COSC 111:
Computer Programming I

Dr. Bowen Hui

University of British Columbia
Okanagan

First half of course

- Software examples
- From English to Java
- Template for building small programs
- Exposure to Java programs
- Fundamental concepts in a programming language

Key programming concepts

- **Classes** – "blueprint"
- **Objects** – instances of a class, has their own attributes and methods
- **Attributes** – "traits" of an object
- **Variables** – models info to be stored, data types, primitives, object variables
- **Methods** – models "abilities" of an object, IPO model, input parameters, return type, header, method calls, variable substitution
- **Statements** – "command" in the program, ends in ;, statement block { }, assignments, arithmetic operations

Key programming concepts

- **Classes** – "blueprint"
- **Objects** – instances of a class, has their own attributes and methods
- **Attributes** – "traits" of an object
- **Variables** – models info to be stored, data types, primitives, object variables
- **Methods** – models "abilities" of an object, IPO model, input parameters, return type, header, method calls, variable substitution
- **Statements** – "command" in the program, ends in ;, statement block { }, assignments, arithmetic operations

Hardest concepts:

1. Class and objects
2. Method calls and variable substitutions

Pre-build Java classes

- **String** – used to compare and change phrases
- **Math** – used to carry out more complicated math operations
- **Random** – used to generate random numbers (of various data types)
- **Scanner** – used to get input from user

Second half of course

- Java nuances
  - Scope
  - Pass by value vs. pass by reference
  - Static
  - Overloading
- Techniques for building more interesting programs
  - Modeling decisions (if statements)
  - Modeling repetitions (for loop, while loop)
  - Modeling collections of data (arrays)
  - If time: graphical user interfaces

Part 1: Scope

- **Scope** defines the area in a program where a label (variable or method) can be referred to
- E.g., you wrote "`int var = 10;`" in the constructor of the `Dog` class.
  Can we use `var` …
  - Anywhere in the same method?
  - In other methods within the same class?
  - In other classes?

Lifetime of a variable

- Scope of a variable defines where that variable is available in the program
- For a variable:
  - Check where it is declared
  - Look for closest enclosing braces
  - That variable is available anywhere within those braces
- Once we are outside those braces, the variable is destroyed; the variable is only “alive” inside those braces

Example: class attributes

```
public class Dog
{
    String name;
    int stomach;
    public Dog( String n )
    {
        name = n;
        stomach = 0;    // empty stomach
    }
    public void eatSnacks( int num ) { … }
}
```

scope

Example: class attributes

```
public class Dog
{
    String name;
    int stomach;
    public Dog( String n )
    {
        name = n;
        stomach = 0;    // empty stomach
    }
    public void eatSnacks( int num ) { … }
}
```

name and stomach are available anywhere in this class

scope

Example: variables inside methods

```
public class Dog
{
    String name;
    int stomach;
    public Dog( String n ) { … }
    public void eatSnacks( int num )
    {
        int half = num/2;
        stomach = stomach + half;
    }
}
```

scope

Example: variables inside methods

```
public class Dog
{
    String name;
    int stomach;
    public Dog( String n ) { … }
    public void eatSnacks( int num )
    {
        int half = num/2;
        stomach = stomach + half;
    }
}
```

scope

half is available inside this method only

Example: input parameter to method

```
public class Dog
{
    String name;
    int stomach;
    public Dog( String n ) { … }
    public void eatSnacks( int num )
    {
        int half = num/2;
        stomach = stomach + half;
    }
}
```

scope

Example: input parameter to method

```
public class Dog
{
    String name;
    int stomach;
    public Dog( String n ) { … }
    public void eatSnacks( int num )
    {
        int half = num/2;
        stomach = stomach + half;
    }
}
```

scope

num is available inside this method only

Where methods can be seen

- Scope of a method defines where that method is available and dictates how the method should be called
- How you call a method:
  - Check which class it is defined in
  - Check which class you want to call that method from
  - If it's the same class, then just call it
    - E.g., eatSnacks(8);
  - If it's a different class, then you can only call it via an object of that class
    - E.g., casper.eatSnacks(8);

Example: within same class

```
public class Dog
{
	String name;
	int stomach;
	public Dog( String n ) { … }
	public void eatSnacks( int num )
	{
		int half = num/2;
		stomach = stomach + half;
		hideSnacks( half );
	}
	private void hideSnacks( int remaining ) { … }
}
```

