Day 3: Object Oriented Programming Structure (OOPS)

- Classes and Objects
- Constructors
- Garbage Collection
- Inheritance
- static
- final
- abstract

Classes and Objects

Class: It is used to create a user define/real time data type.

- It is a collection of member data/attributes and member functions/methods.
- Member data specifies the type of data to be stored.
- Member functions acts as a mediator between user and data.
- Class implements Encapsulation and abstraction.
- Set of rules/specification/layout for user define data type

Object: Instances of class

- Store data and invoke methods.
- Implementation of Rules

Syntax: Class

```
<AccessSpecifier> class <ClassName> {
        //Member data
        // Member functions
}
```

Access Specifier:

- There are 4 type access specifiers in Java.
- These access specifier specifies the scope of member data, member functions, classes, interface and enums

1. private: Within class
2. protected: Within class and derive classes
3. package: Within directory/folder (package is the default access specifier)
4. public: Anywhere

NOTE: All predefined classes make use of TitleCase

Syntax: Object

```
<ClassName> <objectName> = new <ClassName>();          //Declaration and initialization
OR
<ClassName> <objectName>;                              //Declaration
<objectName> = new <ClassName>();                      //Initialization
```

NOTE: Objects makes use of camelNotation

Example: Point

```
import java.util.Scanner;
```

```
class Point {
	private int x;
	private int y;
	public int getX () {
		return x;
	}
	public void setX (int a) {
		x = a;
	}
	public int getY () {
		return y;
	}
	public void setY (int a) {
		y = a;
	}
	public void acceptData() {
		Scanner a = new Scanner(System.in);
		System.out.println("Enter X  : ");
		x = a.nextInt();
		System.out.println("Enter Y  : ");
		y = a.nextInt();
	}
	public void showData() {
		System.out.println("X : " + x);
		System.out.println("Y : " + y);
	}
	public static void main(String args[]) {
		Point p = new Point();
		p.acceptData();
		p.showData();
		Point p1;
		p1 = new Point();
		p1.setX (10);
		p1.setY (20);
		p1.showData();
	}
}
```

Example: Rectangle

```
import java.util.Scanner;
class Rectangle {
	private int width;
	private int height;
	public int getWidth() {
		return width;
	}
	public void setWidth(int w) {
		width = w;
```

```
	}
	public int getHeight() {
		return height;
	}
	public void setHeight(int h) {
		height = h;
	}
	public void acceptData() {
		Scanner a = new Scanner(System.in);
		System.out.println("Enter Width");
		width = a.nextInt();
		System.out.println("Enter Height");
		height = a.nextInt();
	}
	public void showData() {
		System.out.println("Width :" + width);
		System.out.println("Height :" + width);
	}
	public int getArea() {
		return width * height;
	}
	public int getPerimeter() {
		return 2 * (width + height);
	}
	public static void main(String args[]) {
		Rectangle r = new Rectangle();
		r.acceptData();
		r.showData();
		System.out.println("Area is : " + r.getArea());
	}
}
```

Constructors

- Constructors are used to initialization of User define data types in various forms.
- Constructors are special member functions:
  - Same name as of a class.
  - Constructors are executed automatically/implicitly whenever an object is created (new)
  - Does not have any return type, not even "void"
  - Executed only once in the life span of an object.

In Java we do not write a constructor, Java adds a default constructors that initializes, all numeric member data by 0, character by space, boolean by false and reference types by null.

