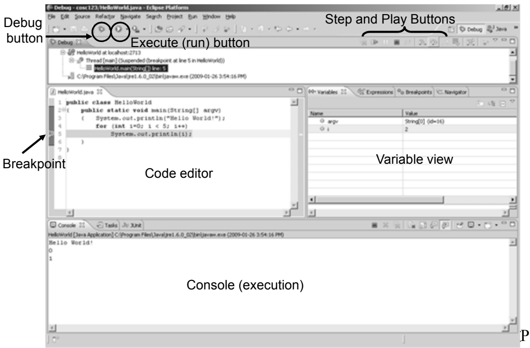


Create a new file resource.

Enter or select the parent folder: cosc123

File name: Test.java

Eclipse
Debugging and Breakpoints


Debug - cosc123/HelloWorld.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug button
Step and Play Buttons
Execute (run) button
Debug
Java
HelloWorld at localhost:2713
Thread [main] (Suspended (breakpoint at line 5 in HelloWorld))
HelloWorld.main(String[]) line: 5
C:\Program Files\Java\jre1.6.0_02\bin\javaw.exe (2009-01-26 3:54:16 PM)
HelloWorld.java
Breakpoint
Code editor
Variables
Expressions
Breakpoints
Navigator
Variable view
Console
Tasks
JUnit
Console (execution)

```
public class HelloWorld
{   public static void main(String[] argv)
    {   System.out.println("Hello World!");
        for (int i=0; i < 5; i++)
            System.out.println(i);
    }
}
```

Name	Value
argv	String[0] (id=16)
i	2

```
HelloWorld [Java Application] C:\Program Files\Java\jre1.6.0_02\bin\javaw.exe (2009-01-26 3:54:16 PM)
Hello World!
0
1
```

Debugging Java Programs

When you write programs, it is very rare that you get the program correct the first time. There are two types of errors:

- ◆1) ***Compile-time errors*** - are language syntax or structure errors detected by the compiler when it compiles your program
  - ⇨A program will not run until all compile-time errors are corrected.
- ◆2) ***Run-time errors*** - are errors that occur while the program is running and often result in incorrect results or program crashes.
  - ⇨Run-time errors are harder to detect because they result from a flaw in your algorithm which is syntactically correct.

Demonstration Exercise
Running HelloWorld in Eclipse

1) Start Eclipse.
2) Create your workspace on **`F:`**.
3) Create a new project called `cosc123`.
4) Download or type in the file **`HelloWorld.java`**.
5) Run the program.

Introduction to Java
Overview

To program in Java you must follow a set of rules for specifying your commands. This set of rules is called a ***syntax***.

Important general rules of Java syntax:

- ◆Java is ***case-sensitive***.
  - ⇨`Main()` is not the same as `main()` or `MAIN()`.
- ◆Java accepts ***free-form layout***.
  - ⇨Spaces and line breaks are not important except to separate words.
  - ⇨You can have as many words as you want on each line or spread them across multiple lines.
  - ⇨However, you should be consistent and follow the programming guidelines given for assignments.
    - It will be easier for you to program and easier for the marker to mark.

Introduction to Java
Your First Java Program

```
public class HelloWorld
{  public static void main(String[] argv)
   {  System.out.println("Hello World!");
   }
}
```

To create this program:

- ◆Create a file called **`HelloWorld.java`** in an Eclipse project and type in the code.

To compile and run this program:

- ◆Press the start button (green arrow) in Eclipse.
- ◆If the code is correct, the program will run, otherwise it will show errors that you must fix first.

Introduction to Java
Your First Java Program - Analysis

```
public class HelloWorld
{  public static void main(String[] argv)
   {  System.out.println("Hello World!");
   }
}
```

The first line of code:

- ◆says you want to create a ***class*** called **`HelloWorld`**
  - ⇨**`HelloWorld`** is the name you have chosen for your class.
    - Class names normally begin with a capital letter.
  - ⇨A class is a blue-print for an object.
    - An object is something that we store or modify in our program.
  - ⇨In this case, class **`HelloWorld`** is the name of our entire program.
    - Notice that we saved the program as `HelloWorld.java` (this is important!)
- ◆the "**`public`**" keyword means the class is usable by the public

Introduction to Java
Your First Java Program - Analysis (2)

```
public class HelloWorld
{  public static void main(String[] argv)
   {  System.out.println("Hello World!");
   }
}
```

The "`{`" and "`}`" characters are used to group commands.

- ◆The first pair of brackets shows what is in class `HelloWorld`.
  - ⇨In this case, the method `main()` is part of the `HelloWorld` class.
- ◆The second pair of brackets indicates what is contained in the method called `main()`.
  - ⇨The statement `System.out.println("Hello World!");` is part of the `main()` method.
- ◆You must ensure that your brackets are properly matched.

Introduction to Java
Your First Java Program - Analysis (3)

```
public class HelloWorld
{  public static void main(String[] argv)
   {  System.out.println("Hello World!");
   }
}
```

The second line of code:

- ◆defines a ***method*** called `main()`
- ◆A ***method*** is a set of commands that tells Java what to do.
  - ⇨Every method must be inside a class in Java.
    - The `main()` method is in the `HelloWorld` class.
  - ⇨The `main()` method is the first method executed in your program.
    - The `main()` method must be in your program for it to work.
    - Memorize the syntax for this method. You will not understand it until later in the course.
- ◆The statements inside the brackets are the commands executed when the method is run.

Introduction to Java
Your First Java Program - Analysis (4)

```
public class HelloWorld
{  public static void main(String[] argv)
   {  System.out.println("Hello World!");
   }
}
```

The third line of code:

- ◆contains a statement executed when the `main()` method is run
- ◆This command calls a built-in method called `println()`.
  - ⇨The `println()` method is in the `System.out` class.
- ◆The method is called with a parameter: `"Hello World!"`.
  - ⇨The parameter to this method is what you want to print.
  - ⇨The parameter is contained in quotes ("") because it is text.
- ◆Note that each statement ends with a semi-colon ("**;**").
- ◆The brackets ("**{**","**}**") denote the start and end of the method.

Output Text to the Screen
`System.out.println`

The `println` method prints output to the screen.

- ⇨The `println` method accepts one `String` variable as output.
- ⇨You can use the + (concatenation) to build an output string that consists of many parts.
- ⇨The `System.out.print` method does not advance to the next line.

Example:

```
public class ThreeplusFour
{  public static void main(String[] args)
   {  System.out.println("3 + 4 is: ");
      System.out.println(3+4);
      System.out.println("6 + 9 is: "+(6+9));
   }
}
```

Question: What is the output of this program? Why?

Reading Data from the User
The Scanner Class

The `Scanner` class reads data entered by the user. Methods:

- ◆`int nextInt()` – reads next integer
- ◆`double nextDouble()` – reads next floating point number
- ◆`String next()` – reads String (up to separator)
- ◆`String nextLine()` – reads entire line as a String

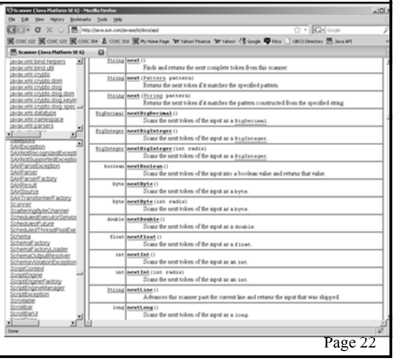
To use must import `java.util.Scanner`.

```
import java.util.Scanner;
public class AddTwoNum
{  public static void main(String[] argv)
   {  // Code reads and adds two numbers
      Scanner sc = new Scanner(System.in);
      int num1 = sc.nextInt();
      int num2 = sc.nextInt();
      int result = num1+num2;
      System.out.println(num1+" + "+num2+" = "+result);
   }
}
```

The Java API

The Java API (Application Programming Interface) defines all the built-in class and methods in Java that you can use.

We are using the Java 6 API at: http://java.sun.com/javase/6/docs/api/



`String`	`next()` Finds and returns the next complete token from this scanner.
`String`	`next(Pattern pattern)` Returns the next token if it matches the specified pattern.
`String`	`next(String pattern)` Returns the next token if it matches the pattern constructed from the specified string.
`BigDecimal`	`nextBigDecimal()` Scans the next token of the input as a `BigDecimal`.
`BigInteger`	`nextBigInteger()` Scans the next token of the input as a `BigInteger`.
`BigInteger`	`nextBigInteger(int radix)` Scans the next token of the input as a `BigInteger`.
`boolean`	`nextBoolean()` Scans the next token of the input into a boolean value and returns that value.
`byte`	`nextByte()` Scans the next token of the input as a `byte`.
`byte`	`nextByte(int radix)` Scans the next token of the input as a `byte`.
`double`	`nextDouble()` Scans the next token of the input as a `double`.
`float`	`nextFloat()` Scans the next token of the input as a `float`.
`int`	`nextInt()` Scans the next token of the input as an `int`.
`int`	`nextInt(int radix)` Scans the next token of the input as an `int`.
`String`	`nextLine()` Advances this scanner past the current line and returns the input that was skipped.
`long`	`nextLong()` Scans the next token of the input as a `long`.

Practice Questions

1) Create a program to ask the user for two numbers, subtract them, and write out the answer.

2) Create a program to ask for a first name then a last name. Output the full name in the form: lastname, firstname.

☆

Values, Variables, and Locations

A ***value*** is a data item that is manipulated by the computer.

A ***variable*** is the name that the programmer users to refer to a location in memory.

A ***location*** has an address in memory and stores a value.

IMPORTANT: The *value* at a given location in memory (named using a variable name) can change using initialization or assignment.

Values, Variables, and Locations Example

We want to store a number that represents the total order value.

Step #1: ***Declare*** the variable by giving it a name and a type.

```
int total;
```

- The computer allocates space for the variable in memory (at some memory address). Every time we give the name `total`, the computer knows what data item we mean.
- The base types we will use are: int, double, and char.

Variable Name Lookup Table

Name	Location	Type
total	16	number

Memory

16	????????
20	
24	
28	

Values, Variables, and Locations Example (2)

Step #2: Initialize the variable to have a starting value

- If you do not initialize your variable to a starting value when you first declare it, the value of the variable is initialized to 0 (for numbers).

Example:

```
total = 1;
```

Variable Name Lookup Table

Name	Location	Type
total	16	number

Memory

16	1
20	
24	
28	

Values, Variables, and Locations Example (3)

Step #3: Value stored in location can be changed throughout the program to whatever we want using ***assignment*** ("**=**" symbol).

```
total = total * 5 + 20;
```

Variable Name Lookup Table

Name	Location	Type
total	16	number

Memory

16	25
20	
24	
28	

Variable Rules

Variables are also called identifiers. An ***identifier*** is a name that ***begins with a letter*** or underscore and cannot contain spaces.

- Every variable in a program must be declared before it is used.
- Variable names **ARE** case-sensitive. Numbers are allowed (but not at the start). Only other symbol allowed is underscore ('_');
- Beware of declaring two variables with the same name.
- Use meaningful variable names.
- Reserved words cannot be used for variable names.
- A ***constant*** is a variable which cannot change in your program. We use the keyword `final` to indicate a constant.

```
final double PST = 0.07;   // Constant
```

- You can declare multiple variables in the same statement:

```
int total = 0, count = 5;
```

The Assignment Statement

An ***assignment statement*** changes the value of a variable.

- ⇨ The variable on the left-hand side of the = is assigned the value from the right-hand side.
- ⇨ The value may be changed to a constant, to the result of an expression, or to be the same as another variable.
- ⇨ The values of any variables used in the expression are always their values before the start of the execution of the assignment.

Examples:

```
int A, B;

A = 5;
B = 10;
A = 10 + 6 / 2;
B = A;
A = 2*B + A - 5;
```

Question: What are the values of A and B?

Expressions

An ***expression*** is a sequence of operands and operators that yield a result. An expression contains:

- ***operands*** - the data items being manipulated in the calculation
  - ⇨ e.g. 5, “Hello, World”, myDouble
- ***operators*** - the operations performed on the operands
  - ⇨ e.g. +, -, /, *, % (modulus - remainder after integer division)

An operator can be:

- ***unary*** - applies to only one operand
  - ⇨ e.g. d = - 3.5;   // “-” is a unary operator, 3.5 is the operand
- ***binary*** - applies to two operands
  - ⇨ e.g d = e * 5.0;   // “*” is binary operator, e and 5.0 are operands

Expressions - Operator Precedence

Each operator has its own priority similar to their priority in regular math expressions:

- ◆1) Any expression in parentheses is evaluated first starting with the inner most nesting of parentheses.
- ◆2) Unary + and unary - have the next highest priorities.
- ◆3) Multiplication and division (*, /, %) are next.
- ◆4) Addition and subtraction (+,-) are then evaluated.

Strings

Strings are sequences of characters inside double quotes.

Example:

```
String personName = "Ramon Lawrence";
personName = "Joe Smith";
```

Question: What is the difference between these two statements?

Strings are objects. Objects have methods.

The ***concatenation operator*** is used to combine two strings into a single string. The notation is a plus sign '**+**'.

```
String firstName = "Ramon", lastName = "Lawrence";
String fullName = firstName+lastName;
```

General Syntax Rules: Comments

Comments are used by the programmer to document and explain the code. Comments are ignored by the computer.

There are two choices for commenting:

- ◆1) One line comment: put "`//`" before the comment and any characters to the end of line are ignored by the computer.
- ◆2) Multiple line comment: put "`/*`" at the start of the comment and "`*/`" at the end of the comment. The computer ignores everything between the start and end comment indicators.

Example:

```
/* This is a multiple line
     comment.
With many lines. */

// Single line comment
// Single line comment again
d = 5.0; // Comment after code
```

Declaration/Initialization Example

```
public class TestInit
{  public static void main(String[] args)
   {  final double d = 5.0;   // d is a constant = 5
      double e;               // Declare double var. e
      int j;                  // Declare int var. j
      String s;               // Declare string var. s

      System.out.println(d);  // Prints 5.0
      System.out.println(j);  // Would not compile!
      j = 25;
      System.out.println(j);  // Prints 25
      s="Test";
      System.out.println(s);  // Prints Test
      e=d;
      System.out.println(e);  // Prints 5.0;
      e=d+20000.5;            // Note: No commas
      System.out.println(e);  // Prints 20005.5;
   }
}
```

Importing Classes

Java provides many classes organized into ***packages***.

To use a class, you must import it. The import syntax is:

```
import packageName.ClassName;
import java.lang.Math;  // Import Math class
                        // java.lang is package
import java.lang.*;     // Import all classes in package
```

The Math class contains methods such as square root or rounding.

```
int num = Math.round(3.5);    // Returns 4
```

Math Operations
Import & Math Function Example

```
import java.lang.Math;

public class TestMath
{  public static void main(String[] args)
   {  double d = 5.0,e=1.5,f;
      int j = 25,k;

      f = -d*e;
      System.out.println(f);            // Prints -7.5
      f = Math.pow(d,2);
      System.out.println(f);            // Prints 25.0
      k = (int) Math.sqrt(j);
      System.out.println(k);            // Prints 5
      System.out.println(Math.sqrt(j)); // Prints 5.0
      d=d*e+j+Math.exp(j);
      System.out.println(d);            // Prints 7.2E10
      System.out.println(k);            // Prints 1
      System.out.println(Math.round(e));// Prints 2
   }
}
```

Compile vs. Run-time Errors

Question: A program is supposed to print the numbers from 1 to 10. It actually prints the numbers from 0 to 9. What type of error is it?

A) Compile-time error

B) Run-time error

Variables – Basic Terminology

Question: Of the following three terms, what is most like a ***box***?

A) value

B) variable

C) location

Variables - Definitions

Question: Which of the following statements is correct?

A) The location of a variable may change during the program.

B) The name of a variable may change during the program.

C) The value of a variable may change during the program.

Variables – Correct Variable Name

Question: Which of the following is a valid Java variable?

A) `aBCde123`

B) `123test`

C) `t_e_s_t!`

Assignment

Question: What are the values of `A` and `B` after this code?

```
int A, B;

A = 2;
B = 4;
A = B + B / A;
B = A * 5 + 3 * 2;
```

A) `A = 6, B = 36`

B) `A = 4, B = 26`

C) `A = 6, B = 66`

String Concatentation

Question: What is the value of result after this code?

```
String st1="Joe", st2="Smith";

String result = st1 + st2;
```

A) `"Joe Smith"`

B) `"JoeSmith"`

String Concatentation (2)

Question: What is the result after this code?

```
String st1="123", st2="456";

String result = st1 + st2;
```

A) `579`

B) `"579"`

C) `"123456"`

Code Output

Question: What is the output of this code if user enters 3 and 4?

```
public class AddTwoNum
{  public static void main(String[] argv)
   {  // Code reads and adds two numbers
      Scanner sc = new Scanner(System.in);
      int num1 = sc.nextInt();
      int num2 = sc.nextInt();
      int result = num1+num2;
      System.out.println(num2+" + "+num1+" = "+result);
   }
}
```

A) 3 + 4 = 7

B) 4 + 3 = 7

C) 4 + + + 3 + = + 7

D) Code has errors and will not compile.

Practice Questions

1) Write a Java program that prompts for a number and outputs the square root of that number.

2) Write a program to read three numbers and then print their sum.

Conclusion

Java is a general-purpose language for building programs. Its performs similar operations as Alice but with different syntax.

Eclipse is a development environment for Java programs. Eclipse is used to write, debug, and run programs.

A Java program consists of ***statements*** separated by semi-colons. ***Variable declaration*** statements require a variable name and type. A string is an example of an object.

Input can be retrieved using the `Scanner` class and data printed using `System.out.println().`

Classes are ***imported*** into the program when required.

Objectives

Key terms:
- JVM, Eclipse, IDE
- variable, value, location, assignment

Java skills:
- Create a workspace and project in Eclipse.
- Create and run Java programs using Eclipse.
- Basic debugging and breakpoints
- Java syntax: statements, variables, expressions, comments
- Output using `System.out.println`
- Input using and `Scanner` class
- Using the Java API for reference
- Strings and concatenation
- Importing classes from packages

Detailed Objectives

- Comparison of Java and Alice syntax for operations.
- Eclipse definitions: workspace, project, perspective, view
- Compile vs. run-time errors and debugging
- Declaring variables and assigning values to variables
- Using the Eclipse IDE
- Output and input of data
- Definitions: declare, assignment, identifier, constant, expression

COSC 123
Computer Creativity

Java Decisions and Loops

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) A decision is made by evaluating a condition in an if/else statement and performing certain actions depending if the condition is true or false.

2) Repetition is performed by using loops that repeat a set of statements multiple times.

Making Decisions

Decisions are used to allow the program to perform different actions in certain conditions.

- For example, if a person applies for a driver’s license and is not 16, then the computer should not give them a license.

To make a decision in a program we must do several things:

- 1) Determine the ***condition*** in which to make the decision.
  - In the license example, we will not give a license if the person is under 16.
- 2) Tell the computer what to do if the condition is true or false.
  - A decision always has a *Boolean value* or true/false answer.

The syntax for a decision uses the ***if*** statement.

Making Decisions
Performing Comparisons

Relational operators compare two items called operands.

- Syntax: *operand1 operator operand2*

Comparison operators in Java:

>	- Greater than
>=	- Greater than or equal
<	- Less than
<=	- Less than or equal
==	- Equal (Note: Not "=" which is used for assignment!)
!=	- Not equal

The result of a comparison is a ***Boolean value*** which is either ***true*** or ***false***.

Making Decisions
Example Comparisons

```
int j=25, k = 45;
double d = 2.5, e=2.51;
boolean result;

result = (j == k);   // false
result = (j <= k);   // true
result = (d == e);   // false (rounding!)
result = (d != e);   // true
result = (k >= 25);  // true
result = (25 == j);  // true
result = (j > k);    // false
result = (e < d);    // false
j = k;
result = (j == k);   // true

// Note: Never compare doubles using "==" due to
//        precision and rounding problems.
```

Making Decisions
Comparing Strings and Objects

Comparing strings and objects is different than numbers.

- Operators such as <, > are not useful for strings and objects.
- Operator “==“ is defined but it is not very useful.
  - The “==“ operator compares if two string/object references refer to the same object **NOT** if the string/object has the same value.

Compare strings using **equals()** and **compareTo()** methods:

```
String str1 = "abc", str2="def";
str1.equals(str2);       // True if str1 is equal to str2
str1.equalsIgnoreCase(str2); // Comparison without case
str1.compareTo(str2); // will be < 0 if str1 < str2
str1.compareTo(str2); // will be > 0 if str1 > str2
str1.compareTo(str2); // will be = 0 if str1 equals str2
```

Making Decisions
Example String Comparisons

```
public class TestStringComparisons
{  public static void main(String[] args)
   {  String st1 = "Hello", st2="Hell", st3="Test",st4;

      System.out.println(st1.equals(st2));      // false
      System.out.println(st1.compareTo(st2));   // 1
      System.out.println(st2.compareTo(st1));   // -1
      System.out.println(st3.compareTo(st1));   // 12
      System.out.println(st3.compareTo("ABC")); // 19
      st4 = st1.substring(0,4);
      System.out.println(st2.equals(st4));      // true
      System.out.println(st2.compareTo(st4));   // 0
      st4 = st4.toUpperCase();
      st2 = st2.toLowerCase();
      System.out.println(st2.equals(st4));      // false
      System.out.println(st2.equalsIgnoreCase(st4));
                                                //true
   }
}
```

String Comparisons

Question: What is the output of this code?

```
String str, str2;
Scanner sc = new Scanner(System.in);
str = sc.nextLine(); // User enters: abc
str2 = sc.nextLine();// User enters: abc

if (str == str2)
   System.out.print("equal");
else
   System.out.print("not equal");
```

A) `equal`

B) `not equal`

Making Decisions
If Statement

To make decisions with conditions, we use the ***if*** statement.

