1. What is Object-Oriented Programming (OOP), and what are its main principles?

Answer: Object-Oriented Programming (OOP) is a programming paradigm that uses "objects" to represent data and methods to manipulate that data. The main principles of OOP are:

- Encapsulation: Bundling data and methods that operate on the data within a class.

- Inheritance: Mechanism by which one class can inherit fields and methods from another class.

- Polymorphism: Ability to present the same interface for different underlying data types.

- Abstraction: Hiding complex implementation details and showing only the essential features of an object.

2. Explain the concept of encapsulation in Java and provide an example.

Answer: Encapsulation is the principle of restricting access to certain components of an object and bundling the data (attributes) and methods (functions) that operate on the data into a single unit called a class. This is achieved using access modifiers (private, public, protected).

Example:

java

public class BankAccount {

private double balance; // Encapsulated variable

public void deposit(double amount) {

if (amount > 0) {

balance += amount; // Method to modify balance

}

}

public double getBalance() {

return balance; // Method to access balance

}

}

3. How does inheritance work in Java? What are its benefits?

Answer: Inheritance allows a new class (subclass) to inherit properties and methods from an existing class (superclass). This promotes code reusability and establishes a hierarchical relationship between classes. Benefits include:

- Reusability of code.

- Method overriding for specific behavior in subclasses.

- Establishing a natural hierarchy.

4. What is polymorphism in Java? Describe the difference between method overloading and method overriding.

Answer: Polymorphism allows methods to perform different tasks based on the object that it is acting upon.

-Method Overloading: Same method name with different parameters within the same class.

- Method Overriding: Subclass provides a specific implementation of a method already defined in its superclass.

Example of Overloading:

java

public void display(int a) { }

public void display(String b) { }

Example of Overriding:

java

class Animal {

void sound() { System.out.println("Animal makes sound"); }

}

class Dog extends Animal {

void sound() { System.out.println("Dog barks"); } // Overriding

}

5. Define abstraction in Java. How can it be implemented using abstract classes and interfaces

Answer: Abstraction is the concept of hiding complex implementation details and exposing only the essential features of an object. It can be implemented using:

- Abstract Classes: Classes that cannot be instantiated and can have abstract methods (without implementation).

- Interfaces: A contract that classes can implement, containing abstract methods.

Example of an abstract class:

java

abstract class Shape {

abstract void draw(); // Abstract method

}

Example of an interface:

java

interface Drawable {

void draw(); // Abstract method

}

6. What are the commonly used String methods in Java? Provide examples of how to use at least three of them.

Answer: Commonly used String methods include:

- `length()`: Returns the length of the string.

- `charAt(int index)`: Returns the character at the specified index.

- `substring(int beginIndex, int endIndex)`: Returns a substring.

Example:

java

String str = "Hello, World!";

System.out.println(str.length()); // Outputs: 13

System.out.println(str.charAt(0)); // Outputs: H

System.out.println(str.substring(0, 5)); // Outputs: Hello

7. How do you convert a String to an integer in Java? What exceptions might you encounter during this process?

Answer: You can convert a String to an integer using `Integer.parseInt(String s)`. The potential exception is `NumberFormatException`, which occurs if the string is not a valid integer.

Example:

try {

int number = Integer.parseInt("123");

} catch (NumberFormatException e) {

System.out.println("Invalid number format.");

}

8. Explain the purpose of the `try`, `catch`, and `finally` blocks in Java exception handling.

Answer:

- try: Contains code that may throw an exception.

- catch: Catches and handles the exception thrown by the try block.

- finally: A block that executes after the try and catch blocks, regardless of whether an exception was thrown or caught. It is typically used for cleanup activities, such as closing resources.

Example:

java

try {

int result = 10 / 0; // This will throw an exception

} catch (ArithmeticException e) {

System.out.println("Cannot divide by zero.");

} finally {

System.out.println("This will always execute.");

}

9. What is the difference between checked and unchecked exceptions in Java?

Answer:

-Checked Exceptions: These are exceptions that are checked at compile-time. The programmer is required to handle these exceptions using try-catch blocks or by declaring them in the method signature with the `throws` keyword. Examples include `IOException` and `SQLException`.

- Unchecked Exceptions: These are exceptions that are not checked at compile-time, and they usually occur due to programming errors, such as `NullPointerException` and `ArrayIndexOutOfBoundsException`. They do not need to be declared or caught.

10. How can you create a custom exception in Java? Provide an example.

Answer: You can create a custom exception by extending the `Exception` class or any of its subclasses.

Example:

java

class InvalidAgeException extends Exception {

public InvalidAgeException(String message) {

super(message);

}

}

public class Test {

public static void validateAge(int age) throws InvalidAgeException {

if (age < 18) {

throw new InvalidAgeException("Age must be 18 or older.");

}

}

}

11. What is the significance of the `throws` keyword in method declarations?

Answer: The `throws` keyword is used in a method declaration to specify that the method can throw one or more exceptions. It informs the caller of the method that they need to handle these exceptions, either with a try-catch block or by declaring them in their own method signature.

Example:

java

public void myMethod() throws IOException {

// Code that may throw IOException

}

12. How can you handle multiple exceptions in a single `catch` block in Java?

Answer: You can handle multiple exceptions in a single catch block by using the pipe (`|`) operator to separate the exception types. This is useful when the handling logic is the same for multiple exceptions.

Example:

java

try {

// Code that may throw multiple exceptions

} catch (IOException | SQLException e) {

System.out.println("An exception occurred: " + e.getMessage());

}

13. What is the role of the `StringBuilder` class in Java, and how does it differ from the `String` class?

Answer: The `StringBuilder` class is used to create mutable strings, which means you can modify the string without creating a new object. In contrast, the `String` class is immutable, meaning any modification creates a new string object. `StringBuilder` is more efficient for string manipulation in scenarios where the string changes frequently.

Example:

java

StringBuilder sb = new StringBuilder("Hello");

sb.append(" World!"); // Modifies the existing object

System.out.println(sb.toString()); // Outputs: Hello World!

14. Describe how to read user input using the Scanner class in Java and handle potential input exceptions.

Answer: You can read user input using the `Scanner` class, which provides methods to read different types of input. To handle potential input exceptions, you can use try-catch blocks.

Example:

java

import java.util.Scanner;

public class UserInput {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

System.out.print("Enter an integer: ");

int number = scanner.nextInt();

System.out.println("You entered: " + number);

} catch (InputMismatchException e) {

System.out.println("Invalid input. Please enter an integer.");

} finally {

scanner.close();

}

}

}

15.What is stream?