Garbage Collector

- Java does not have destructors, rather Java provides a utility called "**Garbage Collector"** that executes periodically and releases the unused memory.
- If you want our statements to be executed along with "**Garbage Collector"** you can provide them within: protected void finalize()

- "**Garbage Collector"** can also be invoked by programmer by calling:
  System.gc();

Getter/Setter VS public

```
class MyClass {
        private int a;
        public int b;
        public int getA() {
                return a;
        }
        public void setA(int a) {
                this.a = a > 100 ? 100 : a < 0 ? 0 : a;
        }
}
MyClass obj = new MyClass();
obj.setA(10000);
obj.b = 10000;
```

- With get/set we have options for read or write or both. But, public data always have both the options (read as well as write).
- With set it is possible to validate the data. But, public data cannot be validated.

this

```
class MyClass {
        private int a;
        public int getA() {
                return a;
        }
        public void setA(int a) {
                this.a = a;
        }
}
MyClass obj1 = new MyClass();
obj1.setA(10);
//In the above line obj1 is the current context. Current context is the object in use.
MyClass obj2 = new MyClass();
obj2.setA(20);
//In the above line obj2 is the current context. Current context is the object in use.
```

In Java, when member data and argument shares a common name, member data is referred using "this". "this" refers to the current context. Current context is the object in use.

Example: Point

```
import java.util.Scanner;
class Point {
        protected int x;
        protected int y;
        public Point() {
        }
        public Point(int x, int y) {
```

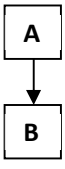
```
                this.x = x;
                this.y = y;
        }
        public int getX () {
                return x;
        }
        public void setX (int x) {
                this.x = x;
        }
        public int getY () {
                return y;
        }
        public void setY (int y) {
                this.y = y;
        }
        public void acceptData() {
                Scanner a = new Scanner(System.in);
                System.out.println("Enter X  : ");
                x = a.nextInt();
                System.out.println("Enter Y  : ");
                y = a.nextInt();
        }
        public void showData() {
                System.out.println("X : " + x);
                System.out.println("Y : " + y);
        }
        public static void main(String args[]) {
                Point p = new Point();
                Point p1 = new Point(10,20);
                p.showData();
                p1.showData();
        }
}
```

Inheritance

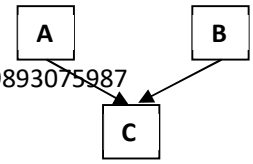
- Passing of properties from one class to another
- Properties include member data and member functions
- The class that gives the properties is called base/super/parent class
- The class that receives the properties is called derive/sub/child class
- Reasons : Reusability, Extending and Overriding

Types of Inheritance

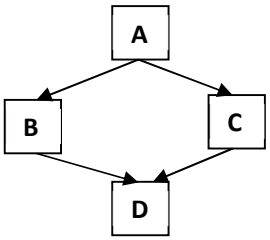
Single



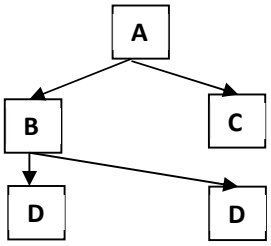
Multiple



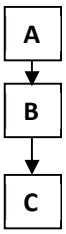
Hybrid



Hierarchical



Multilevel



Function Overloading: Defining Function again with:

- Same name
- Different arguments (number / sequence / data type)
- Same class

Function Overriding: Defining Function again with

- Same name,
- Same arguments (number / sequence/data type)
- Different classes (base and derive)

To inherit a class in Java "extends" keyword is used.

class <DeriveClass> extends <BaseClass>

From the view of derive class there are 3 categories of methods in the base.

1. Constructors: To call constructors of Base class "super" keyword is used, syntax is:

super(<arguments>);

Note: Only Derive class constructors can call constructors of base class (1st line)

2. Overridden Functions: To call overridden functions syntax is:

super.functionName(<args>);

3. Remaining/Normal Functions: To call functions apart from constructors and overridden functions, syntax is:

functionName(<args>);

