3. 2D Arrays

1. **Create and Print a 2D Array:**

   - Write a Java program to create a 2D array of integers and initialize it with values. Print the elements in matrix form.

2. **Sum of Rows and Columns:**

   - Implement a program that calculates the sum of each row and each column of a 2D array.

3. **Transpose of a 2D Matrix:**

   - Write a program to transpose a given 2D matrix (convert rows into columns and vice versa).

4. 3D Arrays

1. **Initialize and Print a 3D Array:**

   - Create a Java program to initialize a 3D array with random numbers between 1 and 100 and print its elements in a formatted way.

2. **Sum of All Elements in a 3D Array:**

   - Write a program that calculates the sum of all elements in a 3D array.

3. **Find Maximum Value in a 3D Array:**

   - Implement a program to find the maximum value in a 3D array.

5. Jagged Arrays

1. **Create and Print a Jagged Array:**

- Write a Java program to create a jagged array where each row has a different number of columns. Initialize the jagged array with random values and print it.

2. **Find Maximum Element in Each Row of a Jagged Array:**

- Implement a program that finds the maximum element in each row of a jagged array.

3. **Sum of All Elements in a Jagged Array:**

- Write a program to calculate the sum of all elements in a jagged array.

6. Array of Objects

1. **Array of `Book` Objects:**

- Create a `Book` class with attributes like `title`, `author`, and `price`. Write a program to create an array of `Book` objects and display their details.

2. **Average Marks of Students:**

- Write a program that creates an array of `Student` objects and calculates the average marks of all students.

3. **Find Highest Salary Employee:**

- Implement a program to create an array of `Employee` objects with attributes like `name`, `id`, and `salary`. Write a method to find the employee with the highest salary.

7. Strings

1. **Check Palindrome:**

- Write a program to check if a given string is a palindrome.

2. **Frequency of Characters:**

- Create a program that counts the frequency of each character in a given string.

3. **Count Vowels and Consonants:**

- Implement a program that takes a sentence as input and counts the number of vowels and consonants.

8. Mutable vs. Immutable Strings

1. **Demonstrate Immutability:**

- Write a program that demonstrates the immutability of the `String` class by attempting to modify a `String` object and observing the results.

2. **Demonstrate Mutability with `StringBuilder`:**

- Create a program that uses `StringBuilder` or `StringBuffer` to modify a string and demonstrate its mutability by appending, reversing, and deleting characters.

9. StringBuffer

1. **Reverse a String Using `StringBuffer`:**

- Implement a program that uses `StringBuffer` to reverse a given string.

2. **Replace Substring Using `StringBuffer`:**

- Write a program to use `StringBuffer` to replace a substring within a string.

3. **Demonstrate `StringBuffer` Methods:**

- Create a program that demonstrates the use of various `StringBuffer` methods like `append()`, `insert()`, `delete()`, `replace()`, and `reverse()`.

10. Static Variables, Methods, and Blocks

1. **Static Variables:**

   - Write a program to demonstrate the use of static variables by creating a class `Counter` that keeps track of the number of objects created.

2. **Static Method for Factorial:**

   - Implement a program that uses a static method to calculate the factorial of a given number.

3. **Static Block Initialization:**

   - Create a program that uses a static block to initialize static variables and prints their values.

11. Encapsulation, Getters, and Setters

1. **Encapsulation with `Person` Class:**

   - Write a Java program that demonstrates encapsulation by creating a `Person` class with private attributes `name` and `age`, and providing public getters and setters for them.

2. **Encapsulation with `BankAccount` Class:**

   - Create a `BankAccount` class with private attributes `accountNumber`, `accountHolderName`, and `balance`. Write appropriate getters and setters to access and update these attributes.

12. `this` Keyword

1. **Using `this` Keyword:**

   - Write a program to demonstrate the use of the `this` keyword to refer to the current object.

2. **Constructor Initialization with `this`:**

- Implement a `Student` class with a constructor that initializes the student's name and ID using the `this` keyword. Write a method to display the student's details.