- If the condition is true, the statement(s) after `if` are executed otherwise they are skipped.
- If there is an **`else`** clause, statements after **`else`** are executed if the condition is false.

Syntax:

```
if (condition)          OR        if (condition)
   statement;                        statement;
                                  else
                                     statement;
```

Example:

```
if (age > 19)           OR        if (age > 19)
   teenager=false;                   teenager=false;
                                  else
                                     teenager=true;
```

Making Decisions
Block Syntax

Currently, using our if statement we are only allowed to execute one line of code (one statement).

- What happens if we want to have more than one statement?

We use the ***block syntax*** for denoting a multiple statement block. A block is started with a "**{**" and ended with a "**}**".

- All statements inside the brackets are grouped together.

Example:

```
if (age > 19)
{  teenager=false;
   hasLicense=true;
   ...
}
```

We will use block statements in many other situations as well.

Making Decisions
If Statement Example

```
int age;
boolean teenager, hasLicense=false;
System.out.print("Enter your age: ");
Scanner sc = new Scanner(System.in);
age = sc.nextInt();

if (age > 19)
{  teenager = false;
   hasLicense = true;
}
else if (age < 13)
{  teenager = false;
   hasLicense = false;
}
else
{  teenager = true;  // Do not know if have license
}
System.out.println("Is teenager: "+teenager);
System.out.println("Has license? "+hasLicense);
```

Making Decisions

Question: What is the output of this code?

```
int num=10;

if (num > 10)
   System.out.println("big");
else
   System.out.println("small");
```

A) `big`

B) `small`

C) `bigsmall`

Making Decisions (2)

Question: What is the output of this code?

```
int num=9;

if (num != 10)
    System.out.print("big");
System.out.println("small");
```

A) `big`

B) `small`

C) `bigsmall`

Making Decisions (3)

Question: What is the output of this code?

```
int num=10;

if (num == 10)
{   System.out.print("big");
    System.out.println("small");
}
```

A) `big`

B) `small`

C) `bigsmall`

Making Decisions
Nested If Statement

We ***nest*** `if` statements for more complicated decisions.

- Verify that you use blocks appropriately to group your code!

Example:

```
if (age > 16)
{  if (sex == "male")
   {  System.out.println("Watch out!");
   }
   else
   {  System.out.println("Great driver!");
   }
}
else
{  System.out.println("Sorry! Too young to drive.");
}
```

Making Decisions
Nested If Statement Example

```
public class NestedIf
{  public static void main(String[] args)
   {  double salary, tax;
      String married;
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter M=married, S=single: ");
      married=sc.next();
      System.out.print("Enter your salary: ");
      salary=sc.nextDouble();

      if (married.equals("S"))
      {  // Single person
         if (salary > 50000)
            tax = salary*0.5;
         else if (salary > 35000)
            tax = salary*0.45;
         else
            tax = salary*0.30;
      }  // End if single person
```

Making Decisions
Nested If Statement Example

```
      else if (married.equals("M"))
      {  // Married person
         if (salary > 50000)
            tax = salary*0.4;
         else if (salary > 35000)
            tax = salary*0.35;
         else
            tax = salary*0.20;
      }  // End if married person
      else  // Invalid input
         tax = -1;

      if (tax != -1)
      {  System.out.println("Salary: "+salary);
         System.out.println("Tax: "+tax);
      }
      else
         System.out.println("Invalid input!");
   }
}
```

Nested Conditions and Decisions
Dangling Else Problem

The ***dangling else problem*** occurs when a programmer mistakes an else clause to belong to a different if statement than it really does.

- Remember, blocks (brackets) determine which statements are grouped together, not indentation!

Example:

Incorrect

```
if (country == "US"))
   if (state == "HI"))
      shipping = 10.00;
else  // Belongs to 2nd if!
   shipping = 20.00; // Wrong!
```

Correct

```
if (country == "US")
{  if (state == "HI")
      shipping = 10.00;
}
else
   shipping = 20.00;
```

Nested Conditions and Decisions
Boolean Expressions

A ***Boolean expression*** is a sequence of conditions combined using AND (**&&**), OR (**||**), and NOT (**!**).

- Allows you to test more complex conditions
- Group subexpressions using parentheses

Syntax: *(expr1)* **&&** *(expr2)* - expr1 AND expr2
(expr1) **||** *(expr2)* - expr1 OR expr2
!(expr1) - NOT expr1

Examples:

```
var b;

1) b = (x > 10) && !(x < 50);
2) b = (month == 1) || (month == 2) || (month == 3);
3) if (day == 28 && month == 2)
4) if !(num1 == 1 && num2 == 3)
5) b = ((10 > 5 || 5 > 10) && ((10>5 && 5>10));// False
```

Boolean Expressions

Question: Is `result` `true` or `false`?

```
int x = 10, y = 20;
int result = (x > 10) || (y < 20);
System.out.println(result);
```

A) `true`

B) `false`

Boolean Expressions (2)

Question: Is `result` `true` or `false`?

```
int x = 10, y = 20;
int result = !(x != 10) && (y == 20);
System.out.println(result);
```

A) `true`

B) `false`

Boolean Expressions (3)

Question: Is `result` `true` or `false`?

```
int x = 10, y = 20;
int result = (x >= y) || (y <= x);
System.out.println(result);
```

A) `true`

B) `false`

Making Decisions (4)

Question: What is the output of this code?

```
int num=12;

if (num >= 8)
    System.out.print("big");
   if (num == 10)
       System.out.print("ten");
else
    System.out.print("small");
```

A) `big`

B) `small`

C) `bigsmall`

D) `ten`

E) `bigten`

Making Decisions (5)
Boolean Expressions

Question: What is the output of this code?

```
int x = 10, y = 20;

if (x >= 5)
{  System.out.print("bigx");
   if (y >= 10)
       System.out.print("bigy");
}
else if (x == 10 || y == 15)
   if (x < y && x != y)
       System.out.print("not equal");
```

A) `bigx`

B) `bigy`

C) `bigxnot equal`

D) `bigxbigynot equal`

E) `bigxbigy`

Making Decisions
Switch Statement

There may be cases where you want to compare a single integer value against many constant alternatives. Instead of using many if statements, you can use a ***switch*** statement.

- ◆If there is no matching case, the default code is executed.
- ◆Execution continues until the **break** statement. (Remember it!)
- ◆Note: You can only use a switch statement if your cases are **integer numbers**. (Characters (‘a’, ‘b’,...,) are also numbers.)

Syntax:

```
switch (integer number)
{  case num1: statement break;
   case num2: statement break;
   ...
   default:   statement break;
}
```

Making Decisions
Switch Statement Example

```
public class TestSwitch
{  public static void main(String[] args)
   {  int num;
      Scanner sc = new Scanner(System.in);

      System.out.println("Enter a day number: ");
      num=sc.nextInt();
      switch (num)
      {  case 1: System.out.println("Sunday"); break;
         case 2: System.out.println("Monday"); break;
         case 3: System.out.println("Tuesday"); break;
         case 4: System.out.println("Wednesday"); break;
         case 5: System.out.println("Thursday"); break;
         case 6: System.out.println("Friday"); break;
         case 7: System.out.println("Saturday"); break;
         default: { System.out.println("Invalid day!");
                    System.out.println("Valid #'s 1-7!");
                  } break;
      }
   }
}
```

Switch Statement

Question: What is the output of this code?

```
int num=2;

switch (num)
{  case 1: System.out.print("one"); break;
   case 2: System.out.print("two"); break;
   case 3: System.out.print("three"); break;
   default: System.out.print("other"); break;
}
```

A) `one`

B) `two`

C) `three`

D) `other`

Switch Statement (2)

Question: What is the output of this code?

```
int num=1;

switch (num)
{  case 1: System.out.print("one");
   case 2: System.out.print("two");
   case 3: System.out.print("three"); break;
   default: System.out.print("other");
}
```

A) `one`

B) `onetwo`

C) `onetwothree`

D) `other`

E) `onetwothreeother`

Decision Practice Questions

1) Write a program that reads an integer *N*.

- ◆If *N* < 0, print “Negative number”, if *N* = 0, print “Zero”, If *N* > 0, print “Positive Number”.

2) Write a program that reads in a number for 1 to 5 and prints the English word for the number. For example, 1 is “one”.

3) Write a program to read in your name and age and print them. Your program should print “Not a teenager” if your age is greater than 19 or less than 13, otherwise print “Still a teenager”.

Iteration and Looping
Overview

A computer does simple operations extremely quickly.

If all programs consisted of simple statements and decisions as we have seen so far, then we would never be able to write enough code to use a computer effectively.

To make a computer do a set of statements multiple times we program ***looping structures***.

A ***loop*** repeats a set of statements multiple times until some condition is satisfied.

- ◆Each time a loop is executed is called an ***iteration***.

The While Loop

The most basic looping structure is the ***while*** loop.

A while loop continually executes a set of statements **while** a condition is true.

Syntax:

```
while (<condition>)
{  <statements>
}
```

Example:

```
int j=0;
while (j <= 5)
{  j=j+1;
   System.out.println(j);
}
```

The ++ and -- Operators

It is very common to subtract 1 or add 1 from the current value of an integer variable.

There are two operators which abbreviate these operations:

- ++- add one to the current integer variable
- -- - subtract one from the current integer variable

Example:

```
int j=0;

j++;      // j = 1;  Equivalent to j = j + 1;
j--;      // j = 0;  Equivalent to j = j - 1;
```

The For Loop

The most common type of loop is the ***for loop***. Syntax:

```
for (<initialization>; <continuation>; <next iteration>)
{  <statement list>
}
```

Explanation:

- 1) initialization section - is executed once at the start of the loop
- 2) continuation section - is evaluated *before* every loop iteration to check for loop termination
- 3) next iteration section - is evaluated *after* every loop iteration to update the loop counter

Iteration & Looping The For Loop

Although Java will allow almost any code in the three sections, there is a typical usage:

```
for (i = start; i < end; i++)
{  statement
}
```

Example:

```
int i;

for (i = 0; i < 5; i++)
{  System.out.println(i);      // Prints 0 to 4
}
```

Java Rules for Loops

The iteration variable is a normal variable that must be declared, but it has the special role of controlling the iteration.

- i, j, and k are the most common choices due to convention and because they are short.

The starting point of the iteration can begin anywhere, including negative numbers.

The continuation/termination test must be an expression that results in a Boolean value. It should involve the iteration variable to avoid an ***infinite loop***.

The next iteration can have any statements, although usually only use the step size to change iteration variable.

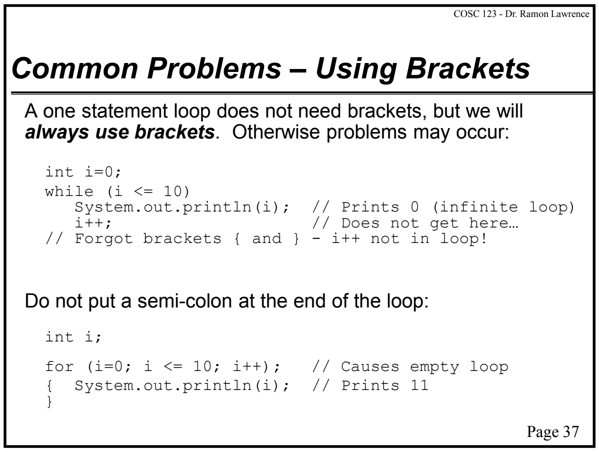
- The step size can be positive or negative and does not always have to be 1.

Common Problems – Infinite Loops

Infinite loops are caused by an incorrect loop condition or not updating values within the loop so that the loop condition will eventually be false.

Examples:

```
int i;

for (i=0; i < 10; i--)      // Should have been i++
{  System.out.println(i);  // Infinite loop: 0,-1,-2,..
}

i = 0;
while (i < 10)
{  System.out.println(i);  // Infinite loop: 0,0,0,..
}                          // Forgot to change i in loop
```



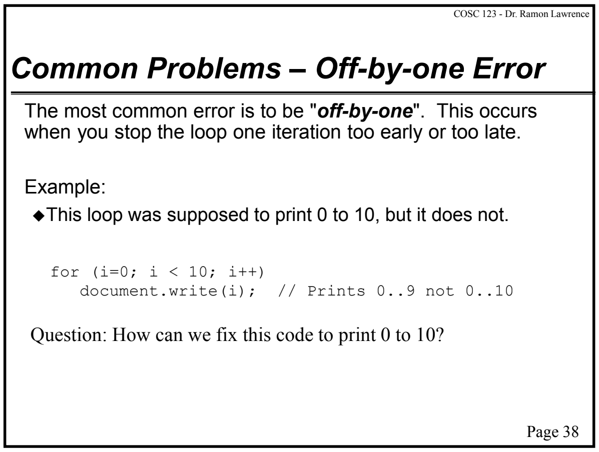
Common Problems – Using Brackets

A one statement loop does not need brackets, but we will ***always use brackets***. Otherwise problems may occur:

```
int i=0;
while (i <= 10)
   System.out.println(i);  // Prints 0 (infinite loop)
   i++;                    // Does not get here…
// Forgot brackets { and } - i++ not in loop!
```

Do not put a semi-colon at the end of the loop:

```
int i;

for (i=0; i <= 10; i++);   // Causes empty loop
{  System.out.println(i);  // Prints 11
}
```



Common Problems – Off-by-one Error

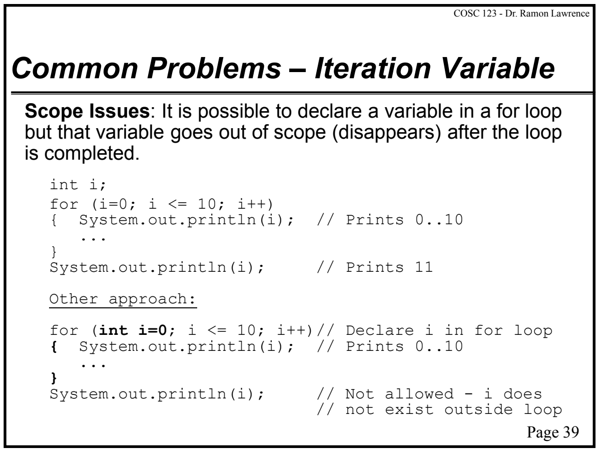
The most common error is to be "***off-by-one***". This occurs when you stop the loop one iteration too early or too late.

Example:

- ◆This loop was supposed to print 0 to 10, but it does not.

```
for (i=0; i < 10; i++)
   document.write(i);  // Prints 0..9 not 0..10
```

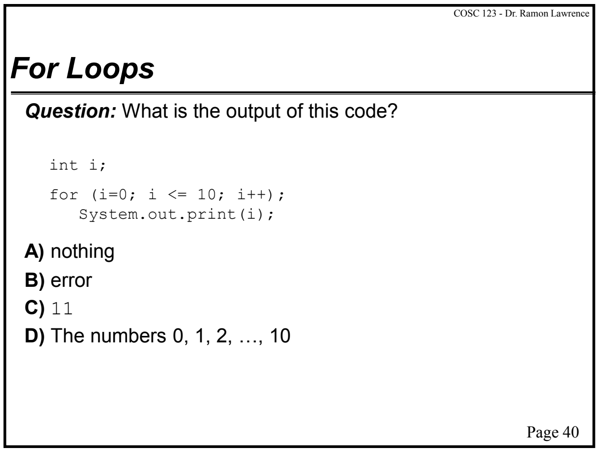
Question: How can we fix this code to print 0 to 10?



Common Problems – Iteration Variable

Scope Issues: It is possible to declare a variable in a for loop but that variable goes out of scope (disappears) after the loop is completed.

```
int i;
for (i=0; i <= 10; i++)
{  System.out.println(i);  // Prints 0..10
   ...
}
System.out.println(i);     // Prints 11

Other approach:

for (int i=0; i <= 10; i++)// Declare i in for loop
{  System.out.println(i);  // Prints 0..10
   ...
}
System.out.println(i);     // Not allowed - i does
                           // not exist outside loop
```



For Loops

Question: What is the output of this code?

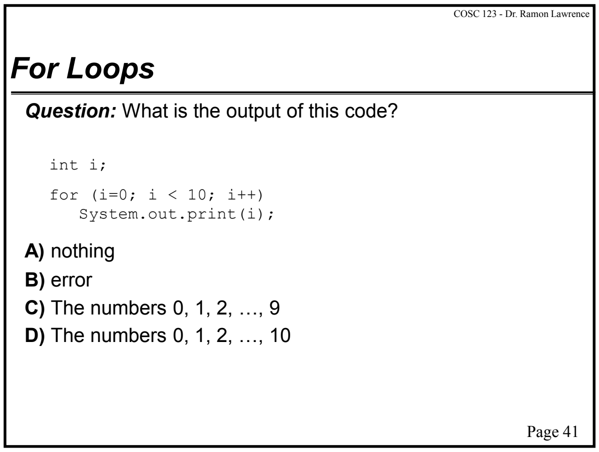
```
int i;

for (i=0; i <= 10; i++);
   System.out.print(i);
```

A) nothing

B) error

C) `11`

D) The numbers 0, 1, 2, …, 10



For Loops

Question: What is the output of this code?

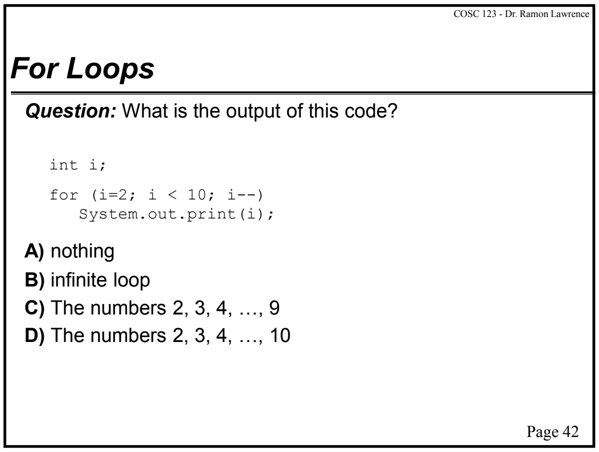
```
int i;

for (i=0; i < 10; i++)
   System.out.print(i);
```

A) nothing

B) error

C) The numbers 0, 1, 2, …, 9

D) The numbers 0, 1, 2, …, 10



For Loops

Question: What is the output of this code?

```
int i;

for (i=2; i < 10; i--)
   System.out.print(i);
```

A) nothing

B) infinite loop

C) The numbers 2, 3, 4, …, 9

D) The numbers 2, 3, 4, …, 10

The do..while Loop

The last looping structure called a ***do..while*** loop. The do..while loop is similar to the while loop except that the loop condition is tested at the bottom of the loop instead of the top.

- ◆This structure is useful when you know a loop must be executed at least once, but you do not know how many times.

Syntax:

```
do
{  statement
} while (condition);
```

Example:

```
do
{  num = num / 2;
} while (num >= 0);
```

Loop Nesting

Similar to decisions statements such as if and switch, it is possible to nest for, while, and do..while loops.

- ◆Note that the loops do not all have to be of the same type.
  - ⇨i.e. You can have a for loop as an outer loop, and a while loop as an inner loop.

Be very careful to include correct brackets when nesting loops.

- ◆It is a good idea to always include brackets in your code to make your code more readable and prevent mistakes.

Nested For/While Loop Example

```
// Prints N x N matrix until N = -1

public class NestedForWhile
{  public static void main(String[] args)
   {  int i, j, num;
      Scanner sc = new Scanner(System.in);

      System.out.print(“Enter a matrix size: “);
      num=sc.nextInt();
      while (num != -1)
      {
         for (i=1; i <= num; i++)
         {  for (j=1; j <= num; j++)
               System.out.print(j+" ");   // No brackets!
            System.out.println();
         }
         System.out.print(“Enter a matrix size: “);
         num=console.readInt();
      }
   }
}
```

Advanced Topic: Break Statement

What happens if you want to exit a loop before the end?

- ◆You can use the ***break*** statement to immediately exit the current loop block.
  - ⇨Note: The break statement exits the current loop. If you have a nested loop, you will need multiple break statements to get out of all loops.

Example:

```
while (true)
{  System.out.print(“Enter a matrix size: “);
   num=console.readInt();
   if (num == -1)
      break;
   ...
}
// After break - execution starts here
```

Advanced Topic: Continue Statement

What happens if you want to quickly skip back to the start of the loop (end the current iteration) while in the middle of the loop statements?

- ◆You can use the ***continue*** statement to immediately stop the current loop iteration and start the next one.
  - ⇨Note: This is rarely used.

Example:

```
for (i=0; i < 5; i++)    // After continue, start i=3
{  if (i  == 2)
      continue;    // For some reason we don’t like 2!
   System.out.println(i);
}

// Question: What is the better way to do this?
```

Looping Review

A loop structure makes the computer repeat a set of statements multiple times.

