# Java

### 1. What are the key features of Java?

- Object-Oriented
- Platform-Independent (Write Once, Run Anywhere)
- Robust and Secure
- Multi-threaded
- Automatic Garbage Collection

**Example Question**: Explain why Java is platform-independent. **Answer**: Java uses bytecode that is executed by the JVM (Java Virtual Machine), making it platform-independent as the bytecode can run on any operating system with a compatible JVM.

---

### 2. Explain the differences between JDK, JRE, and JVM.

- **JDK**: Java Development Kit (tools for developing Java applications).
- **JRE**: Java Runtime Environment (provides the runtime to execute Java applications).
- **JVM**: Java Virtual Machine (responsible for executing bytecode).

---

### 3. What is the difference between `==` and `.equals()` in Java?

- `==`: Compares references (memory locations).
- `.equals()`: Compares the content of the objects.

**Example**:

java
Copy code
```
String s1 = new String("QA");
String s2 = new String("QA");

System.out.println(s1 == s2); // false (different references)
System.out.println(s1.equals(s2)); // true (content is same)
```

---

## 4. What are access modifiers in Java?

- **Public**: Accessible everywhere.
- **Private**: Accessible within the same class only.
- **Protected**: Accessible in the same package or through inheritance.
- **Default**: Accessible in the same package only.

**Interview Tip**: Be ready to explain with a small code snippet.

---

## 5. What is a static keyword in Java?

- **Static Variables**: Shared across all instances of a class.
- **Static Methods**: Called without creating an object.
- **Static Blocks**: Used for static initialization.

**Example**:

java
Copy code
```
class Test {
    static int count = 0;

    public static void increment() {
        count++;
    }
}
```

---

## 6. What is the difference between an abstract class and an interface?

- **Abstract Class**: Can have abstract and concrete methods.
- **Interface**: Contains only abstract methods (before Java 8).
- Since Java 8, interfaces can have default and static methods.

**Example**:

java
Copy code
```
abstract class AbstractClass {
    abstract void display();
    void show() {
```

```
        System.out.println("Concrete method in abstract class.");
    }
}

interface MyInterface {
    void display();
    default void show() {
        System.out.println("Default method in interface.");
    }
}
```

---

## OOP Concepts

## 7. Explain the four principles of OOP.

- **Encapsulation**: Binding data and methods together.
- **Abstraction**: Hiding implementation details.
- **Inheritance**: Reusing code via parent-child relationships.
- **Polymorphism**: One method, multiple behaviors.

---

## 8. What is method overloading and overriding?

- **Overloading**: Same method name, different parameter lists (compile-time polymorphism).
- **Overriding**: Subclass redefines a parent class method (runtime polymorphism).

**Example**:

```
java
Copy code
class Parent {
    void display() {
        System.out.println("Parent display");
    }
}
class Child extends Parent {
    @Override
    void display() {
        System.out.println("Child display");
    }
```

}

---

### 9. What is the difference between `final`, `finally`, and `finalize`?

- **`final`**: A constant variable, class cannot be inherited, or method cannot be overridden.
- **`finally`**: A block in exception handling that always executes.
- **`finalize`**: A method called by the garbage collector before destroying the object.

---

### 10. What is a constructor? Can it be overridden?

- **Constructor**: Special method used to initialize objects.
- **Cannot be overridden**: It is not inherited.

---

## Collections and Data Structures

### 11. What are Java Collections?

- Framework for storing and manipulating data.
- Includes: `List`, `Set`, `Map`, etc.

---

### 12. What is the difference between ArrayList and LinkedList?

- **ArrayList**: Backed by an array, better for retrieval operations.
- **LinkedList**: Doubly-linked list, better for insertion/deletion.

### Write a program to check if a string is a palindrome.

java
Copy code

```
class Palindrome {
   public static void main(String[] args) {
      String str = "madam";
      String reversed = new StringBuilder(str).reverse().toString();
      System.out.println(str.equals(reversed) ? "Palindrome" : "Not a palindrome");
   }
}
```

---

## Write a program to find duplicate elements in an array.

java
Copy code

```
import java.util.HashSet;

class Duplicate {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 2, 3};
        HashSet<Integer> set = new HashSet<>();
        for (int num : arr) {
            if (!set.add(num)) {
                System.out.println("Duplicate: " + num);
            }
        }
    }
}
```

- String Manipulation

## String Manipulation in Java

**Definition**:

String manipulation refers to operations performed on strings to modify, extract, combine, or analyze their content. Strings are immutable in Java, meaning their content cannot be changed once created, but you can create new strings with desired modifications.

