2. CLASSES, CONSTRUCTORS AND METHODS

1. Class:

A class is the collection of data members and member functions. It is a blue print from which objects can be constructred. It is just a logical construct. A class can contain fields and methods to describe the state and behavior of an object respectively. Methods are the members of a class that provide a service for an object or perform some business logic. Current states of a class's corresponding object are stored in the object's instance variables. Methods define the operations that can be performed in java programming. **Java is case-sensitive language**. That is Java fields and member functions names are case sensitive.

A sample of a class is given below:

```
public class Rectangle
{
        int length;
        int breadth;

      public  void area(){
            // code
              }

      public  void perimeter(){
            // code
       }
 }
```

class Rectangle represents the states and behaviour of Rectangle.

2. Object:

Object is known as instance of class. Objects actually have their own states and behavour. For example Rectangle objects have their own length, breadth, area and perimeter. Hence the length and breadth are known as instance variables and methods area and perimeter are known as instance methods. Classes are just a logical construct but Objects are actually loaded into memory. Object are physically present.
In java objects are created dynamically using new keyword.

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' key word is used to create the object.
- **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example of creating an object is given below:

```
public class Rectangle {
	int length;
	int breadth;

	public Rectangle() {
		length=34;
		breadth=35;
	}

	public Rectangle(int length, int breadth) {
		this.length=length;
		this.breadth=breadth;
	}

	public static void main(String args[]) {
		Rectangle rect=new Rectangle() ;	//  call will go to zero constructor
		Rectangle rect1=new Rectangle(23,56);   //  call will go to parameterized constructor
		System.out.println("rect object length and breath is: "+rect.length+" "+rect.breadth);
		System.out.println("rect1 object length and breath is: "+rect1.length+"
"+rect1.breadth);
	}
}
```

In above program, when new statement is get executed, call goes to constructor and is returns the object by initializing it. Thus memory to the objects is dynamically allocated at the time of execution of byte code. By using multiple new statements, we can create multiple objects of the same class.

3. Constructor:

Every class has a constructor. Constructors are required to create objects for a class. Constructors are used to initialize the instance variables of an object. It has a same name as a class name.

Each time a new object is created, at least one constructor will be invoked. If we do not explicitly write a constructor for a class the Java compiler builds a default constructor for that class. A class can have more than one constructor.In java, constructors are implicitly invoked at the time of object creation using new keyword. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Example of a constructor is given below:

```
public class Rectanle{
	int length;
	int breadth;

	public Rectangle() {
			// code for initialization
		}
```

```
    public Rectangle(int length) {      // This constructor has one parameter, length
                // code for initialization
        }
}
```

2. 1 Properties of Constructor:

- Constructor name must be same as its class name
- Constructors used to initialize the data members
- Constructor must have no explicit return type as it implicitly returns object
- Constructors can be declared private. In this case we can not access this from outside of class.
- Constructors can be overloaded.

2.2 Types of java constructors

There are two types of constructors:

- Non-parameterized Constructor also know as Zero constructor
- Parameterized Constructor

1. Zero constructor:

A constructor that have no parameter is known as zero or non parameterized constructor.

Example :

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
public class Rectangle
{
        public Rectangle() {
        System.out.println("Rectangle object initialization");
        }

        public static void main(String args[]) {
                Rectangle rect=new Rectangle()
        }
}
```

Output: Rectangle object initialization

2. Parameterized constructor

A constructor that have parameters is known as parameterized constructor. Parameterized constructor is used to provide different values to the distinct objects.

Example :

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
public class Rectangle {
	int length;
	int breadth;

	public  Rectangle(int length, int breadth) {
		System.out.println("This is parameterized constructor");
	}

	public static void main(String args[]) {
		Rectangle rect=new Rectangle(23,56) ;
	}
}
```

Note: When there is no constructor in class. JVM provides default contructor whose syntax is same as zero contructor.

2.3 Constructor Overloading:

It is one kind of compile time polymorphism where the same constructor name but different type signatures are there. Constructor overloading is used to initialize the objects in different ways. The call to matching constructor is decided at compile time.

