MASTER JAVA WITH CODE

1. Print "Hello, World!"

java

CopyEdit

```
// This is a simple Java program to print "Hello, World!" on the screen.
public class HelloWorld {
    public static void main(String[] args) {
        // Print "Hello, World!" to the console
        System.out.println("Hello, World!");
    }
}
```

Explanation:

1. public class HelloWorld → Defines a class named HelloWorld.
2. public static void main(String[] args) → This is the **main method**, the entry point of the program.
3. System.out.println("Hello, World!"); → Prints "Hello, World!" on the screen.

---

2. Check if a number is even or odd

java

CopyEdit

```
import java.util.Scanner; // Import Scanner class for user input


public class EvenOddCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in); // Create Scanner object
        System.out.print("Enter a number: ");
        int num = sc.nextInt(); // Read integer input


        // Check if the number is even or odd
        if (num % 2 == 0) {
            System.out.println(num + " is Even.");
        } else {
            System.out.println(num + " is Odd.");
        }
        sc.close(); // Close scanner
    }
}
```

Explanation:

1. Scanner sc = new Scanner(System.in); → Reads input from the user.
2. num % 2 == 0 → If the remainder when divided by 2 is 0, it's even; otherwise, it's odd.

---

3. Swap two numbers without using a third variable

java

CopyEdit

```
import java.util.Scanner;

public class SwapNumbers {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter first number (a): ");
        int a = sc.nextInt();
        System.out.print("Enter second number (b): ");
        int b = sc.nextInt();

        System.out.println("Before swapping: a = " + a + ", b = " + b);

        // Swapping without using a third variable
        a = a + b;
        b = a - b;
        a = a - b;

        System.out.println("After swapping: a = " + a + ", b = " + b);
        sc.close();
    }
}
```

Explanation:

1. a = a + b; → Adds both numbers and stores in a.
2. b = a - b; → Now b gets the original value of a.
3. a = a - b; → a gets the original value of b.

---

4. Find the greatest among three numbers

java

CopyEdit

```
import java.util.Scanner;


public class GreatestNumber {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter first number: ");
      int a = sc.nextInt();
      System.out.print("Enter second number: ");
      int b = sc.nextInt();
      System.out.print("Enter third number: ");
      int c = sc.nextInt();

      // Check the greatest number
      int max = (a > b) ? (a > c ? a : c) : (b > c ? b : c);

      System.out.println("The greatest number is: " + max);
      sc.close();
   }
}
```

Explanation:

1. Uses a **ternary operator** to compare three numbers efficiently.

---

5. Print Fibonacci series up to N terms

java

CopyEdit

```
import java.util.Scanner;


public class FibonacciSeries {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter the number of terms: ");
      int n = sc.nextInt();


      int first = 0, second = 1; // First two numbers
      System.out.print("Fibonacci Series: " + first + " " + second);


      for (int i = 2; i < n; i++) {
         int next = first + second;
         System.out.print(" " + next);
         first = second;
         second = next;
      }
      sc.close();
   }
}
```

Explanation:

1. first and second are initialized as 0 and 1.
2. The loop generates the next term by adding previous two numbers.

---

6. Check if a number is prime

java

CopyEdit

```
import java.util.Scanner;

public class PrimeCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        boolean isPrime = true;
        if (num <= 1) {
            isPrime = false;
        } else {
            for (int i = 2; i <= Math.sqrt(num); i++) {
                if (num % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        }

        if (isPrime)
            System.out.println(num + " is Prime.");
        else
            System.out.println(num + " is not Prime.");

        sc.close();
    }
}
```

Explanation:

1. Check divisibility from 2 to √num for efficiency.

---

7. Find sum of digits of a number

java

CopyEdit

```
import java.util.Scanner;


public class SumOfDigits {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        int sum = 0;
        while (num > 0) {
            sum += num % 10; // Extract last digit and add to sum
            num /= 10; // Remove last digit
        }

        System.out.println("Sum of digits: " + sum);
        sc.close();
    }
}
```

---

8. Check if a number is an Armstrong number

java

CopyEdit

```
import java.util.Scanner;

public class ArmstrongNumber {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        int originalNum = num, sum = 0, digits = 0;

        int temp = num;
        while (temp > 0) {
            digits++;
            temp /= 10;
        }

        while (num > 0) {
            int digit = num % 10;
            sum += Math.pow(digit, digits);
            num /= 10;
        }

        if (sum == originalNum)
            System.out.println(originalNum + " is an Armstrong number.");
        else
            System.out.println(originalNum + " is not an Armstrong number.");

        sc.close();
    }
```

```
}
```

---

9. Find factorial using recursion

java

CopyEdit

```
import java.util.Scanner;

public class FactorialRecursion {
    public static int factorial(int n) {
        if (n == 0 || n == 1) return 1;
        return n * factorial(n - 1);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        System.out.println("Factorial of " + num + " is: " + factorial(num));
        sc.close();
    }
}
```

---

10. Check if a number is palindrome

java

CopyEdit

```
import java.util.Scanner;

public class PalindromeCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt(), reversed = 0, original = num;

        while (num > 0) {
            reversed = reversed * 10 + num % 10;
            num /= 10;
        }

        if (original == reversed)
            System.out.println(original + " is a palindrome.");
        else
            System.out.println(original + " is not a palindrome.");

        sc.close();
    }
}
```

Created by Rupesh Sawant

1. Classes and Objects in Java

A **class** is a blueprint for objects, and an **object** is an instance of a class.

java

CopyEdit

```
// Define a class named Car
class Car {
    // Attributes (Variables)
    String brand;
    int speed;

    // Method to display details
    void display() {
        System.out.println("Car Brand: " + brand);
        System.out.println("Car Speed: " + speed + " km/h");
    }
}

public class ClassObjectExample {
    public static void main(String[] args) {
        // Creating an object of Car
        Car myCar = new Car();
        myCar.brand = "Tesla";  // Assigning values to object properties
        myCar.speed = 200;

        // Calling method
        myCar.display();
    }
}
```

Theory & Calculation:

- A **class** contains data members (variables) and methods (functions).
- The **object** is used to access class members.

- Here, Car is a class, and myCar is an **object** that holds brand "Tesla" and speed 200 km/h.

---

2. Constructors in Java

Constructors are special methods used to initialize objects.

java

CopyEdit

```
class Student {
   String name;
   int age;

   // Default constructor
   Student() {
      System.out.println("Default Constructor Called");
      name = "Unknown";
      age = 18;
   }

   // Parameterized constructor
   Student(String name, int age) {
      System.out.println("Parameterized Constructor Called");
      this.name = name;
      this.age = age;
   }

   // Copy constructor
   Student(Student s) {
      System.out.println("Copy Constructor Called");
      this.name = s.name;
      this.age = s.age;
   }
```

```
    void display() {
        System.out.println("Student Name: " + name);
        System.out.println("Student Age: " + age);
    }
}

public class ConstructorExample {
    public static void main(String[] args) {
        Student s1 = new Student();  // Calls Default Constructor
        s1.display();

        Student s2 = new Student("Rupesh", 22);  // Calls Parameterized Constructor
        s2.display();

        Student s3 = new Student(s2);  // Calls Copy Constructor
        s3.display();
    }
}
```

Theory & Calculation:

- **Default Constructor:** No parameters, assigns default values.
- **Parameterized Constructor:** Takes arguments and initializes object.
- **Copy Constructor:** Creates a new object with the same values as another object.

---

3. Method Overloading and Overriding

Method Overloading (Compile-time Polymorphism)

java

CopyEdit

```
class MathOperations {
    // Method with two integers
    int add(int a, int b) {
        return a + b;
    }

    // Method with three integers
    int add(int a, int b, int c) {
        return a + b + c;
    }

    // Method with double values
    double add(double a, double b) {
        return a + b;
    }
}

public class MethodOverloading {
    public static void main(String[] args) {
        MathOperations obj = new MathOperations();
        System.out.println("Addition (int, int): " + obj.add(5, 10));
        System.out.println("Addition (int, int, int): " + obj.add(5, 10, 15));
        System.out.println("Addition (double, double): " + obj.add(5.5, 2.3));
    }
}
```

Method Overriding (Runtime Polymorphism)

java

CopyEdit

```
class Parent {
    void show() {
        System.out.println("Parent class method");
    }
}

class Child extends Parent {
    @Override
    void show() {
        System.out.println("Child class method (Overriding Parent method)");
    }
}

public class MethodOverriding {
    public static void main(String[] args) {
        Parent obj = new Child();  // Runtime polymorphism
        obj.show();  // Calls Child class method
    }
}
```

Theory & Calculation:

- **Overloading:** Methods have the same name but different parameters.
- **Overriding:** A subclass provides a new implementation of a method from the superclass.

---

4. Inheritance in Java

Inheritance allows a child class to use properties and methods of a parent class.

Single Inheritance

java

CopyEdit

```
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}

public class SingleInheritance {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();  // Inherited method
        dog.bark();
    }
}
```

5. Abstract Class in Java

An **abstract class** cannot be instantiated; it can have abstract and non-abstract methods.

java

CopyEdit

```
abstract class Vehicle {
    abstract void start();  // Abstract method

    void fuel() {
        System.out.println("Filling fuel...");
    }
}

class Car extends Vehicle {
    @Override
    void start() {
        System.out.println("Car starts with a key.");
    }
}

public class AbstractExample {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.start();  // Abstract method implemented
        myCar.fuel();
    }
}
```

---

6. Multiple Inheritance Using Interfaces

java

CopyEdit

```
interface A {
   void methodA();
}

interface B {
   void methodB();
}

class C implements A, B {
   public void methodA() {
      System.out.println("Method A from Interface A");
   }

   public void methodB() {
      System.out.println("Method B from Interface B");
   }
}

public class MultipleInheritance {
   public static void main(String[] args) {
      C obj = new C();
      obj.methodA();
      obj.methodB();
   }
}
```

7. Encapsulation with Getters & Setters

java

CopyEdit

```
class Person {
    private String name;  // Private variable

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

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

public class EncapsulationExample {
    public static void main(String[] args) {
        Person p = new Person();
        p.setName("Rupesh");
        System.out.println("Person Name: " + p.getName());
    }
}
```

8. Static Methods and Variables

java

CopyEdit

```
class Counter {
    static int count = 0;  // Static variable

    Counter() {
        count++;  // Increment static variable
    }

    static void displayCount() {
        System.out.println("Total Objects Created: " + count);
    }
}

public class StaticExample {
    public static void main(String[] args) {
        new Counter();
        new Counter();
        new Counter();

        Counter.displayCount();  // Static method call
    }
}
```

---

9. ‘super’ and ‘this’ Keywords

java

CopyEdit

```
class Parent {
   int x = 10;
}

class Child extends Parent {
   int x = 20;

   void show() {
      System.out.println("Child x: " + this.x);
      System.out.println("Parent x: " + super.x);
   }
}

public class SuperThisExample {
   public static void main(String[] args) {
      Child obj = new Child();
      obj.show();
   }
}
```

---

10. Polymorphism (Compile-time and Runtime)

See **Method Overloading** (Compile-time) and **Method Overriding** (Runtime).

