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.

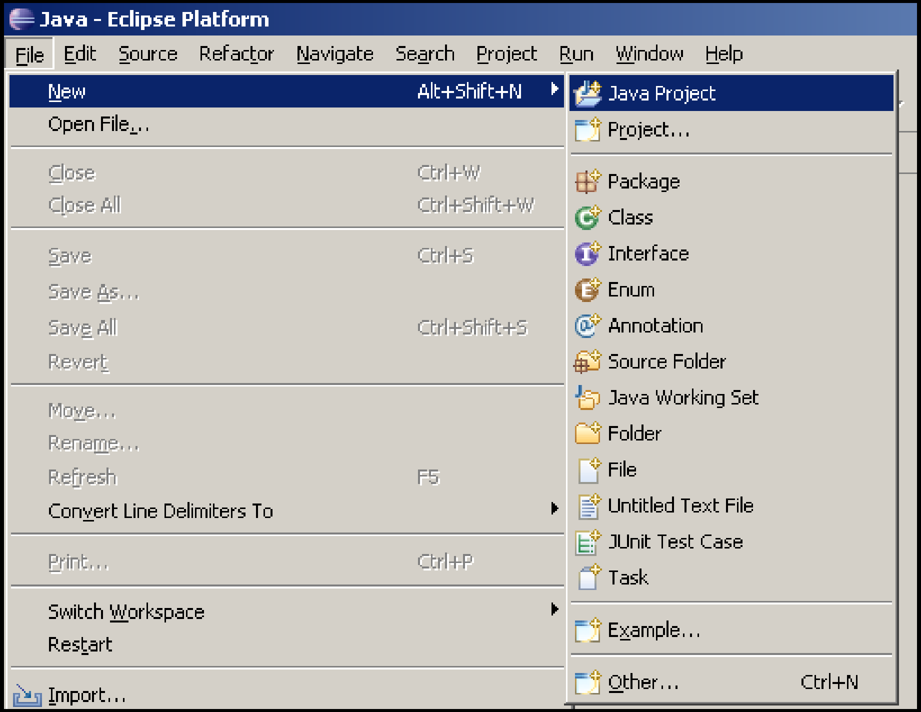
- You will be prompted for your workspace on start up if you have not selected one.

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

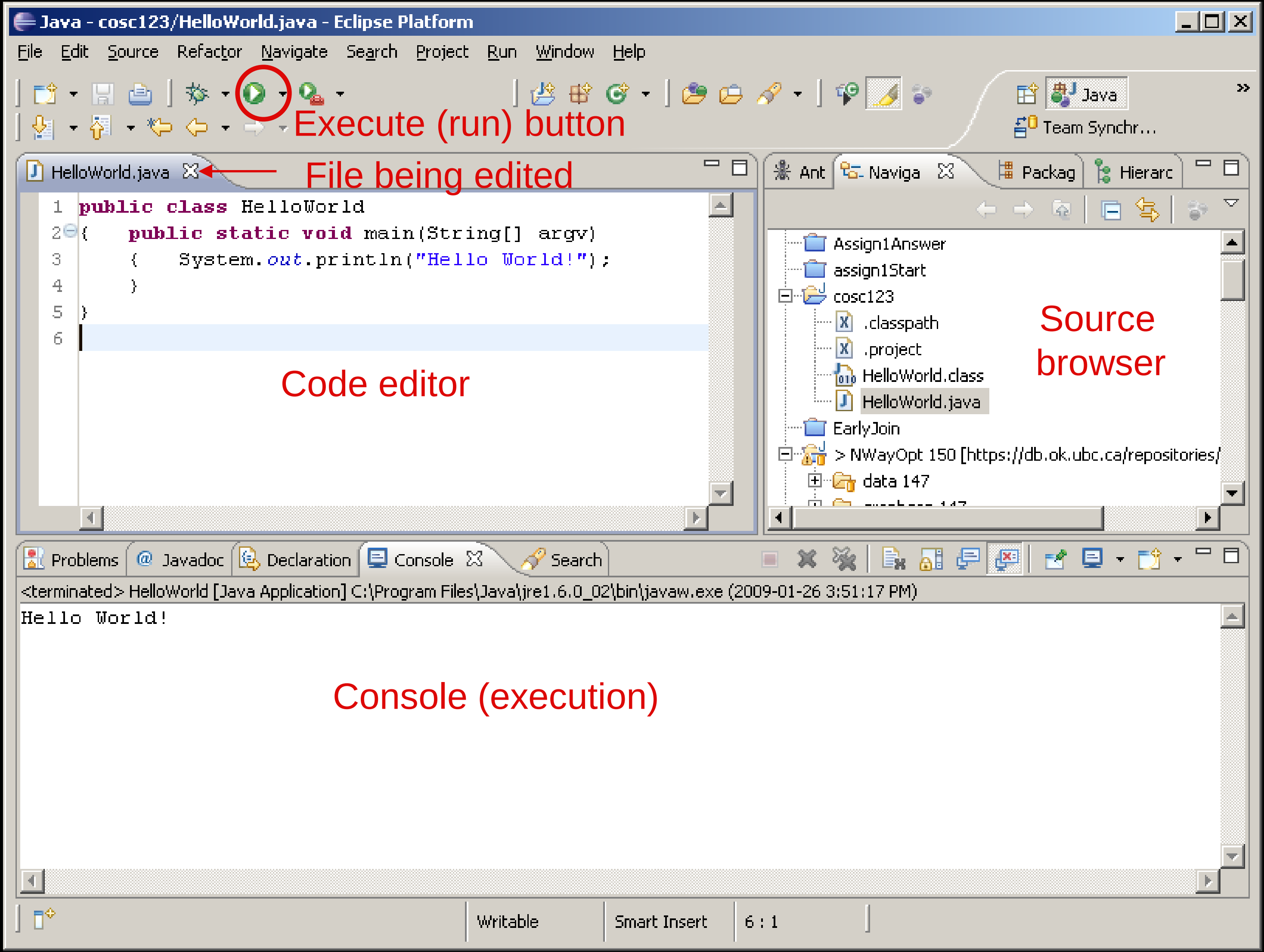
A ***project*** is a group of program files for some purpose. We will create a sample project called **`cosc123`**. You will also create projects for each assignment.