Example :

```
public class Rectangle {
	int length;
	int breadth;

	public Rectangle() {
		length=34;
		breadth=35;
	}

	public Rectangle(int length, int breadth) {
		this.length=length;
		this.breadth=breadth;
	}

	public static void main(String args[]) {
		Rectangle rect=new Rectangle() ;      //  call will go to zero constructor
		Rectangle rect1=new Rectangle(23,56);   //  call will go to parameterized constructor
		System.out.println("rect object length and breath is: "+rect.length+" "+rect.breadth);
		System.out.println("rect1 object length and breath is: "+rect1.length+"
"+rect1.breadth);
	}
```

```
}
```

Output:

rect object length and breath is: 34 35
rect1 object length and breath is: 23 56

Note:

Problem of Instance Variable Hiding: If the local variable name and instance variable name is same then bydefault local variable hides the instance variable in local scope. Thus any assignment from local to instance can not be done.

Solution: differentiate between both variables by referring the current instance variable by using **this** keyword.

```
this.length=length;
this.breadth=breadth;
```

4. Method:

The method of a class contains the code which operates on data. It is part where all logic is written. Methods can be static or not static. Non static methods have own copy for every object of class. Static methods are class methods which have single shared copy.

Syntax:

```
accessmodifer nonaccessmodifer returntype methodname(parameters)
{
        // code of the method
}
```

Example:

```
public void demo() {
        System.out.println("This is a demo method");
}
```

Difference between constructor and method in java:

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type. But is by deafault returns object	Method must have return type or void.
Constructor is invoked implicitly, when new statement is executed	Method is invoked explicitly.

The javaa default constructor if you don't have any constructor.	Method is not provided by java in any case.
Constructor name must be same as the class name.	Method name may not be same as class name.

5. Accessing Instance Variables and Methods:

Instance variables and methods are accessed via created objects. To access an instance variable the fully qualified path should be as follows:

- Create an object of the class
- using object, access the instance variables of that class
- using object, call the instance variables of that class

Example:
This example explains how to access instance variables and methods of a class:

```
public class Rectangle {

    int length;
    int breadth;

        public Rectangle() {
                length=34;
                breadth=35;
        }

        public Rectangle(int length, int breadth) {
                this.length=length;
                this.breadth=breadth;
        }

         public int area() {
                int ar=length*breadth;
                return ar;
         }

        public static void main(String args[]) {

        Rectangle rect=new Rectangle() ;      // this call will go to zero constructor
        Rectangle rect1=new Rectangle(23,56);   // this call will go to parameterized constructor
        System.out.println("rect object length and breath is: "+rect.length+" "+rect.breadth);
        System.out.println("rect1 bject length and breath is: "+rect1.length+" "+rect1.breadth);

                int area1=rect.area();     // calling instance method
                System.out.println("Area of rect :"+area1);
```

```
            int area2=rect1.area();                    // calling instance method
            System.out.println("Area of rect1 :"+area2);
        }
}
```

Output:
rect object length and breath is: 34 35
rect1 bject length and breath is: 23 56
Areae of rect :1190
Areae of rect1 :1288

6. Use of this keyword:

this keyword is used to refer current object. There are several uses of this

- this can be used to refer constructor : generally constrcutors are called explicitly using this keyword to reinitialize the current object.
  Syntax: **this**(parameter1, parameter2,.....);

- this can be used to refer current instance variable : generally it used to differentiate between local variable and instance variable also between current instance variable and other object instance variable.
  Syntax: **this**.variable name

- this can be used to refer current instance method
  Syntax: **this**.method name