- ◆for loop is used when you know exactly how many iterations to perform
- ◆while loop is used when you keep repeating the loop until a condition is no longer true
- ◆a do..while loop is used when a loop has to be performed at least once

When constructing your loop structure make sure that:

- ◆you have the correct brackets to group your statements
- ◆you do not add additional semi-colons that are unneeded
- ◆make sure your loop terminates (no infinite loop)

Remember the operators ++ and -- as short-hand notation.

Continue Statement

Question: How many numbers are printed?

```
for (int i=2; i < 10; i++)
{  if (i % 2 == 0)
      continue;
   System.out.print(i);
}
```

A) 0

B) 4

C) 5

D) 9

Break Statement

Question: How many numbers are printed?

```
for (int i=2; i < 10; i++)
{  if (i > 4)
      break;
   System.out.print(i);
}
```

A) 9

B) 5

C) 4

D) 3

Practice Questions: Iteration

1) How many times does each loop execute:

```
a) for(j=0; j <= 10; j--)
b) for(j=0; j <= 10; j++)
c) for(j=0; j < 10; j++)
d) for(j=-10; j <= 10; j++)
e) for(j=0; j <= 20; j=j+2)
```

2) Write a program to print the numbers from 1 to N.

- ◆a) Modify your program to only print the even numbers.

3) Write a method that builds and prints an integer matrix of the form: (where N is given).

```
1 1 1 … 1
2 2 2 … 2
…
N N N … N
```

Conclusion

A ***decision*** is performed by evaluating a Boolean condition with an `if/else` statement.

A ***loop*** allows the repetition of a set of statements multiple times until some condition is satisfied.

- ◆We will primarily use `for` loops that have 3 components:
  - ⇨initialization - setup iteration variable start point
  - ⇨continuation - use iteration variable to check if should stop
  - ⇨next iteration - increment/decrement iteration variable

Decision and loops can be nested.

Objectives

Java skills:

- ◆Make decisions using if/else statement.
- ◆Use Boolean variables to represent true/false.
- ◆Use relational operators in conditions.
- ◆Comparing Strings and Objects using equals and compareTo.
- ◆Build complex conditions using AND, OR, and NOT.
- ◆Switch statement
- ◆Iteration using three loop constructs:
  - ⇨while statement
  - ⇨for statement
  - ⇨do…while statement
- ◆Break and continue statements
- ◆Nesting of if/else and iteration statements

Detailed Objectives

- ◆Write decisions using the if/else statement.
- ◆Define: Boolean, condition
- ◆List and use the comparison operators.
- ◆Explain the dangling else problem.
- ◆Construct and evaluate Boolean expressions using AND, OR, and NOT.
- ◆Explain why cannot use == with Strings/Objects.
- ◆Define: loop, iteration
- ◆Explain the difference between the while and for loops.
- ◆Explain what ++ and -- operators do.
- ◆Be able to use a for loop structure to solve problems.
- ◆Be aware and avoid common loop problems.
- ◆Define: infinite loop

COSC 123
Computer Creativity

Java Classes

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) Define classes, objects, methods, properties (instance variables), and parameters in Java.

2) Inheritance derives new classes from existing ones. A subclass inherits all methods and variables from its superclass.

3) Create objects from classes using `new`.

4) Explain the difference between an object and an object reference.

5) List the types of variables (instance, static, local, parameter) and explain how the type affects their scope and lifetime.

Java Object-Oriented Terminology

An ***object*** is an instance of a class that has its own properties and methods. Properties and methods define what the object is and what it can do. *Each object has its own area in memory.*

A ***class*** is a generic template (blueprint) for creating an object. All objects of a class have the same methods and properties (although the property values can be different).

A ***property*** (or ***instance variable***) is an attribute of an object.

A ***method*** is a set of statements that performs an action. *A method works on an implicit object and may have parameters.*

A ***parameter*** is data passed into a method for it to use.

Class Definition

To define a class:

- ◆use the keyword **`class`** and provide a name for your class
- ◆enclose the contents of your class in brackets "**{**", "**}**"
- ◆define any properties (*instance variables*) for your class
- ◆define any methods in your class

Example:

```
class classname
{  classname methods
   classname variables
}
```

Variable Definition

To define a variable of a class:

- ◆define the variable as either public or private (access specifier)
- ◆provide the variable type and name as usual

Syntax:

```
class classname
{  accessSpecifier variableType variableName;
   ...
}
```

Example:

```
class MyClass
{  public int num;
   private String st;
   private double value;
}
```

Method Definition

To define a method of a class:

- ◆define the method as either public or private (access specifier)
- ◆provide the method return type, name, and parameters
  - ⇨Each parameter has a type and a name.
  - ⇨A return type of void means return nothing.

Syntax:

```
class classname
{  accessSpec retType methodName(par1, par2, …, parN)
   { method implementation}
}
```

Example:

```
class TestClass
{  public int count(int n)       { return n+1; }
   private void doNothing()      { }
   public String addS(String st) { return st+"S"; }
}
```

Method Definition Parameters

A method may use parameters to perform its operations.

- ◆Each parameter has a type and a name.
- ◆Parameters are separated by commas.
- ◆Parameters can be changed by the method, but their value will not be changed for the caller.

Method Definition Return Types

Use the **`return`** statement to return a method value. Syntax:

```
        return expression;  OR

        return;
```

Example:

```
class TestClass
{  public int retTest(int n)
   {  if (n == 0)
         return 1;
      else
         return n*2+1;
   }
   public void retNothing(String st)
   {  if (st.equals(""))
         return;
      ...
   }
}
```

Your First Java Program (again)

```
public class HelloWorld
{  public static void main(String[] args)
   {  System.out.println("Hello, World!");
   }
}
```

The first line creates a ***public class*** called `HelloWorld` that is the main class of your program and the name of the Java file.

Class `HelloWorld` contains a method `main` that is `public`.

- ◆Since class `HelloWorld` is the public class for this file, it is the class that must contain the `main` method.
- ◆`main` is a method called with one parameter (`String[] args`).
- ◆`main` is a special method because it is automatically called when you run your program.

Class Example `BankAccount` Class

```
public class BankAccount
{
   private double balance;

   public void deposit(double amount)
   {  balance = balance + amount; }

   public void withdraw(double amount)
   {  balance = balance - amount; }

   public double getBalance()
   {  return balance; }
}
```

The `BankAccount` class is used for describing bank accounts.

- ◆The methods defined in the `BankAccount` class are `deposit`, `withdraw`, and `getBalance`.
- ◆The current balance in the account is `private`, so it can only be changed by calling the methods.

Practice Questions

1) Implement a class `Employee`:

- ◆An employee has a name (String) and a salary (`double`).
- ◆Write methods to get/set the name and salary.

2) Implement a class `Purse`:

- ◆A purse holds coins (toonies, loonies, and quarters only).
- ◆Write methods to get/set the number of coins in the purse.
- ◆Write a method called `getValue()` which returns the value of all coins in the purse.

Inheritance Overview

Inheritance is a mechanism for enhancing and extending existing, working classes.

- ⇨In real life, you inherit some of the properties from your parents when you are born. However, you also have unique properties specific to you.
- ⇨In Java, a class that extends another class inherits some of its properties (methods, instance variables) and can also define properties of its own.

`extends` is the key word used to indicate when one class is related to another by inheritance.

Syntax: `class` *`subclass`* **`extends`** *`superclass`*

- ◆The ***superclass*** is the existing, parent class.
- ◆The ***subclass*** is the new class which contains the functionality of the superclass plus new variables and methods.
- ◆A subclass may only inherit from ***one*** superclass.

Why use inheritance?

The biggest reason for using inheritance is to re-use code.

- Once a class has been created to perform a certain function it can be re-used in other programs.
- Further, using inheritance the class can be extended to tackle new, more complex problems without having to re-implement the part of the class that already works.

The alternative is copy and paste which is bad, especially when the code changes.

What is inherited?

When a subclass inherits (or extends) a superclass:

Instance variable inheritance:

- All instance variables of the superclass are inherited by the subclass.
  - ⇨ However, if a variable is **`private`**, it can only be accessed using methods defined by the superclass.

Method inheritance:

- All superclass methods are inherited by the subclass, but they may be ***overridden***.

Inheritance Example

Consider the `BankAccount` class that we created to model bank account objects.

- A bank account has an account number and a balance.

How about if we want to create a special kind of bank account called a `SavingsAccount`?

- A savings account is a special bank account because it also pays interest at a given interest rate.
- Instead of programming the entire `SavingsAccount` class and duplicating features already in the `BankAccount` class, we can extend the `BankAccount` class and inherit its properties when we create a `SavingsAccount`.

BankAccount Code

```
public class BankAccount
{  public void deposit(double amount)
   {  balance = balance + amount; }
   public void withdraw(double amount)
   {  balance = balance - amount; }
   public double getBalance()
   {  return balance; }
   public int getAccount()
   {  return accountNum; }
   public int getLastAccount()
   {  return lastAccountNum; }
   public BankAccount()
   {  this(0); }
   public BankAccount(double b)
   {  balance = b; lastAccountNum++;
      accountNum = lastAccountNum;
   }
   private double balance;
   private int accountNum;
   private static int lastAccountNum = 0; // Static
}
```

SavingsAccount Code

```
public class SavingsAccount extends BankAccount
{
   public void addInterest()
   {  deposit(getBalance()*rate/100); }

   public SavingsAccount()
   {  this(0); }
   public SavingsAccount(double r)
   {  rate = r;   }

   private double rate; // Interest rate paid
}
```

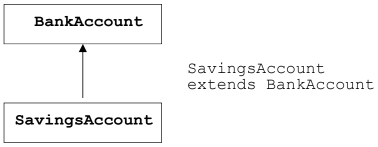
```
Notes:
1) Inherited variables: balance,accountNum,lastaccountNum
2) Inherited methods: deposit, withdraw, getAccount,
                      getLastAccount
3) Inherited variables are private in BankAccount, so we
    cannot access them directly.
   (Use deposit and getBalance methods.)
```

Class Diagrams

Class diagrams display the relationship between related classes using a diagram.

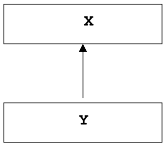
- We will follow the Unified Modeling Language (UML) syntax.
  - ⇨ Each class has its own box.
  - ⇨ There is an arrow from a subclass to a superclass if that class extends the superclass.

Example:



Superclass and Subclass

Question: Which class is the superclass?



A) X
B) Y

Inheritance

Question: Which statement is true?

A) A subclass can access all variables it inherited from the superclass.

B) A subclass can declare an instance variable with the same name as an instance variable in its superclass.

C) A class can have more than one superclass.

Access Specifiers Public and Private

One of the features of object-oriented programming is that not all parts of a program have access to all the data and methods.

Each class, method, and *instance* variable has one of the four ***access specifiers*** to indicate which other objects and methods in your program have access to it. Four types:

- `public` – Accessible by all code (everyone, the public)
- `private` – Only accessible by methods in the class.
- `protected` – Only accessible by methods in the class or classes derived from this class by inheritance.
- default – If nothing is specified, assume package access where all methods in same package (directory) can access it.

Public and Private Examples

```
public class MyClass
{  public void setValue(int n)
   {  num = n; }         // setValue() is a public method

   private void show()   // show() is a private method
   {  st = "Hello"; }

   private int num;      // num is a private variable
   public String st;     // st is a public variable
   double d;             // d has package access
}
```

Summary:
1) Method setValue() is **public**, so it can be accessed from anywhere in the program.
2) Method show() is **private** so only another method in the class MyClass can access it.
3) Variable num is **private**, only methods in MyClass can access it.
4) st is **public**. It is accessible anywhere in the program.
5) d has default (package) access. Any method in a file in the same package (directory) can access it.

Access Specifier Rules

There is one special rule in Java that you must follow:

- There can be only one public class per file, and the name of that class has to be the same as the name of the file.

There are also some common programming rules which you will use in this course:

- Always state if a class/variable/method is `public` or `private`.
- Variables in an object are almost always `private`.
  - ⇨ Other objects/methods do not have access to the data directly.
- Most methods of an object are `public`.
  - ⇨ These methods allow other objects/methods to see/manipulate the data.
- Class names should begin with a capital letter.
- Method and variable names should begin with a small letter.

Inheritance Question

1) Create a `CheckingAccount` class which inherits from `BankAccount`. The `CheckingAccount` class:

- inherits `getBalance()` from `BankAccount`
- overrides `deposit()` and `withdraw()` from `BankAccount`, so it can keep track of the number of transactions (`transactionCount`)
- defines a method `deductFees()` which withdraws $1 for each transaction (`transactionCount`) then resets the # of transactions

Inheritance Questions (2)

2) Create:

- 1) A superclass `Pet`. A `Pet` contains:
  - a `name` and methods to get/set its `name`
- 2) A subclass `Cat` of `Pet`. A `Cat` contains:
  - a boolean variable `hasClaws` which is true if the cat has claws
  - define methods to get/set the `hasClaws` instance variable
- 3) A subclass `Dog` of `Pet` that contains:
  - an integer variable `numTricks` that stores the number of tricks the dog can perform
  - define methods to get/set the value of `numTricks`

Creating and Using Objects

A class is just a blue-print for creating objects.

- By itself, a class performs no work or stores no data.

For a class to be useful, we must create objects of the class.

- Each object created is called an ***object instance***.

To create an object, we use the **`new`** method.

When an object is created using the `new` method:

- Java allocates space for the object in memory.
- The ***constructor*** for the object is called to initialize its contents.
- Java returns a pointer to where the object is stored in memory which we will call an ***object reference***.

Constructors

A ***constructor*** is a method that is called when the object is first created and initializes the variables of an object.

- If you do not supply a constructor for a class, Java supplies a ***default constructor*** which has no parameters.
- You may define your own constructors for your objects to guarantee that an object has the correct initial values.
  - A constructor may have parameters like any other method.

Syntax and Example:

```
class classname                // (Syntax)
{  classname() {}              // Default constructor
   classname(par1, par2, …, parN) {} //Parameters
}
class MyClass                   //(Example)
{  MyClass()      { num = 0; } // Default constructor
   MyClass(int n) { num = n; } // Parameters
   private int num;            // Variable initialized
}
```

Bank Account Example Revisited

```
public class BankAccount
{  public void deposit(double amount)
   {  balance = balance + amount; }
   public void withdraw(double amount)
   {  balance = balance - amount; }
   public double getBalance()
   {  return balance; }

   public BankAccount()          { balance = 0; }
   public BankAccount(double b)  { balance = b; }

   private double balance;
}
```

The `BankAccount` class now defines two constructors:

- Default constructor initializes balance to 0.
- Constructor with parameter initializes balance to a given value.

Creating Objects using `new`

Objects are created using the **`new`** method.

The new method allocates space for the object in memory, calls the appropriate object constructor, and returns an ***object reference*** to be stored in an object reference variable.

Example:

```
BankAccount checking = new BankAccount();
// Creates a BankAccount object referenced by checking
BankAccount savings = new BankAccount();
// Creates a BankAccount object referenced by savings

BankAccount mySavings;  // Declares object reference

mySavings = new BankAccount(); // Creates object
```

Object References

It is important to realize the difference between an object and an object reference.

When you declare an object variable in Java, you are actually declaring an object reference to that particular object type.

- Until you create an object using the `new` method, there is no object in memory which is pointed to by the object reference.

An object is the physical memory representation of the data.

- An object has a location in memory and a type (class).
  - Each object has its own data values.

Changing Object References

Object references are pointers to objects in memory that can be assigned to the same value as another reference using '=' or assigned to `null` (which means they refer to nothing).

Example:

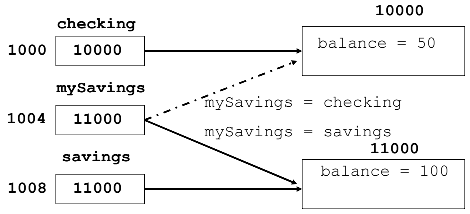
```
BankAccount checking = new BankAccount(50);
// Creates a BankAccount object referenced by checking
BankAccount savings = new BankAccount(100);
// Creates a BankAccount object referenced by savings
BankAccount mySavings;  // Declares object reference

mySavings = savings;    // mySavings points to savings
System.out.println(mySavings.getBalance()); // 100
mySavings = checking;   // mySavings points to checking
System.out.println(mySavings.getBalance()); // 50
```

Objects in Memory

Remember that each object has its own space in memory AND each object reference variable also has its own memory space.

Object references point to objects and can be changed.

Memory diagram based on previous example:



null Object References

Sometimes a programmer wants an object reference to point to nothing. To make an object reference refer to nothing, you assign it a value of `null`.

Example:

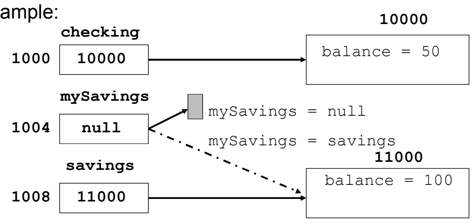
```
BankAccount checking = new BankAccount(50);
BankAccount savings = new BankAccount(100);
BankAccount mySavings;  // Declares object reference

mySavings = savings;    // mySavings points to savings
System.out.println(mySavings.getBalance()); // 100
mySavings = null;       // mySavings now points to null
System.out.println(mySavings.getBalance()); // Error!
```

null Object References Example

A null reference effectively stores the address of **0**. Since this is not a valid memory address for the program, your program will generate a **run-time error** during execution.

- The compiler does not check null references for you!

Example:



Calling Object Methods

A method is called on an object by supplying an object reference and the name and parameters of the method.

Syntax:

objectReference.*methodName*(*parameters*)

Remember:

- Each object has its own class which defines which methods it can perform.
- Each object has its own area of memory storing its data.
- An object reference is a pointer to a particular object in memory, so Java knows which object we are talking about by providing the object reference.
  - The object reference is called an ***implicit parameter***.

Creating and Using Objects Calling Object Methods Example

```
public class TestBankAccount
{  public static void main(String []args)
   {  BankAccount savings = new BankAccount(100);
      BankAccount checking = new BankAccount(50);
      BankAccount myRef;   // No object allocated!

      System.out.println(savings.getBalance()); // 100
      System.out.println(checking.getBalance());// 50
      System.out.println(myRef.getBalance()); // Error!
      savings.deposit(50);
      checking.withdraw(40);
      myRef = savings;
      myRef.withdraw(20);
      System.out.println(savings.getBalance()); // 130
      System.out.println(checking.getBalance()); // 10
      System.out.println(myRef.getBalance()); // 130
      myRef = checking;
      myRef.deposit(50);
      System.out.println(myRef.getBalance()); // 60
   }
}
```

Advanced: Implicit Parameter this

When an object method is called, we tell Java which object to use based on an object reference.

This object reference is then accessible within an object method as the **`this`** reference.

Example:

```
public class TestThis
{  public static void main(String []args)
   {  BankAccount checking = new BankAccount(50);
      BankAccount savings = new BankAccount(100);

      System.out.println(savings.getBalance());
      // this reference set to savings
      System.out.println(checking.getBalance());
      // this reference set to checking
   }
}
```

Implicit Parameter this (2)

```
public class BankAccount
{  public void deposit(double amount)
   {  this.balance = this.balance + amount; }
   public void withdraw(double amount)
   {  this.balance = this.balance - amount; }
   public double getBalance()
   {  return this.balance; }

   // this points to the current object being used
   // Using this is optional because Java assumes you
   // are working with the current object

   public BankAccount()
   { this.balance = 0; }
   public BankAccount(double balance)
   { this.balance = balance; }

   private double balance;
}
```

Access Specifiers Public and Private

Question: A method in class X is defined as private. Can it access a public variable in class Y?

A) Yes

B) No

Access Specifiers Public and Private

Question: Which statement is true?

A) It is a good idea to make all instance variables public.

B) Every parameter should be declared as public or private.

C) A method in class X can call a private method in class Y.

D) A method in class X can access a private instance variable in class X.

Objects and Object References

Question: How many objects are created by this code?

```
BankAccount savings, checking;
BankAccount myAcct, myAcct2;

savings = new BankAccount();
myAcct = savings;
checking = new BankAccount();
```

A) 1

B) 2

C) 3

D) 4

Objects and Object References

Question: How much money is in the account referenced by the `myAcct2` object reference?

```
BankAccount savings, checking;
BankAccount myAcct, myAcct2;

savings = new BankAccount(50);
myAcct = savings;
checking = new BankAccount(100);
savings = checking;
myAcct2 = myAcct;
```

A) unknown

B) 50

C) 100

D) undefined

Practice Questions

1) Explain the difference between a class, an object, and an object reference.

2) Create a program which creates a new BankAccount object called savings with an initial balance of $100. Then, deposit $40, withdraw $20, and print the current balance.

3) Modify the BankAccount class to also store an interest rate.

- Allow the user to specify the interest rate in a constructor.
- Create a method for setting the interest rate.
- Create a method called calcInterest() to update the current balance based on the interest rate.
- Test your class with an account with $1000 and 10% interest rate. Deposit $100, calculate interest, and print balance.

Interfaces

Interfaces are used to allow a class to implement methods of another class without inheriting from it.