---

## String Basics

1. **Creating Strings**:

   Strings can be created using:

   - String literal (stored in the String pool).
   - `new` keyword (creates a new object in heap memory).

**Example**:

```
java
Copy code
String s1 = "Hello";              // Using String literal
String s2 = new String("World");  // Using 'new' keyword
```

2.
3. **Commonly Used Methods**:

    - `length()`: Returns the length of the string.
    - `charAt(int index)`: Retrieves the character at a specific index.
    - `toUpperCase()` **and** `toLowerCase()`: Converts case.
    - `trim()`: Removes leading and trailing spaces.
    - `substring(int start, int end)`: Extracts part of the string.
    - `replace(char oldChar, char newChar)`: Replaces characters.

**Example**:

```
java
Copy code
String str = " Java ";
System.out.println(str.trim());              // "Java"
System.out.println(str.toUpperCase());       // " JAVA "
System.out.println(str.substring(1, 4));     // "Jav"
```

4.

- ---

## String Operations

1. **Concatenation**: Combine two strings using:

    - `+` **operator**
    - `concat()` **method**

**Example**:

```
java
Copy code
String first = "Hello";
String second = "World";
```

```
System.out.println(first + " " + second);      // "Hello World"
System.out.println(first.concat(" ").concat(second)); // "Hello World"
```

2.

- ---

1. **String Comparison**:

    - `equals()`: Compares content (case-sensitive).
    - `equalsIgnoreCase()`: Compares content ignoring case.
    - `compareTo()`: Lexicographically compares two strings.

**Example**:

```
 java
Copy code
String s1 = "test";
String s2 = "Test";
System.out.println(s1.equals(s2));            // false
System.out.println(s1.equalsIgnoreCase(s2));  // true
System.out.println(s1.compareTo(s2));         // Positive value
```

2.

- ---

1. **Searching in Strings**:

    - `indexOf(char)`: Returns the index of the first occurrence of a character.
    - `lastIndexOf(char)`: Returns the index of the last occurrence of a character.
    - `contains()`: Checks if a substring exists.

**Example**:

```
 java
Copy code
String str = "Welcome to Java";
System.out.println(str.indexOf('e'));         // 1
System.out.println(str.lastIndexOf('e'));     // 11
System.out.println(str.contains("Java"));     // true
```

2.

- ---

## Advanced String Manipulation

**Splitting Strings**: Split a string into parts using a delimiter with the `split()` method.

**Example**:

java
Copy code
```
String sentence = "Learn Java Programming";
String[] words = sentence.split(" ");
for (String word : words) {
    System.out.println(word); // "Learn", "Java", "Programming"
}
```

1.

- ---

**String Joining**: Combine multiple strings using `String.join()`.

**Example**:

java
Copy code
```
String joined = String.join(", ", "Apple", "Banana", "Cherry");
System.out.println(joined);  // "Apple, Banana, Cherry"
```

1.

- ---

**String Reversal**: Reverse a string manually or using `StringBuilder`.

**Example**:

java
Copy code
```
String original = "Java";
String reversed = new StringBuilder(original).reverse().toString();
System.out.println(reversed);  // "avaJ"
```

1.

- ---

**Replacing Characters**: Use `replace()` or `replaceAll()` for replacing characters or substrings.

**Example**:

java
Copy code
```
String text = "I like Python";
System.out.println(text.replace("Python", "Java"));  // "I like Java"
```

1.

---

## Interview Tips and Common Questions

## 1. Check for Palindrome

**Question**: Write a program to check if a string is a palindrome.

**Solution**:

java
Copy code
```
String str = "madam";
String reversed = new StringBuilder(str).reverse().toString();
System.out.println(str.equals(reversed) ? "Palindrome" : "Not a Palindrome");
```

---

## 2. Find Duplicate Characters in a String

**Question**: How would you find duplicate characters in a string?

**Solution**:

java
Copy code
```
import java.util.HashMap;

public class DuplicateChars {
```

```
    public static void main(String[] args) {
        String str = "programming";
        HashMap<Character, Integer> map = new HashMap<>();

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

        map.forEach((k, v) -> {
            if (v > 1) {
                System.out.println(k + " appears " + v + " times.");
            }
        });
    }
}
```

---

## 3. Count Vowels and Consonants

**Question**: Write a program to count vowels and consonants in a string.

**Solution**:

```
 java
Copy code
public class VowelConsonant {
    public static void main(String[] args) {
        String str = "Automation";
        int vowels = 0, consonants = 0;

        for (char c : str.toLowerCase().toCharArray()) {
            if ("aeiou".indexOf(c) != -1) {
                vowels++;
            } else if (Character.isLetter(c)) {
                consonants++;
            }
        }
        System.out.println("Vowels: " + vowels + ", Consonants: " + consonants);
    }
}
```

- 

- Array Manipulation

## Array Manipulation in Java

**Definition**:

Array manipulation involves operations performed on arrays to create, access, modify, sort, and analyze their contents. Arrays in Java are fixed-size data structures that store elements of the same data type.

