**Java Programming Test 2024/2025**

**OKORO PRAISE CHIBUIKE**

**2023/254975**

**COMPUTER SCIENCE**

## **Question 1: Write a Java program to print "Hello, World!"**

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

## **Question 2: Explain the difference between == and .equals() in Java. Show with code examples and outputs.**

In Java:  
- `==` checks if two references point to the same object in memory.  
- `.equals()` checks if two objects are logically "equal".  
  
Example:  
```java  
String a = new String("test");  
String b = new String("test");  
  
System.out.println(a == b); // false, different references  
System.out.println(a.equals(b)); // true, same content  
```

## **Question 3: What is the use of the main method in Java?**

The `main` method is the entry point for any standalone Java application. When you run a Java class, the JVM looks for this method to begin execution.  
  
```java  
public static void main(String[] args)  
```  
- `public`: accessible from anywhere.  
- `static`: no need to create an object to call it.  
- `void`: does not return any value.  
- `String[] args`: accepts command-line arguments.

## **Question 4: Write a Java program to add two numbers entered by the user.**

```java  
import java.util.Scanner;  
  
public class AddNumbers {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
  
 System.out.print("Enter first number: ");  
 int num1 = scanner.nextInt();  
 System.out.print("Enter second number: ");  
 int num2 = scanner.nextInt();  
  
 int sum = num1 + num2;  
 System.out.println("Sum: " + sum);  
  
 scanner.close();  
 }  
}  
```

## **Question 5: What is the difference between int, Integer, and String?**

- `int` is a primitive data type representing an integer.  
- `Integer` is a wrapper class for `int` that allows using int as an object.  
- `String` is a class that represents a sequence of characters.  
  
Example:  
```java  
int a = 5;  
Integer b = 10;  
String c = "15";  
```

## **Question 6: Write a program to check if a number is even or odd.**

```java  
import java.util.Scanner;  
  
public class EvenOddCheck {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter a number: ");  
 int num = scanner.nextInt();  
 if (num % 2 == 0) {  
 System.out.println(num + " is Even.");  
 } else {  
 System.out.println(num + " is Odd.");  
 }  
 scanner.close();  
 }  
}  
```

## **Question 7: Write a program to find the largest among three numbers.**

```java  
import java.util.Scanner;  
  
public class LargestOfThree {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter first number: ");  
 int a = scanner.nextInt();  
 System.out.print("Enter second number: ");  
 int b = scanner.nextInt();  
 System.out.print("Enter third number: ");  
 int c = scanner.nextInt();  
  
 int largest = a;  
 if (b > largest) largest = b;  
 if (c > largest) largest = c;  
  
 System.out.println("Largest number is: " + largest);  
 scanner.close();  
 }  
}  
```

## **Question 8: Explain the difference between while, for, and do-while loops in Java.**

- `while` loop: checks the condition before executing the loop body.  
- `for` loop: concise loop with initializer, condition, and increment.  
- `do-while` loop: runs the loop body at least once before checking the condition.  
  
```java  
// while loop  
int i = 0;  
while (i < 3) {  
 System.out.println("While loop: " + i);  
 i++;  
}  
  
// for loop  
for (int j = 0; j < 3; j++) {  
 System.out.println("For loop: " + j);  
}  
  
// do-while loop  
int k = 0;  
do {  
 System.out.println("Do-while loop: " + k);  
 k++;  
} while (k < 3);  
```

## **Question 9: Write a Java program to print the multiplication table of any number.**

```java  
import java.util.Scanner;  
  
public class MultiplicationTable {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter a number: ");  
 int number = scanner.nextInt();  
  
 for (int i = 1; i <= 10; i++) {  
 System.out.println(number + " x " + i + " = " + (number \* i));  
 }  
 scanner.close();  
 }  
}  
```

## **Question 10: Explain the four pillars of OOP in Java.**

1. \*\*Encapsulation\*\*: Wrapping data and methods in a single unit (class).  
2. \*\*Inheritance\*\*: Acquiring properties of one class into another.  
3. \*\*Polymorphism\*\*: One interface, many implementations (method overloading/overriding).  
4. \*\*Abstraction\*\*: Hiding implementation details and showing only essentials.