An ***interface*** is a class where:

- All methods are `public` and `abstract` (no implementation).
- All variables are `static` and `final`. (no instance variables).

A class which implements an interface must implement all methods of the interface.

A class can implement multiple interfaces.

Keyword to indicate implementing an interface is: **`implements`**

Interfaces Example

```
interface Shape {
   int numSides();
   int getArea();
}

class Square implements Shape {
   public int numSides() { return 4; }
   public int getArea()  { return len*height; }

   private int len;
   private int height;
}
```

Object - THE SUPERCLASS

The class **Object** in Java is the root of the inheritance hierarchy or the superclass of all classes.

- That is, every class defined in Java and that you define inherits from the `Object` class.
- If you define a class that does not inherit from another class, your class automatically extends the `Object` class.

The `Object` class has some defined methods:

- `String toString()`
  - ⇨ returns a string representation of the object
- `boolean equals(Object other)`
  - ⇨ tests whether the object equals another object
- `Object clone()`
  - ⇨ makes a full (or deep) copy of the object
    - does not just copy the object reference, copies the entire object

Overriding toString method

```
public class BankAccount
{
   public String toString()
   {  return "BankAccount[balance="+ balance+ "]"; }

   ...

   private double balance;
   private int accountNum;
   private static int lastAccountNum = 0; // Static
}


toString() method:
1) Returns a string representation of your object.
```

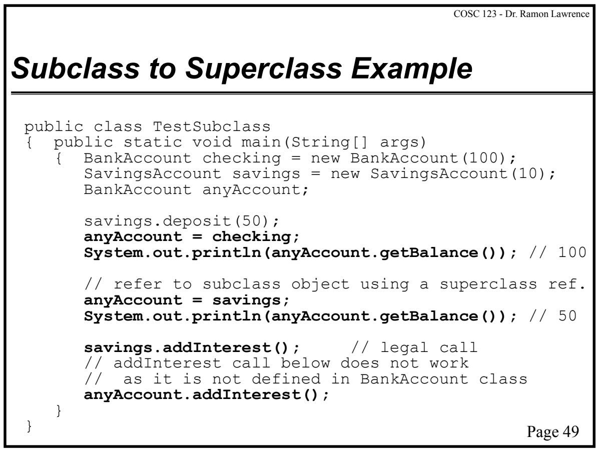
Casting and Method Access

It is possible to assign an object reference of a subclass to the an object reference variable of a superclass.

- This is allowed because a subclass is a special case of the superclass.
- However, you are unable to access any methods/variables in the subclass using a superclass object reference.
- All objects inherit from the **`Object`** class, so they can be assigned to an **`Object`** reference variable.

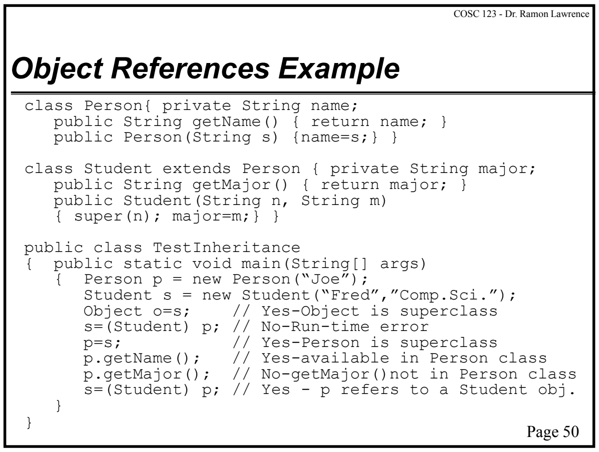
It is possible to explicitly cast an object reference variable for a superclass to a subclass variable only if the superclass variable references a valid subclass instance.

- Otherwise, a run-time error will result.



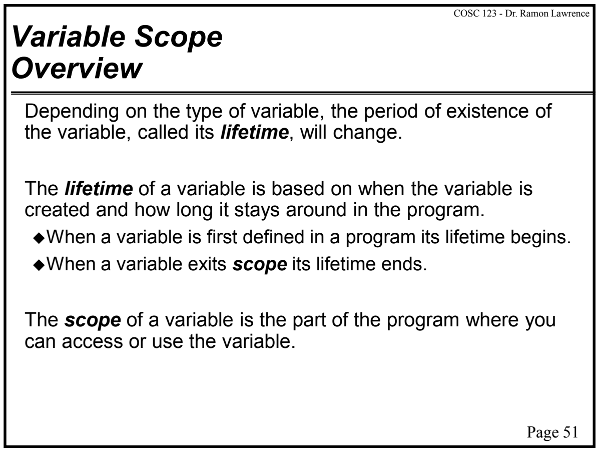
Subclass to Superclass Example

```
public class TestSubclass
{  public static void main(String[] args)
   {  BankAccount checking = new BankAccount(100);
      SavingsAccount savings = new SavingsAccount(10);
      BankAccount anyAccount;

      savings.deposit(50);
      anyAccount = checking;
      System.out.println(anyAccount.getBalance()); // 100

      // refer to subclass object using a superclass ref.
      anyAccount = savings;
      System.out.println(anyAccount.getBalance()); // 50

      savings.addInterest();     // legal call
      // addInterest call below does not work
      //  as it is not defined in BankAccount class
      anyAccount.addInterest();
   }
}
```



Object References Example

```
class Person{ private String name;
   public String getName() { return name; }
   public Person(String s) {name=s;} }

class Student extends Person { private String major;
   public String getMajor() { return major; }
   public Student(String n, String m)
   { super(n); major=m;} }

public class TestInheritance
{  public static void main(String[] args)
   {  Person p = new Person("Joe");
      Student s = new Student("Fred","Comp.Sci.");
      Object o=s;    // Yes-Object is superclass
      s=(Student) p; // No-Run-time error
      p=s;           // Yes-Person is superclass
      p.getName();   // Yes-available in Person class
      p.getMajor();  // No-getMajor()not in Person class
      s=(Student) p; // Yes - p refers to a Student obj.
   }
}
```



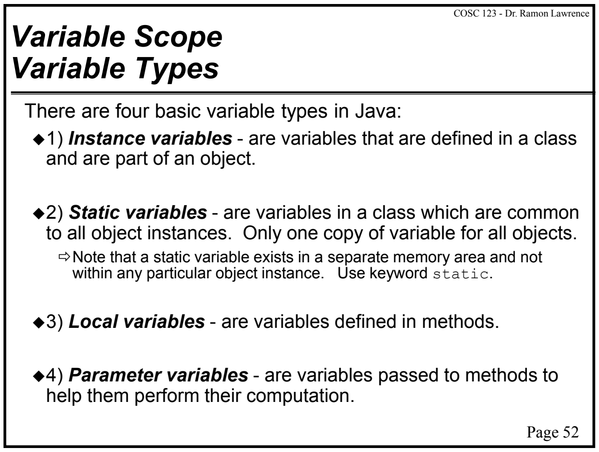
Variable Scope
Overview

Depending on the type of variable, the period of existence of the variable, called its ***lifetime***, will change.

The ***lifetime*** of a variable is based on when the variable is created and how long it stays around in the program.

- When a variable is first defined in a program its lifetime begins.
- When a variable exits ***scope*** its lifetime ends.

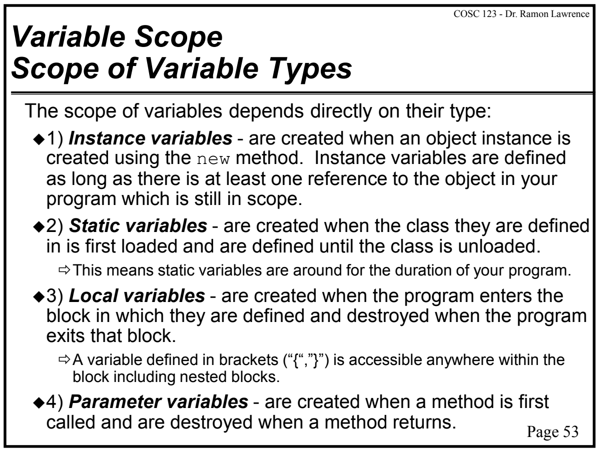
The ***scope*** of a variable is the part of the program where you can access or use the variable.



Variable Scope
Variable Types

There are four basic variable types in Java:

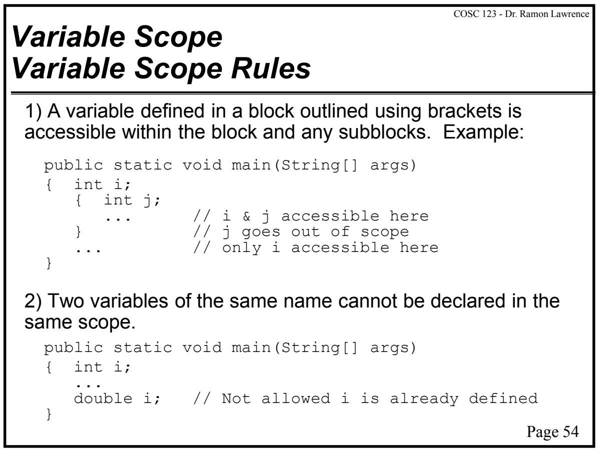
- 1) ***Instance variables*** - are variables that are defined in a class and are part of an object.
- 2) ***Static variables*** - are variables in a class which are common to all object instances. Only one copy of variable for all objects.
  - Note that a static variable exists in a separate memory area and not within any particular object instance. Use keyword `static`.
- 3) ***Local variables*** - are variables defined in methods.
- 4) ***Parameter variables*** - are variables passed to methods to help them perform their computation.



Variable Scope
Scope of Variable Types

The scope of variables depends directly on their type:

- 1) ***Instance variables*** - are created when an object instance is created using the `new` method. Instance variables are defined as long as there is at least one reference to the object in your program which is still in scope.
- 2) ***Static variables*** - are created when the class they are defined in is first loaded and are defined until the class is unloaded.
  - This means static variables are around for the duration of your program.
- 3) ***Local variables*** - are created when the program enters the block in which they are defined and destroyed when the program exits that block.
  - A variable defined in brackets ("{","}") is accessible anywhere within the block including nested blocks.
- 4) ***Parameter variables*** - are created when a method is first called and are destroyed when a method returns.



Variable Scope
Variable Scope Rules

1) A variable defined in a block outlined using brackets is accessible within the block and any subblocks. Example:

```
public static void main(String[] args)
{  int i;
   {  int j;
      ...      // i & j accessible here
   }           // j goes out of scope
   ...         // only i accessible here
}
```

2) Two variables of the same name cannot be declared in the same scope.

```
public static void main(String[] args)
{  int i;
   ...
   double i;   // Not allowed i is already defined
}
```

Variable Scope
Scope Example

```
public class MethodScope
{  public static void main(String[] args)
   {  double amount = 25;  // amount defined in main()
      BankAccount acct = new BankAccount(100);
      acct.withdraw(amount); //amount in main() copied to
      // amt in withdraw() - Not same variable!
   }  // amount, acct go out of scope
      // Object acct can be deleted with variable balance
}

class BankAccount
{  public void withdraw(double amt)
   {  if (amt <= balance)
      {  double newBalance = balance - amt;
         // newBalance is only defined within brackets
         balance = newBalance;
      }  // newBalance goes out of scope and is deleted
   }  // method variable amt goes out of scope

   private double balance; // instance variable
}
```

Variable Scope
Common Scope Errors

1) Variables with the same name in different scopes are different variables!

```
public static double area(Rectangle rect)
{  double r = rect.getWidth() * rect.getHeight();
   return r;
}

public static void main(String[] args)
{  Rectangle r = new Rectangle(5, 10, 20, 30);
   double a = area(r);
}
```

The `double r` in method `area` is a different variable than the variable `Rectangle r` in `main` as they have different scopes.

Variable Scope
Common Scope Errors (2)

2) Beware of scope issues when declaring variables in for loops!

```
public class TestForScope
{  public static void main(String[] args)
   {  for (int i=1; i <= 5; i++)
      {  System.out.println(i);  // 1,2,3,4,5
      }
      // for loop has its own copy of variable i
      // i in for loop goes out of scope
      System.out.println(i);  //Not allowed- i is gone!
   }
}
```

Variable Scope
Advanced Topic: Shadowing

Shadowing occurs when a variable in an inner scope overrides or shadows a variable in an outer scope with the same name.

- Typically, shadowing is an unintended programming mistake.
- Shadowing is possible with variables of different types.

Example:

```
public class Coin
{  public Coin(double aValue, String aName)
   {  value = aValue;
      String name = aName; // Shadows name in class
   }
   ...
   private double value;
   private String name;       // name defined here
}
```

In the example, the programmer accidentally redeclared the string variable name in the constructor which overrides the instance variable in the class.

Advanced:
Method Parameters: Pass-by-value

All method parameters are passed to a method by **value** which means that even if they are changed in a method, they are not updated in the caller method.

To return a value from a method:

- 1) Return a single value using a return type.
- 2) Pass object references to the method which allow object values to be changed.

Note: Although you cannot change the value of any parameters, by passing object references which have access to objects, you can change object data.

- However, you cannot change the object reference value itself.

Variable Scope
Advanced Topic: Garbage Collection

Have you ever wondered what happens to objects that you no longer need after you created them using **`new`**?

- Unlike other languages, a Java programmer is not responsible for deleting or destroying objects that you no longer use.
- When an object has no valid references to it, Java may delete the object in memory in a process called ***garbage collection***.

The lifetime of an object in memory:

- 1) The object is created using `new` and a reference to its location in memory is created.
- 2) The object may have multiple object references during the program execution.
- 3) When all object reference variables go out of scope, the object has no more references and is marked for deletion.
- 4) Java periodically scans memory and deletes objects.

Variable Scope Practice Questions

With this code explain the lifetime and scope of all variables.

```
public class VariableScope
{  public static void main(String[] args)
   {  double amount = 25;
      BankAccount acct = new BankAccount(200);
      for (int i=1; i <= 3; i++)
         acct.deposit(amount);
      System.out.println(acct.getBalance());  // 125.0
   }

   private void doNothing(double a)
   {  int i = 5; return;   }

   public static final int MYNUM = 25;
}
```

Variable Scope Practice Questions (2)

```
class BankAccount
{  public void deposit(double amount)
   {  if (amount <= balance)
      {  double newBalance = balance - amount;
         balance = newBalance;
      }
      double balance = 50;
   }
   public double getBalance()
   {  return balance; }

   public BankAccount(double b)
   {  balance = b; lastAccountNum++;
      accountNum = lastAccountNum;
   }

   private double balance;
   private int accountNum;
   private static int lastAccountNum = 0;
}
```

Conclusion

Key object-oriented terminology:

- ◆***Object*** – an instance of a class.
- ◆***Class*** – an object template with methods and properties.
- ◆***Method*** – a set of statements that performs an action.
- ◆***Parameter*** – data passed into a method.
- ◆***Properties*** – are attributes of objects.

Access specifiers limit what methods can access.

Inheritance is a mechanism for creating a new class by extending the features of an existing class.

Object references point to objects in memory. Use `new` to create objects. Methods are called using an object reference.

The scope and lifetime of a variable depends on its type (instance, static, local, parameter).

Objectives

Definitions: class, object, method, parameter, instance variable, inheritance, superclass, subclass, interface

Java skills:

- ◆Defining a class with variables and methods.
- ◆Meaning of access specifiers: public, protected, private.
- ◆Inheritance using `extends` and method/attribute inheritance.
- ◆Creating objects using `new`.
- ◆Purpose, use, and definition of constructors.
- **◆Difference between objects and object references.**
- ◆Calling methods using object references. Implicit parameter this.
- ◆Parameters are pass by value.
- ◆Variable scope and lifetime for variable types.
- ◆Advanced topics: shadowing, garbage collection

COSC 123
Computer Creativity

Java Lists and Arrays

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Objectives

1) Create and use arrays of base types and objects.

2) Create and use `ArrayList`.

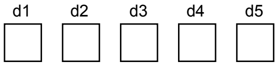
3) Understand the role of generic types to catch and prevent errors.

Arrays Overview

Suppose you need many variables in your program.

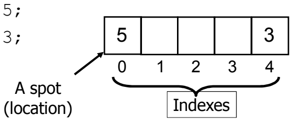
You could either create a separate name for each variable:

- ◆`double d1, d2, d3, d4, d5;`



Or you could create an array that has multiple spots (indexes):

- ◆`double[] myArray = new double[5];`
- ◆`myArray[0] = 5;`
- ◆`myArray[4] = 3;`



Arrays

An ***array*** is a collection of data items of the same type.

An array reference is denoted using the open and close square brackets "`[]`" during declaration.

⇨You can have an array of any data type including the base types (`int`, `double`, `String`) and object-types (`BankAccount`).

- ◆Examples:

```
int[] myArray;
String[] strings;
BankAccount[] accounts;
```

Similar to an object, when you declare an array you are creating a ***reference*** to an array. Until you actually create the space for the array using **`new`**, no array exists in memory.

- ◆`String[] strings = new String[10];`

Array Indexing

When creating an array using `new`, the number in square brackets is the number of elements in the array:

- ◆`double[] values = new double[20]; // 20 items`

Note the first element of the array has index 0 instead of 1.

- ◆In the previous example, the first index is 0 and the last is 19.

When an array is created, its values are initialized to defaults:

- ◆ 0 for numbers, `false` for boolean, `null` for object references

To access or set a value in an array, use its subscript:

- ◆`values[0] = 10;` // Sets first element to 10
- ◆`values[19] = values[0];` // Sets last element same as 1st

Array Details

Java performs automatic ***bound checking*** whenever an array element is referenced.

- ◆If the index is in the valid range, the reference is carried out.
- ◆If the index is not valid, an exception, `ArrayIndexOutOfBoundsException`, is thrown.

To get the length of an array in your program:

- ◆`int[] numbers = new int[25];`
- ◆`int size = numbers.length;` // Returns 25

You can initialize an array with values when you first declare it:

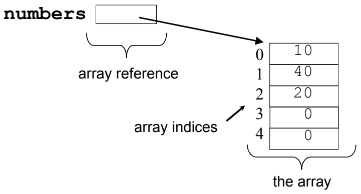
- ◆`int[] primes = {2, 3, 5, 7, 11};`

⇨ `new` is not used with an initializer list. Initializers can only be used during declaration.

Arrays in Memory Diagram Base Types

An array of base types stores the values in the slots in the array. Example:

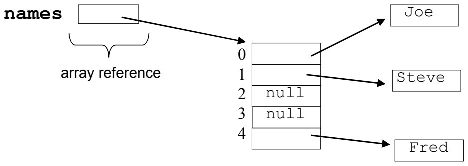
```
int[] numbers = new int[5];
numbers[0] = 10;
numbers[1] = 40;
numbers[2] = numbers[0]+10;
```



Arrays of Objects

An object array is an array of object references. Example:

```
String[] names = new String[5];
names[0] = "Joe";
names[1] = "Steve";
names[4] = "Fred";
```



When allocating an object array, Java does not also allocate space for each object in the array. Each object reference is initialized to null. You must create a new object for each object reference.

Arrays as Parameters and References

An array can be passed as a parameter to a method and returned from a method.

- ◆The values of the array can be changed but not the array reference itself. This is similar to how objects work.

Since an array is just a reference, it is possible to change which array a reference points to using assignment:

- ◆`int[] array1 = new int[10];`
- ◆`int[] array2 = new int[20];`
- ◆`array2 = array1;` // array2 now references array1

Practice Questions

1) Create an `int` array with name `myArray` with 20 elements.

2) Set the value of the 1st element to 10.

3) Set the value of the last element to 1.

4) Create an array that has 10 elements. Put the numbers from 1 to 10 in the array.

5) How do you know how many elements are in an array?

Arrays

Question: What is the size of this array?

```
int[] myArray = new int[10];
```

A) error
B) 10
C) 9
D) 11

Arrays

Question: What are the contents of this array?

```
int[] myArray = new int[4];
myArray[3] = 1;
myArray[2] = 2;
myArray[1] = 3;
myArray[0] = 4;
```

A) error
B) 0, 1, 2, 3
C) 1, 2, 3, 4
D) 4, 3, 2, 1

Java Collections

A ***collection*** is an object that serves as a repository for other objects. A collection provides methods to add, remove, and manage the elements it contains.

The underlying data structure used to implement the collection is independent of the operations the collection provides.

Java Collections API classes defines collection interfaces such as Set, List, SortedSet, Queue, and BlockingQueue.

List collection has two linear data structure implementations:

- ◆**ArrayList** - resizable-array implementation of the List interface.
- ◆**LinkedList** - linked list implementation of the List interface.

ArrayLists

An `ArrayList` implements a *resizable* array of objects.

- ⇨Base types such as `int` are not objects. Use wrapper class `Integer`.

Create an `ArrayList` by:

```
ArrayList names = new ArrayList(); // Size 10 (default)
ArrayList accounts = new ArrayList(5); // Size of 5
```

Add element to an `ArrayList` by:

```
names.add("Joe");       // Add to end of list
names.add(2,"Steve");   // Add at index 2 and shift up
```

Remove element from an `ArrayList` by:

```
names.remove(2);        // Remove index 2 and shift down
```

ArrayLists (2)

Get number of items in list by:

```
 int count = names.size();
```

Get element at an index from an `ArrayList` by:

```
String n = names.get(2);   // Get item at index 2
```

Set element at an index in an `ArrayList` by:

```
names.set(2,"Fred");       // Put Fred at index 2
```

Traversing an ArrayList

A simple way to traverse an `ArrayList` is using a for loop:

```
for (int i=0; i < names.size(); i++)
{  String s = (String) names.get(i);
   System.out.println(s);
}
```

Traversing an ArrayList Iterators

All collections also have iterators which are special classes designed to allow you to traverse through the collection.