Example: within same class

```
public class Dog
{
    String name;
    int stomach;
    public Dog( String n ) { … }
    public void eatSnacks( int num )
    {
        int half = num/2;
        stomach = stomach + half;
        hideSnacks( half );
    }
    private void hideSnacks( int remaining ) { … }
}
```

Called in the same class as where it is defined

Example: in different classes

```
public class TestDog
{
    public static void main( String[] args )
    {
        Dog casper = new Dog( “Casper” );
        casper.eatSnacks();
    }
}
```

Example: in different classes

```
public class TestDog
{
    public static void main( String[] args )
    {
        Dog casper = new Dog( “Casper” );
        casper.eatSnacks();
    }
}
```

Called in a different class from where it is defined

Part 2: modeling decisions

- Sometimes, we want to express:
  "do X if situation A occurs, do Y if situation B occurs, do Z if any other situation occurs"
- Examples:
  - Asks if user wants to play again: if yes, start another round of the game, if no, end game
  - Asks user to guess a number: if correct, user wins, if incorrect, user loses
  - Checking high scores: if new score is higher than current high score, update the high score, otherwise, no changes needed

If statement

- In Java, we use an if-statement also called conditional statement
- Templates:

```
if( condition )
    statement
```

```
if( condition )
    statement
else
    statement
```

where a condition is a boolean expression and a statement can be replaced by a statement block (more than one statements)

High Score example

```
public class Player
{
	int hiScore;
	…
	public void updateScore( int newScore )
	{
		if( newScore > hiScore )
			hiScore = newScore;
		else
			hiScore = hiScore;		// equals itself – redundant
	}
}
```

High Score example

```
public class Player
{
	int hiScore;
	…
	public void updateScore( int newScore )
	{
		if( newScore > hiScore )
			hiScore = newScore;
		else
			hiScore = hiScore;
	}
}
```

Equivalent:

```
If( newScore > hiScore )
	hiScore = newScore;
```

High Score example

```
public class Player
{
	int hiScore;
	…
	public void updateScore( int newScore )
	{
		if( newScore > hiScore )
			hiScore = newScore;
		else
			hiScore = hiScore;
	}
}
```

Equivalent:

```
If( newScore > hiScore )
	hiScore = newScore;
```

Equivalent:

```
If( hiScore < newScore )
	hiScore = newScore;
```

Set Answer example

```
public class Question
{
    String questionWords;
    String option1, option2;
    int correctAnswer;
    …
    public void setAnswer( int i )
    {
        correctAnswer = i;
    }
}
```

Previously:
Sets the correct answer as the i'th option in a multiple choice question
e.g. in test class:
Question q1 = …;
q1.setAnswer(2);

Set Answer example

```
public class Question
{
    String questionWords;
    String option1, option2;
    int correctAnswer;
    …
    public void setAnswer( int i )
    {
        if( i > 0 && i < 3 )            // ensure i is within range
            correctAnswer = i;
        else
            System.out.println( “error: i is out of range” );
    }
}
```

What goes in the conditions?

- Boolean expression is an expression that evaluates to true or false
  - An expression is a piece of code that has a value
    - e.g., a variable, a method call, applying an operator (5 + 2)
- Common use of conditions is to check for equality or relation
  - E.g., is var1 equal to var2?
  - E.g., is var1 greater than 0?
  - E.g., is var1 between 0 and 5?

Equality and relational operators

Operator	Meaning	Example
==	equal to	a == b
!=	not equal to	a != b
<	less than	a < b
>	greater than	a > b
<=	less than or equal to	a <= b
>=	greater than or equal to	a >= b

for primitive types only
not for objects!

Examples

- Lets say we have variables sum, delta, MAX
- if(sum < MAX)
  delta = sum – MAX;

Examples

- Lets say we have variables sum, delta, MAX
- if(sum < MAX)
  delta = sum – MAX;
- if(sum != MAX)
  delta = sum – MAX;
  else
  delta = sum;

Examples

- Lets say we have variables sum, delta, MAX
- if(sum < MAX)
  delta = sum – MAX;
- if(sum != MAX)
  delta = sum – MAX;
  else
  delta = sum;
- if(sum > MAX)
  {
  delta = sum – MAX;
  sum = sum – 1;
  }

Examples

- Lets say we have variables sum, delta, MAX
- if(sum < MAX)
  delta = sum – MAX;
- if(sum != MAX)
  delta = sum – MAX;
  else
  delta = sum;
- if(sum > MAX)
  {
  delta = sum – MAX;
  sum = sum – 1;
  }
- if(sum >= MAX)
  delta = MAX;
  else
  {
  delta = sum;
  sum = sum + MAX;
  }