Example: use of this keyword for object reinitialization

```
public class Rectangle {
   int length;
   int breadth;

        public Rectangle() {
            this(34,35);
        }

        public Rectangle(int length, int breadth) {
            this.length=length;
            this.breadth=breadth;
        }

         public int area() {
            int ar=length*breadth;
            return ar;
         }
```

```
        public static void main(String args[]) {
        Rectangle rect=new Rectangle() ;      // this call will go to zero constructor
        Rectangle rect1=new Rectangle(23,56);   // this call will go to parameterized constructor
        System.out.println("rect object length and breath is: "+rect.length+" "+rect.breadth);
        System.out.println("rect1 bject length and breath is: "+rect1.length+" "+rect1.breadth);

                int area1=rect.area();     // calling instance method
                System.out.println("Area of rect :"+area1);
                int area2=rect1.area();                  // calling instance method
                System.out.println("Area of rect1 :"+area2);
        }
}
```

Output:

rect object length and breath is: 34 35
rect1 bject length and breath is: 23 56
Area of rect :1190
Area of rect1 :1288

7. Parameter Passing :

The parameters can be passed by either

- Call by value
- Call by reference

7.1 Passing Parameters by Value:

While working under calling process, arguments is to be passed. These should be in the same order as their respective parameters in the method specification.
Passing Parameters by Value means calling a method with a parameter. Through this the argument value is passed to the parameter. Example is shown above.

7.2 Passing objects:

In java, objects are implicitly passed by 'call by reference',but no need of & operator.

8. Java instanceOf operator:

The Java instanceof operator is used to test whether the object is an instance specified type (class or subclass or interface).
The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

Example:

Following is the simple example of instance operator where it tests the current class.

```
public class Simple {

        public static void main(String args[]) {
                Simple s=new Simple();
                System.out.println(s instanceof Simple);//true
               }
 }
```

Output: true

instanceof with reference variable :
An object of subclass type is also a type of parent class. For example, if Dog extends Animal then object of Dog can be referred by either Dog or Animal class.

Example:

```
class Animal{}

public class Dog extends Animal{     //Dog inherits Animal

        public static void main(String args[]){
                Animal d=new Dog();

                System.out.println(d instanceof Animal); //true}
        }
}
```

Output: true

instanceof with a variable that have null value :
If we apply instanceof operator with a variable that have null value, it returns false.

Example:

```
public class Simple {

        public static void main(String args[]) {
                Simple s=null;
                System.out.println(s instanceof Simple); //false
        }
 }
```

Output: false

9. Polymorphism:

Polymorphism in java is a concept by which we can perform a similar type of action by different ways. The polymorphism means many forms.

9.1 Types of polymorphism:

There are two types of polymorphism in java:

1. compile time polymorphism
2. runtime polymorphism.

1. Compile time polymorphism:

Method Overloading:

If in a class there are methods with same name but different type signatures i.e. either number of parameters or data types or both, then at compile time compiler decides which method will get called according to matching signatures. This is called method overloading. This is compile time polymorphism.

Example: to calculate areas of different shapes we can give name of the method as area and we can pass different parameters like radius, length, breadth, side to the appropriate methods.

```
public class AreaMethodOverload {

	public double area(double r) {
		double ar=3.14*r*r;
		return ar;
	}
	public void area(int length, double breadth){
		double ar=length*breadth;
		System.out.println("Area of rectangle:"+ar);
	}
	public double area(double side, boolean flag) {
		double ar=side*side;
		return ar;
	}

	public static void main(String args[]) {
		AreaMethodOverload am=new AreaMethodOverload();

	 double ar=am.area(43);
	 System.out.println("Area of circle: "+ar);

	am.area(45,67.6);

	ar=am.area(34,true);
	System.out.println("Area of square: "+34);
```

```
    }
}
```

If only returntype is different and method name and typesignatures are same then this will not be considered as method overloading as compiler does not check the return type it only checks the type signatures and decides which method will get called. Thus only return type is not sufficient for the mehtod overlading. Type signatures are important.

Note: The returntype and acccess modifiers do not matter in method overloading.

2. Runtime Polymorphism:

1. Method Overriding:

It comes under inheritance. If a subclass contains same method name and type signatures as in super class then by default the subclass method overrides superclass same method. That means super class method gets overridden. It is decided at run time.

2. Dynamic Method Dispatch :

Here we use one concept: super class variable can refer subclass object. We can call the methods of super class and subclass dynamically by changing the type of object which is referred by superclass reference variable. The call to the method is decided at run time.
The example is shown in chapter inheritance.

