Day 12

Name :Siddarth S
Date : 09/08/2024

1. Implement Abstract Class with Overloading and Overriding:

Abstract Class: A class that cannot be instantiated and may contain abstract methods that must be implemented by its subclasses.

Overloading: Same method name but different parameters (either in number or type).

Overriding: A subclass provides a specific implementation of a method that is already defined in its superclass.

Code:
```
abstract class Base {
  // Abstract method
  abstract void display();

  // Method Overloading
  void print(String name) {
    System.out.println("Name: " + name);
  }

  void print(int age) {
    System.out.println("Age: " + age);
  }
}

class Derived extends Base {
  // Overriding the abstract method
  @Override
  void display() {
    System.out.println("Display method in Derived class");
  }

  // Overriding a method
  @Override
  void print(String name) {
    System.out.println("Name from Derived class: " + name);
  }
}

public class Main {
  public static void main(String[] args) {
    Derived obj = new Derived();
    obj.display(); // Calls overridden method
    obj.print("Siddarth"); // Calls overridden method
    obj.print(25); // Calls overloaded method
  }
}
```

Output:

```
Display method in Derived class
Name from Derived class: Siddarth
```

Age: 25

Description:

- The `Base` class is an abstract class containing an abstract method `display()` and two overloaded methods `print(String)` and `print(int)`.
- The `Derived` class extends the `Base` class and provides an implementation for the `display()` method and overrides the `print(String)` method.
- Method overloading is demonstrated by the two `print` methods in the `Base` class, while method overriding is demonstrated by the `print(String)` method in the `Derived` class.

2. Implement Multiple Inheritance with Interface
Multiple Inheritance in Java can be achieved using interfaces since Java doesn't allow extending more than one class.

Code

```
interface Printable {
  void print();
}

interface Showable {
  void show();
}

class Document implements Printable, Showable {
  @Override
```

```
    public void print() {
        System.out.println("Printing the document...");
    }

    @Override
    public void show() {
        System.out.println("Showing the document...");
    }
}

public class Main {
    public static void main(String[] args) {
        Document doc = new Document();
        doc.print();
        doc.show();
    }
}
```

Output:

```
Printing the document...
Showing the document...
```

Description:

- The `Printable` and `Showable` interfaces declare methods `print()` and `show()`, respectively.
- The `Document` class implements both interfaces, providing implementations for the `print()` and `show()` methods.

- This demonstrates multiple inheritance in Java, where a class can implement multiple interfaces.

3. Show Final Methods in the Class that Can't Be Overridden

Final Methods: Methods that cannot be overridden by subclasses.

Code

```
class BaseClass {
  final void finalMethod() {
    System.out.println("This is a final method and cannot be
overridden.");
  }

  void regularMethod() {
    System.out.println("This is a regular method in the Base class.");
  }
}

class DerivedClass extends BaseClass {
  // This would cause an error
  // @Override
  // void finalMethod() {
  //   System.out.println("Cannot override final method.");
  // }

  @Override
  void regularMethod() {
    System.out.println("Overridden regular method in Derived class.");
  }
}

public class Main {
  public static void main(String[] args) {
    DerivedClass obj = new DerivedClass();
    obj.finalMethod(); // Calls the final method from BaseClass
    obj.regularMethod(); // Calls the overridden method
  }
}
```

Output:

```
This is a final method and cannot be overridden.
Overridden regular method in Derived class.
```

Description:

- The `BaseClass` has a `final` method `finalMethod()` that cannot be overridden by any subclass.
- The `DerivedClass` attempts to override `finalMethod()`, but this is commented out because it would cause a compilation error.
- The `regularMethod()` in `DerivedClass` is successfully overridden.

Loose Coupling with Base Class

Code:

```
class Base {
  void action() {
    System.out.println("Action in Base class");
  }
}

class Derived1 extends Base {
  @Override
  void action() {
    System.out.println("Action in Derived1 class");
  }
}

class Derived2 extends Base {
  @Override
  void action() {
    System.out.println("Action in Derived2 class");
  }
}

public class Main {
  public static void main(String[] args) {
    Base obj = new Derived1(); // Loose coupling
    obj.action(); // Calls Derived1's action

    obj = new Derived2(); // Loose coupling
    obj.action(); // Calls Derived2's action
    }
}
```

Output:

```
Action in Derived1 class
Action in Derived2 class
```

Description:

- The `Base` class has a method `action()`.
- The `Derived1` and `Derived2` classes extend `Base` and override the `action()` method.
- In the `Main` method, the `Base` class reference is used to create objects of `Derived1` and `Derived2`. This demonstrates loose coupling, allowing for flexible switching between different implementations.