## **Question 11: Create a class Student with properties name, matricNo, and score, and add methods to display the student's info.**

```java  
public class Student {  
 String name;  
 String matricNo;  
 double score;  
  
 public Student(String name, String matricNo, double score) {  
 this.name = name;  
 this.matricNo = matricNo;  
 this.score = score;  
 }  
  
 public void displayInfo() {  
 System.out.println("Name: " + name);  
 System.out.println("Matric No: " + matricNo);  
 System.out.println("Score: " + score);  
 }  
}  
```

## **Question 12: What is method overloading? Give a code example.**

Method overloading means defining multiple methods with the same name but different parameters.  
  
```java  
public class Calculator {  
 int add(int a, int b) {  
 return a + b;  
 }  
  
 double add(double a, double b) {  
 return a + b;  
 }  
}  
```

## **Question 13: What is inheritance? Create a base class Person and a subclass Teacher.**

Inheritance allows a class to inherit properties and methods from another class.  
  
```java  
class Person {  
 String name;  
 int age;  
  
 public void display() {  
 System.out.println("Name: " + name + ", Age: " + age);  
 }  
}  
  
class Teacher extends Person {  
 String subject;  
  
 public void teach() {  
 System.out.println(name + " is teaching " + subject);  
 }  
}  
```

## **Question 14: What does it mean to write “clean code”? Give 3 practices that make code clean and maintainable.**

Clean code is readable, understandable, and easy to maintain. Practices include:  
1. Using meaningful variable and method names.  
2. Keeping methods short and focused.  
3. Writing comments to explain complex logic.

## **Question 15: Why should you avoid writing very long methods in Java programs?**

Long methods are hard to read, test, and debug. Breaking them into smaller methods improves readability, reusability, and maintainability.

## **Question 16: What naming conventions should be followed in Java for: Classes, Variables, Methods.**

- Classes: PascalCase (e.g., `StudentInfo`)  
- Variables: camelCase (e.g., `studentName`)  
- Methods: camelCase (e.g., `calculateGrade`)  
  
```java  
public class StudentInfo {  
 String studentName;  
 int studentAge;  
  
 public void displayInfo() {  
 System.out.println(studentName + ", " + studentAge);  
 }  
}  
```

## **Question 17: What is the importance of breaking your Java program into methods?**

It makes code modular, reusable, easier to test, debug, and understand. Each method handles a specific task.

## **Question 18: Explain the concept of DRY (Don’t Repeat Yourself) with a Java code example.**

DRY principle avoids code duplication.  
  
```java  
public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public void displayAddition(int x, int y) {  
 int result = add(x, y);  
 System.out.println("Sum: " + result);  
 }  
}  
```

## **Question 19: What are the benefits of using classes and objects instead of writing all logic in the main method?**

- Encourages modular design.  
- Improves readability and maintainability.  
- Facilitates code reuse and testing.

## **Question 20: Why is testing important during program development?**

Testing ensures your code works as expected, catches bugs early, and gives confidence during future changes.

## **Question 21: What is the difference between syntax error, runtime error, and logic error?**

- Syntax Error: Mistake in code structure (e.g., missing semicolon).  
- Runtime Error: Occurs during execution (e.g., divide by zero).  
- Logic Error: Code runs but gives wrong result.

## **Question 22: How would you test a method that calculates the average of five numbers?**

- Write test cases with known inputs and expected outputs.  
- Example: For input 10, 20, 30, 40, 50, expect 30.  
  
```java  
public double average(int[] nums) {  
 int sum = 0;  
 for (int n : nums) sum += n;  
 return sum / 5.0;  
}  
```

## **Question 23: Why should Java developers write comments in their code?**

To explain complex logic, improve readability, and help future developers understand the code's purpose.

