Classes and Objects in JAVA

Lecture Outline

1. Classes
   i. Variable Types
   ii. Access Modifiers
   iii. Constructors
2. Objects

Class

A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

```
public class Dog {
   String breed;
   int age;
   String color;

   void barking() {
   }

   void hungry() {
   }

   void sleeping() {
   }
}
```

Class - Variable Types

A class can contain any of the following variable types.

- **Class variables**
- **Instance variables**
- **Local variables**

Class – Class Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- **There would only be one copy of each class variable per class, regardless of how many objects are created from it.**

```
public class VariableExample{
   int myVariable;
   static int data = 30;

   public static void main(String args[]){
      int a = 100;
      VariableExample obj = new VariableExample();

      System.out.println("Value of instance variable
myVariable: "+obj.myVariable);
      System.out.println("Value of static variable data:
"+VariableExample.data);
      System.out.println("Value of local variable a: "+a);
   }
}
```

Class – Instance Variables

- Instance variables are declared in a class, but outside a method.
- When space is allocated for an object in the heap, a slot for each instance variable value is created.
- **Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.**

```
public class VariableExample{
   int myVariable;
   static int data = 30;

   public static void main(String args[]){
      int a = 100;
      VariableExample obj = new VariableExample();

      System.out.println("Value of instance variable
myVariable: "+obj.myVariable);
      System.out.println("Value of static variable data:
"+VariableExample.data);
      System.out.println("Value of local variable a: "+a);
   }
}
```

Class – Local Variables

- Local variables are declared in methods, constructors, or blocks.
- **Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.**

```
public class VariableExample{
   int myVariable;
   static int data = 30;

   public static void main(String args[]){
      int a = 100;
      VariableExample obj = new VariableExample();

      System.out.println("Value of instance variable
myVariable: "+obj.myVariable);
      System.out.println("Value of static variable data:
"+VariableExample.data);
      System.out.println("Value of local variable a: "+a);
   }
}
```

Class – Access Modifiers

Java provides a number of access modifiers to set access levels for classes, variables, methods, and constructors.

The four access levels are −

- Visible to the package, the default. No modifiers are needed.
- Visible to the class only (**private**).
- Visible to the world (**public**).
- Visible to the package and all subclasses (**protected**).

Class – Access Modifiers

Default Access Modifier - No Keyword

- Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
- **A variable or method declared without any access control modifier is available to any other class in the same package.**
- The fields in an interface are implicitly public static final and the methods in an interface are by default public.

```
String version = "1.5.1";

boolean processOrder() {
   return true;
}
```

Class – Access Modifiers

Private Access Modifier - Private

- Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
- Private access modifier is the most restrictive access level. Class and interfaces cannot be private.
- **Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.**
- Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

```
public class Logger {
   private String format;

   public String getFormat()
    {
      return this.format;
    }

   public void setFormat(String
    format)
    {
      this.format = format;
    }
}
```

Class – Access Modifiers

Public Access Modifier - Public

- A class, method, constructor, interface, etc. declared public can be accessed from any other class.
- **Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.**
- However, if the public class we are trying to access is in a different package, then the public class still needs to be imported.
- Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

```
public static void main(String[]
arguments) {
   // ...
}
```

Class – Access Modifiers

Protected Access Modifier - Protected

- **Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.**
- The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.
- Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

```
class AudioPlayer {
   protected boolean
openSpeaker(Speaker sp) {
      // implementation details
   }
}

class StreamingAudioPlayer extends
AudioPlayer {
   boolean openSpeaker(Speaker sp) {
      // implementation details
   }
}
```

Class – Access Modifiers

Access Control and Inheritance

The following rules for inherited methods are enforced −

- Methods declared public in a superclass also must be public in all subclasses.
- Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.
- Methods declared private are not inherited at all, so there is no rule for them.

Class – Constructors

- A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.
- Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other start-up procedures required to create a fully formed object.
- All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.

Class – Constructors

```
class ClassName {
    ClassName() {
    }
}
```

Class – Constructors

No argument Constructors

As the name specifies the no argument constructors of Java does not accept any parameters instead, using these constructors the instance variables of a method will be initialized with fixed values for all objects.

Example

```
Public class MyClass {
   Int num;
  MyClass() {
      num = 100;
   }
}
```

```
public class ConsDemo {
   public static void main(String args[]) {
      MyClass t1 = new MyClass();
      MyClass t2 = new MyClass();
      System.out.println(t1.num + " " + t2.num);
   }
}
```

Class – Constructors

Parameterized Constructors

Most often, you will need a constructor that accepts one or more parameters. Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.

Example

```
// A simple constructor.
class MyClass {
   int x;
   // Following is the constructor
   MyClass(int i ) {
      x = i;
   }
}
```

```
public class ConsDemo {
   public static void main(String args[]) {
      MyClass t1 = new MyClass( 10 );
      MyClass t2 = new MyClass( 20 );
      System.out.println(t1.x + " " + t2.x);
   }
}
```

Objects

Objects have states and behaviors.

Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.

```
public class Puppy {
   public Puppy(String name) {
      // This constructor has one
        parameter, name.
      System.out.println("Passed
        Name is :" + name );
   }

public static void main(String
[]args) {
      // Following statement would
        create an object myPuppy
      Puppy myPuppy = new
        Puppy( "tommy" );
   }
}
```

Sources

https://www.tutorialspoint.com/java/java_object_classes.htm

Thanks