1. Handling ArithmeticException and ArrayIndexOutOfBoundsException

Theory:

- **ArithmeticException** occurs when we divide by zero.
- **ArrayIndexOutOfBoundsException** occurs when accessing an index outside the array's range.

java

CopyEdit

```
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        try {
            // ArithmeticException (Division by zero)
            int a = 10, b = 0;
            int result = a / b;  // This will throw ArithmeticException
            System.out.println("Result: " + result);

            // ArrayIndexOutOfBoundsException
            int[] arr = {1, 2, 3};
            System.out.println(arr[5]);  // Invalid index
        }
        catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero!");
        }
        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Array index out of bounds!");
        }
    }
}
```

2. Implementing a Custom Exception

Theory:

A **custom exception** is created by extending Exception or RuntimeException.

java

CopyEdit

```
// Custom Exception Class
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

public class CustomExceptionExample {
    static void checkAge(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age must be 18 or above.");
        } else {
            System.out.println("Access granted!");
        }
    }

    public static void main(String[] args) {
        try {
            checkAge(16); // Throws exception
        } catch (InvalidAgeException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

3. Try, Catch, Finally, and Throws Keywords

Theory:

- **try:** Code that may cause an exception.
- **catch:** Handles the exception.
- **finally:** Executes always.
- **throws:** Declares exceptions in methods.

java

CopyEdit

```
public class ExceptionExample {
   static void divide(int a, int b) throws ArithmeticException {
      System.out.println("Result: " + (a / b));
   }

   public static void main(String[] args) {
      try {
         divide(10, 0); // Throws exception
      } catch (ArithmeticException e) {
         System.out.println("Caught Exception: " + e.getMessage());
      } finally {
         System.out.println("Finally block executed.");
      }
   }
}
```

---

4. Creating Multiple Threads

java

CopyEdit

```
class MyThread extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(Thread.currentThread().getName() + " - " + i);
        }
    }
}

public class MultiThreadExample {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        MyThread t2 = new MyThread();

        t1.start();
        t2.start();
    }
}
```

---

5. Thread Synchronization

Theory:

Ensures only one thread accesses a shared resource at a time.

java

CopyEdit

```
class Table {
   synchronized void printTable(int n) {  // Synchronized method
      for (int i = 1; i <= 5; i++) {
         System.out.println(n * i);
         try {
            Thread.sleep(500);
         } catch (InterruptedException e) {
            System.out.println(e);
         }
      }
   }
}

public class SynchronizationExample {
   public static void main(String[] args) {
      Table obj = new Table();

      Thread t1 = new Thread(() -> obj.printTable(5));
      Thread t2 = new Thread(() -> obj.printTable(10));

      t1.start();
      t2.start();
   }
}
```

6. Inter-Thread Communication using wait(), notify(), notifyAll()

java

CopyEdit

```
class SharedResource {
   private boolean available = false;

   synchronized void produce() {
      try {
         while (available) {
            wait(); // Wait if resource is available
         }
         System.out.println("Producing...");
         available = true;
         notify(); // Notify consumer
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
   }

   synchronized void consume() {
      try {
         while (!available) {
            wait(); // Wait if no resource
         }
         System.out.println("Consuming...");
         available = false;
         notify(); // Notify producer
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
   }
```

```
}

public class InterThreadExample {
    public static void main(String[] args) {
        SharedResource resource = new SharedResource();

        new Thread(resource::produce).start();
        new Thread(resource::consume).start();
    }
}
```

---

7. Using Executor Framework

java

CopyEdit

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class ExecutorExample {
    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(3);

        for (int i = 1; i <= 5; i++) {
            int taskId = i;
            executor.execute(() -> {
                System.out.println("Executing Task: " + taskId);
            });
        }

        executor.shutdown();
    }
}
```

---

8. Demonstrating Deadlock in Java

java

CopyEdit

```
class Resource {
   void methodA(Resource b) {
      synchronized (this) {
         System.out.println("Method A Locked by " + Thread.currentThread().getName());
         synchronized (b) {
            System.out.println("Method A trying to access Method B");
         }
      }
   }

   void methodB(Resource a) {
      synchronized (this) {
         System.out.println("Method B Locked by " + Thread.currentThread().getName());
         synchronized (a) {
            System.out.println("Method B trying to access Method A");
         }
      }
   }
}

public class DeadlockExample {
   public static void main(String[] args) {
      Resource r1 = new Resource();
      Resource r2 = new Resource();

      Thread t1 = new Thread(() -> r1.methodA(r2));
      Thread t2 = new Thread(() -> r2.methodB(r1));
```

```
        t1.start();
        t2.start();
    }
}
```

---

9. Using Callable and Future

java

CopyEdit

```
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

class MyTask implements Callable<Integer> {
    public Integer call() {
        return 10 + 20;
    }
}

public class CallableExample {
    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newFixedThreadPool(2);
        Future<Integer> result = executor.submit(new MyTask());

        System.out.println("Computed Result: " + result.get());
        executor.shutdown();
    }
}
```

---

10. Producer-Consumer Problem using BlockingQueue

java

CopyEdit

```
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;

class Producer implements Runnable {
    private BlockingQueue<Integer> queue;

    Producer(BlockingQueue<Integer> queue) {
        this.queue = queue;
    }

    public void run() {
        try {
            for (int i = 1; i <= 5; i++) {
                System.out.println("Produced: " + i);
                queue.put(i);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class Consumer implements Runnable {
    private BlockingQueue<Integer> queue;

    Consumer(BlockingQueue<Integer> queue) {
        this.queue = queue;
```

```
    }

    public void run() {
        try {
            while (true) {
                int item = queue.take();
                System.out.println("Consumed: " + item);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class ProducerConsumerExample {
    public static void main(String[] args) {
        BlockingQueue<Integer> queue = new LinkedBlockingQueue<>(3);

        new Thread(new Producer(queue)).start();
        new Thread(new Consumer(queue)).start();
    }
}
```

Created by Rupesh Sawant

1. Implementing ArrayList and Performing Basic Operations

Theory:

- ArrayList is a **dynamic array** that allows **duplicate elements**.
- It provides **fast random access** (O(1) for get/set operations).
- It **automatically resizes** when needed.

java

CopyEdit

```
import java.util.ArrayList;


public class ArrayListExample {
    public static void main(String[] args) {
        // Creating an ArrayList
        ArrayList<String> list = new ArrayList<>();

        // Adding elements
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        // Displaying the list
        System.out.println("ArrayList: " + list);

        // Removing an element
        list.remove("Banana");
        System.out.println("After removing 'Banana': " + list);

        // Getting an element by index
        System.out.println("Element at index 1: " + list.get(1));

        // Checking size
        System.out.println("Size of ArrayList: " + list.size());
```

```
    }
}
```

Output:

yaml

CopyEdit

ArrayList: [Apple, Banana, Cherry]

After removing 'Banana': [Apple, Cherry]

Element at index 1: Cherry

Size of ArrayList: 2

---

2. Implementing LinkedList: Adding & Removing Elements

Theory:

- **LinkedList** is implemented using **doubly linked nodes**.
- **Insertion & deletion** (O(1)) is faster than ArrayList.
- **Random access** is **slower** than ArrayList.

java

CopyEdit

```
import java.util.LinkedList;

public class LinkedListExample {
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<>();

        // Adding elements
        list.add(10);
        list.add(20);
        list.addFirst(5);  // Adds at the beginning
        list.addLast(30);   // Adds at the end

        System.out.println("LinkedList: " + list);
```

```
        // Removing elements
        list.removeFirst(); // Removes first element
        list.removeLast();  // Removes last element

        System.out.println("After removals: " + list);
    }
}
```

Output:

yaml

CopyEdit

LinkedList: [5, 10, 20, 30]

After removals: [10, 20]

---

3. Demonstrating HashSet & TreeSet (Unique Elements)

Theory:

- HashSet: **Unordered, unique elements, fast access (O(1)).**
- TreeSet: **Sorted, unique elements (O(log n)).**

java

CopyEdit

```
import java.util.HashSet;
import java.util.TreeSet;

public class SetExample {
    public static void main(String[] args) {
        // HashSet Example
        HashSet<Integer> hashSet = new HashSet<>();
        hashSet.add(40);
        hashSet.add(10);
        hashSet.add(20);
        hashSet.add(40); // Duplicate ignored
```

```
        System.out.println("HashSet (Unordered): " + hashSet);

        // TreeSet Example
        TreeSet<Integer> treeSet = new TreeSet<>(hashSet);
        System.out.println("TreeSet (Sorted): " + treeSet);
    }
}
```

Output:

```
less
CopyEdit
HashSet (Unordered): [40, 10, 20]
TreeSet (Sorted): [10, 20, 40]
```

---

4. Creating a HashMap (Student ID & Names)

```
java
CopyEdit
import java.util.HashMap;

public class HashMapExample {
    public static void main(String[] args) {
        HashMap<Integer, String> students = new HashMap<>();

        // Adding key-value pairs
        students.put(101, "Alice");
        students.put(102, "Bob");
        students.put(103, "Charlie");

        System.out.println("Student HashMap: " + students);

        // Retrieving a value
        System.out.println("Student with ID 102: " + students.get(102));
```

```
        // Removing an entry
        students.remove(101);
        System.out.println("After removal: " + students);
    }
}
```

Output:

yaml

CopyEdit

```
Student HashMap: {101=Alice, 102=Bob, 103=Charlie}
Student with ID 102: Bob
After removal: {102=Bob, 103=Charlie}
```

---

5. Sorting a TreeMap Based on Keys

java

CopyEdit

```
import java.util.TreeMap;

public class TreeMapExample {
    public static void main(String[] args) {
        TreeMap<Integer, String> map = new TreeMap<>();

        map.put(3, "Banana");
        map.put(1, "Apple");
        map.put(2, "Cherry");

        System.out.println("Sorted TreeMap: " + map);
    }
}
```

Output:

yaml

Created by Rupesh Sawant

CopyEdit

Sorted TreeMap: {1=Apple, 2=Cherry, 3=Banana}

---

6. Implementing PriorityQueue

java

CopyEdit

```
import java.util.PriorityQueue;


public class PriorityQueueExample {
   public static void main(String[] args) {
      PriorityQueue<Integer> pq = new PriorityQueue<>();


      pq.add(30);
      pq.add(10);
      pq.add(20);


      System.out.println("Priority Queue: " + pq);
      System.out.println("Poll: " + pq.poll()); // Removes smallest element
   }
}
```

Output:

yaml

CopyEdit

Priority Queue: [10, 30, 20]

Poll: 10

---

7. Converting an ArrayList to an Array

java

CopyEdit

```
import java.util.ArrayList;
```

```
public class ConvertArrayListToArray {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");

        String[] array = list.toArray(new String[0]);

        System.out.println("Converted Array: " + String.join(", ", array));
    }
}
```

