Objects And Classes

Introduction

- OOP is a technique used to develop programs revolving around the real world entities. In OOPs programming model, programs are developed around data rather than actions and logics.
- In OOP, every real life object has properties and behavior. which is achieved through the class and object creation.
- They contain properties (variables of some type) and behavior (methods).
- OOP provides a better flexibility and compatibility for developing large applications.

- Java is a fully Object Oriented language because object is at the outer most level of data structure in java. No stand alone methods, constants, and variables are there in java. Everything in java is object even the primitive data types can also be converted into object by using the wrapper class.
- **Class:** A class defines the properties and behavior (variables and methods) that is shared by all its objects. It is a blue print for the creation of objects. The primitive data type and keyword void is work as a class object.
- **Object:** the basic entity of object oriented programming language. Class itself does nothing but the real functionality is achieved through their objects. Object is an instance of the class. It takes the properties (variables) and uses the behavior (methods) defined in the class.

Objects

- Consider the real-world, many objects exist around us eg cars, cats, dogs, humans, etc. All these objects have a state and a behavior.
- Example a dog:
  - its state is - name, breed, color,
  - behavior is - barking, wagging the tail, running.
- If you compare the software object with a real-world object, they have very similar characteristics.
- Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.

Classes

- A class is a blueprint from which individual objects are created.

- **Example**

```
public class Dog {
   String breed;
   int age;
   String color;
   void barking() {
   }
   void hungry() {
   }
   void sleeping() {
   }
}
```

Types of variables

- A class can contain any of the following variable types.
- **Local variables** –
  - Are variables defined inside methods, constructors or blocks. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables**
  - are variables within a class but outside any method.
  - Are initialized when the class is instantiated.
  - Can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables**
  - are declared within a class, outside any method, with the *static* keyword.
  - A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Constructors

- A constructor is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes. In Java, a constructor is a block of codes similar to the method.
- Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.
- Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Characteristics of Constructors

- Constructors must have the same name as the class itself.
- Constructors do not have a return type—not even void.
- Constructors are invoked using the *new* operator when an object is created.
- Constructors play the role of initializing objects.

Constructor Example

```
public class Puppy {
   public Puppy() {
   }

   public Puppy(String name) {
      // This constructor has one parameter, name.
   }
}
```

Constructor Example

```
Public class Circle(double r) {
  radius = r;
}
 // This constructor
Circle() {
  radius = 1.0;
}

aCircle = new Circle(5.0);
```

Types of Constructors in Java

- Default Constructor
- Parameterized Constructor
- Copy Constructor

Default Constructor

- This is one that has no parameters
- A default constructor is invisible.

```
// Java Program to demonstrate Default Constructor
import java.io.*;
// Driver class
class myExample{
    // Default Constructor
    myExample ()
 { System.out.println("This is a default constructor"); }

    // Driver function
    public static void main(String[] args)
    {
         myExample hello = new  myExample ();
    }
}
```

Parameterized Constructor

- A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

```
// Java Program for Parameterized Constructor
import java.io.*;
class Example1{
    // data members of the class.
    String name;
    int id;
 Example1(String name, int id)
    {
        this.name = name;
        this.id = id;
    }
}
class Test{
    public static void main(String[] args)
    {
        // This would invoke the parameterized constructor.
        Example1 object1= new Example1("Andrew", 30);
        System.out.println("Your name is:" + object1.name + " and your ID:" + object1.id);
    }
}
```

Copy Constructor

- Unlike other constructors copy constructor is passed with another object which copies the data available from the passed object to the newly created object.

Copy Constructor-Example

```
// Java Program for Copy Constructor
import java.io.*;
class Example2{
    // data members of the class.
    String name;
    int id;

    // Parameterized Constructor
 Example2(String name, int id)
    {
        this.name = name;
        this.id = id;
    }

    // Copy Constructor
 Example2(Example2 obj2)
    {
        this.name = obj2.name;
        this.id = obj2.id;    } }
```

contd

```
class myClass{
    public static void main(String[] args)
    {
        // Invoke the parameterized constructor.
        System.out.println("First Object");
        Example2 obj1= new Example2(“Andrew", 30);
        System.out.println(“Your name :" + object1.name + " and your ID :" + obj1.id);
        System.out.println();
        // Invoke the copy constructor.
 Example2 obj2= new Example2 (obj1);
        System.out.println("Copy Constructor used Second Object");
        System.out.println(“Your name :" + obj2.name + " and GeekId :" + obj2.id);
    }
}
```

Creating an Object

- As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class.
- In Java, the new keyword is used to create new objects.
- There are three steps when creating an object from a class –
- *Declaration* – A variable declaration with a variable name with an object type.
- *Instantiation* – The 'new' keyword is used to create the object.

