Day 14

Topics:

1. **Java OOP Concepts – Encapsulation**
2. **static keyword**
3. **System.out.println()**
4. **public static void main(String args[])**

Encapsulation

Encapsulation is a fundamental Object-Oriented Programming (OOP) concept where code and data are **wrapped together into a single unit**.

This mechanism ensures that sensitive data is hidden from the outside world and only accessible through controlled methods.

Key Points:

1. **All Variables Should Be Private:**
    - All the class variables should be declared as private, ensuring that they are not directly accessible from outside the class.
2. **Getters and Setters for Each Variable:**
    - For each private variable, provide two methods:
        - **Getter Method:** A method to retrieve the value of the variable.
        - **Setter Method:** A method to set or update the value of the variable.
3. **Variables Operated Only Through Methods:**
    - Variables should be accessed and modified only through these getter and setter methods, ensuring that any operation on the data is controlled and secure.

Example Code

Here's an example to demonstrate encapsulation:

```java
public class Person {
    private String name; // Private variable, cannot be accessed directly

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

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

Example Explained

- **Private Variable**:
    - The name variable is private, so it cannot be accessed directly from outside the Person class.
- **Getter Method**:
    - getName() is used to retrieve the value of name.
- **Setter Method**:
    - setName(String newName) is used to set or update the value of name. The this keyword is used to refer to the current object's instance variable.

Accessing Private Variables

In a different class, **you cannot directly access** the private variables:

```java
public class Main {
    public static void main(String[] args) {
        Person myObj = new Person();
        // Direct access to name is not allowed
        // myObj.name = "John";  // error
        // System.out.println(myObj.name); // error

        // Accessing via getter and setter
        myObj.setName("John"); // Sets the name
        System.out.println(myObj.getName()); // Outputs "John"
    }
}
```

Static keyword

The **static** keyword in Java is used to define class-level variables and methods that can be accessed without creating an instance (object) of the class.

When to Use Static Variables and Static Methods?

Static Variables:

- **Shared Data Across All Instances**: Static variables are used when you want a variable to be shared among all instances (objects) of a class. Instead of each object having its own copy of the variable, all objects share the same static variable.

- **Common Properties for All Objects**: Use static variables when there's a property or piece of data that is common to all objects of a class. For example, a counter to keep track of the number of instances created.

- **Memory Efficiency**: Since static variables are shared across all instances, they help in saving memory.

Static Methods:

- Use static methods for **operations that are not dependent on object state** or when creating utility/helper methods.
- Ideal for implementing methods that can be called without creating an instance of the class.

Accessing Static variables and methods:

1. **Static Methods Can Access Static Stuff Directly**:
    - Static methods can directly access other static variables and methods within the same class without needing an object.
2. **Static Methods Can Access Non-Static Stuff Through an Object**:
    - Static methods cannot directly access non-static variables or methods, but they can access them via an object.
3. **Non-Static Methods Can Access Everything Directly**:
    - Non-static methods can access both static and non-static variables and methods directly.

System.out.println()

In Java, **System.out.println()** is a commonly used method to print information to the console.

Let's try to understand with an example.

```
class Test
{
static String s="welcome";
}

Test.s.length()
```

In this example:

- **Test** is a class.
- **s** is a static string variable within the Test class with the value "welcome".
- The **length()** method is called on s, which returns the length of the string.

Similarly,