13. Constructors

- **Default Constructor:**

1. Create a `Car` class with attributes like `model` and `year`. Use a default constructor to initialize these attributes with some default values. Print the details of the car using a method.

2. Create a `Circle` class with a default constructor that initializes the radius to `1.0`. Write a method to calculate and return the area of the circle. Create an object of the `Circle` class and display the area.

3. Write a `Book` class that has a default constructor. The constructor should print a message like "Book object created!" when it is called. Create an object of this class to test if the message is displayed.

- **Parameterized Constructor:**

1. Implement a `Student` class with attributes `name`, `id`, and `marks`. Use a parameterized constructor to initialize these attributes. Write a method to display the student's details.

2. Create a `Rectangle` class with attributes `length` and `breadth`. Write a parameterized constructor to initialize these attributes and a method to calculate and display the perimeter of the rectangle.

3. Write a program that has a `Product` class with a parameterized constructor that accepts parameters like `productName`, `productId`, and `price`. Create an array of `Product` objects and display their details.

- **Constructor Overloading:**

1. Write a Java program to demonstrate constructor overloading by creating a `Person` class with three constructors: one with no parameters, one with one parameter (`name`), and one with two parameters (`name` and `age`). Create objects using different constructors and display their values.

2. Create a `Triangle` class that has constructors to initialize:

   - Only the base of the triangle.

   - Both the base and height of the triangle.

   - Write methods to calculate the area using different constructors.

3. Implement a `BankAccount` class that has overloaded constructors:

   - A default constructor with no parameters.

   - A constructor that initializes `accountNumber` and `accountHolderName`.

   - A constructor that initializes `accountNumber`, `accountHolderName`, and `initialBalance`.

14. Naming Conventions

- **Classes and Interfaces:**

1. Create a Java class that demonstrates proper naming conventions for classes and methods. Follow Java conventions by using PascalCase for classes (`StudentDetails`, `BankAccount`) and interfaces (`Readable`, `Printable`), and camelCase for methods (`calculateTotalMarks`, `displayDetails`).

- **Variables and Methods:**

1. Write a program to demonstrate proper naming conventions for variables and methods. Use camelCase for variable names (`studentName`, `totalMarks`) and method names (`calculateAverageMarks`, `displayStudentInfo`).

- **Constants:**

1. Implement a program where you define constants using the `final` keyword with proper naming conventions (all uppercase letters with underscores). For example, `final int MAX_STUDENTS = 100;` and `final double PI = 3.14159;`.

15. Anonymous Objects

1. **Anonymous Object for Addition:**

   - Create an anonymous object of a `Calculator` class and use it to perform the addition of two numbers. Write a method `add(int a, int b)` in the `Calculator` class and call it using an anonymous object.

2. **Anonymous Object for Circle Area:**

   - Write a Java program that uses an anonymous object to call a method that prints the area of a circle with a given radius.

3. **Anonymous Object for Greeting:**

   - Implement a program that defines a class `Greeting` with a method `sayHello()` that prints a greeting message. Use an anonymous object to call this method.

16. Need for Inheritance in Java

1. **Explain with an example** why inheritance is needed in Java. Write a program that demonstrates code reusability by creating a base class `Animal` with common properties and derived classes like `Dog` and `Cat`.

2. **Write a Java program** that shows how inheritance helps in method overriding. Create a base class `Vehicle` with a method `run()`. Create a derived class `Car` that overrides the `run()` method. Explain why method overriding is useful.

3. **Discuss the need for inheritance** by creating an example where you have a `Shape` class with a method `draw()`. Create subclasses like `Circle` and `Rectangle` that inherit from `Shape` and override the `draw()` method to provide specific implementations.

17. Inheritance in Java

1. **Create a base class `Person`** with attributes like `name` and `age`. Derive a subclass `Student` that adds an attribute `studentId` and a method `study()`. Write a program to demonstrate the concept of inheritance.