- 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


Java - cosc123/HelloWorld.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Execute (run) button
HelloWorld.java
File being edited
Code editor
Ant Naviga Packag Hierarc
Java
Team Synchr...
Source browser
Problems Javadoc Declaration Console Search
Console (execution)
Writable
Smart Insert
6 : 1

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

```
Assign1Answer
assign1Start
cosc123
    .classpath
    .project
    HelloWorld.class
    HelloWorld.java
EarlyJoin
> NWayOpt 150 [https://db.ok.ubc.ca/repositories/
    data 147
```

```
<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

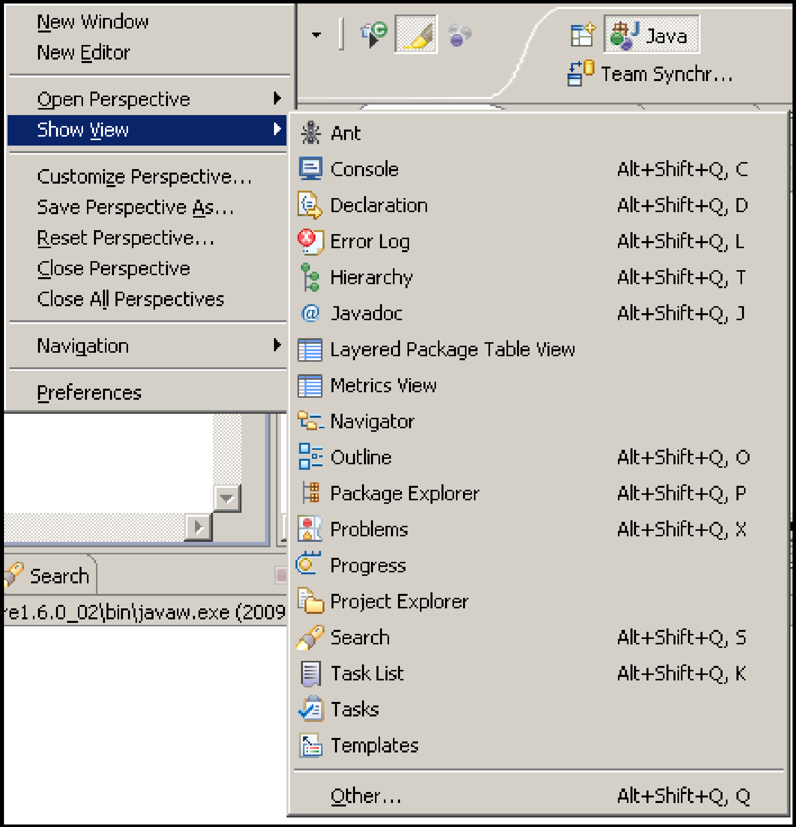
A ***perspective*** is an organization of views to accomplish a certain task (debugging, coding, etc.).

- 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

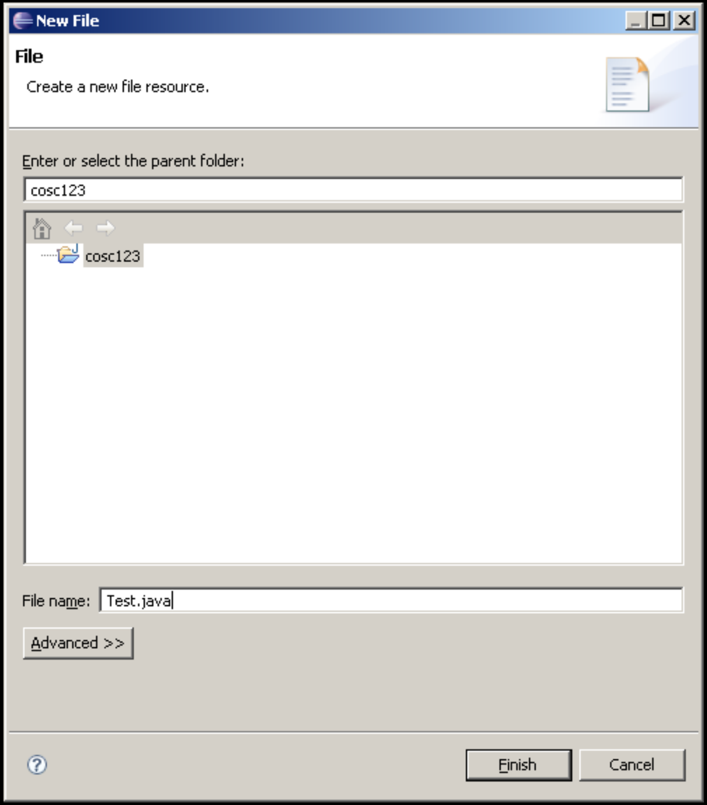
`File->New->Class` and provide a folder and file name.

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`


New File

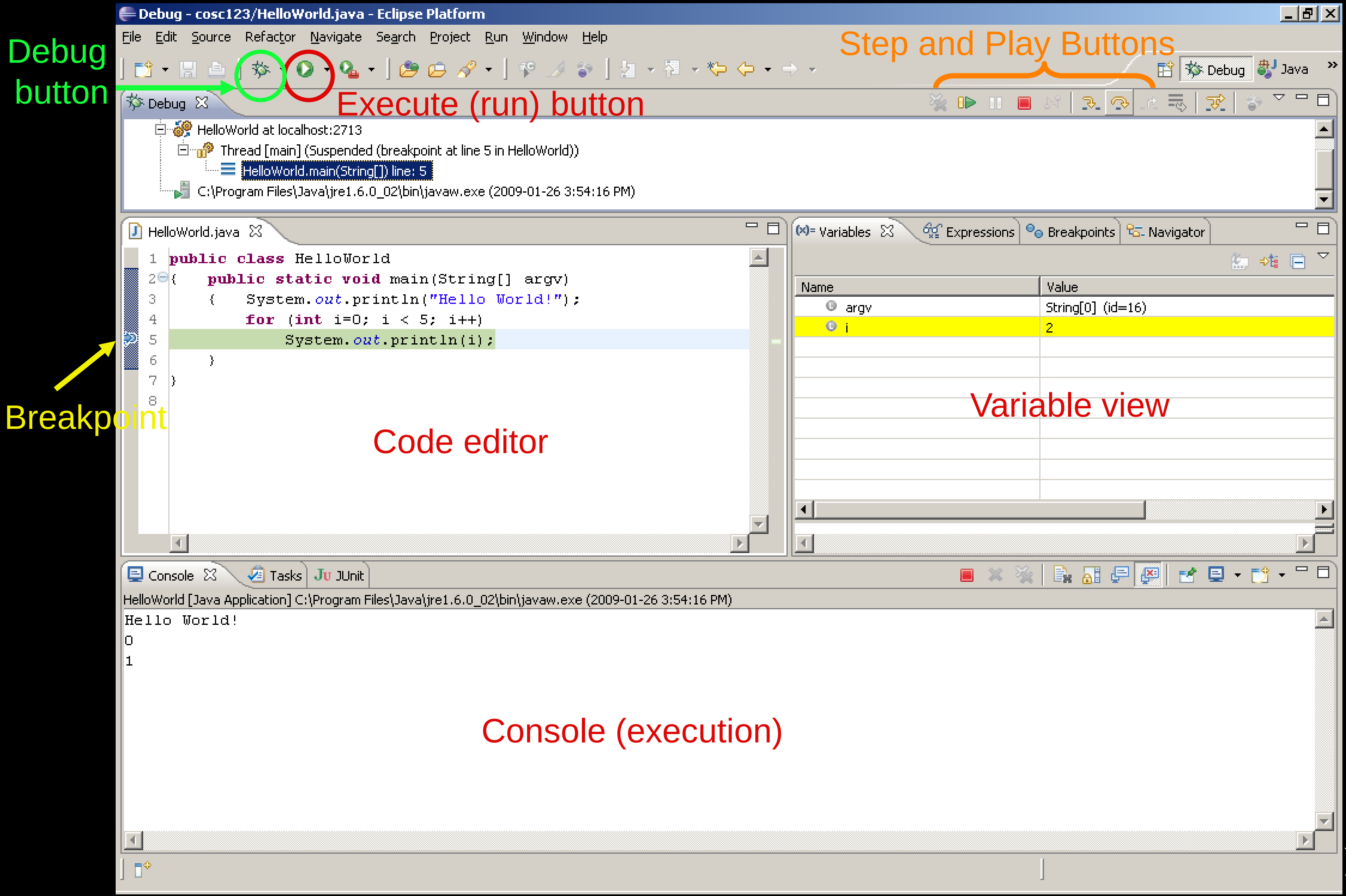
File

Create a new file resource.

Enter or select the parent folder: cosc123

File name: Test.java

Eclipse Debugging and Breakpoints



