Programming I

Section 2 - Classes & Objects

Learning Outcomes

- After this lecture you should be able to:
  - Distinguish between Variables and Constants
  - Name the basic components of object-oriented programming
  - Differentiate classes and objects.
  - Describe the difference between object declaration and creation
  - Understand the meaning of instance variables & methods.
  - ***Reading*** and ***studying*** recommended text is essential to improve your understanding of the above

Variables

- A variable is a named location in a computer's memory that stores a value
- numbers and other data can be *remembered* in the computer's memory and can access that data through program elements called variables
- A variable has three properties:
  - A memory location to store the value,
  - The type of data stored in the memory location, and
  - The name used to refer to the memory location

Java Variable Types

- Instance Variables (Non-Static Fields)
  - objects store their individual states in "non-static fields" (no static keyword). As their values are unique to each instance of a class, non-static fields are also known as instance variables.
- Class variables (Static Fields)
  - any field declared with the static modifier and no matter how many times the class is instantiated there is only one copy of this variable.
- Local Variables
  - local variables store temporary state inside a method and are only visible to the methods in which they are declared.
- Parameters
  - parameters are variables that provide additional information to a method, parameters are always classified as "variables" not "fields"

Naming & Declaring Variables

- Variables are case sensitive, convention to always begin with a letter not _ or $.
  - Use full words as its easier to read and understand.
  - Cannot use keywords or reserved words
  - If variable is only one word use lower case, for variables with more words, capitalise each subsequent letter.

```
int sum; // Declare variable
double number1, number2, sum; // Declare variables
String firstName; // Declare variable
```

  - Variables can be declared and initialised.

```
int i = 10; // Declare & Initialise variable
String lastName = "Wenger"; // Declare & Initialise variable
```

Constants

- We can change the value of a variable. If we want the value to remain the same, use a *constant*.
- For a constant, use upper case and an underscore to separate each subsequent word.

```
final int MINIMUM = 100;
final double PI = 3.14159;
final int MONTH_IN_YEAR = 12;
```

The reserved word **final** is used to declare constants.

These are constants, also called *named constant*.

These are called *literal constant.*

Data Types

- Java is a statically typed language, meaning you need to declare variables before you can use them.
- Primitive data types are predefined by the language and are identified by a reserved keyword.
- The eight primitive data types supported by the Java programming language are:
  - *byte*; *short*; *int*; *long*; *float*; *double*; *boolean* and *char*.
- Non-primitive data types are known as reference data types, variables of reference types store the locations of objects in a computer's memory.
  - *Objects are called reference data types, because the contents are addresses that refer to memory locations where the objects are actually stored*.

OOP Concepts - Objects

- Objects (the classes objects come from) are essentially reusable software components.
- Software objects are used to model real-world objects (nouns)
- Software objects have two characteristics
  - attributes (e.g., name, color and size)
  - behaviors (e.g., calculating, moving and communicating).
- Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.
- object-oriented programs are often easier to understand, correct and modify.

OOP Concepts - Classes

- Class declarations begin with the keyword *public* (an access modifier) and are stored in a file with a .java extension

```
public class Bicycle {
        …
}

File name Bicycle.java
```

- Note the Bicycle class is NOT an application as it does not contain a main method.

OOP Concepts - Classes (contd.)

- the main method is called automatically by the JVM when an application is executed therefore to run the Bicycle class you must either
  - declare a separate class that contains a main method
  - put a main method in the Bicycle class

- Normally write a tester/driver class to test new classes

```
public class BicycleRegistration {

    public static void main(String[] args) {
        …
    }
}
```

OOP Concepts - Methods

- Objects have states and behaviours
  - state are stored in fields (variables)
  - behavior is created through methods (functions)
- Methods operate on an object's internal state and act as the main mechanism for objects to communicate with each other.
- Hiding internal state and ensuring all interaction to be performed through an object's methods is known as data encapsulation

OOP Concepts – Methods (contd.)