```
class System
{
   static PrintStream out;
}

System.out.println()
System.out.print()
```

Here's what's happening:

- **System** is a class provided by Java.
- **out** is a static variable of type **PrintStream** in the **System** class.
- The **println()** and **print()** methods belong to the **PrintStream** class.

public static void main(String[] args)

The **public static void main(String[] args)** method is the entry point of any standalone Java application. Let's break down each part of this method to understand its significance:

1. public

- **Access Modifier**: public is an access modifier that makes the main method accessible from anywhere. This is necessary because the Java runtime needs to access this method to start your application.

2. static

- **Static Method**: static means that the method belongs to the class, not instances of the class. This allows the Java runtime to call the main method without creating an instance of the class. Since main is the starting point of the program, it needs to be static so that it can be called by the Java Virtual Machine (JVM) without needing to instantiate the class first.

3. void

- **Return Type**: void indicates that the main method does not return any value. The purpose of main is to start the application, not to return data to the JVM.

4. main

- **Method Name**: main is the name of the method. This is a special method name in Java, and the JVM looks for this method signature as the entry point when running a Java program.

5. String[] args

- **Parameter**: String[] args is an array of strings that stores command-line arguments passed when the program is executed. If you run your Java program from the command line and provide additional arguments, those arguments are passed as strings in this array.
- For example, if you run java MyClass arg1 arg2, then args will be an array containing ["arg1", "arg2"].
- If no command-line arguments are provided, args will be an empty array.

Quiz on Java Encapsulation, Static Variables, and Static Methods

Questions

1. **What is encapsulation in Java?**
    - a) The process of hiding implementation details and showing only functionality to the user
    - b) The process of inheriting properties from another class
    - c) The process of creating multiple methods with the same name but different parameters
    - d) The process of converting data types
2. **Which access modifier is typically used to achieve encapsulation?**
    - a) public
    - b) protected
    - c) private
    - d) default
3. **What is the purpose of getter methods in encapsulation?**
    - a) To set the values of private variables
    - b) To retrieve the values of private variables
    - c) To perform calculations
    - d) To delete private variables
4. **How do setter methods support encapsulation?**
    - a) By hiding the private variables
    - b) By setting the values of private variables
    - c) By retrieving the values of private variables
    - d) By making private variables public
5. **In encapsulation, how are private variables accessed outside the class?**
    - a) Directly
    - b) Using getter and setter methods
    - c) Using public variables

- d) Using protected methods

6. **What is a static variable in Java?**
   - a) A variable that belongs to an instance of a class
   - b) A variable that can be accessed only within the class
   - c) A variable that belongs to the class itself, rather than any instance
   - d) A variable that changes its value every time it's accessed
7. **How is a static variable declared in Java?**
   - a) static int variableName;
   - b) private int variableName;
   - c) public final int variableName;
   - d) protected int variableName;
8. **What is the primary purpose of a static method in Java?**
   - a) To be able to call the method without creating an instance of the class
   - b) To access instance variables of the class
   - c) To override methods of a superclass
   - d) To implement multiple inheritance
9. **Can a static method access non-static variables directly?**
   - a) Yes
   - b) No
   - c) Only if the non-static variables are declared static
   - d) Only if the static method is inherited
10. **Which keyword is used to define a static method?**
    - a) static
    - b) final
    - c) private
    - d) protected
11. **Can you call a static method from a non-static method?**
    - a) Yes
    - b) No
    - c) Only if the static method is private
    - d) Only if the non-static method is private

12. **What is the output of the following code?**

```
public class Test {
    static int count = 0;

    public static void increment() {
        count++;
    }

    public static void main(String[] args) {
        increment();
        System.out.println(count);
    }
}
```

- a) 0
- b) 1
- c) Compilation error
- d) Runtime error

13. **Which of the following statements about static variables is false?**
    - a) Static variables are shared among all instances of a class
    - b) Static variables are initialized only once
    - c) Static variables can be accessed by non-static methods
    - d) Static variables can have different values for different instances
14. **What is the scope of a static variable in Java?**
    - a) The scope of a static variable is limited to the instance of the class
    - b) The scope of a static variable is the entire class
    - c) The scope of a static variable is the method in which it is declared
    - d) The scope of a static variable is only the main method
15. **Can a static method be overridden in Java?**
    - a) Yes
    - b) No
    - c) Only in an abstract class
    - d) Only if it is declared final
16. **How do you call a static method in Java?**
    - a) Using the class name
    - b) Using an object of the class
    - c) Using a different method
    - d) Using a constructor

17. **Which of the following is a benefit of encapsulation?**
    - a) Increased flexibility
    - b) Decreased complexity
    - c) Enhanced code maintainability
    - d) All of the above
18. **Which of the following statements is true about encapsulation?**
    - a) Encapsulation allows direct access to private variables
    - b) Encapsulation helps in hiding the internal state of an object
    - c) Encapsulation is only used in public methods
    - d) Encapsulation is the same as inheritance
19. **What will happen if you try to access a private variable of a class from outside the class?**
    - a) The program will compile successfully
    - b) The program will compile with a warning
    - c) The program will throw a compilation error
    - d) The program will throw a runtime error
20. **In which scenario would you use static variables?**
    - a) When you need to store values that are common to all instances of a class
    - b) When you need to store values specific to a single instance of a class
    - c) When you need to define methods that are not related to the class
    - d) When you need to override instance methods

Answers

1. a) The process of hiding implementation details and showing only functionality to the user
2. c) private
3. b) To retrieve the values of private variables
4. b) By setting the values of private variables
5. b) Using getter and setter methods
6. c) A variable that belongs to the class itself, rather than any instance
7. a) static int variableName;
8. a) To be able to call the method without creating an instance of the class
9. b) No
10. a) static

11. a) Yes
12. b) 1
13. d) Static variables can have different values for different instances
14. b) The scope of a static variable is the entire class
15. b) No
16. a) Using the class name
17. d) All of the above
18. b) Encapsulation helps in hiding the internal state of an object
19. c) The program will throw a compilation error
20. a) When you need to store values that are common to all instances of a class