Output:

javascript

CopyEdit

Converted Array: Apple, Banana

---

8. Implementing Comparator & Comparable

java

CopyEdit

```
import java.util.*;

class Student implements Comparable<Student> {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int compareTo(Student s) {
```

```
        return this.id - s.id;
    }
}

public class ComparableExample {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student(103, "Charlie"));
        students.add(new Student(101, "Alice"));
        students.add(new Student(102, "Bob"));

        Collections.sort(students);

        for (Student s : students) {
            System.out.println(s.id + " - " + s.name);
        }
    }
}
```

Output:

CopyEdit

101 - Alice

102 - Bob

103 - Charlie

---

9. Removing Duplicates from an ArrayList

java

CopyEdit

```
import java.util.*;

public class RemoveDuplicates {
    public static void main(String[] args) {
```

```
        ArrayList<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3, 2, 4, 1));
        HashSet<Integer> set = new HashSet<>(list);

        System.out.println("List without duplicates: " + set);
    }
}
```

Output:

less

CopyEdit

List without duplicates: [1, 2, 3, 4]

---

10. Implementing Stack & Queue Using Collections

java

CopyEdit

```
import java.util.*;

public class StackQueueExample {
    public static void main(String[] args) {
        // Stack
        Stack<Integer> stack = new Stack<>();
        stack.push(10);
        stack.push(20);
        System.out.println("Stack Pop: " + stack.pop());

        // Queue
        Queue<Integer> queue = new LinkedList<>();
        queue.add(10);
        queue.add(20);
        System.out.println("Queue Poll: " + queue.poll());
    }
}
```

Output:

yaml

CopyEdit

Stack Pop: 20

Queue Poll: 10

1. Create a Text File and Write Data into It

Theory:

- **FileWriter** allows writing data to a file.
- **BufferedWriter** improves efficiency by buffering the output.

java

CopyEdit

```
import java.io.FileWriter;
import java.io.IOException;

public class FileWriteExample {
    public static void main(String[] args) {
        try {
            // Creating FileWriter to write data to "output.txt"
            FileWriter writer = new FileWriter("output.txt");

            // Writing content to the file
            writer.write("Hello, this is a sample text file.\nWelcome to Java File Handling!");

            // Closing the writer to save changes
            writer.close();
            System.out.println("File created and data written successfully.");
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
```

}

Output (in console):

arduino

CopyEdit

File created and data written successfully.

Output (in output.txt file):

arduino

CopyEdit

Hello, this is a sample text file.

Welcome to Java File Handling!

---

2. Read Content from a File Using BufferedReader

Theory:

- **BufferedReader** allows efficient reading of text files line by line.

java

CopyEdit

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReadExample {
    public static void main(String[] args) {
        try {
            // Creating FileReader to read "output.txt"
            FileReader reader = new FileReader("output.txt");
            BufferedReader br = new BufferedReader(reader);

            String line;
            // Reading the file line by line
            while ((line = br.readLine()) != null) {
                System.out.println(line);
```

```
        }

        // Closing resources
        br.close();
      } catch (IOException e) {
        System.out.println("Error reading file: " + e.getMessage());
      }
   }
}
```

Output (in console):

arduino

CopyEdit

Hello, this is a sample text file.

Welcome to Java File Handling!

---

3. Copy Content from One File to Another

Theory:

- **FileReader & FileWriter** are used for reading and writing.
- **Efficient way** to copy content from one file to another.

java

CopyEdit

```
import java.io.*;

public class FileCopyExample {
   public static void main(String[] args) {
      try {
        // Source file to read
        FileReader fr = new FileReader("output.txt");
        BufferedReader br = new BufferedReader(fr);

        // Destination file to write
```

```
            FileWriter fw = new FileWriter("copy.txt");

            String line;
            while ((line = br.readLine()) != null) {
                fw.write(line + "\n");
            }

            br.close();
            fw.close();
            System.out.println("File copied successfully!");
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output (in console):

```
arduino
CopyEdit
File copied successfully!
```

Output (in copy.txt file):

```
arduino
CopyEdit
Hello, this is a sample text file.
Welcome to Java File Handling!
```

4. List All Files in a Directory

Theory:

- **File class** allows directory traversal.

java

CopyEdit

```
import java.io.File;


public class ListFilesExample {
    public static void main(String[] args) {
        // Specify directory path
        File directory = new File("C:/Users/YourUsername/Documents");


        // Get list of files
        File[] files = directory.listFiles();


        if (files != null) {
            for (File file : files) {
                System.out.println(file.getName());
            }
        } else {
            System.out.println("Directory not found!");
        }
    }
}
```

Output (in console, depending on the directory content):

CopyEdit

```
file1.txt
file2.docx
project.java
```

---

5. Implement Serialization and Deserialization

Created by Rupesh Sawant

Theory:

- **Serialization** converts an object into a **byte stream** for storage.
- **Deserialization** reconstructs the object from the byte stream.

java

CopyEdit

```
import java.io.*;

// Class must implement Serializable
class Person implements Serializable {
   String name;
   int age;

   Person(String name, int age) {
      this.name = name;
      this.age = age;
   }
}

public class SerializationExample {
   public static void main(String[] args) {
      try {
         // Creating an object
         Person p1 = new Person("John", 25);

         // Serializing the object
         FileOutputStream fos = new FileOutputStream("person.ser");
         ObjectOutputStream oos = new ObjectOutputStream(fos);
         oos.writeObject(p1);
         oos.close();
         System.out.println("Object Serialized!");
```

```
            // Deserializing the object
            FileInputStream fis = new FileInputStream("person.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            Person p2 = (Person) ois.readObject();
            ois.close();

            System.out.println("Deserialized Object: " + p2.name + ", " + p2.age);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Output (in console):

```
css
CopyEdit
Object Serialized!
Deserialized Object: John, 25
```

---

6. Check If a File Exists or Not

Theory:

- **exists() method** checks file existence.

java

CopyEdit

```
import java.io.File;


public class FileExistsExample {
   public static void main(String[] args) {
      // Create File object
      File file = new File("output.txt");


      // Check if file exists
      if (file.exists()) {
         System.out.println("File exists.");
      } else {
         System.out.println("File does not exist.");
      }
   }
}
```

Output (if output.txt exists):

arduino

CopyEdit

```
File exists.
```

---

7. Append Data to an Existing File

Theory:

- **FileWriter(file, true)** allows appending.

java

CopyEdit

```
import java.io.FileWriter;
import java.io.IOException;


public class FileAppendExample {
   public static void main(String[] args) {
      try {
         FileWriter fw = new FileWriter("output.txt", true);


         // Appending text
         fw.write("\nThis is appended text.");
         fw.close();
         System.out.println("Data appended successfully.");
      } catch (IOException e) {
         System.out.println("Error: " + e.getMessage());
      }
   }
}
```

Output (in output.txt file):

vbnet

CopyEdit

```
Hello, this is a sample text file.
Welcome to Java File Handling!
This is appended text.
```

Created by Rupesh Sawant

JDBC (Java Database Connectivity)

Introduction to JDBC

- JDBC is an **API (Application Programming Interface)** that allows Java applications to interact with relational databases such as **MySQL, PostgreSQL, Oracle, etc.**
- It provides classes and interfaces to establish a connection, execute SQL queries, and process the results.
- **Steps to use JDBC:**
    1. Load the **JDBC Driver**.
    2. Establish a **Connection** to the database.
    3. Create a **Statement** or **PreparedStatement**.
    4. Execute SQL queries.
    5. Retrieve and process the **ResultSet**.
    6. Close the **Connection**.

---

1. Connect Java with MySQL Database

Theory

- Before writing JDBC code, we need:
    - **MySQL Database** installed.
    - **JDBC Driver (MySQL Connector JAR)** → Download from: MySQL Connector
    - Add JAR to your project (Eclipse/IntelliJ → Right-click on Project → Build Path → Add External JARs).

Code: Connect Java to MySQL

java

CopyEdit

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;


public class DatabaseConnection {

   public static void main(String[] args) {

      // Database URL, Username, and Password
```

```
    String url = "jdbc:mysql://localhost:3306/mydatabase"; // Change "mydatabase" to your database name
    String user = "root";  // Change to your MySQL username
    String password = "password";  // Change to your MySQL password

    try {
        // Step 1: Load MySQL JDBC Driver (Optional for modern versions)
        Class.forName("com.mysql.cj.jdbc.Driver");

        // Step 2: Establish Connection
        Connection conn = DriverManager.getConnection(url, user, password);

        if (conn != null) {
            System.out.println("Connected to the database successfully!");
        }

        // Step 3: Close Connection
        conn.close();
    } catch (ClassNotFoundException e) {
        System.out.println("JDBC Driver not found: " + e.getMessage());
    } catch (SQLException e) {
        System.out.println("Connection failed: " + e.getMessage());
    }
  }
}
```

Explanation (Line-by-Line)

1. **Import JDBC classes** (Connection, DriverManager, SQLException).
2. **Define database connection parameters** (URL, username, password).
3. **Load MySQL JDBC driver** (This step is optional for newer Java versions).
4. **Create a connection using DriverManager.getConnection()**.

5. **Print a success message if connected**.
6. **Close the connection using conn.close()**.

Expected Output (if successful)

pgsql

CopyEdit

Connected to the database successfully!