2. **Write a program** that demonstrates inheritance by creating a base class `Employee` with methods like `work()`. Derive two subclasses, `Manager` and `Developer`, each with its own unique method (`manageTeam()` and `writeCode()` respectively).

3. **Implement a class hierarchy** where a base class `Appliance` has a method `turnOn()`. Derive classes like `WashingMachine` and `Refrigerator` that inherit from `Appliance` and add specific methods such as `startWashCycle()` and `setTemperature()`.

18. Single and Multilevel Inheritance

1. **Single Inheritance**:

 - Write a Java program that demonstrates single inheritance by creating a base class `Animal` with a method `makeSound()` and a derived class `Dog` that inherits from `Animal` and has its own method `bark()`.

 - Create a base class `Shape` with a method `draw()`. Derive a subclass `Square` that inherits from `Shape` and has an additional method `calculateArea()`.

2. **Multilevel Inheritance**:

 - Write a program to demonstrate multilevel inheritance where:

  - Class `Vehicle` is the base class.

  - Class `Car` extends `Vehicle`.

  - Class `ElectricCar` extends `Car`.

- Each class should have its own method (`drive()`, `fuelType()`, `chargeBattery()`) to demonstrate multilevel inheritance.

- Implement a `University` class with a method `getDetails()`. Create a subclass `Department` that extends `University`, and a subclass `Course` that extends `Department`. Demonstrate how multilevel inheritance works by creating objects and calling methods from different levels.

19. Multiple Inheritance

- **Explain why multiple inheritance** is not supported in Java with an example.

Method Overriding

1. **Basic Method Overriding**:

- Create a base class called `Vehicle` with a method `void display()`. Create a subclass called `Car` that overrides the `display()` method to print different information. Instantiate both classes and call the `display()` method from each object.

2. **Calling Overridden Methods**:

- Create a class `Base` with a method `void greet()`. Override the `greet()` method in a subclass `Derived` but still want to call the superclass's version of `greet()`. Demonstrate how to achieve this.

Packages in Java

1. **Creating and Using Packages**:

- Create a package named `com.example.utils` with a class `Calculator` that contains a method `int add(int a, int b)`. Create another package named `com.example.main` with a

`Main` class. Use the `Calculator` class from the `utils` package in the `Main` class to perform addition.

2. **Access Control with Packages**:

  - Create two packages: `packageA` and `packageB`. In `packageA`, create a class `ClassA` with methods having different access levels (`public`, `protected`, and default). Create a class `ClassB` in `packageB` and demonstrate which methods of `ClassA` can be accessed.

3. **Importing Static Members**:

  - Create a package `mathoperations` with a class `MathUtils` that has a static method `int multiply(int a, int b)`. In another package, use a static import to directly call the `multiply` method without the class name.

`this` and `super` Keywords

1. **Using `this` to Call Constructors**:

  - Write a class `Person` with two constructors: a default constructor and a parameterized constructor that takes `name` and `age` as parameters. Use `this` to call the parameterized constructor from the default constructor.

2. **Using `this` to Refer to Instance Variables**:

  - Create a class `Employee` with instance variables `name` and `salary`. Write a constructor that initializes these variables using parameters with the same name as the instance variables. Use `this` to distinguish between the parameters and the instance variables.

3. **Using `super` to Call Parent Class Methods**:

- Write a base class `Shape` with a method `void draw()`. Create a subclass `Circle` that overrides the `draw()` method but still calls the `Shape` class's `draw()` method using `super`.

4. **Constructor Chaining with `super`**:

- Create a base class `Parent` with a parameterized constructor. Then, create a subclass `Child` that uses `super` to call the parent class's constructor from its own constructor. Demonstrate the order of constructor calls.

5. **Difference Between `this` and `super`**:

- Write a program to create two classes: `Parent` and `Child`. In the `Child` class, use both `this` and `super` to demonstrate how they differ in referring to class members (methods and variables).