**Encapsulation**

**Assignment**

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

**Ans:** Encapsulation is a mechanism through which we can wrap the data members and member methods of class is a single unit called encapsulation.

It is called Data Hiding because data encapsulation focuses on wrapping (i.e., encapsulating) complex data to offer a simplified perspective to the users, while data hiding focuses on restricting a program's data use to ensure data security. Data encapsulation targets how the data is accessed and how different objects behave.

Note-

1. Declare the class variables as a private.
2. Declare the class methods as a public.
3. What are the important features of Encapsulation?

**Ans:** Combine the data of our application and its manipulation at one place. Encapsulation Allow the state of an object to be accessed and modified through behaviors. Reduce the coupling of modules and increase the cohesion inside them.

1. What are getter and setter methods in Java? Explain with an example?

Ans: Setter methods are used to set the value to the instance variables of the class. Getter methods are used to get the value from the instance variables of the class.

Example:

//setter and getter use

class Student

{

private int age;

private String name;

public int getAge()

{

return age;

}

public void setAge(int age)

{

this.age = age;

}

public String getName()

{

return name;

}

public void setName(String name)

{

this.name = name;

}

public void show()

{

System.out.println(name + " " + age);

}

}

public class dog {

public static void main(String args[])

{

Student obj = new Student();

Student obj1 = new Student();

obj.setAge(18);

obj1.setAge(20);

obj.setName("Prince");

obj1.setName("Kumar");

int stud1Age = obj.getAge();

int stud2Age = obj1.getAge();

String stud1Name = obj.getName();

String stud2Name = obj1.getName();

System.out.println(stud1Age);

System.out.println(stud1Name);

System.out.println(stud2Age);

System.out.println(stud2Name);

}

}

//output: 18

// Prince

// 20

// Kumar

1. What is the use of this keyword explain with an example?

Ans: The use of this keyword is as follows:

1. “this” keyword refers to the current object inside a method or constructor.
2. Whenever the name of instance and local variable both are same then our runtime environment JVM gets confused that which one is local variable and which one is instance variable, to avoid this problem we should use “this” keyword.
3. It is also used when we want to call the default constructor of its own class.
4. It also called parameterized constructor of its own class.

Example:

class Student

{

private String name;

private Integer id;

private String address;

Student(String name,Integer id, String address)

{

this.name=name;

this.id=id;

this.address=address;

}

public void display()

{

System.out.println("Name is    :: "+name);

System.out.println("Id is      :: "+id);

System.out.println("Address is :: "+address);

}

public static void main(String[] args)

{

Student std = new Student("sachin",10,"MI");

std.display();

}

}

Output:

Name is    :: sachin

Id is      :: 10

Address is :: MI

1. What is the advantage of Encapsulation?

Ans: The advantages of encapsulation are:

1. **Data Hiding** – The key benefit of encapsulation is that it hides the internal implementation details from the outside world. The inside of the capsule can be changed without breaking the encapsulated functionality.
2. **Increased Flexibility** –  Since the implementation is hidden, the object can freely change the internals. This allows the object to evolve over time.
3. **Reusability** – Encapsulated objects can be reused in different applications. They provide well-defined interfaces that can be used to connect various applications.
4. **Testing** – Encapsulated code is easy to test. You can test the object by calling its public methods and verifying that they work as intended.
5. **Maintainability** – The encapsulated code is easy to maintain since interacting with the outside world is through a well-defined interface. Any changes made to the internal implementation will not affect the applications using it.
6. How to achieve encapsulation in Java? Give an example?

**Ans:** Encapsulation in Java can be achieved by:

* + 1. Declaring the variables of a class as private.
    2. Providing public setter and getter methods to modify and view the variables values.

Example:

class A

{

private int value;//data hiding

public void setValue(int x)//data abstraction

{

value=x;

}

public int getValue()

{

return value;

}

}

class B

{

public static void main(String[] args)

{

A r = new A();

r.setValue(100);

System.out.print(r.getValue());

}

}

//Output:100

**Constructor**

**Assignment**

1. What is a Constructor?

Ans: Constructor is a special type of method whose name is same as class name. The main purpose of constructor is initializing the object. Every java class has a constructor. A constructor is automatically called at the time of object creation. A constructor never contain any return type include void.