- Methods normally need to be invoked explicitly for them to perform their tasks
  - A method with a public access modifier is available to the public and can be called from methods in other classes.
  - The return type specifies the type of data returned by a method.

```
public String getOwnerName( ) {
    return ownerName;
}
```

  - A method with a void return type will not return a value

```
public void setOwnerName(String name) {
    ownerName = name;
}
```

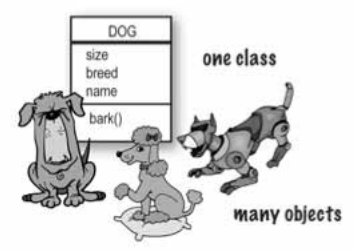
Bicycle Class

```
class Bicycle {

    // Data Member
    private String ownerName;

    //Constructor
    public Bicycle( ) {
        ownerName = "Unassigned";
    }
    //Returns the name of this bicycle's owner
    public String getOwnerName( ) {
        return ownerName;
    }
    //Assigns the name of this bicycle's owner
    public void setOwnerName(String name) {
        ownerName = name;
    }
}
```

Bicycle Tester Class

```
class BicycleRegistration {

    public static void main( String[] args ) {

        String  owner1, owner2; //Declare String variables

        Bicycle bike1 = new Bicycle( ); //Declare and Create Bicycle object
        bike1.setOwnerName(”Sally"); //assign values to bike1 (object.method)

        Bicycle bike2 = new Bicycle( ); //Declare and Create object
        bike2.setOwnerName(”Harry"); //assign values to bike2 (dot notation)

        //Output information on two bicycles
        owner1 = bike1.getOwnerName( ); //gets values for bike2 (dot notation)
        owner2 = bike2.getOwnerName( );

        System.out.println(owner1 + " owns a bicyle");
        System.out.println(owner2 + " also owns a bicycle");
    }
}
```

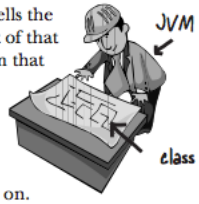
Class vs. Object



A class is not an object.

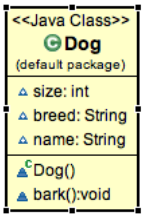
(but it's used to construct them)

A class is a *blueprint* for an object. It tells the virtual machine *how* to make an object of that particular type. Each object made from that class can have its own values for the instance variables of that class. For example, you might use the Button class to make dozens of different buttons, and each button might have its own color, size, shape, label, and so on.

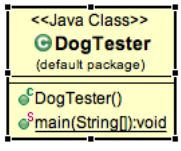


Creating an Object

```
// Step 1 Write Your Class
public class Dog {

    // instance variables
    int size;
    String breed;
    String name;

    //method
    public void bark(){
        System.out.println("Ruff! Ruff!");
    }
}
```




```
// Step 2: Write a tester class
public class DogTester{
    // Main method
    public static void main(String[] args){
        // put your Dog test code here
    }
}
```



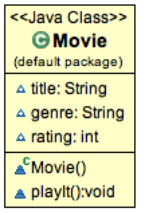
Creating an Object

```
// Step 3: In the tester class, create an object and access the
//         objects variables and methods

public class DogTester{

    // Main method
    public static void main(String[] args){

        // Declaration & creation of a Dog object
        Dog spot = new Dog();

        // Use the dot operator to set the size
        spot.size = 20;

        // Use the dot operator to call bark method
        spot.bark();
    }
}
```

Dot operator (.) gives you access to an object's state and behaviour (instance variables & methods)

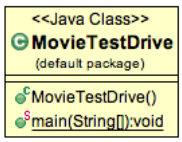
Movie Class

```
class Movie {

    String title;
    String genre;
    int rating;

    void playIt(){
        System.out.println("Playing the Movie");
    }

}
```