Example: Circle

```
import java.util.Scanner;
class Circle extends Point {
        protected float radius;
        public Circle() {
                super(10, 10);
                radius = 10f;
        }
        public Circle(int x,int y) {
                super(x,y);
                radius = 10f;
        }
        public Circle(float radius) {
                super(10, 10);
                this.radius  = radius;
        }
        public Circle(int x,int y,float radius) {
                super(x,y);
```

```
		this.radius = radius;
	}
	public Circle(Point p) {
		super(p.getX(),p.getY());
		radius = 10f;
	}
	public Circle(Point p,float radius) {
		super(p.getX(),p.getY());
		this.radius = radius;
	}
	public float getRadius() {
		return radius;
	}
	public void setRadius(float radius) {
		this.radius = radius;
	}
	@Override
	public void acceptData() {
		super.acceptData();
		Scanner in = new Scanner(System.in);
		System.out.println("Enter Radius : ");
		radius = in.nextFloat();
	}
	@Override
	public void showData() {
		super.showData();
		System.out.println("Radius : " + radius);
	}
	public float getArea() {
		return (float)(Math.PI * Math.pow(radius,2));
	}
	public float getCircumference() {
		return (float)(2 * Math.PI * radius);
	}
	public Point getCenter() {
		int x = getX();
		int y = getY();
		Point p = new Point(x,y);
		return p;
		// return new Point(getX(), getY());
	}
	public static void main(String args[]) {
		Circle c1 = new Circle(10,20,30f);
		Point p = new Point(1,2);
		Circle c2 = new Circle(p,3f);
		c1.showData();
		c2.showData();
		Point p2 = c2.getCenter();
```

```
            p2.showData();
            c2.getCenter().showData();
        }
}
```

static data

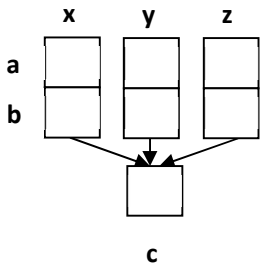
- Any data if declared as static is shared by all the objects.
- Only single copy of static data is created and this copy is owned and maintained by class.
- It is referred as class data rather than object data.
- To access static data, syntax is:

ClassName.dataName

However, it also be accessed using an object

- Benefits:
    - It saves memory.
    - It permits sharing.

```
class Abc {
        int a;
        int b;
        static int c;
}
Abc x = new Abc();
Abc y = new Abc();
Abc z = new Abc();
```



static blocks

- static blocks {} are used to initialize static data of a class.
- These blocks are executed as soon as the class is loaded in memory prior to main.

```
class TestStaticBlock {
        static int a;

        static {
                a = 10;
                System.out.println("A : " + a);
        }
        public static void main(String args[]) {
                a = 20;
                System.out.println("A : " + a);
        }
        static {
                a = 30;
                System.out.println("A : " + a);
        }
}
```

static methods

- Any method that works only and only static data should be declared as static

- To invoke/call a static method, syntax is:

ClassName.methodName(<args>);

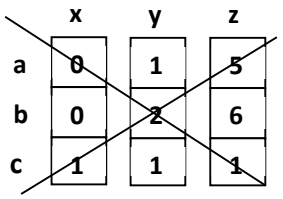
However, it also be invoked using an object

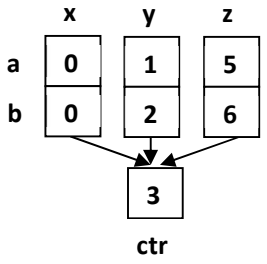
Note: NO OBJECT IS REQUIRED

- Functions that do not work on any member data, they are also declared as static (reusability)

Example: Create a class that counts the number of its objects created

```
import java.util.Scanner;
class MyClassCtr {
        protected int a;
        protected int b;
        protected static int ctr;
        public MyClassCtr() {
                ctr++;
        }
        public MyClassCtr(int a,int b) {
                this.a = a;
                this.b = b;
                ctr++;
        }
        public static int getCtr() {
                return ctr;
        }
        public void getData() {
                Scanner in = new Scanner(System.in);
                System.out.println("Enter A : ");
                a = in.nextInt();
                System.out.println("Enter B : ");
                b = in.nextInt();
        }
        public void showData()  {
                System.out.println("A is : " + a);
                System.out.println("B is : " + b);
        }
        public static void main(String args[]) {
                MyClassCtr x = new MyClassCtr();
                MyClassCtr y = new MyClassCtr(1,2);
                MyClassCtr z = new MyClassCtr(5,6);
                System.out.println("Counter is : " + MyClassCtr.getCtr());
        }
}
```





<u>final data</u>

- Any data if declared as final cannot be changed.
- It is used to declare constants in Java.
- final data if declared within class should be declared as static.