Using an iterator with an `ArrayList`:

```
Iterator it = names.iterator();
while (it.hasNext())
{  String s = (String) it.next();
   System.out.println(s);
}
```

Generic Types

Collections store any type of object as all objects are a subclass of `Object`. It is better to precisely specify what objects are in a collection so that the compiler can check for errors.

All collections support ***generic*** (or parameterized) ***types*** to indicate what type is stored in the collection.

Examples:

```
// ArrayList can ONLY store strings

ArrayList<String> myNames = new ArrayList<String>(5);

// This ArrayList can only store BankAccount objects
ArrayList<BankAccount> accounts
                  = new ArrayList<BankAccount>();
```

ArrayList Example

```
import java.util.ArrayList;

public class TestArrayList
{  public static void main(String[] args)
   {  ArrayList a = new ArrayList();
      BankAccount b1, b = new BankAccount(100);
      SavingsAccount s1, s = new SavingsAccount(5,50);

      a.add(b);   // Add bank account b to list
      a.add(0,s); // Add s to front of list
      b1 = (BankAccount) a.get(1);
      s1 = (SavingsAccount) a.get(0);
      System.out.println(b1.getBalance());
      System.out.println(s1.getBalance());

      a.remove(0); // Remove s from list
      System.out.println(a.size()); // Prints 1
   }
}
```

ArrayList

Question: What is the value of `st`?

```
ArrayList a = new ArrayList();
a.add("Fred");
a.add(0,"Joe");
a.add("Steve");
a.remove(1);
String st = (String) a.get(1);
```

A) Fred

B) Joe

C) Steve

D) error

Practice Questions

1) Write a method **`reverse`** that returns a new array that contains the reverse sequence of numbers.

- Example:
  - ⇨1 4 9 19 9 7 4 9 11 becomes 11 9 4 7 9 19 9 4 1

2) Write a method that reads in strings using Scanner and stores them in an `ArrayList` until "STOP" is entered. Print out the list after you finish reading.

Conclusion

Arrays are a data structure for storing multiple items using the same name.

- ◆An array has a fixed size and is indexed from 0 to size-1.
- ◆An array can store both base types or object references.

A collection is an object that stores other objects and provides methods for adding, removing, and retrieving objects.

An `ArrayList` is a linear collection.

- ◆`ArrayList` has methods for adding, removing, getting, and setting values.
- ◆`ArrayList` can be traversed using a loop or an iterator.
- ◆A generic type ensures the collection only stores the proper objects.

Objectives

Java skills:

- ◆Creating an array
- ◆Array indexing and bounds checking
- ◆Arrays of base types and objects
- ◆Arrays as parameters
- ◆Copying arrays and `System.arraycopy`
- ◆Two-dimensional arrays
- ◆`ArrayList` – create, add, remove, get, set, traversing
- ◆Iterators

COSC 123
Computer Creativity

Graphics and Events

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Key Points

1) Draw shapes, text in various fonts, and colors.

2) Build window applications using `JFrame`/`JPanel` and Swing components.

3) Understand events, event listeners, and event adapters.

4) Write code for handling mouse, keyboard, and window events.

Java Programs Overview

To this point, all our Java programs have received input and displayed output in the console (text window).

Types of Java programs:

- ◆1) **Console applications** - text-based applications which perform input and output using the console
- ◆2) **Graphical applications** - stand-alone Java applications which have a graphic user interface with components such as windows, control buttons, menus, and check boxes.

Graphical Applications Overview

A ***graphical application*** is a Java program with a graphical user interface.

A ***frame window*** is a window on the screen that has a border and a title bar.

A frame window is defined in Java using the `JFrame` class that is present in the `javax.swing` package.

- ◆The `javax.swing` package is also called the ***Swing toolkit***.

Creating a Frame Windows

To create a frame window:

- ◆`import javax.swing.JFrame`
- ◆create our own class (like `MyFrame`) which `extends JFrame`
- ◆provide a constructor for our `MyFrame` class
- ◆set the size of our frame using the `setSize` method
  - ⇨usually performed in `MyFrame` constructor

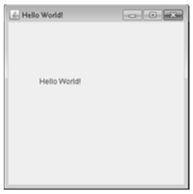
To use the `MyFrame` window:

- ◆define a mainline which instantiates a `MyFrame` instance
- ◆use the `setTitle` method to set the frame title (optional)
- ◆use the `setVisible` method to display the frame on the screen

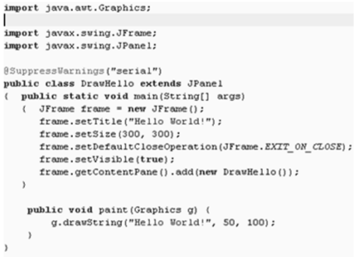
Graphical Applications Creating a Frame Window

```
import javax.swing.JFrame;
public class MyFrame extends JFrame
{  public static void main(String[] args)
   {  MyFrame frame = new MyFrame();
      frame.setTitle("Frame Title");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}

   public MyFrame()
   {  final int DEFAULT_FRAME_WIDTH = 300;
      final int DEFAULT_FRAME_HEIGHT = 300;
      setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HEIGHT);
   }
}
```

A “Hello World!” Window



The window displays `Hello World!`



```
import java.awt.Graphics;

import javax.swing.JFrame;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class DrawHello extends JPanel
{  public static void main(String[] args)
   {  JFrame frame = new JFrame();
      frame.setTitle("Hello World!");
      frame.setSize(300, 300);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
      frame.getContentPane().add(new DrawHello());
   }

    public void paint(Graphics g) {
        g.drawString("Hello World!", 50, 100);
    }
}
```

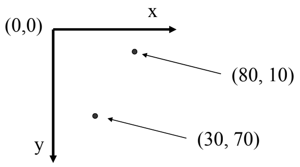
Question: What does `extends` mean?

The Coordinate System

Drawing on the screen is done by specifying coordinates which refer to a location on the screen.

- The ***origin*** is the upper-left hand corner of the screen.
- The x coordinate gets bigger as we move to the right.
- The y coordinate gets bigger as we move down.

Diagram:



`drawString` Method

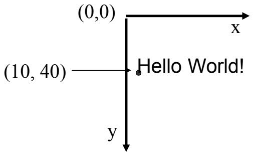
The `drawString` method draws a text string on the screen.

Usage:

- `g.drawString(`*`message`*`, `*`x`*`, `*`y`*`)`
  - ⇨x, y co-ordinates are the base point of the message

Example:

- `g.drawString("Hello World!", 10, 40);`



Drawing

To draw shapes on the screen, we use the `draw` method.

The `draw` method takes a shape that we create and draws it on the screen.

Example:

- `Rectangle box = new Rectangle(10, 10, 20, 30);`
- `g2.draw(box);`

Drawing Shapes

There are several methods to draw shapes:

- 1) Ellipse:
  - ⇨ Ellipse2D.Double egg = new Ellipse2D.Double(topx, topy, width, height);
  - ⇨ Ellipse2D.Double egg = new Ellipse2D.Double(5, 10, 15, 20);
- 2) Rectangle:
  - ⇨Rectangle box = new Rectangle(topx, topy, width, height);
  - ⇨Rectangle box = new Rectangle(10, 10, 20, 30);
- 3) Line:
  - ⇨Line2D.Double = new Line2D.Double(x1, y1, x2, y2);
- 4) Point:
  - ⇨Point2D.Double = new Point2D.Double(x,y);

You can also fill a shape with a color using the `fill` method:

- ⇨g2.fill(box);
- ⇨g2.fill(egg);

Changing Colors

There are 3 basic display colors which are combined to form all colors displayed on a computer.

- Red, green, and blue are used in the RGB color model.
- Any color can be defined by specifying what percentage of red, blue, and green is in the color.

The class for colors in Java is called `Color`.

- `import java.awt.Color;`
- `Color orange = new Color(1.0F, 0.8F, 0.0F);`

Changing what color your text or shapes is drawn in:

- `g.setColor(orange);`

There are also static colors predefined in Java:

- `Color.black`, `Color.red`, `Color.white`, `Color.orange`, etc.

Changing Fonts

The `drawString` method uses a default font if none is given.

A font consists of:

- a font face name (Serif, SansSerif, Monospaced, Dialog, etc.)
- a style (`Font.PLAIN`, `Font.BOLD`, `Font.ITALIC`, etc.)
- a font size (specified in points: 1 inch = 72 points)

The font class in Java is called `Font`:

- `import java.awt.Font;`
- `Font bigFont = new Font("Serif", Font.BOLD, 36);`

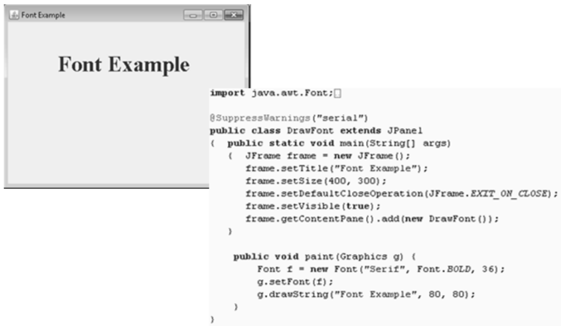
Set the current font:

- `g.setFont(bigFont);`

Then use drawString:

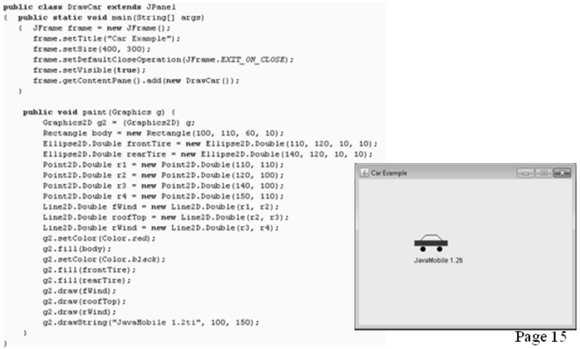
- `g.drawString("Hello World!", 50, 100);`

Drawing Fonts



```
import java.awt.Font;□

@SuppressWarnings("serial")
public class DrawFont extends JPanel
{   public static void main(String[] args)
    {   JFrame frame = new JFrame();
        frame.setTitle("Font Example");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        frame.getContentPane().add(new DrawFont());
    }

     public void paint(Graphics g) {
         Font f = new Font("Serif", Font.BOLD, 36);
         g.setFont(f);
         g.drawString("Font Example", 80, 80);
     }
}
```

Car Drawing Example



```
public class DrawCar extends JPanel
{   public static void main(String[] args)
    {   JFrame frame = new JFrame();
        frame.setTitle("Car Example");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        frame.getContentPane().add(new DrawCar());
    }

     public void paint(Graphics g) {
         Graphics2D g2 = (Graphics2D) g;
         Rectangle body = new Rectangle(100, 110, 60, 10);
         Ellipse2D.Double frontTire = new Ellipse2D.Double(110, 120, 10, 10);
         Ellipse2D.Double rearTire = new Ellipse2D.Double(140, 120, 10, 10);
         Point2D.Double r1 = new Point2D.Double(110, 110);
         Point2D.Double r2 = new Point2D.Double(120, 100);
         Point2D.Double r3 = new Point2D.Double(140, 100);
         Point2D.Double r4 = new Point2D.Double(150, 110);
         Line2D.Double fWind = new Line2D.Double(r1, r2);
         Line2D.Double roofTop = new Line2D.Double(r2, r3);
         Line2D.Double rWind = new Line2D.Double(r3, r4);
         g2.setColor(Color.red);
         g2.fill(body);
         g2.setColor(Color.black);
         g2.fill(frontTire);
         g2.fill(rearTire);
         g2.draw(fWind);
         g2.draw(roofTop);
         g2.draw(rWind);
         g2.drawString("JavaMobile 1.2ti", 100, 150);
     }
}
```

Exercises

1) Draw three circles of different colors.

2) Draw a "better" looking car.

Adding Components to a Frame

Content is added to the frame on the content pane.

One common component to add is a `JPanel` that allows you to draw graphics.

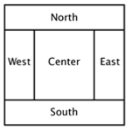
- A ***container*** is a component that can hold other components.

There are five regions of a `JFrame` where you can place components:

- North, West, Center, East, South

Example:

```
Container contentPane = getContentPane();
MyPanel panel = new MyPanel();
contentPane.add(panel, "Center");
```

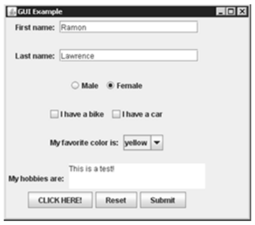


Java Swing Components

The Java Swing package contains the user interface components that we will use in our graphical applications.

Component	**Import Package**
JButton	javax.swing.JButton
ButtonGroup	javax.swing.ButtonGroup
Check box	javax.swing.JCheckBox
Combo box	javax.swing.JComboBox
JFrame	javax.swing.JFrame
JLabel	javax.swing.JLabel
JPanel	javax.swing.JPanel
Radio button	javax.swing.JRadioButton
Text field	javax.swing.JTextField

GUI Example



First name: Ramon

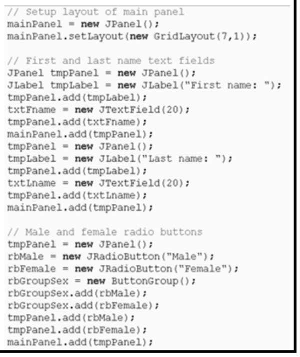
Last name: Lawrence

○ Male ● Female

□ I have a bike □ I have a car

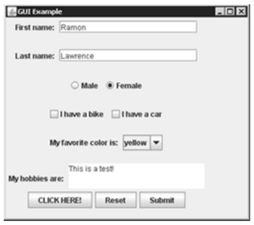
My favorite color is: yellow

My hobbies are: This is a test!



```
// Setup layout of main panel
mainPanel = new JPanel();
mainPanel.setLayout(new GridLayout(7,1));

// First and last name text fields
JPanel tmpPanel = new JPanel();
JLabel tmpLabel = new JLabel("First name: ");
tmpPanel.add(tmpLabel);
txtFname = new JTextField(20);
tmpPanel.add(txtFname);
mainPanel.add(tmpPanel);
tmpPanel = new JPanel();
tmpLabel = new JLabel("Last name: ");
tmpPanel.add(tmpLabel);
txtLname = new JTextField(20);
tmpPanel.add(txtLname);
mainPanel.add(tmpPanel);

// Male and female radio buttons
tmpPanel = new JPanel();
rbMale = new JRadioButton("Male");
rbFemale = new JRadioButton("Female");
rbGroupSex = new ButtonGroup();
rbGroupSex.add(rbMale);
rbGroupSex.add(rbFemale);
tmpPanel.add(rbMale);
tmpPanel.add(rbFemale);
mainPanel.add(tmpPanel);
```

GUI Example (2)



First name: Ramon

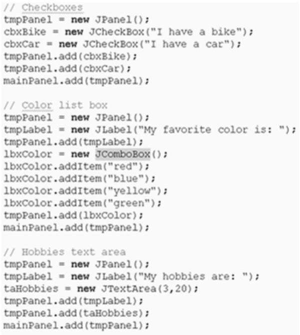
Last name: Lawrence

○ Male ● Female

□ I have a bike □ I have a car

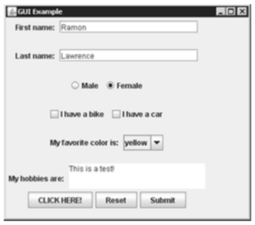
My favorite color is: yellow

My hobbies are: This is a test!



```
// Checkboxes
tmpPanel = new JPanel();
cbxBike = new JCheckBox("I have a bike");
cbxCar = new JCheckBox("I have a car");
tmpPanel.add(cbxBike);
tmpPanel.add(cbxCar);
mainPanel.add(tmpPanel);

// Color list box
tmpPanel = new JPanel();
tmpLabel = new JLabel("My favorite color is: ");
tmpPanel.add(tmpLabel);
lbxColor = new JComboBox();
lbxColor.addItem("red");
lbxColor.addItem("blue");
lbxColor.addItem("yellow");
lbxColor.addItem("green");
tmpPanel.add(lbxColor);
mainPanel.add(tmpPanel);

// Hobbies text area
tmpPanel = new JPanel();
tmpLabel = new JLabel("My hobbies are: ");
taHobbies = new JTextArea(3,20);
tmpPanel.add(tmpLabel);
tmpPanel.add(taHobbies);
mainPanel.add(tmpPanel);
```

GUI Example (3)



First name: Ramon

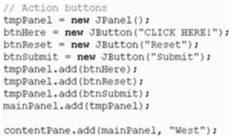
Last name: Lawrence

○ Male ● Female

□ I have a bike □ I have a car

My favorite color is: yellow

My hobbies are: This is a test!



```
// Action buttons
tmpPanel = new JPanel();
btnHere = new JButton("CLICK HERE!");
btnReset = new JButton("Reset");
btnSubmit = new JButton("Submit");
tmpPanel.add(btnHere);
tmpPanel.add(btnReset);
tmpPanel.add(btnSubmit);
mainPanel.add(tmpPanel);

contentPane.add(mainPanel, "West");
```

GUI Components
`JLabel`

The `JLabel` class is used to display a label (or text) on the screen that cannot be edited by the user.

```
JLabel myLabel = new JLabel("My Label",
                            SwingConstants.RIGHT);
```

A label can be aligned by using:

- ◆Center - `SwingConstants.CENTER`
- ◆Right - `SwingConstants.RIGHT`
- ◆Left - `SwingConstants.LEFT`

GUI Components
`JTextField` and `JTextArea`

`JTextField` allows us to read in a single line of text.
`JTextArea` allows us to handle multiple lines of text.

With a `JTextField`, you may give the # of characters:

```
JTextField txtField = new JTextField(5); // 5 chars.
```

With a `JTextArea`, you can give the # of rows/cols:

```
JTextArea txtArea = new JTextArea(5,40);//5 rows, 40 cols
```

GUI Components
`JTextField` and `JTextArea` Methods

Some useful methods for text fields:

```
JTextField txtField = new JTextField();

txtField.setText("Hello World!"); // Set the field text
txtField.setEditable(false);  // Do not allow field edits
txtField.setFont(hugeFont);   // Change the field font
txtField.getText();           // Get current field text
```

GUI Components
`JRadioButton` Overview

The `JRadioButton` class allows the user to select from disjoint inputs (i.e. the user can select only one out of a list).

```
JRadioButton smallButton = new JRadioButton("Small");
JRadioButton mediumButton = new JRadioButton("Medium");
JRadioButton largeButton = new JRadioButton("Large");

ButtonGroup sizeGroup = new ButtonGroup();
sizeGroup.add(smallButton);
sizeGroup.add(mediumButton);
sizeGroup.add(largeButton);
```

The `ButtonGroup` class allows the programmer to specify which buttons are grouped with each other.

You can select buttons or determine if buttons are selected by:

```
smallButton.setSelected(true);
if (smallButton.isSelected()) return "Small";
```

GUI Components
`JCheckBox` Overview

The `JCheckBox` class allows the user to select yes/no valued inputs (i.e. true or false).

```
JCheckBox boldCheckBox = new JCheckBox("Bold");
```

- Note: Do not place check boxes inside a button group because they are not mutually exclusive.

GUI Components
`JComboBox` Overview

The `JComboBox` class allows the user to select from a large list of disjoint inputs where radio buttons are too awkward.

- A `JComboBox` allows you to select an item from the list.
- If the list is editable, you can type in your own selection that may not already be in the list.

```
JComboBox itemCombo = new JComboBox();
itemCombo.addItem("Item 1");
itemCombo.addItem("Item 2");
```

You can get the selected item in the list by:

```
String st = (String) itemCombo.getSelectedItem();
```

- Note that `JComboBox`, `JCheckBox`, and `JRadioButton` all generate action events that should be detected using an action listener.

GUI Components
`JButton` Overview

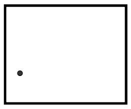
The `JButton` class allows you to put a button on your frame.

When creating a button, it can have just text, just a picture, or a picture and text.

```
leftButton = new JButton("left");
leftButton = new JButton(new ImageIcon("left.gif"));
leftButton = new JButton("left",newImageIcon("left.gif"));
```

Coordinates

Question: Select from the coordinates below the pair that best describes this point's location. Assume box is 100 by 100.



A) (10,80)
B) (80,10)
C) (10,20)
D) (20,10)

Components

Question: What is the best component to use if the user can select yes/no to multiple items independently?

A) `JRadioButton`
B) `JComboBox`
C) `JCheckBox`
D) `JButton`

Components

Question: What is the best component to use if the user must pick only one item from 50 possible choices?

A) `JRadioButton`
B) `JComboBox`
C) `JCheckBox`
D) `JButton`

Events and Event Handling GUI Programming Philosophy

In ***graphical applications***, the programmer must ***react*** instead of ***dictate*** the events that occur in a program.