- *Initialization* – The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Creating an Object Example

```
public class Employee {
   public Employee(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 myEmp
 Employee myEmp = new Employee( “Andrew" );
   }
}
```

Accessing Instance Variables and Methods

- Instance variables and methods are accessed via created objects.
- To access an instance variable, following is the fully qualified path

```
/* First create an object */
ObjectReference = new Constructor();

/* Then call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

Example

- This example explains how to access instance variables and methods of a class.

```
public class Employee {
   int employeeAge;

   public Employee (String name) {
      // This constructor has one parameter, name.
      System.out.println("Name chosen is :" + name );   }
   public void setAge( int age ) {
      employeeAge = age;   }
   public int getAge( ) {
      System.out.println(" Employee 's age is :" + employeeAge );
      return employeeAge;   }
   public static void main(String []args) {
      /* Object creation */
```

Example..contd

```
Employee myEmployee = new Employee( “Simon" );

    /* Call class method to set employee's age */
    my Employee.setAge( 15 );

    /* Call another class method to get employee's age */
    myEmployee.getAge( );

    /* You can access instance variable as follows as well */
    System.out.println("Variable Value :" + my Employee.employeeAge );
  }
}
```

Case Study

- Creating two classes Employee and EmployeeTest.
- The Employee class has four instance variables - name, age, designation and salary.
- The class has one explicitly defined constructor, which takes a parameter.

Class Employee

```
import java.io.*;
class Employee {
   String name;
   int age;
   String designation;
   double salary;
   // This is the constructor of the class Employee
   public Employee(String name) {
      this.name = name;   }
   // Assign the age of the Employee  to the variable age.
   public void empAge(int empAge) {
      age = empAge;   }
   /* Assign the designation to the variable designation.*/
   public void empDesignation(String empDesig) {
      designation = empDesig;   }
   /* Assign the salary to the variable                  salary.*/
   public void empSalary(double empSalary) {
      salary = empSalary;   }

   /* Print the Employee details */
   public void printEmployee() {
      System.out.println("Name:"+ name );
      System.out.println("Age:" + age );
      System.out.println("Designation:" + designation );
      System.out.println("Salary:" + salary);   }}
//continued on next page
```

Contnued

```
import java.io.*;
public class EmployeeTest {
   public static void main(String args[]) {
      /* Create two objects using constructor */
      Employee empOne = new Employee(“Alice Kamau");
      Employee empTwo = new Employee(“Felix Otieno");
      // Invoking methods for each object created
      empOne.empAge(36);
      empOne.empDesignation(“Programmer");
      empOne.empSalary(10000);
      empOne.printEmployee();
      empTwo.empAge(31);
      empTwo.empDesignation(“Network Engineer");
      empTwo.empSalary(20000);
      empTwo.printEmployee();
   }
}
```

Methods

- A method is a named sequence of code that can be invoked by other Java code.
- A method takes some parameters, performs some computations and then optionally returns a value (or object).
- Methods can be used as part of an expression statement.
-

```
public float convertCelsius(float tempC){
        return( ((tempC * 9.0f) / 5.0f) + 32.0 );
        }
```

Method Signatures

- A method signature specifies:
  - The name of the method.
  - The type and name of each parameter.
  - The type of the value (or object) returned by the method.
  - The checked exceptions thrown by the method.
  - Various method modifiers.
  - modifiers type name (parameter list) [throws exceptions]

Example

public float convertCelsius (float tCelsius) {}

public boolean setUserInfo (int i, int j, String name) throws IndexOutOfBoundsException {}

Example: Circle Class

```
// Circle.java: Contains both Circle class and its user class
//Add Circle class code here
class MyMain{
public static void main(String args[]) {
Circle aCircle; // creating reference
aCircle = new Circle(); // creating object
aCircle.x = 10; // assigning value to data field
aCircle.y = 20;
aCircle.r = 5;
double area = aCircle.area(); // invoking method
double circumf = aCircle.circumference();
System.out.println("Radius="+aCircle.r+" Area="+area);
System.out.println("Radius="+aCircle.r+" Circumference ="+circumf);
```

this keyword

- "this" is a special variable in Java, which does not have to be declared. Java makes it available automatically in instance methods and constructors. It holds a reference to the object that is being constructed or that contains the instance method that is being executed (or, in terms of messages, the object that received the message that is being processed).
- It provides a way to refer to "this object." If x is an instance variable, it can also be referred to as this.x within the same class. If doSomething() is an instance method, it can also be called as this.doSomething() within the same class. (Personally, I would be happier with Java if it required the use of "this" instead of using it implicitly.) e.g.