Syntax:- class class\_name

{

Class\_name()

{

……

…….

}

}

1. What is Constructor Chaining?

Ans: Constructor Chaining is a technique of calling another constructor from one constructor. this() is used to call the same class constructor whereas super() is used to call super class constructor.

1. Can we call a subclass constructor from a superclass constructor?

Ans: No, there is no way in java to call a subclass constructor from a superclass constructor.

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

Ans: Ideally, Constructor must not have a return type. By definition, if a method has a return type, it’s not a constructor. (JLS8.8 Declaration) It will be treated as a normal method. But compiler gives a warning saying that method has a constructor name.

Example:

class Demo

{

int Demo()

{

return 0; // Warning for the return type

}

}

1. What is No-arg constructor?

Ans: Constructor without arguments is called a no-arg constructor. Default constructor in java is always a no-arg constructor.

class Demo{ public Demo() { //No-arg constructor

}}

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

Ans: In Java, a default constructor is a constructor that is automatically provided by the compiler if no other constructors are defined in the class. A default constructor takes no arguments and has no implementation.A no-arg constructor, on the other hand, is a constructor that explicitly takes no arguments. This means that the developer has defined it in the class and it may have some implementation

1. When do we need Constructor Overloading?

Ans: Constructor overloading in Java is useful in several situations:

1. **Different initialization options**: Constructor overloading allows you to provide different ways to initialize objects of a class. By defining multiple constructors with varying parameter lists, you can offer different initialization options based on the needs of the client code. This can be helpful when objects may be created with different sets of initial values or when certain parameters are optional.
2. **Default values:** Constructor overloading can be used to provide default values for certain parameters. By defining constructors with different parameter combinations, you can set default values for optional parameters, allowing clients to omit those parameters when creating objects.
3. **Encapsulation and abstraction:** Constructor overloading can help enforce encapsulation and abstraction principles by providing different levels of access to object initialization. For example, a class may have a public constructor that accepts all necessary parameters and a private constructor that only accepts a subset of those parameters. This allows the class to control the initialization process and provide a more abstract interface to the client code.
4. Polymorphism and inheritance: Constructor overloading plays a role in polymorphic scenarios, especially when dealing with inheritance. Subclasses can have their own constructors in addition to inheriting constructors from the superclass. This allows you to create objects of different subclasses while still adhering to the base class interface.
5. Code reuse and convenience: Constructor overloading enables code reuse by allowing constructors to call each other. You can have a constructor that performs common initialization tasks and have other constructors delegate to it, reducing code duplication. Additionally, constructor overloading provides convenience to the client code by offering different ways to create objects based on the provided parameters.

In summary, constructor overloading in Java is useful when you want to provide different initialization options, set default values, enforce encapsulation and abstraction, enable polymorphism and inheritance, facilitate code reuse, and provide convenience to the client code.

1. What is Default constructor? Explain with an Example?

Ans:  A constructor which does not have any parameter is called default constructor.

1. For every java class constructor concept is applicable.
2. If we don't write any constructor, then the compiler will generate a default constructor.
3. If we write at least one constructor then the compiler won't generate any default constructor, so we say every java class will have a compiler generated default constructor or programmer written constructor but not both simultaneously.

Example:

class A

{

int a;

String b;

boolean c;

A()//default constructor

{

a=100;

b="Raju Kushwah";

c=true;

}

void Disp()

{

System.out.print(a+" "+b+" "+c);

}

}

public class DefaultCons

{

public static void main(String[] args)

{

A r=new A();

r.Disp();

}

}

//Output:

// 100 Raju Kushwah true

**Static Keyword**

**Assignment**

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

Ans: The static keyword in Java is used to define members (variables and methods) that belong to the class itself rather than individual instances (objects) of the class. Here are a few reasons why we need the static keyword in Java:

1. Accessing members without creating objects: When a member (variable or method) is declared as static, it can be accessed directly using the class name, without the need to create an instance of the class. This is useful when you want to access a member that is common to all instances of the class or when you want to perform some operation without the need for an object.

