# 12TH JUNE ASSIGNMENT

1. **What is a Constructor?**

**Ans:** A constructor in Java is a special method that is used to initialize objects of a class. It has the same name as the class and does not have a return type. The purpose of a constructor is to set the initial state of an object when it is created.

2. **What is Constructor Chaining?**

**Ans:** Constructor chaining refers to the process of calling one constructor from another constructor in the same class or in the superclass. This allows the initialization tasks to be shared among different constructors, reducing code duplication. It is achieved using the "this" keyword to invoke another constructor within the same class, or using the "super" keyword to invoke a constructor in the superclass.

**3. Can we call a subclass constructor from a superclass constructor?**

**Ans:** No, you cannot directly call a subclass constructor from a superclass constructor. The constructor call must always be the first statement in a constructor, and it can either call another constructor in the same class using "this" or call a constructor in the superclass using "super". Subclass constructors implicitly call the superclass constructor as the first step of their execution.

**4. What happens if you keep a return type for a constructor?**

**Ans:** Constructors do not have a return type. If you mistakenly include a return type for a constructor, it will be treated as a regular method and will not be considered a constructor. The compiler will report an error, and the code will fail to compile.

**5. What is No-arg constructor?**

**Ans:** A no-arg constructor is a constructor that takes no arguments. It does not have any parameters and can be called without providing any values. It is also known as a default constructor because if a class does not explicitly define any constructors, a default no-arg constructor is automatically provided by the compiler.

**6. How is a No-argument constructor different from the default Constructor?**

**Ans:** A no-argument constructor and a default constructor are essentially the same thing. Both refer to a constructor that takes no arguments. The term "no-argument constructor" emphasizes the absence of parameters, while "default constructor" emphasizes the automatic provision by the compiler when no constructors are explicitly defined in a class.

**7. When do we need Constructor Overloading?**

**Ans:** Constructor overloading is used when we want to create multiple constructors in a class with different parameter lists. It allows objects to be initialized in different ways, depending on the arguments passed to the constructor. Constructor overloading is useful when we want to provide flexibility and convenience to the users of our class by allowing them to choose different initialization options.

**8. What is Default constructor Explain with an Example**

**Ans:** A default constructor is a constructor that is automatically provided by the compiler if a class does not have any explicitly defined constructors. It is a no-arg constructor that initializes the object with default values or performs no initialization. Here's an example:

```
public class MyClass {
    // Default constructor
    public MyClass() {
        // Initialization code (if any)
    }
}
```

**9. What is Encapsulation in Java? Why is it called Data hiding?**

**Ans:** Encapsulation in Java is a mechanism of wrapping data (variables) and methods together into a single unit, called a class. It provides data hiding by declaring the variables as private and providing public methods (getters and setters) to access and modify the data. Encapsulation is called data hiding because it hides the internal details of the class and prevents direct access to the variables, allowing controlled access through methods.

**10. What are the important features of Encapsulation?**

**Ans:** The important features of encapsulation are:

Data hiding: Encapsulation hides the internal representation of an object and provides controlled access to it.

Access control: Encapsulation allows you to control the visibility of variables and methods by using access modifiers (e.g., public, private, protected).

Code reusability: Encapsulation promotes code reusability by creating reusable classes with well-defined interfaces.

Maintainability: Encapsulation makes it easier to maintain and modify the internal implementation of a class without affecting other parts of the code.

**11. What are getter and setter methods in Java Explain with an example**

**Ans:** Getter and setter methods, also known as accessors and mutators, are used to access and modify the values of private variables in a class, respectively. They are part of the encapsulation mechanism in Java. Here's an example:

```
public class Person {
    private String name;

    // Getter method
    public String getName() {
        return name;
```

```
    }

    // Setter method
    public void setName(String name) {
        this.name = name;
    }
}

// Usage:
Person person = new Person();
person.setName("John");
System.out.println(person.getName()); // Output: John
```

**12. What is the use of this keyword explain with an example**

**Ans:** The "this" keyword in Java is a reference to the current object. It is used to refer to the instance variables and methods of the current object. It is often used to disambiguate between instance variables and parameters with the same name. Here's an example:

