1. Explain the concept of method overloading with an appropriate example.

2. Let there is a **class A** which has a member function **showA()** which displays "49 intake!!! Best of Luck for your exam !!!". **Class C** and **class D** inherits **class A** and **class E** inherits both **class C** and **class D**. Will there any problem if we call **showA()** function from the object of **class E**? If yes, then how can we solve it? Write down the solved program.

3. **Car** and **Bus** are different types of vehicles. **Car** has its property **speed** and **no_ of passengers**. The **Bus** also has the same properties. Both the vehicle's speed depends on its engine and passengers depend on number of sits given when a **Car** or a **Bus object** is created. You are assigned to the task of comparing the two types of vehicles by their speed.

   Now **translate** the scenario to a java program, by creating the classes and comparing the speed of two objects of the classes **Bus** and **Car**.

4. Write a program using three classes named **Grandfather**, **Father** and **Son** where the constructor of each class takes one integer **Age** and one string **Name** value as argument. All the constructors show the Name and Age. Complete the program considering the following cases:

   - Father inherits Grandfather and Son inherits Father.

   Now **show** the output for the above case. Also implement **super** and **this** keyword in the above program.

5. Create an abstract class named **Store**. Store will have one non-abstract function **showItem()** and two abstract functions **get_price()** and **display_info()**. Create two derived classes **Bookstore** and **Ricestore**. In the **main()** function, **design Store** class in such a way that implements the idea of abstraction.

6. Create a base class called **shape** which has two double type properties and a function **setdata()** which is used to set the values of those two properties. There should be another member function **display_area()** in **shape** to compute and display the area of figures.

   Derive two specific classes called **triangle** and **rectangle** from the base **shape**. Use these three classes that implements the idea of dynamic method dispatch

7. In package **mathoperations**, there is an interface **MathOperations** with an abstract method **Add(int a, int b)** that performs addition.
   In package **advancedoperations**, create another interface **AdvancedMathOperations** that extends **MathOperations** and has two abstract methods:
   Subtract(int a, int b) (performs subtraction) and **Power(int base, int exponent)** (raises the base to the power of the exponent).
   In package **implementations**, create a class **BasicMath** that implements the **MathOperations** interface,
   BasicMath must have a concrete method **Multiply(int a, int b)** that performs multiplication.
   In the same package, create another class **AdvancedMath** that implements the **AdvancedMathOperations** interface.**AdvancedMath** must also implement a method **Divide(int a, int b)** that performs integer division and checks for division by **zero**. Implement the code for the given scenario.

8. How can you declare "PI" as constant for the following code? public class MathConstants {
   public double PI = 3.14159; [3]
   }

9. Indicate what is wrong with the code below.

```
interface MyInterface {
        public MyInterface() {
                System.out.println("Constructor in an interface");
                }
        }
```

10. You are developing a multi-tiered payment system with different levels of subscriptions.

    There is a base class **Subscription**, which handles common features like calculating the basic fee. The **PremiumSubscription** class extends **Subscription** and adds premium features, while the **VIPSubscription** class further extends **PreiniumSubscription** and adds VIP-only features. The base class Subscription has a constructor that takes a **double baseFee** and assigns it to a class variable.

    The **PremiumSubscription** class has an additional feature for a **double premiumFee**. Its constructor should invoke the **Subscription** class's constructor using **super()** and initialize the premium fee. The **VIPSubscription** class further adds a **double vipFee**, Its constructor should invoke the **PremiumSubscription** constructor using **super()** and initialize the VIP fee.

    Implement the classes **Subscription**, **PremiumSubscription**, and **VIPSubscription**. Use the **super** keyword appropriately to call constructors and methods. Implement the code for the given scenario.

11. Implement a class **Car** that has a static variable count and a static method **getCount()**.

12. There is a class called **BankUtils** that contains a method that prints "Welcome to the bank". This class should not be extended by any other class, Inside the BankAccount class, there's a method called **calculateInterest()** that provides the logic for calculating interest. This method should not be overridden by any subclass. Each **BankAccount** object should have a unique account number that

cannot be changed once it's set during object creation. A **Transaction** class should have a method process Transaction() that is only accessible within the same package, representing a default access control method.

13. Create a class named **ElectricCar** that both extends an abstract class and implements two interfaces. Specifically: The abstract class Vehicle should contain abstract methods **start()** and **stop()**. The first interface Electric should define a method **chargeBattery()**. The second interface Autonomous should define a method **enableAutoPilot()**.

14. There is a top-level interface **Appliance** that provides a method for turning an appliance on or off. The **KitchenAppliance** interface extends Appliance and adds a method for setting the temperature of the appliance. The **EntertainmentAppliance** interface also extends Appliance and adds a method for adjusting the volume of the appliance. The class **SmartDevice** should implement both **KitchenAppliance** and **EntertainmentAppliance** interfaces. This device can control both types of appliances, adjusting temperature and volume as required

    Create the interfaces **Appliance**, **KitchenAppliance**, and **EntertainmentAppliance** to represent the functionality described. Implement the **SmartDevice** class that handles both the temperature and volume