Output (if database is down)

pgsql

CopyEdit

Connection failed: Access denied for user 'root'@'localhost' (using password: YES)

---

2. Insert Records into a Database Using JDBC

Theory

- We use the **INSERT** SQL statement to add records into a table.
- The Statement or PreparedStatement interface executes the query.
- **Example Table: students**

sql

CopyEdit

```
CREATE TABLE students (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(50),
   age INT
);
```

Code: Insert Data into MySQL Table

java

CopyEdit

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
```

```
import java.sql.SQLException;

public class InsertRecord {
   public static void main(String[] args) {
      String url = "jdbc:mysql://localhost:3306/mydatabase";
      String user = "root";
      String password = "password";

      try {
         // Step 1: Establish Connection
         Connection conn = DriverManager.getConnection(url, user, password);

         // Step 2: Prepare SQL Query
         String query = "INSERT INTO students (name, age) VALUES (?, ?)";
         PreparedStatement pstmt = conn.prepareStatement(query);

         // Step 3: Set Values
         pstmt.setString(1, "John Doe"); // First parameter: name
         pstmt.setInt(2, 22); // Second parameter: age

         // Step 4: Execute Update
         int rowsInserted = pstmt.executeUpdate();

         if (rowsInserted > 0) {
            System.out.println("Record inserted successfully!");
         }

         // Step 5: Close Connection
         pstmt.close();
         conn.close();
      } catch (SQLException e) {
```

```
            System.out.println("Error inserting record: " + e.getMessage());
        }
    }
}
```

Explanation

1. **Establish a connection** to the database.
2. **Prepare an SQL query** using PreparedStatement (for security).
3. **Set values dynamically** using pstmt.setString() and pstmt.setInt().
4. **Execute the query** using executeUpdate().
5. **Print success message** if record is inserted.

Expected Output (if successful)

mathematica

CopyEdit

Record inserted successfully!

Database Table After Execution

id	name	age
1	John Doe	22

---

3. Fetch Records from Database

Theory

- The SELECT query retrieves records from a table.
- The ResultSet object stores the results.

Code: Retrieve Data from MySQL Table

java

CopyEdit

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

Created by Rupesh Sawant

```
import java.sql.SQLException;
import java.sql.Statement;

public class FetchRecords {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String user = "root";
        String password = "password";

        try {
            // Step 1: Establish Connection
            Connection conn = DriverManager.getConnection(url, user, password);

            // Step 2: Create Statement
            Statement stmt = conn.createStatement();

            // Step 3: Execute SELECT Query
            String query = "SELECT * FROM students";
            ResultSet rs = stmt.executeQuery(query);

            // Step 4: Process the ResultSet
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                int age = rs.getInt("age");
                System.out.println("ID: " + id + ", Name: " + name + ", Age: " + age);
            }

            // Step 5: Close Resources
            rs.close();
            stmt.close();
```

```
            conn.close();
        } catch (SQLException e) {
            System.out.println("Error fetching records: " + e.getMessage());
        }
    }
}
```

Expected Output

yaml

CopyEdit

ID: 1, Name: John Doe, Age: 22

4. Update and Delete Records in the Database

Theory

- **UPDATE Statement** modifies existing records in the table.
- **DELETE Statement** removes records from the table.
- Both use PreparedStatement for secure execution.

Code: Update a Record in MySQL Table

java

CopyEdit

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class UpdateRecord {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String user = "root";
        String password = "password";
```

```
        try {
            // Step 1: Establish Connection
            Connection conn = DriverManager.getConnection(url, user, password);

            // Step 2: Create SQL Update Query
            String query = "UPDATE students SET age = ? WHERE name = ?";
            PreparedStatement pstmt = conn.prepareStatement(query);

            // Step 3: Set New Values
            pstmt.setInt(1, 25);  // Set age = 25
            pstmt.setString(2, "John Doe");  // Update record where name = John Doe

            // Step 4: Execute Update
            int rowsUpdated = pstmt.executeUpdate();

            if (rowsUpdated > 0) {
                System.out.println("Record updated successfully!");
            }

            // Step 5: Close Connection
            pstmt.close();
            conn.close();
        } catch (SQLException e) {
            System.out.println("Error updating record: " + e.getMessage());
        }
    }
}
```

Explanation

1. Establish a **connection** to the database.
2. Create an **UPDATE query** with placeholders (?).
3. Use pstmt.setInt() and pstmt.setString() to replace placeholders.
4. Execute the update using executeUpdate().
5. Print a **success message** if update is successful.

Expected Output

mathematica

CopyEdit

Record updated successfully!

Code: Delete a Record from MySQL Table

java

CopyEdit

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class DeleteRecord {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String user = "root";
        String password = "password";

        try {
            // Step 1: Establish Connection
            Connection conn = DriverManager.getConnection(url, user, password);

            // Step 2: Create SQL DELETE Query
```

```
        String query = "DELETE FROM students WHERE name = ?";
        PreparedStatement pstmt = conn.prepareStatement(query);

        // Step 3: Set Value
        pstmt.setString(1, "John Doe");

        // Step 4: Execute Delete
        int rowsDeleted = pstmt.executeUpdate();

        if (rowsDeleted > 0) {
            System.out.println("Record deleted successfully!");
        }

        // Step 5: Close Connection
        pstmt.close();
        conn.close();
    } catch (SQLException e) {

        System.out.println("Error deleting record: " + e.getMessage());
    }
  }
}
```

Expected Output

mathematica

CopyEdit

Record deleted successfully!

---

5. Prevent SQL Injection Using Prepared Statements

Theory

- **SQL Injection** is a hacking technique where an attacker inserts malicious SQL queries.
- **Example of vulnerable code:**

java

CopyEdit

Statement stmt = conn.createStatement();

String query = "SELECT * FROM users WHERE username = '" + userInput + "'";

ResultSet rs = stmt.executeQuery(query);

If userInput = "' OR '1'='1" is provided, it bypasses authentication.

- **Solution:** Use PreparedStatement to **prevent direct query manipulation**.

Secure Code Using PreparedStatement

java

CopyEdit

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;


public class SecureLogin {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String user = "root";
        String password = "password";

        String inputUsername = "admin";
        String inputPassword = "admin123";

        try {
```

```
        // Step 1: Establish Connection
        Connection conn = DriverManager.getConnection(url, user, password);

        // Step 2: Secure SQL Query
        String query = "SELECT * FROM users WHERE username = ? AND password = ?";
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setString(1, inputUsername);
        pstmt.setString(2, inputPassword);

        // Step 3: Execute Query
        ResultSet rs = pstmt.executeQuery();

        if (rs.next()) {
            System.out.println("Login successful!");
        } else {
            System.out.println("Invalid credentials!");
        }

        // Step 4: Close Resources
        rs.close();
        pstmt.close();
        conn.close();
    } catch (SQLException e) {
        System.out.println("Error: " + e.getMessage());
    }
  }
}
```

Expected Output (For Correct Credentials)

nginx

CopyEdit

Login successful!

Expected Output (For Incorrect Credentials)

nginx

CopyEdit

Invalid credentials!

---

6. Implement Transaction Management in JDBC

Theory

- A **transaction** is a group of SQL statements executed as a single unit.
- **ACID Properties** ensure:
    - **Atomicity:** Either all or no operations occur.
    - **Consistency:** Data remains correct.
    - **Isolation:** Transactions don't interfere.
    - **Durability:** Changes are permanent.
- Use **commit()** to save changes, **rollback()** to undo.

Code: Transaction in JDBC

java

CopyEdit

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class TransactionExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String user = "root";
        String password = "password";

        try {
```

```
        // Step 1: Establish Connection
        Connection conn = DriverManager.getConnection(url, user, password);

        // Step 2: Disable Auto-Commit
        conn.setAutoCommit(false);

        // Step 3: Execute First Query (Insert Student)
        String query1 = "INSERT INTO students (name, age) VALUES (?, ?)";
        PreparedStatement pstmt1 = conn.prepareStatement(query1);
        pstmt1.setString(1, "Alice");
        pstmt1.setInt(2, 21);
        pstmt1.executeUpdate();

        // Step 4: Execute Second Query (Insert Student)
        String query2 = "INSERT INTO students (name, age) VALUES (?, ?)";
        PreparedStatement pstmt2 = conn.prepareStatement(query2);
        pstmt2.setString(1, "Bob");
        pstmt2.setInt(2, 23);
        pstmt2.executeUpdate();

        // Step 5: Commit Transaction
        conn.commit();
        System.out.println("Transaction successful!");

        // Step 6: Close Connections
        pstmt1.close();
        pstmt2.close();
        conn.close();
    } catch (SQLException e) {
        System.out.println("Transaction failed! Rolling back...");
        e.printStackTrace();
```

```
        }
    }
}
```

Expected Output

nginx

CopyEdit

Transaction successful!

If an error occurs, rollback is executed:

nginx

CopyEdit

Transaction failed! Rolling back...

7. Servlets & JSP

1. Create a Simple Servlet and Deploy on Tomcat

Steps:

1. Install **Apache Tomcat** (e.g., C:\Tomcat).
2. Place Java files in webapps/MyApp/WEB-INF/classes/.
3. Configure web.xml.
4. Deploy in Tomcat and access via **http://localhost:8080/MyApp/HelloServlet**.

Code: Simple Servlet (HelloServlet.java)

java

CopyEdit

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

// Annotation-based Servlet Mapping
@WebServlet("/HelloServlet")
public class HelloServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

   // Handles GET request
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html"); // Set response content type
      PrintWriter out = response.getWriter(); // Get writer to print response

      // Print Hello Message
```

```
        out.println("<html><body>");
        out.println("<h2>Hello, World! This is a Simple Servlet.</h2>");
        out.println("</body></html>");
    }
}
```

---

web.xml Configuration

xml

CopyEdit

```
<web-app>
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/HelloServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Output:

URL: http://localhost:8080/MyApp/HelloServlet

csharp

CopyEdit

Hello, World! This is a Simple Servlet.

---

2. Implement doGet() and doPost()

Code: Handling GET & POST Requests (FormServlet.java)

java

CopyEdit

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/FormServlet")
public class FormServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

   // Handles GET Request
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      // Display form for POST submission
      out.println("<html><body>");
      out.println("<form action='FormServlet' method='POST'>");
      out.println("Name: <input type='text' name='username'><br>");
      out.println("<input type='submit' value='Submit'>");
      out.println("</form>");
      out.println("</body></html>");
   }
```

```
    // Handles POST Request
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Get user input
        String name = request.getParameter("username");

        out.println("<html><body>");
        out.println("<h2>Welcome, " + name + "!</h2>");
        out.println("</body></html>");
    }
}
```

Output:

1. Open **http://localhost:8080/MyApp/FormServlet**
2. Enter "Rupesh" and Submit.
3. **Response:**

CopyEdit

Welcome, Rupesh!

---

3. Session Management Using HttpSession

Code: LoginServlet.java

java

CopyEdit

```
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String username = request.getParameter("username");
      HttpSession session = request.getSession(); // Create session
      session.setAttribute("user", username); // Store user data

      out.println("<html><body>");
      out.println("<h2>Login Successful! Welcome, " + username + ".</h2>");
      out.println("<a href='DashboardServlet'>Go to Dashboard</a>");
      out.println("</body></html>");
   }
}
```

Code: DashboardServlet.java

java

CopyEdit

```
@WebServlet("/DashboardServlet")
public class DashboardServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
```

```
        HttpSession session = request.getSession(false);
        String user = (String) session.getAttribute("user");

        if (user != null) {
            out.println("<h2>Welcome, " + user + "! This is your dashboard.</h2>");
            out.println("<a href='LogoutServlet'>Logout</a>");
        } else {
            out.println("No session found! Please login.");
        }
    }
}
```

Output:

1. **Login (POST)** → "Welcome, Rupesh!"
2. **Dashboard (GET)** → "Welcome, Rupesh! This is your dashboard."
3. **Logout (GET)** → "You have logged out successfully."

