COMP1409: Introduction to Software Development I

Mike Mulder (mmulder10@bcit.ca)

Week 5

Agenda

- Quiz
  - Quiz 4
  - Review Answers
- Assignment 1 Questions
- Lab 4B Concepts
  - Overloading
  - Local Variables and Constants
- Lesson 5
  - Arithmetic operators
  - Incrementing and decrementing
  - switch/case statement
  - Calling internal methods
  - while loops
- Lab 5

Quiz 4

Closed book, laptop, phone, etc.

You have a maximum of 20 minutes to complete

Raise your hand when you are done, and I will retrieve your paper

We will review the answers afterwards

Lab 4B

Key Concepts:

- Calling Setters from the Constructor
- Overloading Constructors
- Overloading Methods
- Local Variables and Constants

Calling Setters from Constructor

- Calling the setter methods from the Constructor is a form of code reuse.
- All the validation of the parameters is centralized in the setter method.
  - Less chance of copy and paste errors
  - Only one place to change if the validation changes
- public **final** setFirstName(String firstName)
  - This is a best practice for methods called from a Constructor
  - The final prevents the method from being Overridden – this is an advanced OOP topic not in scope for this course.

Overloading

- Constructors and methods can both be overloaded
- Overloading allows a class to have two or more methods having same name, if their signatures are different
- Three ways in which a signature can be different:
  1. Different **number** of parameters
  2. Different **type** of parameters

  and/or
  3. Different **order** of parameters.

Overloading constructors

public Book()

public Book(String title)

public Book(int numPages)

public Book(String title, int numPages)

public Book(int numPages, String title)

Overloading methods

```
public void withdraw(double amountUSD){}
public void withdraw(double amountUSD, int yourPIN){}
public void withdraw(String acctNumber, double amountUSD){}
public void withdraw(String acctNumber, double amountUSD, int thePin){}
public void withdraw(int amountCAD){}
```

Local variables vs. instance variables

- Scope
- Lifetime

	Instance Variable	Local Variable
Scope	Class	Method
Lifetime	As long as the Object exists	Each time the method is called

```
public class Point {

   private double x = 0.0;
   private double y = 0.0;

   public Point(double x, double y) {
      this.x = x;
      this.y = y;
   }

   public double distance() {
      double distance =
         sqrt((this.x * this.x) +
              (this.y * this.y));

      return distance;
   }
}
```

Instance Variables

Local Variable

Static and Final Variables

Constants (use in place of Magic Numbers)

```
public static final int NUMBER_OF_BEERS = 6;
```

(Note: public if accessible outside of the class)