As a programmer, you design a graphical user interface with windows, buttons, and components that the user can interact with. You do not know the order or the sequence of events the user will generate, but you must be able to react to them.

Events and Event Handling Overview

An ***event*** is a notification to your program that something has occurred.

- ◆For graphical events (mouse click, data entry), the Java window manager notifies your program that an event occurred.
  - ⇨There are different ***kinds*** of events such as keyboard events, mouse click events, mouse movement events, etc.

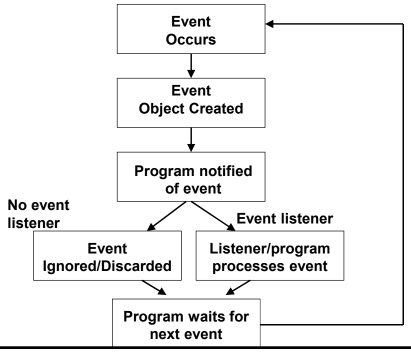
An ***event handler*** or ***listener*** is part of your program that is responsible for "listening" for event notifications and handling them properly.

- ◆An event listener often only listens for certain types of events.

An ***event source*** is the user interface component that generated the event.

- ◆A button, a window, and scrollbars are all event sources.

Event Handling Overview



Mouse Event Example

Handling mouse click events requires three classes:

- ◆1) The **event class** - that stores information about the event.
  - ⇨For mouse clicks, this class is `MouseEvent`.
  - ⇨The `MouseEvent` class has methods `getX()` and `getY()` that indicate the position of the mouse at the time the event was generated.
  - ⇨Each event class has the method `Object getSource()` that returns the source of the event.
- ◆2) The **listener class** - allows your program to detect events. Building your own listener class requires implementing a pre-defined interface.
  - ⇨For mouse clicks, the interface is `MouseListener`. `MouseAdapter` is a class that implements the `MouseListener` interface.
- ◆3) The **event source** - is the component in your GUI that generated the event.

MouseListener Interface

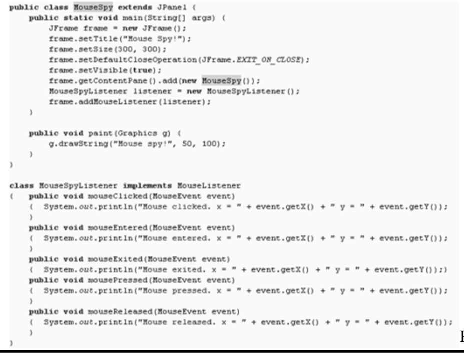
The `MouseListener` interface must be implemented by your class that handles mouse events. It has the methods:

```
public interface MouseListener
{   void mouseClicked(MouseEvent event);
    // Called when the mouse has been clicked on component
    void mouseEntered(MouseEvent event);
    // Called when the mouse enters a component
    void mouseExited(MouseEvent event);
    // Called when the mouse exits a component
    void mousePressed(MouseEvent event);
    // Called when a mouse button pressed on a component
    void mouseReleased(MouseEvent event);
    // Called when mouse button released on a component
}
```

To add a listener, use the method:

- ◆`addMouseListener(`*`listener_name`*`);`

Mouse Event Example Code



```
public class MouseSpy extends JPanel {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setTitle("Mouse Spy!");
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        frame.getContentPane().add(new MouseSpy());
        MouseSpyListener listener = new MouseSpyListener();
        frame.addMouseListener(listener);
    }

    public void paint(Graphics g) {
        g.drawString("Mouse spy!", 50, 100);
    }
}

class MouseSpyListener implements MouseListener
{   public void mouseClicked(MouseEvent event)
    {  System.out.println("Mouse clicked. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseEntered(MouseEvent event)
    {  System.out.println("Mouse entered. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseExited(MouseEvent event)
    {  System.out.println("Mouse exited. x = " + event.getX() + " y = " + event.getY());}
    public void mousePressed(MouseEvent event)
    {  System.out.println("Mouse pressed. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseReleased(MouseEvent event)
    {  System.out.println("Mouse released. x = " + event.getX() + " y = " + event.getY());
    }
}
```

Event Listeners and Inner Classes

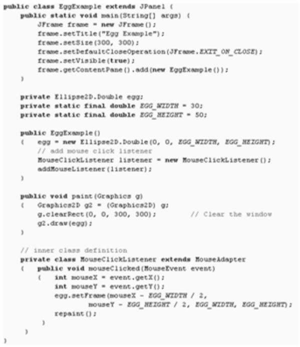
Typically, your event listener class will perform some function based on the user input.

- ◆This function often involves accessing the private variables of the `Frame` class you defined.
- ◆However, if the listener class is implemented outside of the `Frame` class, it has no more access rights to the private instance variables of that class then any other class.
- ◆The solution to this problem is to use inner classes.

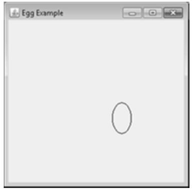
An ***inner class*** is a class that is defined inside another class.

- ◆The methods of the inner class have access to the private variables of the outer class.
- ◆The inner class is typically defined as `private`.
- ◆An inner class object remembers the object that created it.

Egg Draw Example Code



```
public class EggExample extends JPanel {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setTitle("Egg Example");
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        frame.getContentPane().add(new EggExample());
    }

    private Ellipse2D.Double egg;
    private static final double EGG_WIDTH = 30;
    private static final double EGG_HEIGHT = 50;

    public EggExample()
    {   egg = new Ellipse2D.Double(0, 0, EGG_WIDTH, EGG_HEIGHT);
        // add mouse click listener
        MouseClickListener listener = new MouseClickListener();
        addMouseListener(listener);
    }

    public void paint(Graphics g)
    {   Graphics2D g2 = (Graphics2D) g;
        g.clearRect(0, 0, 300, 300);        // Clear the window
        g2.draw(egg);
    }

    // inner class definition
    private class MouseClickListener extends MouseAdapter
    {   public void mouseClicked(MouseEvent event)
        {   int mouseX = event.getX();
            int mouseY = event.getY();
            egg.setFrame(mouseX - EGG_WIDTH / 2,
                    mouseY - EGG_HEIGHT / 2, EGG_WIDTH, EGG_HEIGHT);
            repaint();
         }
     }
}
```



Draws an ellipse where the user clicks.

- ◆ The mouse listener is an inner class.
- ◆ Every time the user clicks the mouse, the listener repositions the ellipse and calls `repaint` to redraw.

WindowListener Interface

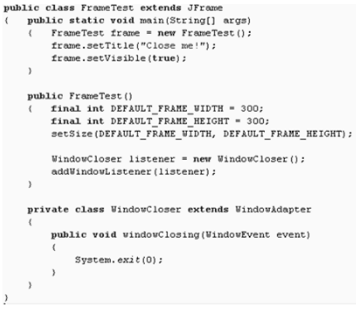
The `WindowListener` interface must be implemented by your frame class to handle its events. It has the methods:

```
public interface WindowListener
{   void windowOpened(WindowEvent e);
    void windowClosed(WindowEvent e);
    void windowActivated(WindowEvent e);
    void windowDeactivated(WindowEvent e);
    void windowIconified(WindowEvent e);
    void windowDeiconfied(WindowEvent e);
    void windowClosing(WindowEvent e);
}
```

Most programs only care about the window closed event.

- ◆That is where `System.exit(0)` is typically placed.
- ◆There is also a `WindowAdapter` class that can be extended instead of implementing all interface methods.

Frame Window Event Example



```
public class FrameTest extends JFrame
{   public static void main(String[] args)
    {   FrameTest frame = new FrameTest();
        frame.setTitle("Close me!");
        frame.setVisible(true);
    }

    public FrameTest()
    {   final int DEFAULT_FRAME_WIDTH = 300;
        final int DEFAULT_FRAME_HEIGHT = 300;
        setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HEIGHT);

        WindowCloser listener = new WindowCloser();
        addWindowListener(listener);
    }

    private class WindowCloser extends WindowAdapter
    {
        public void windowClosing(WindowEvent event)
        {
            System.exit(0);
        }
    }
}
```

Note the use of `WindowAdapter` as we only care about the window closing event.

Action Listeners

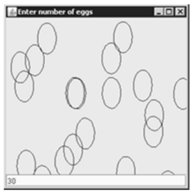
GUI components like buttons, text fields, combo boxes, and check boxes all generate action events.

The **`ActionListener`** interface has a single method:

```
public interface ActionListener
{   public void actionPerformed(ActionEvent event);
}
```

An action event is generated when you click on the control or press Enter for text fields.

Eggs.Java Example



In this example, we will create a `JFrame` with a `JPanel` and a `JTextField` and ask the user for the number of ellipses ("eggs") to draw on the screen.

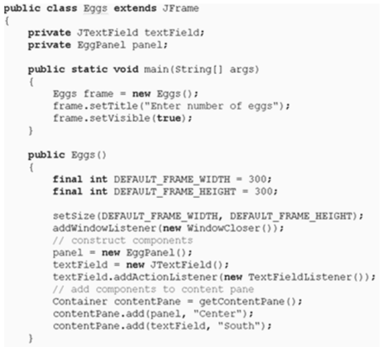
Notes:

1) The `JPanel` component paints itself in the `paintComponent` method. This method MUST call `super.paintComponent` as the first line.

- ◆ Note that this is different than the `paint` method in applets.

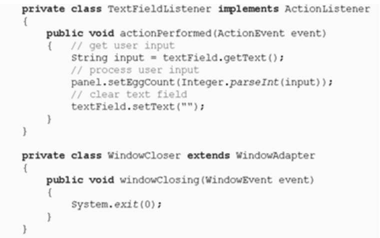
2) When the user changes the value in the text field, you must call `repaint` to get the `JPanel` to repaint itself.

Eggs.Java Example Code



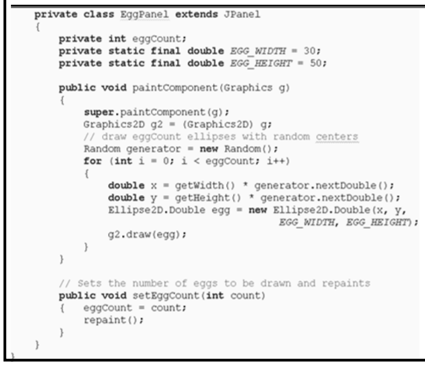
```
public class Eggs extends JFrame
{
    private JTextField textField;
    private EggPanel panel;

    public static void main(String[] args)
    {
        Eggs frame = new Eggs();
        frame.setTitle("Enter number of eggs");
        frame.setVisible(true);
    }

    public Eggs()
    {
        final int DEFAULT_FRAME_WIDTH = 300;
        final int DEFAULT_FRAME_HEIGHT = 300;

        setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HEIGHT);
        addWindowListener(new WindowCloser());
        // construct components
        panel = new EggPanel();
        textField = new JTextField();
        textField.addActionListener(new TextFieldListener());
        // add components to content pane
        Container contentPane = getContentPane();
        contentPane.add(panel, "Center");
        contentPane.add(textField, "South");
    }
```

Eggs.Java Example Code (2)



```
    private class TextFieldListener implements ActionListener
    {
        public void actionPerformed(ActionEvent event)
        {   // get user input
            String input = textField.getText();
            // process user input
            panel.setEggCount(Integer.parseInt(input));
            // clear text field
            textField.setText("");
        }
    }

    private class WindowCloser extends WindowAdapter
    {
        public void windowClosing(WindowEvent event)
        {
            System.exit(0);
        }
    }
```

Eggs.Java Example Code (3)



```
    private class EggPanel extends JPanel
    {
        private int eggCount;
        private static final double EGG_WIDTH = 30;
        private static final double EGG_HEIGHT = 50;

        public void paintComponent(Graphics g)
        {
            super.paintComponent(g);
            Graphics2D g2 = (Graphics2D) g;
            // draw eggCount ellipses with random centers
            Random generator = new Random();
            for (int i = 0; i < eggCount; i++)
            {
                double x = getWidth() * generator.nextDouble();
                double y = getHeight() * generator.nextDouble();
                Ellipse2D.Double egg = new Ellipse2D.Double(x, y,
                                                EGG_WIDTH, EGG_HEIGHT);
                g2.draw(egg);
            }
        }

        // Sets the number of eggs to be drawn and repaints
        public void setEggCount(int count)
        {   eggCount = count;
            repaint();
        }
    }
}
```

`JButton` and `ActionListener`

When a button is clicked, it sends an action event that must be captured using an action listener.

```
leftButton = new JButton("left");
ActionListener listener = new ButtonListener();
leftButton.addActionListener(listener);
```

You may either create one action listener for all buttons (which uses the `event.getSource` method to determine the button pressed) or create a separate listener for each button.

One Button Action Listener

```
public class MyFrame
{  public MyFrame()
   {  ...
      upButton = new JButton("Up");
      ActionListener listener = new UpListener();
      upButton.addActionListener(listener);
      ...
   }
   ...
   private JButton upButton;
   ...
   private class UpListener implements ActionListener
   {  public void actionPerformed(ActionEvent event)
      {  // performs action when up button is clicked
      }
   }
}
```

Multiple Button Action Listener

```
public class MyFrame
{  public MyFrame()
   {  ...
      upButton = new JButton("Up");
      downButton = new JButton("Down");
      leftButton = new JButton("Left");
      rightButton = new JButton("Right");
      ActionListener listener = new DirectionListener();
      upButton.addActionListener(listener);
      downButton.addActionListener(listener);
      leftButton.addActionListener(listener);
      rightButton.addActionListener(listener);
      // create a Panel containing all buttons and add
      // to content pane
   }
   private JButton upButton, downButton, leftButton;
   private JButton rightButton;
```

Multiple Button Action Listener (2)

```
   ...
   private class DirectionListener implements
                                    ActionListener
   {  public void actionPerformed(ActionEvent event)
      {  // performs action when any button is clicked
         Object source = event.getSource();
         if (source == upButton)
         // Perform up action
         else if (source == downButton)
         // Perform down action
         else if (source == leftButton)
         // Perform left action
         else if (source == rightButton)
         // Perform right action
      }
   }
```

Listeners and Adapters

Question: Which one is a true statement?

A) To handle mouse events, create a class that `extends MouseListener`.

B) To handle mouse events, create a class that `implements MouseAdapter`.

C) To handle mouse events, create a class that `extends MouseAdapter`.

D) You must implement all event methods when your class extends `MouseAdapter`.

Practice Questions

1) Create a program that displays the string "Hello world" at the location where the user clicks the mouse.

- Notes:
  - When the user clicks a mouse, move the location of "Hello World!".
  - Use `MouseAdapter` and inner classes.

2) Create a program that opens up a window with "1" as the title. Then,

- If the user clicks on the window, a new window is opened with value of "2".
- If the user clicks on either open window, a new window is opened with value of "3". This may repeat for any # of windows.
- When a window is closed, all other windows stay open.
- When the last window is closed, the program quits.

Menus Overview

Menus allow the user to select options without using buttons and fields.

- A menu is located at the top of the frame in a menu bar.

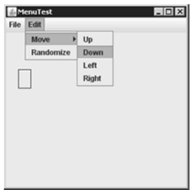
A ***menu*** is a collection of menu items and more menus.

- You add menu items and submenus with the add method.

When a menu item is selected, it generates an action event.

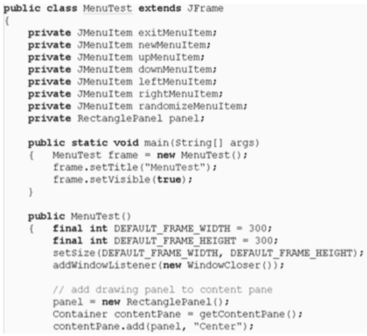
- Thus, each menu item should have a listener defined.

Menus Example



In this example, we will create a menu that allows us to move a rectangle around the window based on user selections.

Menus Example Code

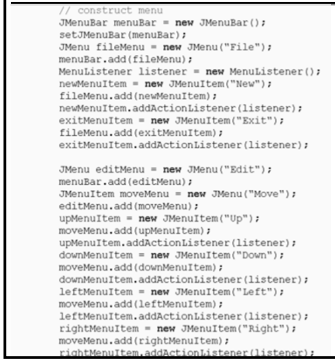


```
public class MenuTest extends JFrame
{
    private JMenuItem exitMenuItem;
    private JMenuItem newMenuItem;
    private JMenuItem upMenuItem;
    private JMenuItem downMenuItem;
    private JMenuItem leftMenuItem;
    private JMenuItem rightMenuItem;
    private JMenuItem randomizeMenuItem;
    private RectanglePanel panel;

    public static void main(String[] args)
    {   MenuTest frame = new MenuTest();
        frame.setTitle("MenuTest");
        frame.setVisible(true);
    }

    public MenuTest()
    {   final int DEFAULT_FRAME_WIDTH = 300;
        final int DEFAULT_FRAME_HEIGHT = 300;
        setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HEIGHT);
        addWindowListener(new WindowCloser());

        // add drawing panel to content pane
        panel = new RectanglePanel();
        Container contentPane = getContentPane();
        contentPane.add(panel, "Center");
```

This is the basic setup for creating a frame.

Note the `JMenuItem` instance variables, one for each menu item.

Menus Example Code (2)



```
        // construct menu
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);
        JMenu fileMenu = new JMenu("File");
        menuBar.add(fileMenu);
        MenuListener listener = new MenuListener();
        newMenuItem = new JMenuItem("New");
        fileMenu.add(newMenuItem);
        newMenuItem.addActionListener(listener);
        exitMenuItem = new JMenuItem("Exit");
        fileMenu.add(exitMenuItem);
        exitMenuItem.addActionListener(listener);

        JMenu editMenu = new JMenu("Edit");
        menuBar.add(editMenu);
        JMenuItem moveMenu = new JMenu("Move");
        editMenu.add(moveMenu);
        upMenuItem = new JMenuItem("Up");
        moveMenu.add(upMenuItem);
        upMenuItem.addActionListener(listener);
        downMenuItem = new JMenuItem("Down");
        moveMenu.add(downMenuItem);
        downMenuItem.addActionListener(listener);
        leftMenuItem = new JMenuItem("Left");
        moveMenu.add(leftMenuItem);
        leftMenuItem.addActionListener(listener);
        rightMenuItem = new JMenuItem("Right");
        moveMenu.add(rightMenuItem);
        rightMenuItem.addActionListener(listener);
```

Still in the constructor, this code begins by creating a `JMenuBar` and setting it as the frame's menu bar.

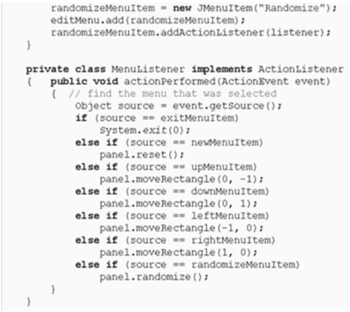
Then, the file menu is created with two items: new and exit.

Note that an `ActionListener` is added for each menu item, and it is the same listener object.

Later we will see how the listener determines what menu item was selected.

The next code creates the edit menu. Note that the move menu is a submenu of the edit menu.

Menus Example Code (3)



```
        randomizeMenuItem = new JMenuItem("Randomize");
        editMenu.add(randomizeMenuItem);
        randomizeMenuItem.addActionListener(listener);
    }

    private class MenuListener implements ActionListener
    {   public void actionPerformed(ActionEvent event)
        {  // find the menu that was selected
            Object source = event.getSource();
            if (source == exitMenuItem)
                System.exit(0);
            else if (source == newMenuItem)
                panel.reset();
            else if (source == upMenuItem)
                panel.moveRectangle(0, -1);
            else if (source == downMenuItem)
                panel.moveRectangle(0, 1);
            else if (source == leftMenuItem)
                panel.moveRectangle(-1, 0);
            else if (source == rightMenuItem)
                panel.moveRectangle(1, 0);
            else if (source == randomizeMenuItem)
                panel.randomize();
        }
    }
```

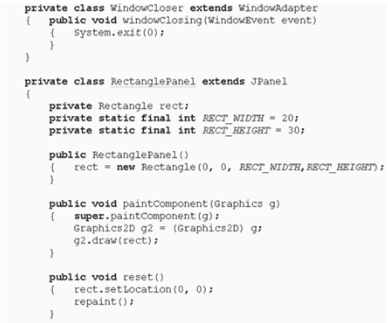
The top of the code finishes the edit menu by adding the randomize menu item.

The `MenuListener` is the class that is used to respond to menu action events.

Note that the `getSource` method is used to determine the menu item selected which is then compared with all the menu items created.

Once the appropriate menu item is found, the correct method is called to perform the menu action.

Menus Example Code (4)



```
    private class WindowCloser extends WindowAdapter
    {   public void windowClosing(WindowEvent event)
        {   System.exit(0);
        }
    }

    private class RectanglePanel extends JPanel
    {
        private Rectangle rect;
        private static final int RECT_WIDTH = 20;
        private static final int RECT_HEIGHT = 30;

        public RectanglePanel()
        {   rect = new Rectangle(0, 0, RECT_WIDTH,RECT_HEIGHT);
        }

        public void paintComponent(Graphics g)
        {   super.paintComponent(g);
            Graphics2D g2 = (Graphics2D) g;
            g2.draw(rect);
        }

        public void reset()
        {   rect.setLocation(0, 0);
            repaint();
        }
```

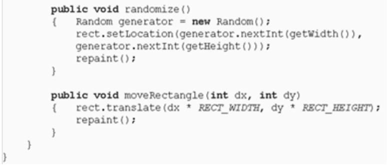
A standard class extending `WindowAdapter` is used to detect when the window is closed and to terminate the application.