```
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)
```

```java
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

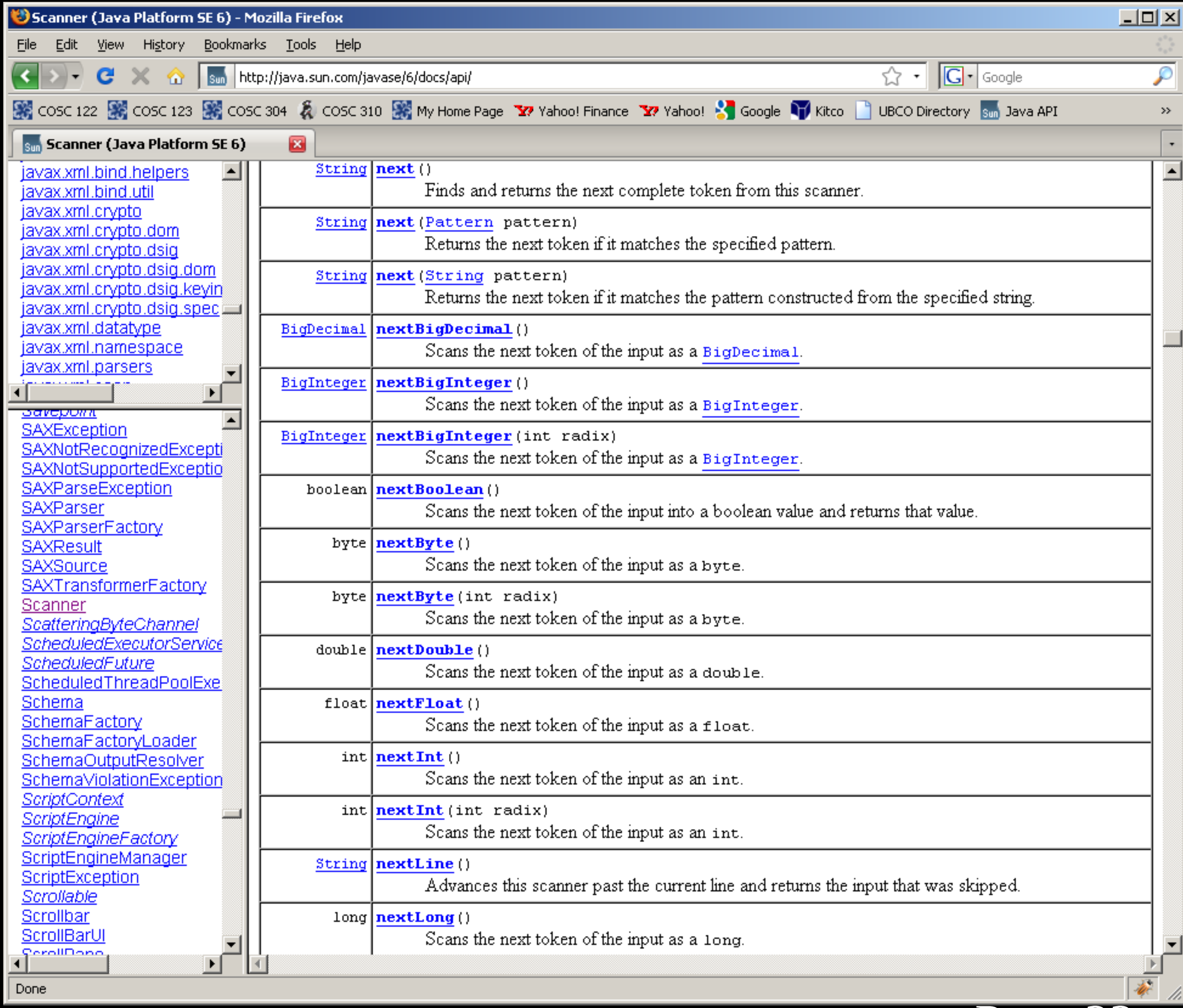
To use must import `java.util.Scanner`.

```
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/


Scanner (Java Platform SE 6) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://java.sun.com/javase/6/docs/api/
Google
COSC 122 COSC 123 COSC 304 COSC 310 My Home Page Yahoo! Finance Yahoo! Google Kitco UBCO Directory Java API
Scanner (Java Platform SE 6)
Done

javax.xml.bind.helpers
javax.xml.bind.util
javax.xml.crypto
javax.xml.crypto.dom
javax.xml.crypto.dsig
javax.xml.crypto.dsig.dom
javax.xml.crypto.dsig.keyin
javax.xml.crypto.dsig.spec
javax.xml.datatype
javax.xml.namespace
javax.xml.parsers

SAXException
SAXNotRecognizedExcepti
SAXNotSupportedExceptio
SAXParseException
SAXParser
SAXParserFactory
SAXResult
SAXSource
SAXTransformerFactory
Scanner
ScatteringByteChannel
ScheduledExecutorService
ScheduledFuture
ScheduledThreadPoolExe
Schema
SchemaFactory
SchemaFactoryLoader
SchemaOutputResolver
SchemaViolationException