10. Static Binding and Dynamic Binding:

Bindind the reference and type of object refered together is known as Binding.
There are two types of binding

1. static binding (also known as early binding).
2. dynamic binding (also known as late binding).

10.1 Static binding

When type of the object is determined at compiled time(by the compiler), it is known as static binding.
If there is any private, final or static method in a class are statically bounded. Because compiler does not have any difficulty to determine object of the class.

Example:

```
class Shape
{
        private void dimension() {
                //code
        }
        static void property() {
```

```
            // code
        }
        public static void main(String[] args) {
            Shape sh=new Shape();    //static binding
            sh.dimension();
            Shape.property();     // statically bounded
        }
}
```

10.2 Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

Example:

```
class Shape
{
        public  void dimension() {
            //code
        }
        static void property() {
            // code
        }
}

class Rectangle extends Shape
{
        public  void dimension(){
            System.out.println("Rectangle Dimension");
        }
        public static void main(String[] args) {
            Shape.property();

            Shape sh1=new Rectangle();  // dynamic binding
            sh1.dimension();
        }
}
```

11. Instance Initializer block:

This is also known as annonymous block. Instance initializer block is a mechanism provided by java compiler to define a group of statements common to all constructors at a single place. At the compilation time, compiler moves these statements at the beginning of all constructors after super. It can also be used to initialize the instance variable.

Example:

```
class Display {
```

```
    int a, b;

    //Anonymous or instance initializer Block
    {
        System.out.println("AnonumousBlock called.");
        a = 10;
    }

   //default constructor
   Display(){
     System.out.println("default constructor called.");
   }

   //one argument constructor
   Display(int num){
     System.out.println("one parameter constructor called.");
     b = num;
   }

   //method to display values
   public void display(){
         System.out.println("a = " + a);
         System.out.println("b = " + b);
   }
}

public class AnnonymousBlockDemo {

          public static void main(String args[]){
        Display obj1 = new Display();
        obj1.display();

        Display obj2 = new Display(20);
        obj2.display();
  }
}
```

Output:

AnonumousBlock called.
default constructor called.
a = 10
b = 0

AnonumousBlock called.
one parameter constructor called.
a = 10
b = 20

If two Anonymous Blocks are used then they will execute in the same order in which they are appear.

Example:

```
class Display {
   int a, b;
   //first Anonymous or instance initializer Block
   {
      System.out.println("First AnonumousBlock called.");
      a = 10;
   }

   //Second Anonymous or instance initializer Block
   {
      System.out.println("Second AnonumousBlock called.");
      b = 20;
   }

  //default constructor
  Display(){
    System.out.println("default constructor called.");
  }

  //one argument constructor
  Display(int num){
    System.out.println("one parameter constructor called.");
    b = num;
  }

  //method to display values
  public void display(){
        System.out.println("a = " + a);
        System.out.println("b = " + b);
  }
}

public class AnnonymousBlockDemo {
```

```
        public static void main(String args[]){
        Display obj1 = new Display();
        obj1.display();

        Display obj2 = new Display(20);
        obj2.display();
  }
}
```

Output:

```
First AnonumousBlock called.
Second AnonumousBlock called.
default constructor called.
a = 10
b = 20
First AnonumousBlock called.
Second AnonumousBlock called.
one parameter constructor called.
a = 10
b = 20
```

If static and non-static Anonymous Blocks are used then static Anonymous Block is executed only once.

```
class Display {
   int a, b;
   //static  Block
   static {
         System.out.println("Static AnonumousBlock called.\n");
   }
   //first Anonymous or instance initializer Block
   {
      System.out.println("First AnonumousBlock called.");
      a = 10;
   }

   //Second Anonymous or instance initializer Block
   {
      System.out.println("Second AnonumousBlock called.");
      b = 20;
   }
  //default constructor
  Display(){
```

```
    System.out.println("default constructor called.");
  }
  //one argument constructor
  Display(int num){
    System.out.println("one parameter constructor called.");
    b = num;
  }
  //method to display values
  public void display(){
        System.out.println("a = " + a);
        System.out.println("b = " + b);
  }
}

public class AnnonymousBlockDemo {
        public static void main(String args[]){
      Display obj1 = new Display();
      obj1.display();

      Display obj2 = new Display(20);
      obj2.display();
 }
}
```