## **Question 24: What are JavaDoc comments and how are they different from regular comments?**

JavaDoc comments (`/\*\* ... \*/`) generate documentation automatically. Regular comments (`//` or `/\* \*/`) are for internal code explanation.

## **Question 25: Write a sample Java method with JavaDoc comments.**

```java  
/\*\*  
 \* Adds two integers and returns the result.  
 \* @param a First integer  
 \* @param b Second integer  
 \* @return Sum of a and b  
 \*/  
public int add(int a, int b) {  
 return a + b;  
}  
```

## **Question 26: What is version control and why is it important in team projects?**

Version control tracks changes, helps collaboration, avoids conflicts, and maintains history (e.g., Git).

## **Question 27: How would you explain the concept of “code refactoring” to a junior developer?**

Refactoring is restructuring code without changing its behavior to improve readability, performance, or maintainability.

## **Question 28: What tools can Java developers use to collaborate on large projects? Attach screenshots of 3 examples.**

- GitHub: Code hosting and version control.  
- Git: Local version control system.  
- IntelliJ IDEA: IDE with collaboration features.  
  
(Screenshots Placeholder)

## **Question 29: Mention 5 best practices you follow when developing a Java program.**

1. Use meaningful names.  
2. Keep methods short.  
3. Write comments and documentation.  
4. Follow naming conventions.  
5. Write unit tests.

## **Question 30: What is code readability, and why is it more important than “smart” code?**

Readable code is easy to understand. It helps others (and your future self) maintain and debug code. Smart code may be efficient but hard to follow.

## **Question 31: Build a command-line application that keeps track of student grades and allows adding, updating, and viewing records.**

```java  
import java.util.\*;  
  
class Student {  
 String name;  
 int grade;  
  
 Student(String name, int grade) {  
 this.name = name;  
 this.grade = grade;  
 }  
}  
  
public class GradeManager {  
 static Map<String, Student> records = new HashMap<>();  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 while (true) {  
 System.out.println("1. Add  
2. Update  
3. View  
4. Exit");  
 int choice = scanner.nextInt();  
 scanner.nextLine();  
  
 if (choice == 1) {  
 System.out.print("Name: ");  
 String name = scanner.nextLine();  
 System.out.print("Grade: ");  
 int grade = scanner.nextInt();  
 records.put(name, new Student(name, grade));  
 } else if (choice == 2) {  
 System.out.print("Name: ");  
 String name = scanner.nextLine();  
 if (records.containsKey(name)) {  
 System.out.print("New Grade: ");  
 int grade = scanner.nextInt();  
 records.put(name, new Student(name, grade));  
 } else System.out.println("Student not found.");  
 } else if (choice == 3) {  
 for (Student s : records.values()) {  
 System.out.println(s.name + ": " + s.grade);  
 }  
 } else break;  
 }  
 scanner.close();  
 }  
}  
```

## **Question 32: Write a program that simulates a basic ATM system (check balance, deposit, withdraw).**

```java  
import java.util.Scanner;  
  
public class ATM {  
 static double balance = 1000;  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 while (true) {  
 System.out.println("1. Check Balance  
2. Deposit  
3. Withdraw  
4. Exit");  
 int choice = scanner.nextInt();  
  
 if (choice == 1) {  
 System.out.println("Balance: " + balance);  
 } else if (choice == 2) {  
 System.out.print("Enter amount: ");  
 double amount = scanner.nextDouble();  
 balance += amount;  
 } else if (choice == 3) {  
 System.out.print("Enter amount: ");  
 double amount = scanner.nextDouble();  
 if (amount <= balance) balance -= amount;  
 else System.out.println("Insufficient funds.");  
 } else break;  
 }  
 scanner.close();  
 }  
}  
```