MovieTestDrive Class

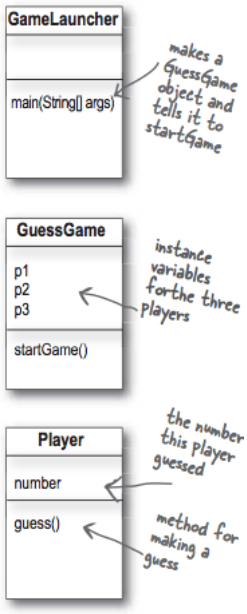
```
public class MovieTestDrive {

    public static void main(String [] args){

        Movie one = new Movie();
        one.title = "Gone with the ...";
        one.genre = "Black Comedy";
        one.rating = -1;

        Movie two =  new Movie();
        two.title = "Young Frankenstien";
        two.genre = "Comedy";
        two.rating = 4;
        two.playIt();

    }

}
```



Guessing Game

```
public class Player {
    int number = 0;
    public void guess()
    {
        number = (int) (Math.random() * 10);
        System.out.println("I'm guessing " + number);
    }
}
```

```
public class GameLauncher {
    public static void main (String[] args) {
        GuessGame game = new GuessGame();
        game.startGame();
    }
}
```



GuessGame Class

```
public class GuessGame {
    // Create 3 instance variables for the 3 player objects
    Player p1;
    Player p2;
    Player p3;

    public void startGame() {

    // Create 3 player objects & assign them to the 3 player instance variables
       p1 = new Player();
       p2 = new Player();
       p3 = new Player();

    // Declare 3 variables to hold the player's guesses
       int guessp1 = 0;
       int guessp2 = 0;
       int guessp3 = 0;

    // Declare 3 variables to hold a true/false based on player's answer
       boolean p1isRight = false;
       boolean p2isRight = false;
       boolean p3isRight = false;
```

Guessing Game

```
    // Generate the number the players have to guess
      int targetNumber = (int) (Math.random() * 10);

      System.out.println("I'm thinking of a number between 0 and 9...");

      while(true) {
         System.out.println("Number to guess is " + targetNumber);

         // Call each player's guess() method
         p1.guess();
         p2.guess();
         p3.guess();

         // assign P1's number (the result of guess()) to guess
         // variable by accessing the number variable of each player
         guessp1 = p1.number;
         System.out.println("Player one guessed " + guessp1);

         guessp2 = p2.number;
         System.out.println("Player two guessed " + guessp2);

         guessp3 = p3.number;
         System.out.println("Player three guessed " + guessp3);
```

GuessGame Class contd.

```
// if player's guess matches targetNumber set variable to true
       if (guessp1 == targetNumber) { p1isRight = true;}
       if (guessp2 == targetNumber) { p2isRight = true;}
       if (guessp3 == targetNumber) { p3isRight = true;}

// if player one OR player two OR player three is correct (|| operator means OR)
       if (p1isRight || p2isRight || p3isRight) {
          System.out.println("We have a winner!");
          System.out.println("Player one got it right? " + p1isRight);
          System.out.println("Player two got it right? " + p2isRight);
          System.out.println("Player three got it right? " + p3isRight);
          System.out.println("Game is over");
          break; // game over, break out of loop
       }
       else {
          // stay in loop and keep playing nobody guessed correctly
          System.out.println("Players will have to try again.");
       }
     } // end loop
   } // end method
} // end class
```

Garbage Collection

- Objects when created go on the *heap* (an area of memory)
- Java - garbage-collectable heap
- Java allocates memory on the heap according to how much is each object needs
- When the JVM sees that an object cannot be used again, the object becomes available for garbage collection
- When you are running out of memory, garbage collection will run and throw out unreachable objects, thus freeing up space

Summary

- In this lecture we have covered the following:
  - OOP enables program extension without having to alter previously tested working code
  - All Java code is defined in a class
  - A class is like a blueprint, you can make an object of that class type
  - A class knows things and does things
  - Things an object knows about itself are called instance variables (represent the state of the class)

– *continued*

Summary (contd.)

- Things an object can do are called methods (represent the behaviour of an object)
- Separate test classes can be used to create objects of your class type
- At runtime, a Java program consists of objects speaking to other objects.