Output:

```
Static AnonumousBlock called.
First AnonumousBlock called.
Second AnonumousBlock called.
default constructor called.
a = 10public class AnnonymousBlockDemo {
        public static void main(String args[]){
      Display obj1 = new Display();
      obj1.display();

      Display obj2 = new Display(20);
      obj2.display();
 }
}
```

Output:

```
First AnonumousBlock called.
Second AnonumousBlock called.
default constructor called.
```

```
a = 10
b = 20
First AnonumousBlock called.
Second AnonumousBlock called.
one parameter constructor called.
a = 10
b = 20
```

If static and non-static Anonymous Blocks are used then static Anonymous Block is executed only once.

```
class Display {
   int a, b;
   //static  Block
   static {
         System.out.println("Static AnonumousBlock called.\n");
   }
   //first Anonymous or instance initializer Block
   {
      System.out.println("First AnonumousBlock called.");
      a = 10;
   }

   //Second Anonymous or instance initializer Block
   {
      System.out.println("Second AnonumousBlock called.");
      b = 20;
   }
  //default constructor
  Display(){
    System.out.println("default constructor called.");
  }
  //one argument constructor
  Display(int num){
    System.out.println("one parameter constructor called.");
    b = num;
  }
  //method to display values
  public void display(){
        System.out.println("a = " + a);
        System.out.println("b = " + b);
  }
}
```

```
public class AnnonymousBlockDemo {
        public static void main(String args[]){
      Display obj1 = new Display();
      obj1.display();

      Display obj2 = new Display(20);
      obj2.display();
  }
}
```

Output:

```
Static AnonumousBlock called.
First AnonumousBlock called.
Second AnonumousBlock called.
default constructor called.
a = 10
b = 20
First AnonumousBlock called.
Second AnonumousBlock called.
one parameter constructor called.
a = 10
b = 20
```

ASSIGNMENTS

1. Define the class Vehical and describe the states and behaviour of Vehical.
2. Overload the constructor of Vehical class by passing different parameters.
3. Determine the speedUp of Vehical which represents behaviour of Vehical and which can get increased.
4. Define the common states and behaviours for the Vehical.
5. Define constants in for Vehical i.e. the states those are fix for all Vehicals.
6. Show the parameter passing in Vehical.
7. Use in instaceOf operator in Vehical.
8. Overload the method fuelCapacity by passing the capacity in terms of litres and mililitres.
9. Show the use of instance initializer block in Vehical class.
10. Implement the object array of User accounts. Perform following actions by taking user input
    1. Adding new account
    2. Updating existing account
    3. Deleting existing account

4. Searching particular account
5. Displaying all accounts
6. Depositing money in particular account
7. Withdrawing money from particular account

b = 20
First AnonumousBlock called.
Second AnonumousBlock called.
one parameter constructor called.
a = 10
b = 20

ASSIGNMENTS

11. Define the class Vehical and describe the states and behaviour of Vehical.
12. Overload the constructor of Vehical class by passing different parameters.
13. Determine the speedUp of Vehical which represents behaviour of Vehical and which can get increased.
14. Define the common states and behaviours for the Vehical.
15. Define constants in for Vehical i.e. the states those are fix for all Vehicals.
16. Show the parameter passing in Vehical.
17. Use in instaceOf operator in Vehical.
18. Overload the method fuelCapacity by passing the capacity in terms of litres and mililitres.
19. Show the use of instance initializer block in Vehical class.
20. Implement the object array of User accounts. Perform following actions by taking user input
    1. Adding new account
    2. Updating existing account
    3. Deleting existing account
    4. Searching particular account
    5. Displaying all accounts
    6. Depositing money in particular account
    7. Withdrawing money from particular account