---

## Basic Array Operations

1. **Declaring and Initializing Arrays**:

Single-dimensional array:

java
Copy code
```
int[] numbers = new int[5];          // Declaration with size
int[] nums = {10, 20, 30, 40, 50};  // Declaration with values
```

- 

Multi-dimensional array:

java
Copy code
```
int[][] matrix = new int[3][3];  // 2D Array
int[][] matrixValues = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

- 

- ---

1. **Accessing and Modifying Array Elements**:

    - Access elements by index (0-based).
    - Modify elements by assigning new values.

**Example**:

```
 java
Copy code
int[] arr = {10, 20, 30};
System.out.println(arr[1]);  // Output: 20

arr[1] = 25;                 // Modify element
System.out.println(arr[1]); // Output: 25
```

2.

---

1. **Traversing Arrays**:
    - Using loops (`for`, `for-each`, `while`).

**Example**:

```
 java
Copy code
int[] nums = {10, 20, 30, 40};
for (int i = 0; i < nums.length; i++) {
    System.out.println(nums[i]); // Access elements
}

// Using for-each loop
for (int num : nums) {
    System.out.println(num);
}
```

2.

---

## Common Array Operations

**Finding the Length**: Use the `.length` property.

```
 java
Copy code
int[] arr = {10, 20, 30};
System.out.println(arr.length); // Output: 3
```

1.
2. **Copying Arrays**:
    - Using `System.arraycopy()`.
    - Using `Arrays.copyOf()`.

**Example**:

java
Copy code
```
int[] source = {1, 2, 3};
int[] dest = new int[source.length];
System.arraycopy(source, 0, dest, 0, source.length);

System.out.println(Arrays.toString(dest)); // Output: [1, 2, 3]
```

3.
4. **Sorting Arrays**:
    - Use `Arrays.sort()`.

**Example**:

java
Copy code
```
int[] nums = {40, 10, 30, 20};
Arrays.sort(nums);
System.out.println(Arrays.toString(nums)); // Output: [10, 20, 30, 40]
```

5.
6. **Searching in Arrays**:
    - Use `Arrays.binarySearch()` (array must be sorted).

**Example**:

java
Copy code
```
int[] nums = {10, 20, 30, 40};
int index = Arrays.binarySearch(nums, 30);
System.out.println(index); // Output: 2
```

7.

- ---

## Advanced Array Operations

**Reversing an Array**: Reverse elements in place using a loop.

java
Copy code

```
int[] nums = {10, 20, 30, 40};
int start = 0, end = nums.length - 1;
while (start < end) {
    int temp = nums[start];
    nums[start] = nums[end];
    nums[end] = temp;
    start++;
    end--;
}
System.out.println(Arrays.toString(nums)); // Output: [40, 30, 20, 10]
```

1.

**Removing Duplicates from an Array**: Use a `Set` to remove duplicates.

**Example**:

java
Copy code

```
int[] nums = {10, 20, 20, 30, 30, 40};
Set<Integer> set = new LinkedHashSet<>();
for (int num : nums) {
    set.add(num);
}
System.out.println(set); // Output: [10, 20, 30, 40]
```

2.

**Merging Two Arrays**: Combine two arrays into one.

**Example**:

java
Copy code

```
int[] arr1 = {1, 2, 3};
int[] arr2 = {4, 5, 6};
int[] merged = new int[arr1.length + arr2.length];

System.arraycopy(arr1, 0, merged, 0, arr1.length);
System.arraycopy(arr2, 0, merged, arr1.length, arr2.length);

System.out.println(Arrays.toString(merged)); // Output: [1, 2, 3, 4, 5, 6]
```

3.

**Rotating an Array**: Rotate the elements by a given number of positions.

java
Copy code

```
public class RotateArray {
    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 4, 5};
        int k = 2; // Number of positions to rotate

        int n = nums.length;
        int[] rotated = new int[n];

        for (int i = 0; i < n; i++) {
            rotated[(i + k) % n] = nums[i];
        }

        System.out.println(Arrays.toString(rotated)); // Output: [4, 5, 1, 2, 3]
    }
}
```

4.

- ---

## Multi-Dimensional Arrays

**Traversing a 2D Array**: Use nested loops.

**Example**:

java
Copy code

```
int[][] matrix = {
```

```
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println();
}
```

1.

**Flattening a 2D Array**: Convert it to a 1D array.