<u>final methods</u>

- Any method if declared as final cannot be overridden.

final class

- Any class if declared as final cannot be inherited.
- It is used to end the hierarchy.

Example: TestCircle

```
import java.util.Scanner;
class TestCircle {
	protected float radius;
	public static final float PI = 3.14f;
	public TestCircle() {
	}
	public TestCircle(float radius) {
		this.radius = radius;
	}
	public float getRadius() {
		return radius;
	}
	public void setRadius(float radius) {
		this.radius = radius;
	}
	public void getData() {
		Scanner a = new Scanner(System.in);
		System.out.println("Enter Radius :");
		radius = a.nextFloat();
	}
	public void showData() {
		System.out.println("radius is : " + radius);
	}
	public final float getArea() {
		return (float)(PI * Math.pow(radius,2));
	}
	public final float getCircumference() {
		return (float)(2 * PI * radius);
	}
	public final static void main(String args[]) {
		TestCircle c1 = new TestCircle();
		c1.getData();
		c1.showData();
		System.out.println("Area is :  " + c1.getArea());
	}
}
```

abstract methods

- If any method is declared as abstract, it should be compulsorily overridden.
- These methods does not contain any statements, they only enforces symmetry in hierarchy.

abstract classes

- If any class contains at least one abstract method, it should be compulsorily declared as abstract.

- class created without abstract methods can also be declared as abstract.
- No object of abstract class can be created; it can only store reference to derive class objects.
- abstract class can have data constructors and methods that are used by derive classes.
- These classes are used to start the hierarchy and define rules for the hierarchy in form of abstract methods.

Example: AbstractA.java

```
abstract class AbstractA {
        public abstract void acceptData();
        public abstract void showData();
}
```

Example: DerivedA.java

```
import java.util.Scanner;
class DerivedA extends AbstractA {
        protected int a;
        public DerivedA(){}
        public DerivedA(int a) {
                this.a = a;
        }
        public void acceptData() {
                Scanner in = new Scanner(System.in);
                System.out.println("Enter A :");
                a = in.nextInt();
        }
        public void showData()  {
                System.out.println("A is : " + a);
        }
        public static void main(String args[]) {
                AbstractA a = new DerivedA(10);
                DerivedA b = new DerivedA(20);
// WRONG        AbstractA c = new AbstractA();
                a.showData();
                b.showData();
        }
}
```

Assignment

Theory Questions

1. Describe OOPS.
2. What are classes? Why do we write a constructor in a class? Does Java provide destructors?
3. What are access specifiers? Explain the types of access specifiers available in Java.
4. What is heritance and what are its types? Explain the benefits of inheritance.
5. Explain differences between function overloading and function overriding.
6. Explain static data, static blocks and static functions.
7. What are abstract functions and abstract classes?
8. Explain why we need final data, final functions and final classes.

Practical Questions

1. Create a class called Student that stores student name, class, section and marks in 4 subjects. Write constructors, getter/setters, behaviour, and operations to accept and display the data. Also provide operations that return total and percentage. Test the class by creating an implementation function (main).
2. Create a class called Employee that stores its name, department, designation and basic salary. Write constructors, getter/setters, behaviour, and operations to accept and display the data. Also write methods that return his incentive. The incentives are HRA (20%), DA (10%), CA (10%).Test the above code by creating an implementation program.
3. Create a class called Rectangle derived from Point class. Apart from data of Point class, Rectangle should store its width and height. Write constructors, getter/setters, behaviour, and operations to accept and display the data. Also write methods that return its area and perimeter. Test the class by creating an implementation program.
4. Create a class called Employee that stores his code and name. Create two derive classes of Employee named TempEmp (temporary employee) and PerEmp (permanent employee). TempEmp should store wage grade and number of days worked whereas PerEmp should store department, designation and basic salary. Write constructors, getter/setters and appropriate operations.
5. Create a class that counts the number of its object created. If the objects counter it less than 5, it should display a message “Too Less”, when it is equal to 5 “Will do” should be displayed and if more that 5 objects are created it should display “Exceeding the Limits”. Write suitable constructors and methods. Also write a method that returns the number of objects created.