ScriptContext
ScriptEngine
ScriptEngineFactory
ScriptEngineManager
ScriptException
Scrollable
Scrollbar
ScrollBarUI

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.

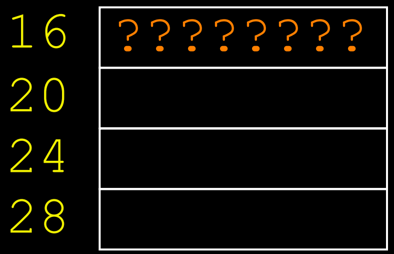
```
int total;
```

- 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



Values, Variables, and Locations Example (2)

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

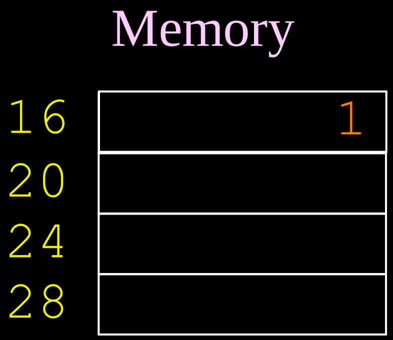
- If you do not initialize your variable to a starting value when you first declare it, the value of the variable is initialized to 0 (for numbers).

Example:

```
total = 1;
```

Variable Name Lookup Table

Name	Location	Type
total	16	number



Values, Variables, and Locations Example (3)

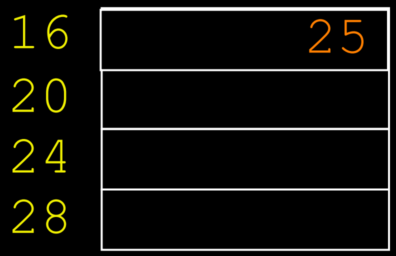
Step #3: Value stored in location can be changed throughout the program to whatever we want using ***assignment*** ("**=**" symbol).

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

Variable Name Lookup Table

Name	Location	Type
total	16	number

Memory



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