Example:

public class MathUtils

{

public static int add(int a, int b)

{

return a + b;

}

}

// Usage

int sum = MathUtils.add(5, 3);

In this example, the add method in the MathUtils class is declared as static. It can be invoked using the class name MathUtils.add(5, 3) without creating an object of the MathUtils class.

1. Sharing data among instances: static variables are shared among all instances of a class. When a member is declared as static, there is only one copy of that variable or method in memory, regardless of how many objects of that class are created. This can be useful when you want to maintain a shared state or when you want to keep track of information that should be consistent across all instances.

Example:

public class Counter

{

private static int count = 0;

public Counter()

{

count++;

}

public static int getCount()

{

return count;

}

}

// Usage

Counter c1 = new Counter();

Counter c2 = new Counter();

System.out.println(Counter.getCount()); // Output: 2

In this example, the count variable is declared as static. Each time a Counter object is created, the constructor increments the count variable. Since count is shared among all instances, it keeps track of the total number of Counter objects created.

1. Utility methods and constants: static methods can be used to define utility methods or helper functions that are not specific to any particular instance. They can be called directly on the class itself without the need for an object. Similarly, static variables can be used to define constants that are common to all instances of a class.

Example:

public class StringUtils

{

public static boolean isEmpty(String str)

{

return str == null || str.isEmpty();

}

public static final double PI = 3.14159;

}

// Usage

String name = "John";

boolean isEmpty = StringUtils.isEmpty(name);

double radius = 5.0;

double circumference = 2 \* StringUtils.PI \* radius;

In this example, the isEmpty method in the StringUtils class is a utility method that can be called directly on the class. The PI variable is a constant that holds the value of π and can be used in calculations without the need for an object.

In summary, the static keyword in Java allows members to be accessed without creating objects, facilitates sharing data among instances, enables the definition of utility methods and constants, and provides a way to organize code at the class level.

1. What is class loading and how does the Java program actually execute?

Ans: Class Loading: The JVM is responsible for loading classes into memory as they are needed during program execution. The class loading process involves three steps: loading, linking, and initialization.

a. **Loading**: The class loader searches for the bytecode of the required class in the classpath. It can load classes from various sources, such as a local file system, network, or even dynamically generated at runtime. The loaded bytecode is then stored in the JVM's method area.

b. **Linking**: The linking phase consists of three sub-phases: verification, preparation, and resolution.

**Verification:** The bytecode is verified to ensure its integrity and security. This step checks for compliance with the Java language rules and detects any potential security issues.

**Preparation:** Memory is allocated for static variables and initialized with their default values. This includes static variables declared in the class and its superclasses.

**Resolution**: Symbolic references in the bytecode are resolved to actual references. This involves locating the referenced classes, methods, and fields.

c. **Initialization**: In this step, the static variables and static initialize blocks of the class are executed. Static variables are assigned their explicit values, and static initializer blocks are executed in the order they appear in the code.

1. Can we mark a local variable as static?

Ans: No, it is not possible to mark a local variable as static in Java.

The static keyword is used to define members (variables and methods) at the class level, not at the local level. Local variables are variables declared inside a method, constructor, or block, and they exist only within the scope of that particular block or method.

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

Ans: In Java, the static block is executed before the main method because it is part of the class initialization process. When a class is loaded and initialized by the Java Virtual Machine (JVM), it follows a specific sequence of steps, which includes the execution of static blocks before the main method is invoked.

Example:

public class MyClass

{

static

{

System.out.println("Static block executed");

}

public static void main(String[] args)

{

System.out.println("Main method executed");

}

}

//Output:

// Static block executed

// Main method executed

1. Why a static method is also called a class method?

Ans: A static method in Java is also called a "class method" because it is associated with the class itself rather than with individual instances (objects) of the class. It is called a class method because it can be invoked directly on the class, without the need to create an object of the class.

Eaxmple:

public class MathUtils

{

public static int sum(int a, int b)

{

return a + b;

}

}

// Usage

int result = MathUtils.sum(5, 3);

1. What is the use of static blocks in java?

Ans: In Java, static blocks are used to initialize static variables or perform one-time initialization tasks that are associated with the class itself rather than with individual instances (objects) of the class. They are executed when the class is loaded into memory, before any static method or the main method is invoked.