Day 4: enum and interface

- enum
- interface

enum

- an enumeration is a list of named constants
- enum was introduced with jdk5
- enum is used to store one out of multiple pre-defined set of values
- for each enum, internally Java creates a cass file
- each constant declared within enum is declared as public static final object of the class within the class created

Example:

```
public enum Colors { RED, GREEN, BLUE };
class Test1 {
	public static void main(String args[]) {
		Colors color = Colors.RED;
		System.out.println(color);
	}
}
```

- How to declare and use enum
- enum can be declared within class or outside class
- if enum is declared outside class and is public, enum name and filename should be same
- enum makes use of proper case

Example:

```
class Test2 {
	enum Colors {
		RED("#FF0000"), GREEN("#00FF00"), BLUE("#0000FF");
		Colors(String hexCode) {
			this.hexCode = hexCode;
		}
		public String getHexCode() {
			return hexCode;
		}
		private String hexCode;
	}
	public static void main(String args[]) {
		Colors color = Colors.RED;
		System.out.println(color);
		System.out.println(color.getHexCode());
	}
}
```

- enum can have data, constructors and methods
- options declared within enum should be declared within the first line
- enum cannot have public constructors

Example:

```
class Test3 {
	enum Colors { RED, GREEN, BLUE }
	public static void main(String args[]) {
		for(Colors c : Colors.values())
			System.out.println(c + " : " + c.ordinal());

		System.out.println(Colors.valueOf("RED"));
		System.out.println(Colors.valueOf("YELLOW"));
	}
}
```

- valueOf(), values() and ordinal()

Example:

```
class Test4 {
	enum Colors {RED, GREEN, BLUE};
	public static void main(String args[]) {
		Colors color = Colors.RED;
		if(color == Colors.RED)
			System.out.println("This is Red Color.");
		else
			System.out.println("This is not the Red Color.");

		if(color.equals(Colors.RED))
			System.out.println("This is Red Color.");
		else
			System.out.println("This is not the Red Color.");

		switch(color) {
			case RED : System.out.println("This is Red Color.");
				break;
			case GREEN : System.out.println("This is Green Color.");
				break;
			case BLUE : System.out.println("This is Blue Color.");
				break;
		}
	}
}
```

- equals(), ==, switch works with enum

Example:

```
import java.util.Scanner;
enum Gender {
	Male, Female;
	public static Gender acceptGender() {
		Scanner in = new Scanner(System.in);
		String str = in.nextLine();
		if(str.trim().toLowerCase().equals("male"))
			return Male;
```

```
                else
                        return Female;
        }
};
class Person {
        protected String name;
        protected int age;
        protected Gender gender;
        public Person() {
                name = new String();
                gender = Gender.Male;
        }
        public Person(String name, int age , Gender gender) {
                this.name = name;
                this.age = age;
                this.gender = gender;
        }
        public void acceptData() {
                Scanner in = new Scanner(System.in);
                System.out.println("Enter Name : ");
                name = in.nextLine();
                System.out.println("Enter Age : ");
                age = in.nextInt();
                System.out.println("Enter Gender : ");
                gender = Gender.acceptGender();
        }
        public void showData() {
                System.out.println("Name : " + name);
                System.out.println("Age : " + age);
                System.out.println("Gender : " + gender);
        }
        public static void main(String args[]) {
                Person p = new Person();
                p.acceptData();
                p.showData();
        }
}
```