```
void changeRadius(double radius) {
//change the instance variable to the argument value
this.radius = radius;
}
```

- this.radius refers to the instance variable, and radius the parameter

Visibility Control

- Java provides control over the visibility of variables and methods, encapsulation, safely sealing data within the capsule of the class.
- Prevents programmers from relying on details of class implementation, so you can update without worry
- Helps in protecting against accidental or wrong usage.
- Keeps code elegant and clean (easier to maintain)

Visibility/Accessibility Modifiers

- **Public:** makes it available/visible everywhere. Applied to a method or variable, it makes it completely visible.
- **Default** (No visibility modifier is specified): it behaves like public in its package and private in other packages.
- Variables and methods can be declared without any modifiers, as in the following examples:

```
String version = "1.5.1";
boolean processOrder() {
return true;
}
```

Private

- **Private**: fields or methods for a class only visible within that class. Private members are not visible within subclasses, and are not inherited.
- The following class uses private access control:

```
public class Logger {
private String format;
public String getFormat() {
return this.format;
}
public void setFormat(String format) {
this.format = format;
}
}
```

Protected

- Protected members of a class are visible within the class, subclasses and also within all classes that are in the same package as that class.
- The following parent class uses protected access control, to allow its child class override

  openSpeaker() method:

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

Access modifier example

```
public class Circle {
private double x,y,r;
// Constructor
public Circle (double x, double y, double r) {
this.x = x;
this.y = y;
this.r = r;}
//Methods to return circumference and area
public double circumference() { return 2*3.14*r;}
public double area()
{ return 3.14 * r * r;}
}
```

```
//Complete program with a constructor
class Circle {
public double x, y; // centre of the circle
public double r; // radius of circle
// Constructor
public Circle (double x, double y, double r) {
this.x = x;
this.y = y;
this.r = r;
}
//Methods to return circumference and area
public double circumference() {
return 2*3.14*r;
}
public double area() {
return 3.14 * r * r;}
}
```

```
// Circle.java: Contains both Circle class and its user class
//Add Circle class code here
class MyMain_Cons{
public static void main(String args[]) {
Circle aCircle; // creating reference
aCircle = new Circle(10,20,5); // creating object
//aCircle.x = 10; // assigning value to data field
//aCircle.y = 20;
// aCircle.r = 5;
double area = aCircle.area(); // invoking method
double circumf = aCircle.circumference();
System.out.println("Radius="+aCircle.r+" Area="+area);
System.out.println("Radius="+aCircle.r+" Circumference ="+circumf);}
}
```

Other Modifier - final

- It applies to classes, methods, and variables.
- Final features may not be changed.
- A final class may not be subclassed.
- A final variable may not be modified once it has been assigned a value.
- A final method may not be overridden.
- If a final variable is a reference to an object, it is the reference that must stay the same, not the object. Common usage: declaring constants

o Example: final int x = 78;

Other Modifier - abstract

- It applies to classes and methods.
- A class that is abstract may not be instantiated (that is, you may not call its constructor).
- Abstract classes provide a way to defer implementation to subclasses.
- If a class contains at least one abstract method, the compiler insists that the class must be declared abstract.
- Any subclass of abstract classes must provide an implementation of abstract method or declare itself to be abstract.
- abstract is the opposite of final: a final class, for example, may not be subclassed, an abstract class must be subclassed.

Other Modifier - static

- It applies to variables, methods, and even a strange kind of code that is not part of a method called static initializer.
- A static feature belongs to a class, not an instance of the class.(One copy per class)
- Static member is allocated when class is loaded, the initialization also happens at class-load time.

Example:

```
public class SomeClass {
static int i =48; int j = 1;
public static void main(string[] args) {
i += 100;
//j *= 5; //non-static variable j cannot be referenced from a static context
SomeClass s1 = new SomeClass();
s1 .j *= 5; //correct
}}
```

Automatic garbage collection

- Java automatically collects garbage periodically and releases the memory used to be used in the future. The technique that accomplishes this is called garbage collection.
- When an object is no longer referenced, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed.
- Garbage collection occurs only at the time of execution of the program

Example of garbage collection in java

```
public class TestGarbage{
public void finalize()
{System.out.println("object is garbage collected");}
public static void main(String args[]){
TestGarbage s1=new TestGarbage();
TestGarbage s2=new TestGarbage();
s1=null;
s2=null;
System.gc();
} }
```

Exercise on classes

1. A program is required that accepts the basic salary, allowance and tax rate from the keyboard and computes the tax amount and net salary of an employee. The computations are done as follows: -

    tax amount = tax rate/100 * (basic salary + allowance)

    net salary = basic salary – tax amount

- Write a complete program that uses a class called Salary to implement the above requirements. The class should have the following specifications.
- Member variables: -
    - basic_salary
    - allowance
    - tax_rate
    - tax_amount
    - net_salary
- Member methods/functions: -
    - input – for inputting details
    - output – for outputting details
- compute – for computing the tax amount and the net salary.
- In addition, the class should have a constructor that initializes basic salary, allowance and tax amount to specified values or 0 if no value is provided.

2. Write a simple Java program that can be used to compute the area of a rectangle given the length and width.

- The program should have a class Rectangle with private instance variables length and width, method computeArea and two constructors.
- One constructor accepts no parameters but initializes length and width to 0. The other constructor must take two parameters.
- The class should have a main method that is used to create two object references and instantiate two objects.
- Use the object references to call computeArea and to print the results appropriately in a dialog box.