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name; // "this.name" refers to the instance variable
    }
}
```

In the example above, "this.name" is used to assign the value of the parameter "name" to the instance variable "name".

**13. What is the advantage of Encapsulation?**

**Ans:** The advantage of encapsulation is that it provides data hiding and protects the internal state of an object from direct access. By encapsulating data and exposing only necessary methods, encapsulation helps maintain the integrity of the object and prevents unauthorized access or modification. It also allows the internal implementation of a class to be changed without affecting other parts of the code that use the class.

**14. How to achieve encapsulation in Java? Give an example.**

**Ans:** To achieve encapsulation in Java, you need to declare the variables of a class as private and provide public getter and setter methods to access and modify those variables. Here's an example:

```
public class Circle {
    private double radius;

    // Getter method
    public double getRadius() {
        return radius;
    }

    // Setter method
    public void setRadius(double radius) {
        if (radius > 0) {
            this.radius = radius;
        }
    }
}

// Usage:
Circle circle = new Circle();
circle.setRadius(5.0);
System.out.println(circle.getRadius()); // Output: 5.0
```

In the example above, the "radius" variable is declared as private, so it cannot be accessed directly from outside the class. The getter method "getRadius()" provides controlled access to the value of the radius, and the setter method "setRadius()" allows the radius to be modified only if the provided value is valid.

**15. Create a class that keeps track of the number of instances created. Implement a static variable and method to accomplish this.**

**Ans:** Here's an example of a class that keeps track of the number of instances created using a static variable and method:

```
public class InstanceCounter {
    private static int count = 0;

    public InstanceCounter() {
        count++; // Increment count when an instance is created
    }
```

```
    public static int getCount() {
        return count; // Return the number of instances created
    }
}

// Usage:
InstanceCounter obj1 = new InstanceCounter();
InstanceCounter obj2 = new InstanceCounter();
System.out.println(InstanceCounter.getCount()); // Output: 2
```

In the example above, the static variable "count" is incremented in the constructor every time a new instance of the class is created. The static method "getCount()" returns the current value of the count, which represents the number of instances created.

**16. Write a program and create a constructor with parameters and initialise the variable using a constructor.**

**Ans:** Here's an example of a constructor with parameters that initializes the variables:

```
public class Rectangle {
    private int width;
    private int height;

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    // Other methods...
}

// Usage:
Rectangle rectangle = new Rectangle(10, 5);
```

In the example above, the constructor of the Rectangle class takes two parameters, width and height. The values provided when creating a Rectangle object are used to initialize the corresponding instance variables.

**17. Use a private keyword for a variable and use setter and getter methods to initialise and print the values.**

**Ans:** Here's an example of using the private keyword for a variable and using setter and getter methods to initialize and print the values:

```
public class Person {
    private String name;

    // Setter method
    public void setName(String name) {
        this.name = name;
    }

    // Getter method
    public String getName() {
        return name;
    }
}

// Usage:
Person person = new Person();
person.setName("John");
System.out.println(person.getName()); // Output: John
```

In the example above, the name variable is declared as private, so it cannot be accessed directly from outside the Person class. The setName() method is used to set the value of name, and the getName() method is used to retrieve the value.

**18. Write a program to call an method without creating an object of a class**

**Ans:** You can call a method without creating an object of a class if the method is declared as static. Static methods belong to the class itself rather than individual instances, so they can be called directly using the class name. Here's an example:

```
public class MathUtils {
    public static int add(int a, int b) {
        return a + b;
    }
}
```

```
// Usage:
int result = MathUtils.add(5, 3);
System.out.println(result); // Output: 8
```

In the example above, the add() method in the MathUtils class is declared as static. It can be called using the class name without creating an object of the class.

**19. Write a program which has static block and constructor overloading, initialise variables using constructors and print it.**

**Ans:** Here's an example of a program that includes a static block, constructor overloading, initializes variables using constructors, and prints the values:

```
public class MyClass {
    private int value;

    // Static block
    static {
        System.out.println("Static block");
    }

    // Constructor with parameter
    public MyClass(int value) {
        this.value = value;
    }