---

4. MVC Architecture Using Servlets & JSP

Code: index.jsp (View)

jsp

CopyEdit

```
<form action="LoginController" method="POST">
    Username: <input type="text" name="username">
    <input type="submit" value="Login">
</form>
```

Code: LoginController.java (Controller)

java

CopyEdit

```
@WebServlet("/LoginController")
public class LoginController extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
        throws ServletException, IOException {
    String username = request.getParameter("username");

    if (username.equals("admin")) {
        request.setAttribute("message", "Login Successful");
    } else {
        request.setAttribute("message", "Invalid User");
    }

    request.getRequestDispatcher("dashboard.jsp").forward(request, response);
  }
}
```

Code: dashboard.jsp (View)

jsp

CopyEdit

```
<%= request.getAttribute("message") %>
```

Output:

- **Enter "admin"** → "Login Successful"
- **Enter "user"** → "Invalid User"

---

8. Hibernate (ORM Framework)

1. Hibernate Configuration & Connection

hibernate.cfg.xml

xml

CopyEdit

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mydb</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">root</property>
```

```
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
    </session-factory>
</hibernate-configuration>
```

---

2. Hibernate CRUD Operations

Code: Student.java (Entity)

java

CopyEdit

```
@Entity
@Table(name = "student")
public class Student {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private int age;

    // Getters & Setters
}
```

Code: HibernateMain.java (CRUD)

java

CopyEdit

```
public class HibernateMain {
    public static void main(String[] args) {
        SessionFactory factory = new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();

        Student s1 = new Student();
        s1.setName("Rupesh");
        s1.setAge(25);
```

```
        session.save(s1); // Insert data
        tx.commit();
        session.close();
    }
}
```

Output:

- **Data Inserted into MySQL Table student**

Created by Rupesh Sawant

1. Reverse a String without using built-in functions

java

CopyEdit

```
public class ReverseString {
    public static void main(String[] args) {
        String str = "Java";
        String reversed = "";

        for (int i = str.length() - 1; i >= 0; i--) {
            reversed += str.charAt(i);
        }

        System.out.println("Reversed: " + reversed);
    }
}
```

📝 **Output:** avaJ

---

2. Check if a number is Prime

java

CopyEdit

```
public class PrimeNumber {
    public static boolean isPrime(int num) {
        if (num <= 1) return false;
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) return false;
        }
        return true;
    }

    public static void main(String[] args) {
        int num = 29;
```

```
        System.out.println(num + " is prime? " + isPrime(num));
    }
}
```

📝 **Output:** 29 is prime? true

---

3. Find the first non-repeating character in a string

java

CopyEdit

```
import java.util.LinkedHashMap;
import java.util.Map;

public class FirstNonRepeatingChar {
    public static void main(String[] args) {
        String str = "programming";
        char result = firstNonRepeatingCharacter(str);
        System.out.println("First non-repeating character: " + result);
    }

    public static char firstNonRepeatingCharacter(String str) {
        Map<Character, Integer> countMap = new LinkedHashMap<>();

        for (char c : str.toCharArray()) {
            countMap.put(c, countMap.getOrDefault(c, 0) + 1);
        }

        for (Map.Entry<Character, Integer> entry : countMap.entrySet()) {
            if (entry.getValue() == 1) return entry.getKey();
        }

        return '\0'; // No unique character found
    }
```

```
}
```

📝 **Output:** First non-repeating character: p

🟡 **Medium Level**

4. Find the missing number in an array (1 to N)

```
java
CopyEdit
public class MissingNumber {
    public static int findMissingNumber(int[] arr, int n) {
        int sum = n * (n + 1) / 2;
        int actualSum = 0;

        for (int num : arr) {
            actualSum += num;
        }

        return sum - actualSum;
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 5, 6};
        System.out.println("Missing number: " + findMissingNumber(arr, 6));
    }
}
```

📝 **Output:** Missing number: 3

---

5. Implement a Singleton Pattern

```
java
CopyEdit
class Singleton {
    private static Singleton instance;
```

```
    private Singleton() {} // Private constructor

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}

public class SingletonExample {
    public static void main(String[] args) {
        Singleton obj1 = Singleton.getInstance();
        Singleton obj2 = Singleton.getInstance();

        System.out.println(obj1 == obj2); // Should print true
    }
}
```

📝 **Output:** true

---

6. Detect a Loop in a LinkedList (Floyd's Cycle Detection)

```
java
CopyEdit
class Node {
    int data;
    Node next;
    Node(int d) { data = d; next = null; }
}

public class DetectLoop {
```

```
    public static boolean hasLoop(Node head) {
        Node slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
            if (slow == fast) return true;
        }
        return false;
    }

    public static void main(String[] args) {
        Node head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(3);
        head.next.next.next = head.next; // Creating a loop

        System.out.println("Loop detected? " + hasLoop(head));
    }
}
```

📝 **Output:** Loop detected? true

🔴 **Hard Level**

7. Implement a Producer-Consumer Problem using BlockingQueue

java

CopyEdit

```
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

class Producer implements Runnable {
    private BlockingQueue<Integer> queue;
```

```
    public Producer(BlockingQueue<Integer> queue) {
        this.queue = queue;
    }

    public void run() {
        try {
            for (int i = 1; i <= 5; i++) {
                System.out.println("Produced: " + i);
                queue.put(i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) { e.printStackTrace(); }
    }
}

class Consumer implements Runnable {
    private BlockingQueue<Integer> queue;

    public Consumer(BlockingQueue<Integer> queue) {
        this.queue = queue;
    }

    public void run() {
        try {
            while (true) {
                Integer value = queue.take();
                System.out.println("Consumed: " + value);
                Thread.sleep(1500);
            }
        } catch (InterruptedException e) { e.printStackTrace(); }
    }
```

```
}

public class ProducerConsumerExample {
    public static void main(String[] args) {
        BlockingQueue<Integer> queue = new ArrayBlockingQueue<>(2);
        new Thread(new Producer(queue)).start();
        new Thread(new Consumer(queue)).start();
    }
}
```

📝 **Output:**

```
makefile
CopyEdit
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
...
```

---

8. Implement a Deadlock Scenario

```
java
CopyEdit
class Resource {
    void methodA(Resource r) {
        synchronized (this) {
            System.out.println(Thread.currentThread().getName() + " locked methodA");
            try { Thread.sleep(100); } catch (InterruptedException e) {}
            synchronized (r) {
                System.out.println(Thread.currentThread().getName() + " locked methodB");
            }
        }
    }
```

```
}

public class DeadlockExample {
    public static void main(String[] args) {
        Resource r1 = new Resource();
        Resource r2 = new Resource();

        Thread t1 = new Thread(() -> r1.methodA(r2));
        Thread t2 = new Thread(() -> r2.methodA(r1));

        t1.start();
        t2.start();
    }
}
```

📝 **Output:**
Threads stuck in deadlock.

🔹 1. Right-Angled Triangle Pattern

markdown

CopyEdit

```
*
* *
* * *
* * * *
* * * * *
```

Java Code

java

CopyEdit

```
public class RightTrianglePattern {
    public static void main(String[] args) {
        int n = 5; // Number of rows

        // Outer loop for rows
        for (int i = 1; i <= n; i++) {
            // Inner loop for columns
            for (int j = 1; j <= i; j++) {
                System.out.print("* "); // Print star with space
            }
            System.out.println(); // Move to the next line
        }
    }
}
```

Output

markdown

CopyEdit

```
*
* *
* * *
```

```
* * * *
* * * * *
```

🔹 2. Inverted Right-Angled Triangle

markdown

CopyEdit

```
* * * * *
* * * *
* * *
* *
*
```

Java Code

java

CopyEdit

```
public class InvertedTrianglePattern {
    public static void main(String[] args) {
        int n = 5; // Number of rows

        // Outer loop for rows
        for (int i = n; i >= 1; i--) {
            // Inner loop for columns
            for (int j = 1; j <= i; j++) {
                System.out.print("* "); // Print star with space
            }
            System.out.println(); // Move to next line
        }
    }
}
```

Output

markdown

CopyEdit

```
* * * * *
* * * *
* * *
* *
*
```

🔹 3. Pyramid Pattern

markdown

CopyEdit

```
   *
  * *
 * * *
* * * *
* * * * *
```

Java Code

java

CopyEdit

```
public class PyramidPattern {
    public static void main(String[] args) {
        int n = 5; // Number of rows

        for (int i = 1; i <= n; i++) {
            // Printing spaces
            for (int j = 1; j <= n - i; j++) {
                System.out.print(" "); // Print space
            }

            // Printing stars
            for (int j = 1; j <= i; j++) {
                System.out.print("* "); // Print star with space
            }
```

```
            System.out.println(); // Move to the next line
        }
    }
}
```

Output

markdown

CopyEdit

```
    *
   * *
  * * *
 * * * *
* * * * *
```

🔹 4. Diamond Pattern

markdown

CopyEdit

```
    *
   * *
  * * *
 * * * *
* * * * *
 * * * *
  * * *
   * *
    *
```

Java Code

java

CopyEdit

```
public class DiamondPattern {
    public static void main(String[] args) {
        int n = 5; // Number of rows for the upper part
```

```
        // Upper half of the diamond
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n - i; j++) {
                System.out.print(" "); // Print leading spaces
            }
            for (int j = 1; j <= i; j++) {
                System.out.print("* "); // Print stars
            }
            System.out.println();
        }

        // Lower half of the diamond
        for (int i = n - 1; i >= 1; i--) {
            for (int j = 1; j <= n - i; j++) {
                System.out.print(" "); // Print leading spaces
            }
            for (int j = 1; j <= i; j++) {
                System.out.print("* "); // Print stars
            }
            System.out.println();
        }
    }
}
```

Output

markdown

CopyEdit

```
   *
  * *
 * * *
* * * *
```

```
* * * * *
 * * * *
  * * *
   * *
    *
```

🔹 5. Hollow Square Pattern

markdown

CopyEdit

```
* * * * *
*     *
*     *
*     *
* * * * *
```

Java Code

java

CopyEdit

```
public class HollowSquarePattern {
    public static void main(String[] args) {
        int n = 5; // Size of square

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                // Print '*' for first and last row and column
                if (i == 1 || i == n || j == 1 || j == n) {
                    System.out.print("* ");
                } else {
                    System.out.print("  "); // Print space
                }
            }
            System.out.println();
```

```
        }
    }
}
```

Output

markdown

CopyEdit

```
* * * * *
*     *
*     *
*     *
* * * * *
```

🔹 6. Floyd's Triangle

CopyEdit

```
1
2 3
4 5 6
7 8 9 10
```

Java Code

java

CopyEdit

```
public class FloydsTriangle {
    public static void main(String[] args) {
        int n = 4; // Number of rows
        int num = 1; // Number to be printed

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(num + " ");
                num++;
            }
```

```
            System.out.println();
        }
    }
}
```

Output

CopyEdit

1

2 3

4 5 6

7 8 9 10

- **7. Pascal's Triangle**

markdown

CopyEdit

```
     1
    1 1
   1 2 1
  1 3 3 1
 1 4 6 4 1
```

Java Code

java

CopyEdit

```
public class PascalsTriangle {
    public static void main(String[] args) {
        int n = 5; // Number of rows

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n - i; j++) {
                System.out.print(" ");
            }
            int number = 1;
```