`RectanglePanel` is the panel where the rectangle is drawn.

The rectangle has a fixed size and starts off at (0,0).

The `reset` method is called when the new menu item is selected. It places the rectangle back at (0,0) and calls `repaint` to make sure the panel is redrawn to reflect the changes.

Menus Example Code (5)



```
        public void randomize()
        {   Random generator = new Random();
            rect.setLocation(generator.nextInt(getWidth()),
            generator.nextInt(getHeight()));
            repaint();
        }

        public void moveRectangle(int dx, int dy)
        {   rect.translate(dx * RECT_WIDTH, dy * RECT_HEIGHT);
            repaint();
        }
    }
}
```

The `randomize` method places the rectangle at a random location in the window and redraws the panel.

The `moveRectangle` method moves the rectangle an amount left/right (`dx`) or up/down (`dy`) from its current location.

Exercise

Create an application that has a File menu and an edit menu.

- ◆The file menu should have an exit item that closes the application.
- ◆The edit menu should have two subitems:
  - ⇨shape – has submenu of rectangle, square, and circle
  - ⇨color – has submenu of red, green, blue, yellow
- ◆When the use selects a shape and color, remember the shape and color. Default is rectangle and red.
- ◆When the user clicks on a place on the screen, draw that shape in that color.

Timer

A ***timer*** can be used to create events at set times. A timer generates `ActionEvents`.

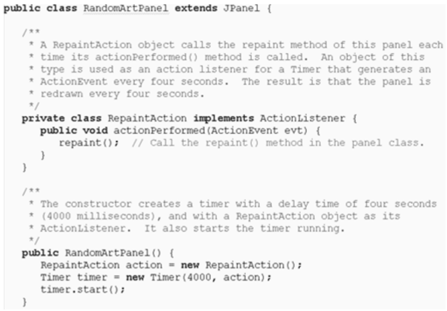
Creating a timer:

```
Timer timer = new Timer(1000, listener);
   // The timer fires every 1000 ms (1 second).
   // The listener class is called every time.
```

Starting and stopping a timer:

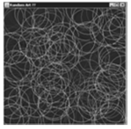
```
timer.start();
timer.stop();
```

Timer Example Code



```
public class RandomArtPanel extends JPanel {

   /**
    * A RepaintAction object calls the repaint method of this panel each
    * time its actionPerformed() method is called.  An object of this
    * type is used as an action listener for a Timer that generates an
    * ActionEvent every four seconds.  The result is that the panel is
    * redrawn every four seconds.
    */
   private class RepaintAction implements ActionListener {
      public void actionPerformed(ActionEvent evt) {
         repaint();  // Call the repaint() method in the panel class.
      }
   }

   /**
    * The constructor creates a timer with a delay time of four seconds
    * (4000 milliseconds), and with a RepaintAction object as its
    * ActionListener.  It also starts the timer running.
    */
   public RandomArtPanel() {
      RepaintAction action = new RepaintAction();
      Timer timer = new Timer(4000, action);
      timer.start();
   }
```

This draws a random picture every 4 seconds.



This is the listener for the timer which just calls `repaint`.

Note the creation and starting of the timer.

Keyboard Events

A ***keyboard event*** occurs when a keyboard key is pressed.

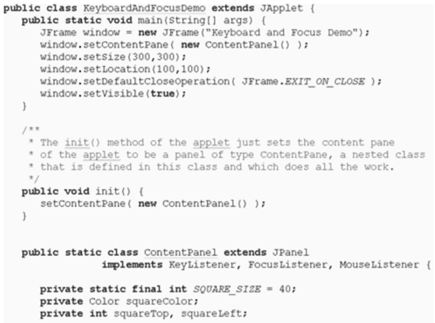
Key events allow a program to respond immediately as the user presses keys.

A listener responds when any key is pressed, then decides what to do based on the specific key pressed.

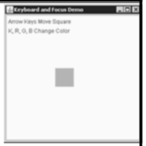
Keyboard events:

```
public void keyPressed(KeyEvent evt);
public void keyReleased(KeyEvent evt);
public void keyTyped(KeyEvent evt);
```

Keyboard Example Code



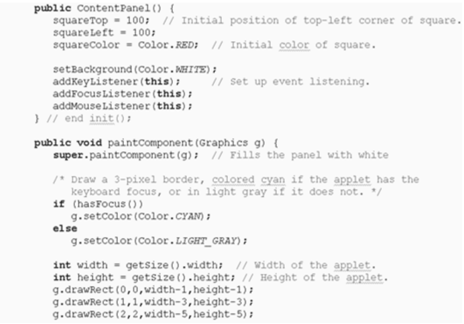
```
public class KeyboardAndFocusDemo extends JApplet {
   public static void main(String[] args) {
      JFrame window = new JFrame("Keyboard and Focus Demo");
      window.setContentPane( new ContentPanel() );
      window.setSize(300,300);
      window.setLocation(100,100);
      window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      window.setVisible(true);
   }

   /**
    * The init() method of the applet just sets the content pane
    * of the applet to be a panel of type ContentPane, a nested class
    * that is defined in this class and which does all the work.
    */
   public void init() {
      setContentPane( new ContentPanel() );
   }


   public static class ContentPanel extends JPanel
               implements KeyListener, FocusListener, MouseListener {

      private static final int SQUARE_SIZE = 40;
      private Color squareColor;
      private int squareTop, squareLeft;
```



This allows the user to move a square and change its color by pressing keys.

Note that the panel is setup to listen for keyboard, mouse, and focus events.

Keyboard Example Code (2)

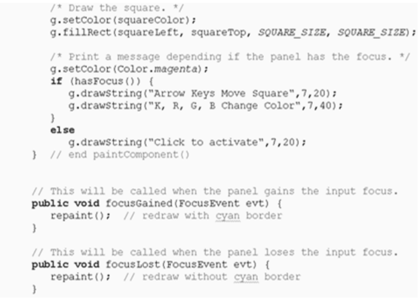


```
      public ContentPanel() {
         squareTop = 100;  // Initial position of top-left corner of square.
         squareLeft = 100;
         squareColor = Color.RED;  // Initial color of square.

         setBackground(Color.WHITE);
         addKeyListener(this);     // Set up event listening.
         addFocusListener(this);
         addMouseListener(this);
      } // end init();

      public void paintComponent(Graphics g) {
         super.paintComponent(g);  // Fills the panel with white

         /* Draw a 3-pixel border, colored cyan if the applet has the
            keyboard focus, or in light gray if it does not. */
         if (hasFocus())
            g.setColor(Color.CYAN);
         else
            g.setColor(Color.LIGHT_GRAY);

         int width = getSize().width;  // Width of the applet.
         int height = getSize().height; // Height of the applet.
         g.drawRect(0,0,width-1,height-1);
         g.drawRect(1,1,width-3,height-3);
         g.drawRect(2,2,width-5,height-5);
```

The constructor for the panel adds listeners for the events.

The paintComponent method draws the panel. It also draws the rectangle.

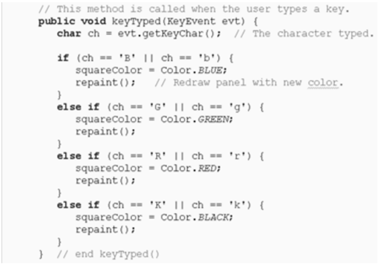
Keyboard Example Code (3)



```
         /* Draw the square. */
         g.setColor(squareColor);
         g.fillRect(squareLeft, squareTop, SQUARE_SIZE, SQUARE_SIZE);

         /* Print a message depending if the panel has the focus. */
         g.setColor(Color.magenta);
         if (hasFocus()) {
            g.drawString("Arrow Keys Move Square",7,20);
            g.drawString("K, R, G, B Change Color",7,40);
         }
         else
            g.drawString("Click to activate",7,20);
      }  // end paintComponent()


      // This will be called when the panel gains the input focus.
      public void focusGained(FocusEvent evt) {
         repaint();  // redraw with cyan border
      }

      // This will be called when the panel loses the input focus.
      public void focusLost(FocusEvent evt) {
         repaint();  // redraw without cyan border
      }
```

If the panel gains or loses focus, repaint is called to update the graphics on the panel.

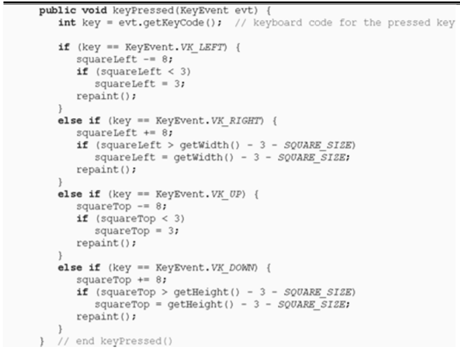
Keyboard Example Code (4)



```
// This method is called when the user types a key.
public void keyTyped(KeyEvent evt) {
   char ch = evt.getKeyChar();  // The character typed.

   if (ch == 'B' || ch == 'b') {
      squareColor = Color.BLUE;
      repaint();   // Redraw panel with new color.
   }
   else if (ch == 'G' || ch == 'g') {
      squareColor = Color.GREEN;
      repaint();
   }
   else if (ch == 'R' || ch == 'r') {
      squareColor = Color.RED;
      repaint();
   }
   else if (ch == 'K' || ch == 'k') {
      squareColor = Color.BLACK;
      repaint();
   }
}  // end keyTyped()
```

The keyTyped method detects when a user types a key and changes the color of the square accordingly.

Keyboard Example Code (5)



```
public void keyPressed(KeyEvent evt) {
   int key = evt.getKeyCode();  // keyboard code for the pressed key

   if (key == KeyEvent.VK_LEFT) {
      squareLeft -= 8;
      if (squareLeft < 3)
         squareLeft = 3;
      repaint();
   }
   else if (key == KeyEvent.VK_RIGHT) {
      squareLeft += 8;
      if (squareLeft > getWidth() - 3 - SQUARE_SIZE)
         squareLeft = getWidth() - 3 - SQUARE_SIZE;
      repaint();
   }
   else if (key == KeyEvent.VK_UP) {
      squareTop -= 8;
      if (squareTop < 3)
         squareTop = 3;
      repaint();
   }
   else if (key == KeyEvent.VK_DOWN) {
      squareTop += 8;
      if (squareTop > getHeight() - 3 - SQUARE_SIZE)
         squareTop = getHeight() - 3 - SQUARE_SIZE;
      repaint();
   }
}  // end keyPressed()
```

The keyPressed method detects when a key is pressed and moves the square.

Graphical User Interfaces Conclusion

Buttons, text fields, check boxes, combo boxes, and menus are all components in the Java Swing package that can be used to developed a GUI for your application.

Components generate events (usually action events) to indicate when they have been clicked on or accessed by the user.

- We handle the events using listeners and adapters.

The important thing about Swing is not memorizing the components and their methods, but understanding how the components work and generate events.

- Focus on event handling and the concept of using components, not on the definition of the components!

Objectives

Definitions: event, event handler/listener, event source

Java skills:

- Create applets and place on web pages.
- Use the Java coordinate system.
- Draw basic shapes, change colors and fonts.
- Window applications using `JFrame` and `JPanel`.
- Java Swing components: `JButton`, `JCheckBox`, `JComboBox`, `JLabel`, `JPanel`, `JRadioButton`, `JTextField`, `JTextArea`
- Event listeners versus event adapters
- Mouse events: `MouseListener`, `MouseAdapter`
- Window events: `WindowListener`, `WindowAdapter`
- `ActionListener` and use with `JButton`

Objectives (2)

Java skills (cont.):

- Using inner classes.
- Menus: `JMenu`, `JMenuItem`, `JMenuBar`
- `Timer` and timer events
- Keyboard events: `KeyListener`

COSC 123
Computer Creativity

I/O Streams and Exceptions

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Objectives

Explain the purpose of exceptions.

Examine the try-catch-finally statement for handling exceptions.

Show how to throw exceptions to other methods.

Identify I/O streams with specific focus on reading and writing text files and handling I/O exceptions.

Exception Handling

An ***exception*** is an error situation that must be handled or the program will fail.

- ***Exception handling*** is a mechanism for communicating error conditions between methods of your program.

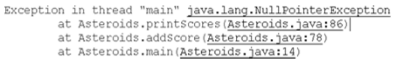
Examples:

- Attempting to divide by zero
- An array index that is out of bounds
- A specified file that could not be found
- A requested I/O operation that could not be completed normally
- Attempting to follow a null reference
- Attempting to execute an operation that violates some kind of security measure

Uncaught Exceptions

If a program does not handle the exception, it will terminate abnormally and produce the message that describes the exception that occurred and where in the code it was produced.

Example:



```
Exception in thread "main" java.lang.NullPointerException
        at Asteroids.printScores(Asteroids.java:86)
        at Asteroids.addScore(Asteroids.java:78)
        at Asteroids.main(Asteroids.java:14)
```

- The output is the call stack trace that indicates where the exception occurred.

The try-catch Statement

The ***try-catch statement*** identifies a block of statements that may throw an exception.

A ***catch clause*** defines how a particular kind of exception is handled. Each catch clause is called an ***exception handler***.

When the try-catch statement is executed, the statements in the `try` block are executed.

If an exception is thrown at any point during the execution of the `try` block, control is immediately transferred to the appropriate `catch` handler.

Catching Exceptions Example Code

```
try
{
   Scanner sc = new Scanner(System.in);
   System.out.print("Enter your age? ");
   int age = sc.nextInt();
   System.out.println("You are: "+age+" years old!!!");
}
catch (InputMismatchException e)
{  System.out.println("Input was not a number.");
}
```

The finally Clause

A `try-catch` statement can have an optional `finally` clause which defines a section of code that is executed *no matter* how the try block is exited.

If no exception is generated, the statements in the `finally` clause are executed after the `try` block is complete.

If an exception is generated in the `try` block, control first transfers to the appropriate `catch` clause, then to `finally` clause.

Finally Example

```
try
{
   Scanner sc = new Scanner(System.in);
   System.out.println("Enter your age?");
   int age = sc.nextInt();
   System.out.println("You are: "+age+" years old!!!");
}
catch (InputMismatchException e)
{  System.out.println("Input was not a number.");
}
finally
{  System.out.println("We always go in here!");
}
```

Throwing Exceptions

Your method has two ways of handling exceptions:

- ◆1) It can handle them inside the method using a `try-catch-finally` block.
- ◆2) It can throw the exception to the method that called it and force that method to handle it.

To throw an exception you must do two things:

- ◆1) List the type of exception that is thrown in the method header.
- ◆2) Not catch an exception (do not use try-catch block) or create a new exception and call `throw` to pass it to the caller.

When an exception is thrown, the **method exits immediately** similar to a return statement.

Throwing Exceptions Example Code

```
public class ThrowException
{
   public static void main(String[] args)
   {
      System.out.println("This isn't smart...");
      doSomethingDumb();
   }

   public static int doSomethingDumb()
                        throws ArithmeticException
   {
      int num1 = 5, num2 = 0;
      int result = num1/num2; // Divide by zero
      return result;
   }
}
```

Checked and Unchecked Exceptions

Checked exceptions are exceptions that you must tell the compiler how your code is handling them.

- ◆A checked exception *must* be either caught or thrown.
  - ⇨Checked exceptions are typically exceptions that are not your fault.
  - ⇨e.g. `IOException` (and all its subclasses)

Unchecked exceptions are exceptions that the compiler does not force your program to handle.

- ◆An unchecked exception is automatically passed to the caller method if it is not handled by the method that generated the exception.
  - ⇨Unchecked exceptions include `NumberFormatException`, `IllegalArgumentException`, and `NullPointException`.
  - ⇨Exceptions that are a subclass of `RuntimeException` are unchecked.

Exceptions

Question: TRUE or FALSE: A good programmer can always avoid exceptions.

A) TRUE

B) FALSE

Exceptions

Question: TRUE or FALSE: An uncaught exception may be passed through several methods before the program crashes.

A) TRUE

B) FALSE

Exceptions

Question: What does this code output if the user enters "32"?

```
try
{  Scanner sc = new Scanner(System.in);
   System.out.print("Enter a number: ");
   int num = sc.nextInt();
   System.out.print(num+" ");
}
catch (InputMismatchException e)
{  System.out.print("Input was not a number. ");
}
finally
{  System.out.print("HELLO!");
}
```

A) nothing

B) `32`

C) `Input was not a number.`

D) `32 HELLO!`

Exceptions

Question: What does this code output if the user enters "abc"?

```
try
{  Scanner sc = new Scanner(System.in);
   System.out.print("Enter a number: ");
   int num = sc.nextInt();
   System.out.print(num+" ");
}
catch (InputMismatchException e)
{  System.out.print("Input was not a number. ");
}
finally
{  System.out.print("HELLO!");
}
```

A) `abc`

B) `Input was not a number.`

C) `abc HELLO!`

D) `Input was not a number. HELLO!`

Java File Input/Output

A ***stream*** is an ordered sequence of bytes.

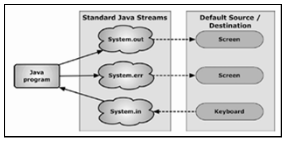
A stream may be either an input stream or an output stream.

An ***input stream*** is a stream from which information is read.

An ***output stream*** is a stream to which information is written.

Streams may generate exceptions such as `IOException`.

The `System` class contains three object reference variables:



Reading and Writing Text Files

A file is opened as a stream for reading or writing.

Programmers need to know the contents of the file and how to translate it to a usable form.

If for some reason there is a problem finding or opening a file, the attempt to create a `File` object will throw an `IOException`.

- To put a backslash ("\") in a filename string, you must enter each backslash TWICE as backslash is an escape character.
  - e.g. `File in = new File("c:\\homework\\input.dat");`

Output file streams should be explicitly closed or they may not correctly retain the data written to them.

Read Text File with `Scanner`

```
Scanner sc = null;
try
{  sc = new Scanner(new File("MyFile.txt"));
   while (sc.hasNextLine())
   {  String st = sc.nextLine();
      System.out.println(st);
   }
}
catch (FileNotFoundException e)
{  System.out.println("Did not find input file: "+e);
}
finally
{  if (sc != null)
      sc.close();
}
```

Note: The Scanner class handles some exceptions for you.

Write Text File with `PrintWriter`

```
PrintWriter out = null;
try
{
   out = new PrintWriter("output.txt");
   // Write the numbers 1 to 10 in the file
   for (int i=1; i <=10; i++)
      out.println(i);
}
catch (FileNotFoundException e)
{  System.out.println("Could not create output file: "+e);
}
finally
{  if (out != null)
      out.close();
}
```

Streams and Exceptions Practice Question

1) Write a program that prompts the user for a filename then opens the text file and counts the number of lines in the file.

Conclusions

An ***exception*** is an error situation that must be handled or the program will fail.

- ***Exception handling*** is a mechanism for communicating error conditions between methods of your program.

There are two ways for handling exceptions:

- 1) Instead method using a `try-catch-finally` block.
- 2) By throwing it to the caller method.
- Checked exceptions must always be handled.

A stream is a sequential sequence of bytes which can be used for input or output. Files are streams as is `System.out`.

Reading from text files can be done using `Scanner` class similar to reading from `System.in`.

Writing to text files is done using the `PrintWriter` class.

- Make sure to close all files!

Objectives

Key terms:

- exceptions and exception handling

Java skills:

- exception handling using try-catch-finally statement
- uncaught exceptions and the call stack trace
- throwing exceptions (`throws` in method header)
- checked vs. unchecked exception
- streams and the standard I/O streams in the System class
- Reading from a text file using `Scanner`
- Writing to a text file using `PrintWriter`

COSC 123
Computer Creativity

Course Review

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Reading Data from the User
The Scanner Class

The `Scanner` class reads data entered by the user. Methods:

- ◆`int nextInt()` - reads next integer
- ◆`double nextDouble()` – reads next floating point number
- ◆`String next()` – reads String (up to separator)
- ◆`String nextLine()` – reads entire line as a String

To use must import `java.util.Scanner`.

```
import java.util.Scanner;
public class AddTwoNum
{  public static void main(String[] argv)
   {  // Code reads and adds two numbers
      Scanner sc = new Scanner(System.in);
      int num1 = sc.nextInt();
      int num2 = sc.nextInt();
      int result = num1+num2;
      System.out.println(num1+" + "+num2+" = "+result);
   }
}
```

Values, Variables, and Locations

A ***value*** is a data item that is manipulated by the computer.

A ***variable*** is the name that the programmer users to refer to a location in memory.

A ***location*** has an address in memory and stores a value.

IMPORTANT: The *value* at a given location in memory (named using a variable name) can change using initialization or assignment.

Compile vs. Run-time Errors

Question: A program is supposed to print the numbers from 1 to 10. It actually prints the numbers from 0 to 9. What type of error is it?

A) Compile-time error

B) Run-time error

Variables - Definitions

Question: Which of the following statements is correct?

A) The location of a variable may change during the program.

B) The name of a variable may change during the program.