```
 java
Copy code
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
int[] flat = Arrays.stream(matrix).flatMapToInt(Arrays::stream).toArray();
System.out.println(Arrays.toString(flat)); // Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.

- ---

## Interview Questions

**Find the Second Largest Element**:

```
 java
Copy code
int[] nums = {10, 20, 30, 40, 50};
Arrays.sort(nums);
System.out.println(nums[nums.length - 2]); // Output: 40
```

1.

**Find Missing Number in Array**:

```
 java
Copy code
int[] nums = {1, 2, 3, 5};
int n = nums.length + 1; // Array should have n elements
int sum = n * (n + 1) / 2; // Sum of first n natural numbers

for (int num : nums) {
    sum -= num;
}
System.out.println("Missing number: " + sum); // Output: 4
```

2.

**Check if Array is Sorted**:

```
 java
Copy code
boolean isSorted = true;
for (int i = 1; i < nums.length; i++) {
    if (nums[i] < nums[i - 1]) {
        isSorted = false;
        break;
    }
}
System.out.println(isSorted ? "Sorted" : "Not Sorted");
```

3.

**Find Duplicates**:

```
 java
Copy code
Set<Integer> seen = new HashSet<>();
for (int num : nums) {
    if (!seen.add(num)) {
        System.out.println("Duplicate: " + num);
    }
}
```

4.

- Number Operation

## 1. Prime Number Check

**Definition**:

A prime number is a number greater than 1 that has no divisors other than 1 and itself.

**Algorithm**:

1. Check if the number is less than 2 (not prime).
2. Iterate from 2 to the square root of the number.
3. If any number divides the input number, it is not prime.

**Code Example**:

```
 java
Copy code
public class PrimeNumberCheck {
    public static void main(String[] args) {
        int num = 29; // Input number
        boolean isPrime = true;

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

        System.out.println(num + " is " + (isPrime ? "a Prime Number" : "not a Prime Number"));
    }
}
```

- **Example Input**: `29`

  **Example Output**: `29 is a Prime Number`

---

## 2. Fibonacci Series

**Definition**:

A Fibonacci series is a sequence where each number is the sum of the two preceding numbers, starting with 0 and 1.

**Algorithm**:

1. Start with `a = 0` and `b = 1`.
2. Generate the next number by adding `a` and `b`.
3. Repeat for the desired length of the series.

**Code Example**:

```
 java
Copy code
public class FibonacciSeries {
    public static void main(String[] args) {
        int n = 10; // Number of terms
        int a = 0, b = 1;

        System.out.println("Fibonacci Series:");
        for (int i = 0; i < n; i++) {
            System.out.print(a + " ");
            int next = a + b;
            a = b;
            b = next;
        }
    }
}
```

- **Example Input**: `n = 10`

  **Example Output**: `0 1 1 2 3 5 8 13 21 34`

---

## 3. Factorial Calculation

**Definition**:

The factorial of a non-negative integer `n` is the product of all positive integers less than or equal to `n` (denoted as `n!`).

**Algorithm**:

1. Initialize a result variable as 1.
2. Multiply it with numbers from 1 to `n`.

**Code Example**:

```
 java
Copy code
public class FactorialCalculation {
    public static void main(String[] args) {
        int num = 5; // Input number
        long factorial = 1;

        for (int i = 1; i <= num; i++) {
            factorial *= i;
        }

        System.out.println("Factorial of " + num + " is " + factorial);
    }
}
```

- **Example Input**: `5`

  **Example Output**: `Factorial of 5 is 120`

---

## 4. Armstrong Number Check

**Definition**:

An Armstrong number (or narcissistic number) is a number that is equal to the sum of its digits raised to the power of the number of digits.

**Algorithm**:

1. Count the digits of the number.
2. Compute the sum of each digit raised to the power of the total digits.
3. Check if the sum equals the original number.

**Code Example**:

```
 java
Copy code
```

```
public class ArmstrongNumberCheck {
    public static void main(String[] args) {
        int num = 153; // Input number
        int original = num;
        int sum = 0;

        int digits = String.valueOf(num).length(); // Count digits

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

        System.out.println(original + " is " + (sum == original ? "an Armstrong Number" : "not an
Armstrong Number"));
    }
}
```

- **Example Input**: `153`

**Example Output**: `153 is an Armstrong Number`

---

## Summary of Outputs

| Operation | Input | Output |
|---|---|---|
| Prime Number Check | `29` | `29 is a Prime Number` |
| Fibonacci Series | `n = 10` | `0 1 1 2 3 5 8 13 21 34` |
| Factorial Calculation | `5` | `Factorial of 5 is 120` |
| Armstrong Number Check | `153` | `153 is an Armstrong Number` |

Java Concept

## 1. Inheritance

**Definition**:

Inheritance allows a class (child or subclass) to acquire properties and methods of another class (parent or superclass). It promotes code reusability and establishes a parent-child relationship.