Interface

- Interface is a collection of method declarations/prototype
- If any class implements an interface, it provides definition for all the methods declared within interface.
- A class can implement, one or more interfaces.
- Apart from interface methods class can also have its own method.
- An interface can inherit one or more interfaces.
- By default all methods declared within interface are public and abstract.
- By default all data declared within interface is public, static and final.

```
interface Ia {
        void a();                       //public abstract void a();
```

```
	int x = 10;				//public static final int x = 10;
}

class Ca implements Ia {
	public void a(){....}
}

interface Ib {
	void b();
}

class Cab implements Ia,Ib {
	public void a(){..}
	public void b(){..}
	public void c(){..}
}
```

NOTE:

- Interfaces make use of ProperCase
- Interface name and filename should be same

Example: **EmpInterface.java**

```
interface EmpInterface {
	void showData();
	void acceptData();
}
```

Example: **Employee.java**

```
import java.util.Scanner;
class Employee implements EmpInterface
	protected int code;
	protected String name;
	protected String dept;
	protected String desig;
	protected float basic;
	public Employee() {
		name = new String();
		dept = new String();
		desig = new String();
	}
	public void acceptData() {
		Scanner a = new Scanner(System.in);
		System.out.println("Enter Code : ");
		code = a.nextInt();
		a.nextLine();
		System.out.println("Enter name : ");
		name = a.nextLine();
		System.out.println("Enter Department : ");
		dept = a.nextLine();
```

```
            System.out.println("Enter Designation : ");
            desig = a.nextLine();
            System.out.println("Enter Basic : ");
            basic= a.nextFloat();
        }
        public void showData() {
            System.out.println("Code:  " + code);
            System.out.println("Name:  " + name);
            System.out.println("Department:  " + dept);
            System.out.println("Designation:  " + desig);
            System.out.println("Basic: " + basic);
        }
        public float getHra() {
            return 0.2f * basic;
        }
        public static void main(String args[])
            Employee e = new Employee();
            e.getData();
            e.showData();
            System.out.println("HRA is : " + e.getHra());
            EmpInterface e1 = new Employee();
            e1.getData();
            e1.showData();
            // WRONG e1.getHra();
        }
}
```

Default Interface Methods

- The release of JDK 8 has changed this by adding a new capability to interface called the default method (also referred as extension method).
- A default method lets you define a default implementation for an interface method. In other words, by use of a default method, it is now possible for an interface method to provide a body, rather than being abstract.

```
interface Stack {
        void push(int element);
        void pop();
        default void clear() {
            System.out.println("Clear Implementation not provided");
        }
}

public class StackImpl implements Stack {
        private static final int capacity = 3;
        int arr[] = new int[capacity];
        int top = -1;
        public void push(int pushedElement) {
            if (top < capacity - 1) {
                    top++;
                    arr[top] = pushedElement;
```

```
                System.out.println("Element " + pushedElement + " is pushed to Stack !");
                printElements();
            } else {
                System.out.println("Stack Overflow !");
            }
        }
        public void pop() {
            if (top >= 0) {
                top--;
                System.out.println("Pop operation done !");
            } else {
                System.out.println("Stack Underflow !");
            }
        }
        public void printElements() {
            if (top >= 0) {
                System.out.println("Elements in stack :");
                for (int i = 0; i <= top; i++) {
                    System.out.println(arr[i]);
                }
            }
        }
//      public void clear() {
//          arr = new int[capacity];
//          top  = -1;
//      }
        public static void main(String[] args) {
            StackImpl stackDemo = new StackImpl();
            stackDemo.pop();
            stackDemo.push(23);
            stackDemo.push(2);
            stackDemo.push(73);
            stackDemo.push(21);
            stackDemo.pop();
            stackDemo.pop();
            stackDemo.pop();
            stackDemo.pop();
        }
}
```