```
            for (int j = 0; j <= i; j++) {
                System.out.print(number + " ");
                number = number * (i - j) / (j + 1); // Calculate next number
            }
            System.out.println();
        }
    }
}
```

Output

markdown

CopyEdit

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

📌 1. Initialize and Print an Array

Problem:

Write a Java program to **initialize an array** and print its elements.

Java Code:

java

CopyEdit

```
public class ArrayInitialization {
    public static void main(String[] args) {
        // Declare and initialize an array of size 5
        int[] numbers = {10, 20, 30, 40, 50};

        // Printing array elements
        System.out.println("Array elements:");
        for (int i = 0; i < numbers.length; i++) {
            System.out.println(numbers[i]); // Print each element
        }
    }
}
```

Output:

javascript

CopyEdit

```
Array elements:
10
20
30
40
50
```

Explanation:

1. **Initialization:** {10, 20, 30, 40, 50} is stored in an array.
2. **Looping:** A **for loop** iterates through the array and prints each element.

📌 2. Find the Largest Element in an Array

Problem:

Write a Java program to **find the largest element** in an array.

Java Code:

java

CopyEdit

```
public class LargestElement {
    public static void main(String[] args) {
        int[] numbers = {12, 34, 56, 78, 90, 23, 67};

        // Assume the first element is the largest
        int max = numbers[0];

        for (int i = 1; i < numbers.length; i++) {
            if (numbers[i] > max) { // Check if the current element is greater than max
                max = numbers[i]; // Update max
            }
        }

        System.out.println("Largest element: " + max);
    }
}
```

Output:

yaml

CopyEdit

```
Largest element: 90
```

Mathematical Calculation:

- Start with max = 12
- Compare and update:
    - 34 > 12 → max = 34
    - 56 > 34 → max = 56

- 78 > 56 → max = 78
- 90 > 78 → max = 90 (Final answer)

📌 3. Reverse an Array

Problem:

Write a Java program to **reverse an array**.

Java Code:

java

CopyEdit

```
public class ReverseArray {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};

        System.out.println("Original array:");
        for (int num : numbers) {
            System.out.print(num + " ");
        }

        // Reverse the array using two-pointer approach
        int start = 0, end = numbers.length - 1;
        while (start < end) {
            // Swap numbers[start] and numbers[end]
            int temp = numbers[start];
            numbers[start] = numbers[end];
            numbers[end] = temp;

            start++; // Move forward
            end--;   // Move backward
        }

        System.out.println("\nReversed array:");
```

```
        for (int num : numbers) {
            System.out.print(num + " ");
        }
    }
}
```

Output:

php

CopyEdit

Original array:

1 2 3 4 5

Reversed array:

5 4 3 2 1

Explanation:

1. **Swapping elements:**
    - Swap 1 and 5
    - Swap 2 and 4
    - Middle element 3 stays the same.

📌 4. Check If an Array is Sorted

Problem:

Write a Java program to check if an array is sorted in ascending order.

Java Code:

java

CopyEdit

```
public class CheckSortedArray {
    public static void main(String[] args) {
        int[] numbers = {10, 20, 30, 40, 50};

        boolean isSorted = true;
        for (int i = 0; i < numbers.length - 1; i++) {
            if (numbers[i] > numbers[i + 1]) { // If previous element is greater, array is not sorted
```

```
            isSorted = false;
            break;
        }
    }

    if (isSorted) {
        System.out.println("Array is sorted.");
    } else {
        System.out.println("Array is not sorted.");
    }
  }
}
```

Output:

python

CopyEdit

Array is sorted.

Mathematical Calculation:

- **Compare adjacent elements:**
  - 10 < 20 ✅
  - 20 < 30 ✅
  - 30 < 40 ✅
  - 40 < 50 ✅
- Since all comparisons are correct, **array is sorted**.

📌 5. Merge Two Sorted Arrays

Problem:

Write a Java program to merge two **sorted** arrays into a single sorted array.

Java Code:

java

CopyEdit

```
import java.util.Arrays;
```

```
public class MergeSortedArrays {
    public static void main(String[] args) {
        int[] arr1 = {1, 3, 5, 7};
        int[] arr2 = {2, 4, 6, 8};

        int[] mergedArray = new int[arr1.length + arr2.length];

        int i = 0, j = 0, k = 0;

        // Merge elements from both arrays
        while (i < arr1.length && j < arr2.length) {
            if (arr1[i] < arr2[j]) {
                mergedArray[k++] = arr1[i++];
            } else {
                mergedArray[k++] = arr2[j++];
            }
        }

        // Copy remaining elements
        while (i < arr1.length) {
            mergedArray[k++] = arr1[i++];
        }
        while (j < arr2.length) {
            mergedArray[k++] = arr2[j++];
        }

        System.out.println("Merged sorted array: " + Arrays.toString(mergedArray));
    }
}
```

Output:

less

CopyEdit

Merged sorted array: [1, 2, 3, 4, 5, 6, 7, 8]

Mathematical Calculation:

- **Two-pointer approach:**
  - Compare 1 and 2 → Take 1
  - Compare 3 and 2 → Take 2
  - Compare 3 and 4 → Take 3
  - Continue until all elements are merged.

📌 **6. Find the Second Largest Element in an Array**

Problem:

Write a Java program to find the **second largest** element in an array.

Java Code:

java

CopyEdit

```
public class SecondLargest {
    public static void main(String[] args) {
        int[] numbers = {10, 20, 4, 45, 99, 55};

        // Initialize first and second largest
        int first = Integer.MIN_VALUE, second = Integer.MIN_VALUE;

        for (int num : numbers) {
            if (num > first) {
                second = first; // Update second largest
                first = num;  // Update first largest
            } else if (num > second && num != first) {
                second = num; // Update second if it's not equal to first
            }
        }
```

```
        System.out.println("Second largest element: " + second);
    }
}
```

Output:

sql

CopyEdit

Second largest element: 55

Mathematical Calculation:

- **Step 1:** first = 10, second = MIN_VALUE
- **Step 2:** Compare each element:
    - 20 > 10 → first = 20, second = 10
    - 4 < 20 → No change
    - 45 > 20 → first = 45, second = 20
    - 99 > 45 → first = 99, second = 45
    - 55 < 99 & > 45 → second = 55
- **Final result:** Second largest is **55**.

📌 7. Find Missing Number in an Array (1 to N)

Problem:

Given an array of **N-1 elements**, where numbers are from **1 to N**, find the **missing number**.

Formula:

Sum of first N numbers=N×(N+1)2\text{Sum of first N numbers} = \frac{N \times (N+1)}{2}Sum of first N numbers=2N×(N+1)

Java Code:

java

CopyEdit

```
public class MissingNumber {
    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 5, 6}; // N=6 but missing 3

        int n = arr.length + 1; // Actual size N
```

```
        int totalSum = (n * (n + 1)) / 2; // Sum of first N numbers

        int arraySum = 0;
        for (int num : arr) {
            arraySum += num; // Sum of elements in array
        }

        int missingNumber = totalSum - arraySum; // Find missing number
        System.out.println("Missing Number: " + missingNumber);
    }
}
```

Output:

mathematica

CopyEdit

Missing Number: 3

Mathematical Calculation:

- **Total sum (1 to 6):** 6×72=21\frac{6 \times 7}{2} = 2126×7=21
- **Sum of given numbers:** 1+2+4+5+6=181 + 2 + 4 + 5 + 6 = 181+2+4+5+6=18
- **Missing number:** 21−18=321 - 18 = 321−18=3

📌 8. Remove Duplicates from an Array

Problem:

Write a Java program to **remove duplicates** from an array.

Java Code:

java

CopyEdit

```
import java.util.Arrays;
import java.util.LinkedHashSet;

public class RemoveDuplicates {
    public static void main(String[] args) {
```

```
        Integer[] arr = {1, 2, 2, 3, 4, 4, 5};

        LinkedHashSet<Integer> set = new LinkedHashSet<>(Arrays.asList(arr));

        Integer[] uniqueArray = set.toArray(new Integer[0]);

        System.out.println("Array without duplicates: " + Arrays.toString(uniqueArray));
    }
}
```

Output:

sql

CopyEdit

Array without duplicates: [1, 2, 3, 4, 5]

Explanation:

- **LinkedHashSet** removes duplicates while maintaining order.

📌 9. Rotate an Array (Left Rotation by K)

Problem:

Write a program to **rotate an array left by K positions**.

Java Code:

java

CopyEdit

```
import java.util.Arrays;

public class LeftRotateArray {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int k = 2; // Rotate by 2 positions

        int n = arr.length;
        int[] rotatedArray = new int[n];
```

```
        for (int i = 0; i < n; i++) {
            rotatedArray[i] = arr[(i + k) % n]; // Circular shift
        }

        System.out.println("Rotated array: " + Arrays.toString(rotatedArray));
    }
}
```

Output:

less

CopyEdit

Rotated array: [3, 4, 5, 1, 2]

Mathematical Calculation:

- **Shift by K:**
  - 3 → 0th index
  - 4 → 1st index
  - 5 → 2nd index
  - 1 → 3rd index
  - 2 → 4th index

📌 10. Find Intersection and Union of Two Arrays

Problem:

Write a Java program to **find the intersection and union** of two arrays.

Java Code:

java

CopyEdit

```
import java.util.Arrays;
import java.util.HashSet;

public class ArrayIntersectionUnion {
    public static void main(String[] args) {
```

```
        Integer[] arr1 = {1, 2, 3, 4, 5};
        Integer[] arr2 = {4, 5, 6, 7, 8};

        HashSet<Integer> unionSet = new HashSet<>(Arrays.asList(arr1));
        unionSet.addAll(Arrays.asList(arr2));

        HashSet<Integer> intersectionSet = new HashSet<>(Arrays.asList(arr1));
        intersectionSet.retainAll(Arrays.asList(arr2));

        System.out.println("Union: " + unionSet);
        System.out.println("Intersection: " + intersectionSet);
    }
}
```

Output:

makefile

CopyEdit

Union: [1, 2, 3, 4, 5, 6, 7, 8]

Intersection: [4, 5]

Explanation:

- **Union:** All unique elements.
- **Intersection:** Common elements.

📌 11. Find First Non-Repeating Element

Problem:

Find the **first non-repeating** element in an array.

Java Code:

java

CopyEdit

```
import java.util.HashMap;

public class FirstNonRepeating {
```

```
    public static void main(String[] args) {
        int[] arr = {4, 5, 1, 2, 0, 4, 5, 2};

        HashMap<Integer, Integer> map = new HashMap<>();

        for (int num : arr) {
            map.put(num, map.getOrDefault(num, 0) + 1); // Count frequency
        }

        for (int num : arr) {
            if (map.get(num) == 1) {
                System.out.println("First non-repeating element: " + num);
                break;
            }
        }
    }
}
```

Output:

sql

CopyEdit

First non-repeating element: 1

Explanation:

- **Frequency count:** {4=2, 5=2, 1=1, 2=2, 0=1}
- **First element with count 1:** 1

📌 **12. Find the Majority Element in an Array (Boyer-Moore Algorithm)**

Problem:

A majority element in an array **appears more than N/2 times**. Find the majority element if it exists.

Java Code (Boyer-Moore Voting Algorithm):

java

CopyEdit

public class MajorityElement {