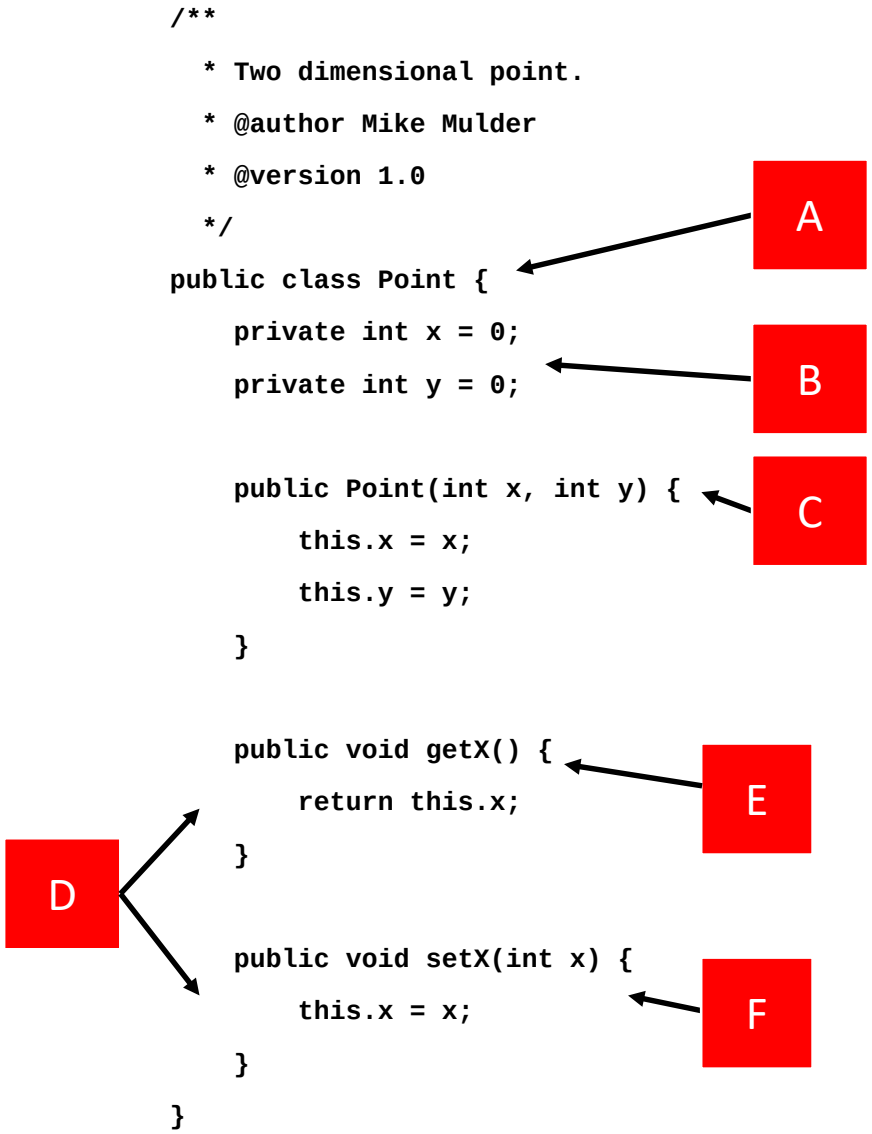
Class Variables (the same for all instances of a class, but changeable)

```
private static int numberOfBeers = 6;
```

Final Instance Variables (set once in the constructor, then unchangeable)

```
private final int numberOfBeers;
```

Review - Anatomy of a Java Class



A – Class Name (and Visibility)

B – Instance Variables (including visibility and type)

C – Constructor (including visibility and parameters)

D – Methods

E – Accessor/Getter/Get Method

F – Mutator/Setter/Set Method

Learning outcomes: lesson 5

- Arithmetic operators
- Incrementing and decrementing
- switch/case statement
- Calling internal methods
- while loops

Arithmetic operators

+	addition
-	subtraction
*	multiplication
/	division
%	modulus (remainder after division)
++	increment
--	decrement

Arithmetic operators with assignment operator

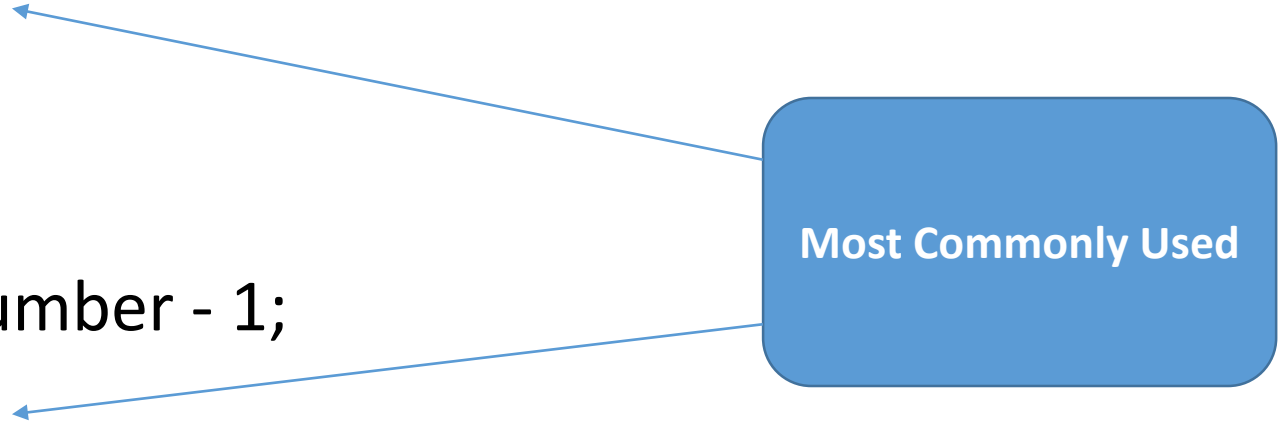
+=	add then assign	
-=	subtract then assign	
*=	multiply then assign	// don’t
/=	divide then assign	// do
%=	remainder then assign	// these