    // Default constructor
    public MyClass() {
        this(0); // Call the constructor with parameter
    }

    // Method to print value
    public void printValue() {
        System.out.println("Value: " + value);
    }
}
```

```
// Usage:
MyClass obj1 = new MyClass();
obj1.printValue(); // Output: Value: 0

MyClass obj2 = new MyClass(5);
obj2.printValue(); // Output: Value: 5
```

In the example above, the static block is executed when the class is loaded. The MyClass class has two constructors: a constructor with a parameter and a default constructor. The default constructor calls the constructor with a parameter using this() to initialize the value variable. The printValue() method is used to print the value of the value variable.

**20. Why do we need static keyword in Java Explain with an example?**

**Ans:** The static keyword in Java is used to declare members (variables and methods) that belong to the class itself, rather than individual instances of the class. Here's an example to explain its usage:

```
public class Circle {
    private static double pi = 3.14159;
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public static double getPi() {
        return pi;
    }

    public double getArea() {
        return pi * radius * radius;
    }
}

// Usage:
Circle circle = new Circle(5.0);
```

System.out.println(Circle.getPi()); // Output: 3.14159

System.out.println(circle.getArea()); // Output: 78.53975

In the example above, the pi variable is declared as static, meaning it belongs to the class Circle itself and is shared among all instances. The getPi() method is also declared as static because it operates on the class level rather than individual objects. The radius variable, on the other hand, is an instance variable and has different values for each instance of the Circle class.

**21. What is class loading and how does the Java program actually executes?**

**Ans:** Class loading is the process of loading a Java class file into memory so that it can be executed by the Java Virtual Machine (JVM). When a Java program is executed, the JVM performs three main steps: loading, linking, and initializing. During the loading phase, the JVM locates and reads the bytecode of the class and creates a representation of it in memory. The linking phase involves verifying and preparing the class for execution. The initialization phase initializes static variables and executes static initialization blocks.

**22. Can we mark a local variable as static**

**Ans:** No, a local variable cannot be marked as static. The static keyword is used to declare members of a class (variables and methods) that belong to the class itself, not to individual instances. Local variables are specific to a particular method or block and have a limited scope within that context.

**23. Why is the static block executed before the main method in java?**

**Ans:** The static block is executed before the main method in Java because it is part of the class loading process. When a class is loaded into memory, the static block is executed to perform any necessary initialization tasks for the class, such as initializing static variables. Once the static block completes, the main method is invoked to start the execution of the program.

**24. Why is a static method also called a class method?**

**Ans:** A static method is also called a class method because it belongs to the class itself, not to individual instances of the class. It can be invoked using the class name, without the need to create an object of the class. Static methods can only directly access other static members (variables or methods) of the class and cannot access instance variables or methods without a reference to an instance of the class.

**25. What is the use of static blocks in java?**

**Ans:** Static blocks in Java are used to initialize static variables or perform other one-time initialization tasks for a class. They are executed when the class is loaded into memory, before any static methods or the main method are called. Static blocks are particularly useful when you need to perform some initialization that is common to all instances of the class.

**26. Difference between Static and Instance variables**

**Ans:** The main difference between static and instance variables is their association with the class and instances, respectively:

Static variables: Also known as class variables, static variables belong to the class itself, not to individual instances. They are shared among all instances of the class and are accessed using

the class name. Changes to static variables are visible across all instances. They are declared using the static keyword.

Instance variables: Instance variables belong to individual instances of a class. Each instance has its own copy of the instance variables. They are accessed using the instance name and can have different values for each instance. Instance variables are declared without the static keyword.

**27. Difference between static and non static members**

**Ans:** The main difference between static and non-static members (variables or methods) is their association with the class and instances, respectively:

Static members: Static members belong to the class itself, not to individual instances. They are shared among all instances of the class and can be accessed using the class name. Static members are declared using the static keyword.

Non-static members: Non-static members belong to individual instances of a class. Each instance has its own copy of non-static members. They are accessed using the instance name and can have different values or behaviors for each instance. Non-static members are not declared with the static keyword.