Building more elaborate conditions

- Use logical operators to combine boolean expressions
- Logical NOT `!` takes 1 operand
- Logical AND `&&` takes 2 operands
- Logical OR `||` takes 2 operands

Logical NOT

- A unary operator
- Meaning negation (also called complement)
- Possible values represented as a truth table

NOT:

a	!a
true	false
false	true

Using NOT

- E.g.:

```
boolean found = true;
if( found )
    System.out.println( “book was found” );
else
    System.out.println( “book was not found” );

// the else part is equivalent to the following
if( !found )
    System.out.println( “book was not found” );
```

Logical AND and OR

- Binary operators
- AND: means when both operands are true
- OR: means at least one operand is true

AND and OR

a	b	a && b	a || b
true	true	true	true
	false	false	true
false	true	false	true
	false	false	false

alternate to get all combos

if both true then true otherwise false

↳ if both false then false otherwise true

Examples

```
int x = 5;
int y = 2;
boolean found = true;
int total = x + y;
int max = 10;
if(( total < ( max + 5 )) && !found )
      max = 1;
else
      max = 2;
if( total > max || x <= y )
      max++;
else
      max = 0;
```

Examples

```
int x = 5;
int y = 2;
boolean found = true;
int total = x + y;
int max = 10;
if(( total < ( max + 5 )) && !found )
      max = 1;
else
      max = 2;
if( total > max || x <= y )
      max++;
else
      max = 0;
```

becomes: (7 < (10+5)) && !true
becomes: true && false
becomes: false
evaluate: max = 2;

Examples

```
int x = 5;
int y = 2;
boolean found = true;
int total = x + y;
int max = 10;
if(( total < ( max + 5 )) && !found )
      max = 1;
else
      max = 2;
if( total > max || x <= y )
      max++;
else
      max = 0;
```

becomes: (7 > 2 || 5 <= 2)
becomes: true && false
becomes: true
evaluate: max++;

Changes in control flow

- Previously:
  1. Start inside the main() method
  2. Evaluate top-down
  3. If method call, then go to that method definition and evaluate its statements top-down, come back
  4. Continue (repeat 3 if necessary)

Changes in control flow

- Previously:
  1. Start inside the main() method
  2. Evaluate top-down
  3. If method call, then go to that method definition and evaluate its statements top-down, come back
  4. Continue (repeat 3 if necessary)
- With if statements:
  1. Evaluate condition
  2. If true – evaluate then-statement(s), ignore else part
  3. If false – evaluate else-statement(s), ignore then part
- Also called branching, when the control flow branches out to different parts of the code

Using statement blocks

- if you want to include multiple statements in either the then or else part, use a statement block
- E.g.:

```
if( total > max )
{
        x++;
        y = x/2;
}
else
{
        x--;
        y = x;
}
```

Handling multiple conditions

- Recall the syntax is:

```
if( condition )
    statement
else
    statement
```

- That means you can put another if statement inside the then or else part

Examples

```
// do something to x depending on how
// total compares to max
if( total > max )
    x = x+1;
else if( total == max )
    x = y;
else
    x--;
```

- Only one if-statement
- Only one branch of statements will be evaluated, depending on which condition is true

Nested if-statement

```
if( total > max )
{
      if( total > (max*2) )
            x = x*x;
      else
            x = x+1;
}
else if( total == max )
      x = y;
else
      x--;
```

- Only one if-statement
- Only one branch of statements will be evaluated

Activity: find smallest among 3 numbers

- Say you have integers num1, num2, num3
- Say you have integer min
- Write an if-statement that compares these three numbers (there are several combinations, so you will have several conditions)
- Assign min to either num1, or num2, or num3, depending on which condition is true

Solution 1

```
if ( num1 < num2 )
{
        if ( num1 < num3 )
            min = num1;
        else
            min = num3;
}
else
{
        if ( num2 < num3)
            min = num2;
        else
            min = num3;
}
```

① < ②

① < ③ (②)

③ = ① < ②

③ < ① < ②

① = 2 or ① > ②

② < ③ (<= ①)

③ = ② <= ①

③ < ② <= ①

Solution 2

```
if((num1 < num2) && (num1 < num3))
    min = num1;
else if((num1 < num2) && (num1 >= num3))
    min = num3;
else if((num1 >= num2) && (num2 < num3))
    min = num2;
else if( (num1 >= num2) && (num2 >= num3))
    min = num3;
```

Last "else" is omitted

do this only if you're sure you've covered all combos