Example

```
int number = 1;
int value = 3;

(a) number = number + value;
or
(b) number += value;                    // does the same thing as (a)

(c) number = number - value;
or
(d) number -= value;                    // does the same thing as (c)
```

Incrementing and decrementing

```
int number = 6;

Increment:   number = number + 1;
or:          number++;
or:          ++number;

Decrement: number = number - 1;
or:          number--;
or:          --number;
```



switch/case statement

- Alternative to “if” statements
- Allows a variable to be tested for equality against a list of discrete values
- Each value is called a case
- The variable used can only be **int**, **char**, **String**, byte, short, or enum
- Caution: Statements will execute until a break statement is reached
- Can have an optional **default** case, which can only appear at the end of the entire switch.

switch/case statement

```
public static void test(String direction){
      switch(direction){
         case "north":
            System.out.println("going up");
            break;          // if you remove this break, “north” prints
                            // “going up going down”
         case "south":
            System.out.println("going down");
            break;
         case "west":
            System.out.println("going left");
            break;
         case "east":
            System.out.println("going right");
            break;
         default:
            System.out.println("error");
            break;
      }
   }
```

Same:

```
public static void test(String direction){
                if(direction.equals("north")){
                        System.out.println("going up");
                }else if(direction.equals("south“)){
                        System.out.println("going down");
                }else if(direction.equals("west")){
                        System.out.println("going left");
                }else if(direction.equals("east")){
                        System.out.println("going right");
                }else{
                        System.out.println("error");
                }
}
```

switch/case statement

```
switch(selection)
{
    case 1:
        //do something here
        break;

    case 2:
        //do something here
        break;

    case 3:
        //do something here
        break;

    default:
        //do something here when all other cases fail
        break;
}
```

*note the variable *selection* is of type int

switch/case using “if” instead

*note the variable *selection* is of type int

```
if(selection == 1){
        // do something here
}else if(selection == 2){
        // do something here
}else if(selection == 3){
        // do something here
}else{
        // do something here when all other     // cases fail
}
```

switch/case statement

```
// prints the range of grades that gives that letter (e.g. if the parameter is 'A' or 'a' (case insensitive!) then the method prints "80 to 100", etc…. A is 80 to 100;  B is 60 to 79;  C is 50 to 59;  F is 0 to 49; other grades just print "invalid letter grade"
public void printGradeRange(char letterGrade){
        switch(letterGrade){
                case('A'):
                case('a'):
                        System.out.println("80 to 100");
                        break;
                case('B'):
                case('b'):
                        System.out.println("60 to 79");
                        break;
                case('C'):
                case('c'):
                        System.out.println("50 to 59");
                        break;
                case('F'):
                case('f'):
                        System.out.println("0 to 49");
                        break;
                default:
                        System.out.println("invalid letter grade");
                        break;
        }
}
```