**Example**:

java
Copy code

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

**Output**:

Copy code

```
This animal eats food.
The dog barks.
```

- **Interview Question**:
    1. What is the difference between single and multiple inheritance?

---

## 2. Polymorphism

**Definition**:

Polymorphism allows methods to take many forms. It is achieved through **method overloading** (compile-time polymorphism) and **method overriding** (runtime polymorphism).

## a. Method Overloading:

The same method name with different parameter lists within the same class.

**Example**:

java
Copy code

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }
}

public class MethodOverloadingExample {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println("Sum (int): " + calc.add(5, 10));
        System.out.println("Sum (double): " + calc.add(5.5, 10.5));
    }
}
```

**Output**:

sql
Copy code

```
Sum (int): 15
Sum (double): 16.0
```

## b. Method Overriding:

A child class provides a specific implementation of a method already defined in its parent class.

**Example**:

java

Copy code
```
class Animal {
    void sound() {
        System.out.println("Some generic animal sound.");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks.");
    }
}

public class MethodOverridingExample {
    public static void main(String[] args) {
        Animal myAnimal = new Dog(); // Runtime polymorphism
        myAnimal.sound();
    }
}
```

**Output**:

Copy code
```
Dog barks.
```

- **Interview Question**:

    1. Can static methods be overridden in Java? Why or why not?

---

## 3. Encapsulation

**Definition**:

Encapsulation involves bundling data (variables) and methods that operate on the data into a single unit (class). It restricts direct access to some components, making the class more secure.

**Example**:

java
Copy code
```
class Employee {
```

```
    private String name;
    private int age;

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if (age > 18) { // Validation
            this.age = age;
        } else {
            System.out.println("Age must be greater than 18.");
        }
    }
}

public class EncapsulationExample {
    public static void main(String[] args) {
        Employee emp = new Employee();
        emp.setName("John");
        emp.setAge(25);

        System.out.println("Employee Name: " + emp.getName());
        System.out.println("Employee Age: " + emp.getAge());
    }
}
```

**Output**:

```
 yaml
Copy code
Employee Name: John
Employee Age: 25
```

- **Interview Question**:

    1. How does encapsulation improve code maintainability?

---

## 4. Abstraction

**Definition**:

Abstraction hides the implementation details and shows only essential features. It is achieved using **abstract classes** and **interfaces**.

## a. Abstract Classes

An abstract class is a class that cannot be instantiated and may contain abstract methods (methods without a body).

**Example**:

java
Copy code

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

    void stop() { // Concrete method
        System.out.println("The vehicle stops.");
    }
}

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

public class AbstractClassExample {
    public static void main(String[] args) {
        Vehicle myCar = new Car();
        myCar.start();
        myCar.stop();
    }
}
```

**Output**:

vbnet
Copy code
The car starts with a key.
The vehicle stops.

## b. Interfaces

An interface is a contract that defines methods that must be implemented by a class.

**Example**:

java
Copy code

```
interface Animal {
    void eat();
    void sleep();
}

class Cat implements Animal {
    @Override
    public void eat() {
        System.out.println("The cat eats fish.");
    }

    @Override
    public void sleep() {
        System.out.println("The cat sleeps in a basket.");
    }
}

public class InterfaceExample {
    public static void main(String[] args) {
        Animal myCat = new Cat();
        myCat.eat();
        myCat.sleep();
    }
}
```

**Output**:

bash
Copy code

The cat eats fish.
The cat sleeps in a basket.

- **Interview Question**:

    1. What is the difference between an abstract class and an interface?

- ---

## Comparison Table: Abstract Classes vs. Interfaces

| Feature | Abstract Class | Interface |
|---|---|---|
| **Methods** | Can have both abstract and concrete methods. | Only abstract methods (default methods allowed from Java 8). |
| **Variables** | Can have instance variables. | Only static and final variables. |
| **Inheritance** | Supports single inheritance. | Supports multiple inheritance. |
| **Access Modifiers** | Can use any access modifier. | All methods are `public` by default. |

- ---

## Common Interview Questions

1. What are the key principles of OOP, and how are they implemented in Java?
2. Can you explain the difference between method overloading and method overriding?
3. Why is encapsulation important in Java?
4. When should you use abstract classes vs. interfaces?