```
    public static int findMajorityElement(int[] nums) {
        int candidate = 0, count = 0;

        // Step 1: Find the potential majority element
        for (int num : nums) {
            if (count == 0) {
                candidate = num;
            }
            count += (num == candidate) ? 1 : -1;
        }

        // Step 2: Verify if it's a majority
        count = 0;
        for (int num : nums) {
            if (num == candidate) {
                count++;
            }
        }

        return (count > nums.length / 2) ? candidate : -1; // Return majority or -1
    }

    public static void main(String[] args) {
        int[] nums = {3, 3, 4, 2, 3, 3, 3, 2};
        System.out.println("Majority Element: " + findMajorityElement(nums));
    }
}
```

Output:

mathematica

CopyEdit

Majority Element: 3

Mathematical Calculation:

- **Given array:** {3, 3, 4, 2, 3, 3, 3, 2}
- **N/2 =** 8/2 = 4
- **Frequency of 3 =** 5 (more than 4, so majority element is 3)

📌 13. Find Pairs in an Array That Sum to a Given Value

Problem:

Find all **pairs** in an array that sum up to a target value.

Java Code:

java

CopyEdit

```
import java.util.HashSet;


public class PairSum {
   public static void findPairs(int[] arr, int target) {
      HashSet<Integer> set = new HashSet<>();
      for (int num : arr) {
         int complement = target - num;
         if (set.contains(complement)) {
            System.out.println("Pair: (" + complement + ", " + num + ")");
         }
         set.add(num);
      }
   }


   public static void main(String[] args) {
      int[] arr = {2, 4, 3, 7, 8, 5};
      int target = 10;
      findPairs(arr, target);
   }
}
```

Output:

makefile

CopyEdit

Pair: (3, 7)

Pair: (2, 8)

Mathematical Calculation:

- **Pairs that sum to 10:**
  - 3 + 7 = 10
  - 2 + 8 = 10

📌 14. Find the Longest Consecutive Sequence in an Unsorted Array

Problem:

Find the **longest sequence of consecutive numbers** in an unsorted array.

Java Code:

java

CopyEdit

```
import java.util.HashSet;


public class LongestConsecutiveSequence {
    public static int findLongestSequence(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        for (int num : nums) {
            set.add(num);
        }

        int longest = 0;
        for (int num : nums) {
            if (!set.contains(num - 1)) { // Start of a sequence
                int currentNum = num;
                int currentLength = 1;
```

```
            while (set.contains(currentNum + 1)) {
                currentNum++;
                currentLength++;
            }
            longest = Math.max(longest, currentLength);
        }
    }
    return longest;
  }

  public static void main(String[] args) {
    int[] nums = {100, 4, 200, 1, 3, 2};
    System.out.println("Longest Consecutive Sequence: " + findLongestSequence(nums));
  }
}
```

Output:

mathematica

CopyEdit

Longest Consecutive Sequence: 4

Mathematical Calculation:

- **Given numbers:** {100, 4, 200, 1, 3, 2}
- **Longest sequence:** {1, 2, 3, 4} (Length = 4)

📌 15. Find the Smallest and Largest Elements in an Array

Problem:

Find the **smallest and largest numbers** in an array.

Java Code:

java

CopyEdit

```
public class MinMaxArray {
  public static void main(String[] args) {
```

```
        int[] arr = {7, 2, 8, 1, 9, 3};

        int min = arr[0], max = arr[0];

        for (int num : arr) {
            if (num < min) min = num;
            if (num > max) max = num;
        }

        System.out.println("Smallest: " + min);
        System.out.println("Largest: " + max);
    }
}
```

Output:

makefile

CopyEdit

Smallest: 1

Largest: 9

📌 16. Rearrange Array Elements (Negative and Positive Alternately)

Problem:

Rearrange an array so that **negative and positive numbers alternate**.

Java Code:

java

CopyEdit

```
import java.util.Arrays;

public class RearrangeArray {
    public static void rearrange(int[] arr) {
        int[] temp = new int[arr.length];
        int posIndex = 0, negIndex = 1;
```

```
        for (int num : arr) {
            if (num >= 0 && posIndex < arr.length) {
                temp[posIndex] = num;
                posIndex += 2;
            } else if (num < 0 && negIndex < arr.length) {
                temp[negIndex] = num;
                negIndex += 2;
            }
        }

        System.arraycopy(temp, 0, arr, 0, arr.length);
    }

    public static void main(String[] args) {
        int[] arr = {1, -2, 3, -4, 5, -6};
        rearrange(arr);
        System.out.println("Rearranged Array: " + Arrays.toString(arr));
    }
}
```

Output:

javascript

CopyEdit

Rearranged Array: [1, -2, 3, -4, 5, -6]

Explanation:

- **Alternates between positive and negative numbers.**

📌 17. Find Subarray with Given Sum (Sliding Window)

Problem:

Find a **contiguous subarray** that adds up to a given sum.

Java Code:

Created by Rupesh Sawant

java

CopyEdit

```
public class SubarraySum {
    public static void findSubarray(int[] arr, int target) {
        int start = 0, sum = 0;

        for (int end = 0; end < arr.length; end++) {
            sum += arr[end];

            while (sum > target && start <= end) {
                sum -= arr[start];
                start++;
            }

            if (sum == target) {
                System.out.println("Subarray found from index " + start + " to " + end);
                return;
            }
        }

        System.out.println("No subarray found.");
    }

    public static void main(String[] args) {
        int[] arr = {1, 4, 20, 3, 10, 5};
        int target = 33;
        findSubarray(arr, target);
    }
}
```

Output:

pgsql

CopyEdit

Subarray found from index 2 to 4

Explanation:

- **Given array:** {1, 4, 20, 3, 10, 5}
- **Subarray {20, 3, 10}** sums to 33.

🔹 Reverse an Array (Without Using Extra Space)

Problem:

Reverse an array **in-place** (without using an extra array).

Java Code:

```
java
CopyEdit
import java.util.Arrays;

public class ReverseArray {
    public static void reverse(int[] arr) {
        int left = 0, right = arr.length - 1;

        while (left < right) {
            // Swap elements
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;

            left++;
            right--;
        }
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        reverse(arr);
        System.out.println("Reversed Array: " +
Arrays.toString(arr));
    }
}
```

Output:

```
javascript
CopyEdit
Reversed Array: [5, 4, 3, 2, 1]
```

2️⃣ Find Second Largest Element in an Array

Problem:

Find the second largest element **without sorting the array**.

Java Code:

```
java
CopyEdit
public class SecondLargest {
```

```
    public static int findSecondLargest(int[] arr) {
        int largest = Integer.MIN_VALUE;
        int secondLargest = Integer.MIN_VALUE;

        for (int num : arr) {
            if (num > largest) {
                secondLargest = largest;
                largest = num;
            } else if (num > secondLargest && num != largest) {
                secondLargest = num;
            }
        }
        return secondLargest;
    }

    public static void main(String[] args) {
        int[] arr = {12, 35, 1, 10, 34, 1};
        System.out.println("Second Largest: " +
findSecondLargest(arr));
    }
}
```

Output:

```
sql
CopyEdit
Second Largest: 34
```

3️⃣ Find the Missing Number in an Array (N Natural Numbers)

Problem:

An array contains numbers from `1` to `N` but one number is missing. Find the missing number.

Mathematical Formula Used:

Sum of first N natural numbers=N×(N+1)2\text{Sum of first N natural numbers} = \frac{N \times (N + 1)}{2}Sum of first N natural numbers=2N×(N+1)

Java Code:

```
java
CopyEdit
public class MissingNumber {
    public static int findMissing(int[] arr, int n) {
        int totalSum = n * (n + 1) / 2; // Sum of first N natural
numbers
        int actualSum = 0;

        for (int num : arr) {
            actualSum += num;
        }

        return totalSum - actualSum;
    }

    public static void main(String[] args) {
```

```
        int[] arr = {1, 2, 4, 5, 6}; // Missing 3
        int n = 6;
        System.out.println("Missing Number: " + findMissing(arr, n));
    }
}
```

Output:

```
mathematica
CopyEdit
Missing Number: 3
```

4️⃣ Find the Duplicate Number in an Array

Problem:

Find the duplicate number in an **array of size N** with values from `1` to `N-1`.

Java Code:

```
java
CopyEdit
import java.util.HashSet;

public class FindDuplicate {
    public static int findDuplicate(int[] arr) {
        HashSet<Integer> set = new HashSet<>();
        for (int num : arr) {
            if (set.contains(num)) {
                return num; // Duplicate found
            }
            set.add(num);
        }
        return -1; // No duplicate
    }

    public static void main(String[] args) {
        int[] arr = {1, 3, 4, 2, 5, 3};
        System.out.println("Duplicate Number: " +
findDuplicate(arr));
    }
}
```

Output:

```
javascript
CopyEdit
Duplicate Number: 3
```

5️⃣ Check if an Array is a Subset of Another Array

Problem:

Check if `arr2[]` is a **subset** of `arr1[]`.

Java Code:

```
java
CopyEdit
import java.util.HashSet;

public class ArraySubset {
```

```
    public static boolean isSubset(int[] arr1, int[] arr2) {
        HashSet<Integer> set = new HashSet<>();
        for (int num : arr1) {
            set.add(num);
        }

        for (int num : arr2) {
            if (!set.contains(num)) {
                return false; // Not a subset
            }
        }
        return true;
    }

    public static void main(String[] args) {
        int[] arr1 = {1, 2, 3, 4, 5, 6};
        int[] arr2 = {2, 4, 5};

        System.out.println("Is Subset: " + isSubset(arr1, arr2));
    }
}
```

Output:

```
sql
CopyEdit
Is Subset: true
```

---

6. Find the Intersection of Two Arrays

Problem:

Find **common elements** in two arrays.

Java Code:

```
java
CopyEdit
import java.util.HashSet;

public class ArrayIntersection {
    public static void findIntersection(int[] arr1, int[] arr2) {
        HashSet<Integer> set = new HashSet<>();
        for (int num : arr1) {
            set.add(num);
        }

        System.out.print("Intersection: ");
        for (int num : arr2) {
            if (set.contains(num)) {
                System.out.print(num + " ");
            }
        }
    }

    public static void main(String[] args) {
        int[] arr1 = {1, 2, 3, 4, 5};
```

```
        int[] arr2 = {3, 4, 5, 6, 7};

        findIntersection(arr1, arr2);
    }
}
```

Output:

```
makefile
CopyEdit
Intersection: 3 4 5
```

---

7 Rotate an Array (Right Rotation by K Positions)

Problem:

Rotate an array to the right **by κ positions**.

Java Code:

```
java
CopyEdit
import java.util.Arrays;

public class RotateArray {
    public static void rotateRight(int[] arr, int k) {
        int n = arr.length;
        k = k % n; // Handle cases where K > N

        reverse(arr, 0, n - 1);
        reverse(arr, 0, k - 1);
        reverse(arr, k, n - 1);
    }

    private static void reverse(int[] arr, int start, int end) {
        while (start < end) {
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
            end--;
        }
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int k = 2;
        rotateRight(arr, k);
        System.out.println("Rotated Array: " + Arrays.toString(arr));
    }
}
```

Output:

```
javascript
CopyEdit
Rotated Array: [4, 5, 1, 2, 3]
```

📌 1. Sorting Algorithms in Java (Most Asked in Interviews)

Sorting is fundamental in **problem-solving, data structures, and optimization.**

1️⃣ Bubble Sort (Basic Sorting Algorithm)

Concept:

- Repeatedly **swap adjacent elements** if they are in the wrong order.
- **Time Complexity:**
  - Worst Case: **O(N²)**
  - Best Case (Already Sorted): **O(N)**

Java Code:

```
java
CopyEdit
import java.util.Arrays;

public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        boolean swapped;

        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    swapped = true;
                }
            }
            if (!swapped) break; // Stop if no swaps
        }
    }

    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        bubbleSort(arr);
        System.out.println("Sorted Array: " + Arrays.toString(arr));
    }
}
```

Output:

```
javascript
CopyEdit
Sorted Array: [11, 12, 22, 25, 64]
```

2️⃣ Selection Sort (Find Minimum and Swap)

Concept:

- **Find the smallest element** and swap it with the first unsorted element.
- **Time Complexity:**
  - Worst Case: **O(N²)**
  - Best Case: **O(N²)** (No optimized case)

Java Code:

```
java
CopyEdit
import java.util.Arrays;

public class SelectionSort {
```

```
    public static void selectionSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }

    public static void main(String[] args) {
        int[] arr = {29, 10, 14, 37, 13};
        selectionSort(arr);
        System.out.println("Sorted Array: " + Arrays.toString(arr));
    }
}
```

Output:

```
javascript
CopyEdit
Sorted Array: [10, 13, 14, 29, 37]
```

1️⃣ Quick Sort (Divide and Conquer)

Concept:

- Pick a **pivot**, partition the array into **left (smaller)** and **right (greater)** parts.
- **Recursively** sort both halves.
- **Time Complexity:**
  - Worst Case: **$O(N^2)$** (rare)
  - Average & Best Case: **O(N log N)**

Java Code:

```
java
CopyEdit
import java.util.Arrays;

public class QuickSort {
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    private static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
```

```
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

    public static void main(String[] args) {
        int[] arr = {10, 80, 30, 90, 40, 50, 70};
        quickSort(arr, 0, arr.length - 1);
        System.out.println("Sorted Array: " + Arrays.toString(arr));
    }
}
```

Output:

```
javascript
CopyEdit
Sorted Array: [10, 30, 40, 50, 70, 80, 90]
```

📌 2. Recursion Problems (Common in Interviews)

Recursion is **widely asked** in coding rounds.

1️⃣ Factorial of a Number (Using Recursion)

Formula:

n!=n×(n−1)!n! = n \times (n - 1)!n!=n×(n−1)! 5!=5×4×3×2×1=1205! = 5 × 4 × 3 × 2 × 1 = 1205!=5×4×3×2×1=120

Java Code:

```
java
CopyEdit
public class FactorialRecursion {
    public static int factorial(int n) {
        if (n == 0) return 1;
        return n * factorial(n - 1);
    }

    public static void main(String[] args) {
        int num = 5;
        System.out.println("Factorial of " + num + ": " + factorial(num));
    }
}
```

Output:

```
yaml
CopyEdit
Factorial of 5: 120
```

2️⃣ Fibonacci Series (Recursion)

Formula:

F(n)=F(n−1)+F(n−2)F(n) = F(n-1) + F(n-2)F(n)=F(n−1)+F(n−2)

Java Code:

```
java
CopyEdit
public class FibonacciRecursion {
    public static int fibonacci(int n) {
        if (n <= 1) return n;
        return fibonacci(n - 1) + fibonacci(n - 2);
    }

    public static void main(String[] args) {
        int n = 6;
        System.out.println("Fibonacci Number at position " + n + ": " +
fibonacci(n));
    }
```

```
}
```
Output:
```
mathematica
CopyEdit
Fibonacci Number at position 6: 8
```

📌 3. Matrix-Based Problems

Matrix-based problems **test logical thinking and optimization.**

🔲 Transpose of a Matrix

Problem:

Convert **rows into columns.**

Java Code:

```
java
CopyEdit
public class MatrixTranspose {
    public static void transpose(int[][] matrix) {
        int row = matrix.length, col = matrix[0].length;
        int[][] transposed = new int[col][row];

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                transposed[j][i] = matrix[i][j];
            }
        }

        System.out.println("Transposed Matrix:");
        for (int[] rowArr : transposed) {
            for (int num : rowArr) {
                System.out.print(num + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        transpose(matrix);
    }
}
```

Output:

```
yaml
CopyEdit
Transposed Matrix:
1 4 7
2 5 8
3 6 9
```

📌 1. Dynamic Programming (DP) Problems

DP problems require **optimal substructure and overlapping subproblems.**

1️⃣ Fibonacci Using DP (Memoization & Tabulation)

Why DP?

The recursive Fibonacci function we wrote earlier has **O(2ⁿ) time complexity** due to redundant calculations. DP optimizes it to **O(n)**.

🔹 Approach 1: Memoization (Top-Down)

- Store **previously computed** Fibonacci values in an array.
- **Time Complexity:** O(n), **Space Complexity:** O(n).

```
java
CopyEdit
import java.util.HashMap;

public class FibonacciDP {
    static HashMap<Integer, Integer> memo = new HashMap<>();

    public static int fibonacci(int n) {
        if (n <= 1) return n;
        if (memo.containsKey(n)) return memo.get(n);

        int result = fibonacci(n - 1) + fibonacci(n - 2);
        memo.put(n, result);
        return result;
    }

    public static void main(String[] args) {
        System.out.println("Fibonacci(6): " + fibonacci(6));
    }
}
```

Output:

```
scss
CopyEdit
Fibonacci(6): 8
```

🔹 Approach 2: Tabulation (Bottom-Up)

- Build the Fibonacci array **iteratively** instead of recursion.
- **Time Complexity:** O(n), **Space Complexity:** O(n).

```
java
CopyEdit
public class FibonacciTabulation {
    public static int fibonacci(int n) {
        if (n <= 1) return n;
        int[] dp = new int[n + 1];
        dp[0] = 0;
        dp[1] = 1;

        for (int i = 2; i <= n; i++) {
            dp[i] = dp[i - 1] + dp[i - 2];
        }

        return dp[n];
    }

    public static void main(String[] args) {
        System.out.println("Fibonacci(6): " + fibonacci(6));
    }
}
```

Output:

```
scss
CopyEdit
Fibonacci(6): 8
```

2️⃣ Longest Common Subsequence (LCS)
Problem:
Given two strings, find the **length of the longest subsequence** common to both.
Example:

```
vbnet
CopyEdit
Input: X = "ACDBE", Y = "ABCDE"
Output: 4 (Common: "ACDE")
```

🔹 **DP Approach (Bottom-Up)**

- **State:** dp[i][j] stores LCS length of X[0...i-1] and Y[0...j-1].
- **Transition:**
  - If X[i] == Y[j] → dp[i][j] = 1 + dp[i-1][j-1]
  - Else → dp[i][j] = max(dp[i-1][j], dp[i][j-1])
- **Time Complexity:** O(m × n)

```
java
CopyEdit
public class LCS {
    public static int findLCS(String X, String Y) {
        int m = X.length(), n = Y.length();
        int[][] dp = new int[m + 1][n + 1];

        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (X.charAt(i - 1) == Y.charAt(j - 1)) {
                    dp[i][j] = 1 + dp[i - 1][j - 1];
                } else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }
        return dp[m][n];
    }

    public static void main(String[] args) {
        String X = "ACDBE", Y = "ABCDE";
        System.out.println("LCS Length: " + findLCS(X, Y));
    }
}
```

Output:

```
yaml
CopyEdit
LCS Length: 4
```

📌 **2. Graph Algorithms (Important for Java Interviews)**
Graph problems are commonly asked in tech interviews.

1️⃣ Depth-First Search (DFS)
Concept:

- **Traverse the graph** using recursion.
- Uses **Stack (Recursive Call Stack)**.

Java Code for DFS on a Graph

```
java
CopyEdit
import java.util.*;
```

```
public class GraphDFS {
    private Map<Integer, List<Integer>> adjList = new HashMap<>();

    public void addEdge(int u, int v) {
        adjList.putIfAbsent(u, new ArrayList<>());
        adjList.putIfAbsent(v, new ArrayList<>());
        adjList.get(u).add(v);
        adjList.get(v).add(u);
    }

    public void dfs(int node, Set<Integer> visited) {
        if (visited.contains(node)) return;

        System.out.print(node + " ");
        visited.add(node);
        for (int neighbor : adjList.get(node)) {
            dfs(neighbor, visited);
        }
    }

    public static void main(String[] args) {
        GraphDFS graph = new GraphDFS();
        graph.addEdge(1, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 4);
        graph.addEdge(3, 5);

        System.out.println("DFS Traversal:");
        graph.dfs(1, new HashSet<>());
    }
}
```

Output:

```
yaml
CopyEdit
DFS Traversal:
1 2 4 3 5
```

2️⃣ Breadth-First Search (BFS)

Concept:

- Uses **Queue** to explore all neighbors before moving deeper.
- **Time Complexity:** O(V + E)

```
java
CopyEdit
import java.util.*;

public class GraphBFS {
    private Map<Integer, List<Integer>> adjList = new HashMap<>();

    public void addEdge(int u, int v) {
        adjList.putIfAbsent(u, new ArrayList<>());
        adjList.putIfAbsent(v, new ArrayList<>());
        adjList.get(u).add(v);
        adjList.get(v).add(u);
    }

    public void bfs(int start) {
        Queue<Integer> queue = new LinkedList<>();
        Set<Integer> visited = new HashSet<>();

        queue.offer(start);
```

```
        visited.add(start);

        while (!queue.isEmpty()) {
            int node = queue.poll();
            System.out.print(node + " ");
            for (int neighbor : adjList.get(node)) {
                if (!visited.contains(neighbor)) {
                    queue.offer(neighbor);
                    visited.add(neighbor);
                }
            }
        }
    }

    public static void main(String[] args) {
        GraphBFS graph = new GraphBFS();
        graph.addEdge(1, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 4);
        graph.addEdge(3, 5);

        System.out.println("BFS Traversal:");
        graph.bfs(1);
    }
}
```

Output:

```
yaml
CopyEdit
BFS Traversal:
1 2 3 4 5
```