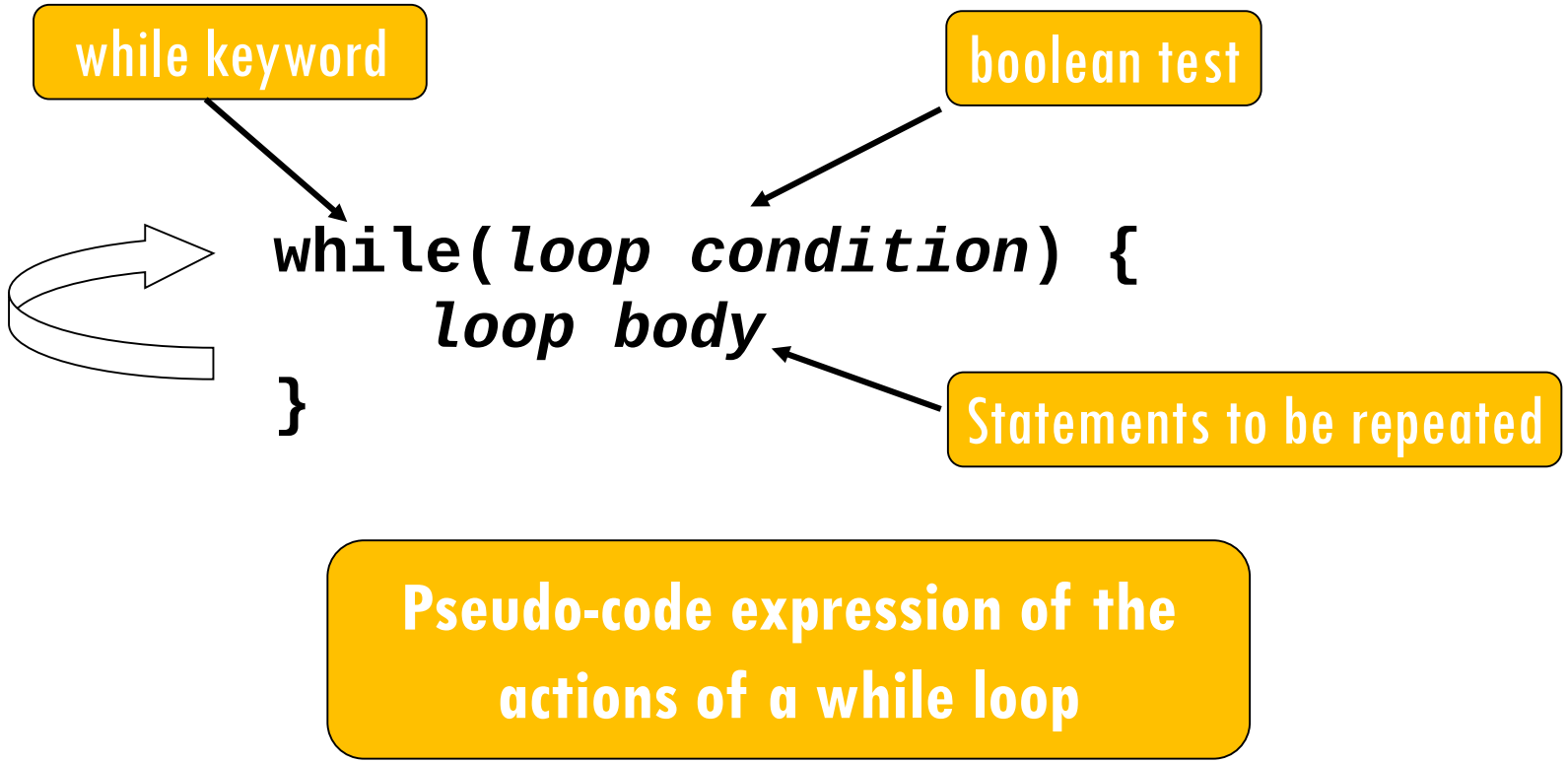
Calling own methods

```
public BankAccount(int thePIN, double theBalanceUSD){
    if(isPinValid(thePIN)){
        PIN = thePIN;
    }
}

private boolean isPinValid(int aPIN){
    if((aPIN < 0) || (aPIN > 9999)){
        return false;
    }else{
        return true;
    }
}
```

