Programming Assignment: Object-Oriented Programming in Java

1. Write a Java program to print "Hello, World!" and display the version of Java being used.
   (Hint: Use System.getProperty("java.version") to get the version.)

2. Write a Java program to display the default values of different data types in Java.
   (Hint: Use instance variables for primitive data types.)

3. Write a Java program to swap two numbers using a temporary variable and also without using a temporary variable.

4. Write a Java program to create a one-dimensional array, initialize it with values, and find the largest and smallest elements in the array.

5. Implement a Java program to perform arithmetic operations (+, -, , /, %) using user-defined methods and the appropriate operators.

6. Write a Java program to check whether a given number is prime or not using a loop.

7. Write a menu-driven Java program using a switch statement to perform the following operations:
   a. Check if a number is even or odd
   b. Find the factorial of a number
   c. Exit the program
8. Create a class Student with instance variables: name, rollNo, and marks. Write a Java program to create an object of Student, initialize values, and display the details.

9. Demonstrate method overloading by defining three methods with the same name but different parameters (int, double, and two int parameters).

10. Create two classes: Parent and Child. Inherit Child from Parent and demonstrate method overriding. Show how super can be used to access the parent class method.

11. Write a Java program that illustrates the use of the final keyword with a variable, method, and class.

12. Define a package named mypackage that contains a class Calculator with basic arithmetic methods (add, subtract, multiply, divide). Write another Java program to import this package and use the Calculator class.

13. Write a Java program to demonstrate multiple inheritance using interfaces. Create an interface Animal with a method makeSound(), and another interface Pet with a method play(). Implement both interfaces in a class Dog.

1. Implement a Custom Dynamic Array (Arrays, Classes, and Objects)

Problem Statement:
Design a **Custom Dynamic Array** class in Java that:

- Allows adding elements dynamically.
- Doubles its size when the capacity is full.
- Provides methods to add, remove, and get elements by index.
- Implements a `toString()` method to display the elements.

Hint:
Use an internal array and implement dynamic resizing when the array is full.

---

2. Banking System with Inheritance and Method Overriding

Problem Statement:
Design a **Banking System** in Java with the following structure:

- A base class **`BankAccount`** with attributes: `accountNumber`, `accountHolderName`, and `balance`.
- A method `deposit(amount)` to add money and `withdraw(amount)` to deduct money (ensure balance is sufficient).
- A derived class **`SavingsAccount`** that:
  - Overrides `withdraw(amount)` to allow withdrawal only if the balance remains above ₹1000.
  - Implements an interest calculation method (`applyInterest()`).

Hint:

- Use the `super` keyword for method calls from the parent class.
- Create objects and demonstrate transactions.

---

3. Student Management System using Packages and CLASSPATH

Problem Statement:
Create a **Student Management System** with the following:

1. A package `studentdata` that contains a class `Student` with attributes: `name`, `rollNumber`, `marks`.
2. A method `displayDetails()` to show student details.
3. Another package `studentoperations` that contains a class `Result` with a method `calculateGrade()` based on marks:
   - `marks >= 90` → A
   - `marks >= 80` → B
   - `marks >= 70` → C

    - Otherwise → F
4. Create a main program to:
    - Import both packages.
    - Create multiple student objects.
    - Display details and grades for each student.

Hint:
Use `import` to access classes from different packages and run the program by setting the CLASSPATH.

---

4. Multiple Interfaces for Vehicle System

Problem Statement:
Design a **Vehicle System** using **multiple interfaces**:

1. Interface `Vehicle` with a method `void start()`.
2. Interface `FuelVehicle` with a method `void refuel()`.
3. Class `Car` that implements both `Vehicle` and `FuelVehicle`, providing implementations for `start()` and `refuel()`.
4. Class `ElectricCar` that implements only `Vehicle`, overriding `start()` with a different implementation.

Hint:

- Use `implements` to implement multiple interfaces.
- Create objects of `Car` and `ElectricCar` and test their methods.

---

5. Exception Handling in Employee Payroll System

Problem Statement:
Develop an **Employee Payroll System** that:

1. Accepts `employeeName`, `employeeID`, and `salary` as inputs.
2. Throws a custom exception `InvalidSalaryException` if the salary is **negative**.
3. Catches and handles exceptions properly.
4. Uses a **try-catch-finally** block to ensure salary input is always validated.

Hint:

- Define a custom exception class extending `Exception`.
- Throw the exception when an invalid salary is entered.
- Use `finally` to display a message that execution is complete.