**Uses:**

1. **Initializing static variables:**

Ex.

public class MyClass

{

static int count;

static

{

count = 10;

}

}

1. **Loading resources or setting up configurations:**

**Ex.**

public class DatabaseConnection

{

static Connection connection;

static

{

// Load configuration from file

// Initialize the database connection

// Set up other necessary resources

}

}

1. **Exception handling:**

**Ex.**

public class MyClass

{

static

{

Try

{

// Perform initialization tasks

}

catch (Exception e)

{

// Handle the exception

// Log the error message

// Take appropriate actions

}

}

}

1. Difference between Static and Instance variable?

Ans:

| **Static Variable** | **Instance Variable** |
| --- | --- |
| Static Variables are declared using keyword 'static'. | Instance Variables are declared without using keyword 'static'. |
| All objects of a class share the same copy of static variables. | Each object of the class gets its own copy of instance variables. |
| Static Variables can be accessed using the class name or object.  ***ObjectReference.VariableName*** | Instance Variables can be accessed only through an object of the class.  ***ClassName.VariableName*** |

1. Difference between static and non static member?

Ans:

| **Static Data Members** | **Non-Static Data Members** |
| --- | --- |
| They are declared using keyword 'static'. | They are declared without using keyword 'static'. |
| All objects of a class share the same copy of Static data members. | Each object of the class gets its own copy of Non-Static data members. |
| They can be accessed using the class name or object. | They can be accessed only through an object of the class. |

**Practical**

**Assignment**

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

Ans:

public class StaticAssignment

{

private static int instanceCount = 0;

public StaticAssignment()

{

instanceCount++;

}

public static int getInstanceCount()

{

return instanceCount;

}

public static void main(String[] args)

{

StaticAssignment c1 = new StaticAssignment();

StaticAssignment c2 = new StaticAssignment();

StaticAssignment c3 = new StaticAssignment();

System.out.println("Number of instances created: " + StaticAssignment.getInstanceCount());

}

}

//output:

// Number of instances created: 3

1. Write a program and create a constructor with parameters and initialize the variable using a constructor?

Ans:

class Parameterized1

{

int x,y;

Parameterized1(int a, int b)

{

x=a; y=b;

}

Parameterized1(int a, String b)

{

System.out.println(a+" "+b);

}

void show()

{

System.out.println(x+" "+y);

}

}

class Parameterized

{

public static void main(String[] args)

{

Parameterized1 r=new Parameterized1(100,200);

r.show();

Parameterized1 ref=new Parameterized1(10,"ankush");

}

}

//Output

// 100 200

// 10 ankush

1. Use a private keyword for a variable and use setter and getter methods to initialize and print the values?

Ans:

class A

{

private int value;//data hiding

public void setValue(int x)//data abstraction

{

value=x;

}

public int getValue()

{

return value;

}

}

class B

{

public static void main(String[] args)

{

A r = new A();

r.setValue(100);

System.out.print(r.getValue());

}

}

//Output:100

1. Write a program to call an method without creating an object of a class?

Ans:

Class Demo

{

Int a=10;

String b=”Ankus”;

Void Show()

{

System.out.println(a+” ”+b);

}

}

Class Test

{

public static void main(String[] args)

{

Demo r;

r= new Demo();

r.Show();

}

}

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

Ans:

import java.util.stream.Stream;

public class StaticBlock

{

public String name;

public int age;

public StaticBlock()

{

System.out.println("No arg constructor");

}

public StaticBlock(int age)

{

super();

this.age = age;

}

public StaticBlock(String name)

{

super();

this.name = name;

}

public StaticBlock(String name, int age)

{

super();

this.name = name;

this.age = age;

}

public String toString()

{

return "StaticBlock [name=" + name + ", age=" + age + "]";

}

static

{

System.out.println("Static block");

}

public static void main(String[] args)

{

StaticBlock block = new StaticBlock(20);

System.out.println(block.age);

StaticBlock block2 = new StaticBlock("Kohli");

System.out.println(block2.name);

StaticBlock block3 = new StaticBlock("Mahi", 30);

System.out.println(block3.toString());

}

}

//Output:

// Static block

// 20

// Kohli

// StaticBlock [name=Mahi, age=30]