while loops

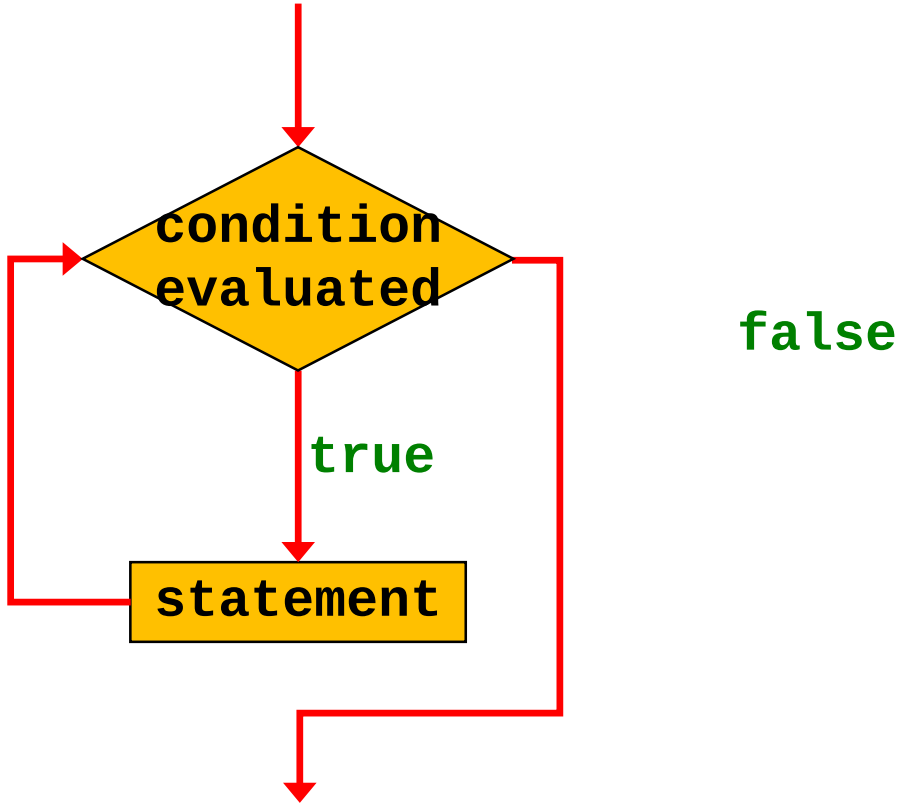
- *Repetition statements* allow us to execute a statement multiple times
- Like “if” statements, loops are controlled by boolean expressions
- Java has several kinds of repetition statements, such as `while loops, for loops, for-each loops, and do-while loops`
- Let’s look at the `while` loop.

while loop



"while the condition is true, do the things in the loop body."

while loop



while loop

```
while(condition)
{
    statement 1;
    statement 2;
    etc…
}
```

1. If the condition is true, the statements are executed
2. Then the condition is evaluated again, and if it is still true, the statements are executed again
3. Repeat until the condition becomes false.

while loop

```
// Print even numbers from 2 to 30

int index = 2;
while(index <= 30) {
    System.out.println(index);
    index = index + 2;
}
```

while loop

```
// Sum of the even numbers from 2
to 30, inclusive
int index = 2;
int sum = 0;
while(index <= 30) {
    sum += index;
    index = index + 2;
}
System.out.println(sum);
```

Lab 5

5A (In Class)

- More Overloading Constructors and Methods
- Case/Switch Statements and While Loop
- Incrementing
- Calling Internal Methods

5B (Due Wednesday at Midnight)

- Arithmetic Operators
- **Make sure you test your code before you submit it to D2L!**