C) The value of a variable may change during the program.

Assignment

Question: What are the values of `A` and `B` after this code?

```
int A, B;

A = 6;
B = 3;
A = 3 * B + A / B;
B = A + 5 * 3 * B;
```

A) `A = 6, B = 3`

B) `A = 11, B = 56`

C) `A = 5, B = 90`

Code Output

Question: What is the output of this code if user enters 3 and 4?

```
public class AddTwoNum
{  public static void main(String[] argv)
   {  // Code reads and adds two numbers
      Scanner sc = new Scanner(System.in);
      int num1 = sc.nextInt();
      int num2 = sc.nextInt();
      int result = num1+num2;
      System.out.println(num2+" - "+num1+" = "+result);
   }
}
```

A) 3 + 4 = 7

B) 4 + 3 = 7

C) 3 – 4 = 7

D) 4 – 3 = 7

Practice Question

1) Create a program to ask the user for two numbers, a operation to perform (+,-,/,*), and then do that operation.

Making Decisions

Decisions are used to allow the program to perform different actions in certain conditions.

To make a decision in a program we must do several things:

- 1) Determine the ***condition*** in which to make the decision.
- 2) Tell the computer what to do if the condition is true or false.
  - ⇨ A decision always has a *Boolean value* or true/false answer.

The syntax for a decision uses the ***if*** statement:

```
if (age > 19)          OR       if (age > 19)
   teenager=false;                 teenager=false;
                                else
                                   teenager=true;
```

Making Decisions Block Syntax

Use the ***block syntax*** for denoting a multiple statement block. A block is started with a “**{**“ and ended with a “**}**”.

- All statements inside the brackets are grouped together.

Example:

```
if (age > 19)
{  teenager=false;
   hasLicense=true;
   ...
}
```

The ***dangling else problem*** occurs when a programmer mistakes an else clause to belong to a different if statement than it really does.

- Remember, blocks (brackets) determine which statements are grouped together, not indentation!

Nested Conditions and Decisions Boolean Expressions

A ***Boolean expression*** is a sequence of conditions combined using AND (**&&**), OR (**||**), and NOT (**!**).

- Allows you to test more complex conditions
- Group subexpressions using parentheses

Syntax:	*(expr1)* **&&** *(expr2)*	- expr1 AND expr2
	(expr1) **||** *(expr2)*	- expr1 OR expr2
	!(expr1)	- NOT expr1

Examples:

```
var b;
1) b = (x > 10) && !(x < 50);
2) b = (month == 1) || (month == 2) || (month == 3);
3) if (day == 28 && month == 2)
4) if !(num1 == 1 && num2 == 3)
5) b = ((10 > 5 || 5 > 10) && ((10>5 && 5>10));// False
```

Making Decisions Switch Statement

There may be cases where you want to compare a single integer value against many constant alternatives. Instead of using many if statements, you can use a ***switch*** statement.

- If there is no matching case, the default code is executed.
- Execution continues until the **break** statement. (Remember it!)
- Note: You can only use a switch statement if your cases are **integer numbers**. (Characters (‘a’, ‘b’,...,) are also numbers.)

Syntax:

```
switch (integer number)
{  case num1: statement break;
   case num2: statement break;
   ...
   default:   statement break;
}
```

Making Decisions (3)

Question: What is the output of this code?

```
int num=10;

if (num == 10)
{   System.out.print("big");
    System.out.println("small");
}
```

A) `big`

B) `small`

C) `bigsmall`

Making Decisions (4)

Question: What is the output of this code?

```
int num=10;

if (num >= 8)
   System.out.print("big");
   if (num == 10)
       System.out.print("ten");
else
     System.out.print("small");
```

A) `big`

B) `small`

C) `bigsmall`

D) `ten`

E) `bigten`

Switch Statement (2)

Question: What is the output of this code?

```
int num=2;

switch (num)
{  case 1: System.out.print("one");
   case 2: System.out.print("two");
   case 3: System.out.print("three"); break;
   default: System.out.print("other");
}
```

A) `onetwo`

B) `two`

C) `twothree`

D) `other`

E) `onetwothreeother`

Decision Practice Question

1) Write a program that reads a number *N*.

- ◆If *N* has 1 digit, print 1 digit.
- ◆If *N* has 2 digits, print 2 digits.
- ◆Otherwise if *N* has 3 or more digits, print 3 or more digits.

The For Loop

The most common type of loop is the ***for loop***. Syntax:

```
for (<initialization>; <continuation>; <next iteration>)
{  <statement list>
}
```

Explanation:

- ◆1) initialization section - is executed once at the start of the loop
- ◆2) continuation section - is evaluated *before* every loop iteration to check for loop termination
- ◆3) next iteration section - is evaluated *after* every loop iteration to update the loop counter

Example:

```
int i;

for (i = 0; i < 5; i++)
{  System.out.println(i);      // Prints 0 to 4
}
```

For Loops

Question: What is the output of this code?

```
for (i=0; i < 6; i++)
   System.out.print(i);
```

A) nothing

B) error

C) The numbers 0, 1, 2, …, 5

D) The numbers 0, 1, 2, …, 6

For Loops

Question: What is the output of this code?

```
for (i=2; i < 20; i--)
   System.out.print(i);
```

A) nothing

B) infinite loop

C) The numbers 2, 3, 4, …, 19

D) The numbers 2, 3, 4, …, 20

Loops Practice Question

1) Write a program that reads a number *N* and calculate the sum of the numbers from 1 to *N*.

If that is too easy for you, calculate only the sum of the even numbers from 1 to *N*!

Java Object-Oriented Terminology

An ***object*** is an instance of a class that has its own properties and methods. Properties and methods define what the object is and what it can do. *Each object has its own area in memory.*

A ***class*** is a generic template (blueprint) for creating an object. All objects of a class have the same methods and properties (although the property values can be different).

A ***property*** (or ***instance variable***) is an attribute of an object.

A ***method*** is a set of statements that performs an action. *A method works on an implicit object and may have parameters.*

A ***parameter*** is data passed into a method for it to use.

Access Specifiers Public and Private

One of the features of object-oriented programming is that not all parts of a program have access to all the data and methods.

Each class, method, and variable needs to have one of the four ***access specifiers*** defined that indicate which other objects and methods in your program have access to it. Four types:

- `public` – Accessible by all code (everyone, the public)
- `private` – Only accessible by methods in the class.
- `protected` – Only accessible by methods in the class or classes derived from this class by inheritance.
- default – If nothing is specified, assume package access where all methods in same package can access it.

Class Example `BankAccount` Class

```
public class BankAccount
{
   private double balance;

   public void deposit(double amount)
   {  balance = balance + amount; }

   public void withdraw(double amount)
   {  balance = balance - amount; }

   public double getBalance()
   {  return balance; }
}
```

The `BankAccount` class is used for describing bank accounts.

- The methods defined in the `BankAccount` class are `deposit`, `withdraw`, and `getBalance`.
- The current balance in the account is `private`, so it can only be changed by calling the methods.

Class Practice Questions

1) Write the `setBalance` method for the `BankAccount` class.

2) Add an instance variable called name for the name of the owner of the `BankAccount`. Add get/set methods for this instance variable.

3) Add a default constructor (no parameters) and an overloaded constructor that accepts balance and name as parameters.

Creating and Using Objects

A class is just a blue-print for creating objects.

- By itself, a class performs no work or stores no data.

For a class to be useful, we must create objects of the class.

- Each object created is called an ***object instance***.

To create an object, we use the `new` method.

When an object is created using the `new` method:

- Java allocates space for the object in memory.
- The ***constructor*** for the object is called to initialize its contents.
- Java returns a pointer to where the object is stored in memory which we will call an ***object reference***.

Objects and Object References

Question: How many object references are in this code?

```
BankAccount savings, checking;
BankAccount myAcct, myAcct2;

savings = new BankAccount();
myAcct = savings;
checking = new BankAccount();
```

A) 1

B) 2

C) 3

D) 4

Objects and Object References

Question: How much money is in the account referenced by the `myAcct2` object reference?

```
BankAccount savings, checking;
BankAccount myAcct, myAcct2;

savings = new BankAccount(50);
myAcct = savings;
savings = null;
checking = new BankAccount(100);
savings = checking;
myAcct2 = myAcct;
```

A) unknown

B) 50

C) 100

D) undefined

Variable Scope
Scope of Variable Types

The scope of variables depends directly on their type:

- 1) ***Instance variables*** - are created when an object instance is created using the `new` method. Instance variables are defined as long as there is at least one reference to the object in your program which is still in scope.
- 2) ***Static variables*** - are created when the class they are defined in is first loaded and are defined until the class is unloaded.
  - This means static variables are around for the duration of your program.
- 3) ***Local variables*** - are created when the program enters the block in which they are defined and destroyed when the program exits that block.
  - A variable defined in brackets ("{","}") is accessible anywhere within the block including nested blocks.
- 4) ***Parameter variables*** - are created when a method is first called and are destroyed when a method returns.

Variable Scope
Practice Questions

With this code explain the lifetime and scope of all variables.

```
public class VariableScope
{  public static void main(String[] args)
   {  double amount = 25;
      BankAccount acct = new BankAccount(200);
      for (int i=1; i <= 3; i++)
         acct.deposit(amount);
      System.out.println(acct.getBalance());  // 125.0
   }

   private void doNothing(double a)
   {  int i = 5; return;   }

   public static final int MYNUM = 25;
}
```

Variable Scope
Practice Questions (2)

```
class BankAccount
{  public void deposit(double amount)
   {  if (amount <= balance)
      {  double newBalance = balance - amount;
         balance = newBalance;
      }
      double balance = 50;
   }
   public double getBalance()
   {  return balance; }

   public BankAccount(double b)
   {  balance = b; lastAccountNum++;
      accountNum = lastAccountNum;
   }

   private double balance;
   private int accountNum;
   private static int lastAccountNum = 0;
}
```

Inheritance Overview

Inheritance is a mechanism for enhancing and extending existing, working classes.

- ⇨ In real life, you inherit some of the properties from your parents when you are born. However, you also have unique properties specific to you.
- ⇨ In Java, a class that extends another class inherits some of its properties (methods, instance variables) and can also define properties of its own.

`Extends` is the key word used to indicate when one class is related to another by inheritance.

Syntax: `class` *`subclass`* **`extends`** *`superclass`*

- ◆The ***superclass*** is the existing, parent class.
- ◆The ***subclass*** is the new class which contains the functionality of the superclass plus new variables and methods.

Inheritance Question

1) Create a `CheckingAccount` class which inherits from `BankAccount`. The `CheckingAccount` class:

- ◆inherits `getBalance()` from `BankAccount`
- ◆overrides `deposit()` and `withdraw()` from `BankAccount`, so it can keep track of the number of transactions (`transactionCount`)
- ◆defines a method `deductFees()` which withdraws $1 for each transaction (`transactionCount`) then resets the # of transactions

Arrays

An ***array*** is a collection of data items of the same type.

An array reference is denoted using the open and close square brackets "`[]`" during declaration.

- ⇨ You can have an array of any data type including the base types (`int`, `double`, `String`) and object-types (`BankAccount`).

◆Examples:

```
int[] myArray;
String[] strings;
BankAccount[] accounts;
```

Similar to an object, when you declare an array you are creating a ***reference*** to an array. Until you actually create the space for the array using **`new`**, no array exists in memory.

- ◆`String[] strings = new String[10];`

Arrays

Question: What is the size of this array?

```
int[] myArray = new int[10];
```

A) error

B) 10

C) 9

D) 11

Arrays

Question: What are the contents of this array?

```
int[] myArray = new int[4];
myArray[0] = 1;
myArray[3] = 2;
myArray[2] = 3;
myArray[0] = 4;
```

A) error

B) 0, 1, 2, 3

C) 1, 2, 3, 4

D) 4, 0, 3, 2

ArrayLists

An `ArrayList` implements a *resizable* array of objects.

- ⇨ Base types such as int are not objects. Use wrapper class Integer.

Create an `ArrayList` by:

```
ArrayList names = new ArrayList(); // Size 10 (default)
ArrayList accounts = new ArrayList(5); // Size of 5
```

Add element to an `ArrayList` by:

```
names.add("Joe");       // Add to end of list
names.add(2,"Steve");   // Add at index 2 and shift down
```

Remove element from an `ArrayList` by:

```
names.remove(2);        // Remove index 2 and shift up
```

ArrayLists (2)

Get number of items in list by:

```
int count = names.size();
```

Get element at an index from an `ArrayList` by:

```
String n = names.get(2);   // Get item at index 2
```

Set element at an index in an `ArrayList` by:

```
names.set(2,"Fred");       // Put Fred at index 2
```

A simple way to traverse an `ArrayList` is using a for loop:

```
for (int i=0; i < names.size(); i++)
{  String s = (String) names.get(i);
   System.out.println(s);
}
```

ArrayList

Question: What is the value of `st`?

```
ArrayList a = new ArrayList();
a.add("Fred");
a.add(1,"Joe");
a.add(1,"Steve");
a.remove(0);
String st = (String) a.get(0);
```

A) Fred

B) Joe

C) Steve

D) error

ArrayList Practice Question

1) Write a method that takes an `ArrayList` as a parameter and returns an `ArrayList` with all the items in reverse order.

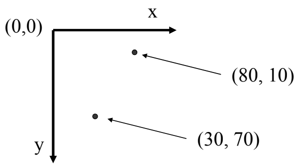
For example, if the list was `{Joe, Fred, Smith}` then after reverse the list is `{Smith, Fred, Joe}`.

The Coordinate System

Drawing on the screen is done by specifying coordinates which refer to a location on the screen.

- The ***origin*** is the upper-left hand corner of the screen.
- The x coordinate gets bigger as we move to the right.
- The y coordinate gets bigger as we move down.

Diagram:



Drawing Methods

- 1) Ellipse:
  - ⇨ Ellipse2D.Double egg = new Ellipse2D.Double(topx, topy, width, height);
  - ⇨ Ellipse2D.Double egg = new Ellipse2D.Double(5, 10, 15, 20);
- 2) Rectangle:
  - ⇨ Rectangle box = new Rectangle(topx, topy, width, height);
  - ⇨ Rectangle box = new Rectangle(10, 10, 20, 30);
- 3) Line:
  - ⇨ Line2D.Double = new Line2D.Double(x1,y1, x2, y2);
- 4) Point:
  - ⇨ Point2D.Double = new Point2D.Double(x,y);

You can also fill a shape with a color using the `fill` method:

- ⇨ g2.fill(box);
- ⇨ g2.fill(egg);

Drawing Methods (2)

- Change colors:
  - ⇨ `g2.setColor(Color.orange);`
- Draw a string
  - ⇨ `g2.drawString("Hello", 50, 100); // message, x, y`

Java Swing Components

The Java Swing package contains the user interface components that we will use in our graphical applications.

Component

JButton

ButtonGroup

Check box

Combo box

JFrame

JLabel

JPanel

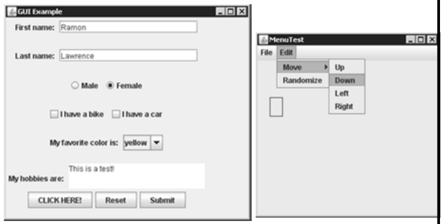
Radio button

Text field

JMenuBar

JMenu

JMenuItem



GUI Example

First name: Ramon

Last name: Lawrence

○ Male ● Female

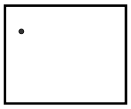
□ I have a bike □ I have a car

My favorite color is: yellow

My hobbies are: This is a test!

Coordinates

Question: Select from the coordinates below the pair that best describes this point's location. Assume box is 100 by 100.



A) (10,80)

B) (80,10)

C) (10,20)

D) (20,10)

Events and Event Handling Overview

An ***event*** is a notification to your program that something has occurred.

- For graphical events (mouse click, data entry), the Java window manager notifies your program that an event occurred.
  - There are different ***kinds*** of events such as keyboard events, mouse click events, mouse movement events, etc.

An ***event handler*** or ***listener*** is part of your program that is responsible for "listening" for event notifications and handling them properly.

- An event listener often only listens for certain types of events.

An ***event source*** is the user interface component that generated the event.

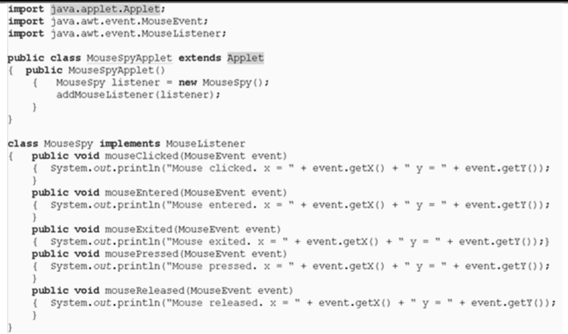
- A button, a window, and scrollbars are all event sources.

Mouse Event Example

Handling mouse click events requires three classes:

- 1) **The event class** - that stores information about the event.
  - For mouse clicks, this class is `MouseEvent`.
  - The `MouseEvent` class has methods `getX()` and `getY()` that indicate the position of the mouse at the time the event was generated.
- 2) **The listener class** - allows your program to detect events. Building your own listener class requires implementing a pre-defined interface.
  - For mouse clicks, the listener interface is `MouseListener`.
  - An ***event listener*** is a class that implements all methods of an event interface.
  - An ***event adapter*** extends a class and only requires you implement method for the events that you are interested in.
- 3) **The event source** - is the component in your GUI that generated the event.

Mouse Event Example Code



```
import java.applet.Applet;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class MouseSpyApplet extends Applet
{  public MouseSpyApplet()
    {   MouseSpy listener = new MouseSpy();
        addMouseListener(listener);
    }
}

class MouseSpy implements MouseListener
{   public void mouseClicked(MouseEvent event)
    {  System.out.println("Mouse clicked. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseEntered(MouseEvent event)
    {  System.out.println("Mouse entered. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseExited(MouseEvent event)
    {  System.out.println("Mouse exited. x = " + event.getX() + " y = " + event.getY());}
    public void mousePressed(MouseEvent event)
    {  System.out.println("Mouse pressed. x = " + event.getX() + " y = " + event.getY());
    }
    public void mouseReleased(MouseEvent event)
    {  System.out.println("Mouse released. x = " + event.getX() + " y = " + event.getY());
    }
}
```

Exception Handling

An ***exception*** is an error situation that must be handled or the program will fail. ***Exception handling*** is how your program deals with exceptions when they occur.

Two ways of handling exceptions:

- 1) *Handle* them inside the method using a `try-catch-finally` block.
- 2) *Throw* the exception to the method that called it and force that method to handle it.

Two types of exceptions:

- ***Checked exceptions*** are exceptions that you must tell the compiler how your code is handling them. (e.g. `IOException`)
  - A checked exception *must* be either caught or thrown.
- ***Unchecked exceptions*** are exceptions that the compiler does not force your program to handle.

The try-catch-finally Statement

The ***try-catch-finally statement*** identifies a block of statements that may throw an exception and provides code to handle exceptions if they occur.

Three components:

- ***try block*** - has statements to execute that may cause exceptions. Each statement is executed one at a time. If an exception occurs, jump out of try block to a catch clause. If no exception, go to finally clause (if it exists).
- ***catch block*** – handles a particular kind of exception and has code that performs the desired action if it occurs. Only one catch clause is every executed and are not executed if an exception does not occur.
- ***finally block*** – code that is always executed regardless if all statements completed successfully or an exception occurred

Exceptions

Question: TRUE or FALSE: An uncaught exception may be passed through several methods before the program crashes.

A) TRUE

B) FALSE

Exceptions

Question: What does this code output if the user enters "32"?

```
try
{  Scanner sc = new Scanner(System.in);
   System.out.print("Enter a number: ");
   int num = sc.nextInt();
   System.out.print(num+" ");
}
catch (InputMismatchException e)
{  System.out.print("Input was not a number. ");
}
finally
{  System.out.print("HELLO!");
}
```

A) nothing

B) `32`

C) `Input was not a number.`

D) `32 HELLO!`

Read Text File with `Scanner`

```
Scanner sc = null;
try
{  sc = new Scanner(new File("MyFile.txt"));
   while (sc.hasNextLine())
   {  String st = sc.nextLine();
      System.out.println(st);
   }
}
catch (FileNotFoundException e)
{  System.out.println("Did not find input file: "+e); }
finally
{  if (sc != null)
      sc.close();
}
```

`Note: The Scanner class handles some exceptions for you and makes it easier to read numbers and other types that are not Strings. It should be the one used.`

Streams and Exceptions Practice Question

1) Write a program that opens up the file "test.txt" that contains numbers and computes a sum where every odd number is added and every even number is subtracted.

Putting it all together...

Computer programming is the art and science of solving problems on the computer.

- As you have seen, there are many different ways to approach and solve the same problem.
  - ⇨ Each technique may have different benefits and performance.

Computer science is about learning how to make the correct and most efficient decisions on how to solve problems.

- ⇨ Anyone can program on a computer, but computer scientists know why they are programming a solution and what they are doing.

The most exciting aspect of programming is the satisfaction of building a program to solve a problem.

- Whether it is a simple algorithm or a program that runs a nuclear power plant.
  - ⇨ You do not quite have the skills or experience to solve the large problems, but you can solve smaller problems and appreciate the difficulty inherent in solving the larger ones.