Java Fundamentals: Questions 1–32

# Basics & Syntax

**1. Print "Hello, World!"**

public class HelloWorld {  
 public static void main(String[] args) {  
 System.out.println("Hello, World!");  
 }  
}

**2. Difference between == and .equals()**

String a = new String("test");  
String b = new String("test");  
  
System.out.println(a == b); // false  
System.out.println(a.equals(b)); // true

- '==' checks reference equality.  
- '.equals()' checks object content equality.

**3. Purpose of the main Method**

- It's the entry point of a Java application.  
- Syntax: public static void main(String[] args)

**4. Add Two Numbers Entered by User**

import java.util.Scanner;  
  
public class AddNumbers {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.in);  
 System.out.print("Enter two numbers: ");  
 int a = sc.nextInt(), b = sc.nextInt();  
 System.out.println("Sum = " + (a + b));  
 }  
}

**5. Difference Between int, Integer, String**

- int: primitive type  
- Integer: wrapper class of int  
- String: object for text

# Control Structures

**6. Even or Odd**

int num = 6;  
System.out.println(num % 2 == 0 ? "Even" : "Odd");

**7. Largest of Three Numbers**

int a = 5, b = 8, c = 3;  
int max = Math.max(a, Math.max(b, c));  
System.out.println("Max = " + max);

**8. Loop Differences**

- while: condition checked before loop body  
- do-while: condition checked after loop body  
- for: best for known iteration counts

**9. Multiplication Table**

int n = 4;  
for (int i = 1; i <= 10; i++)  
 System.out.println(n + " x " + i + " = " + (n \* i));

# OOP Concepts

**10. Four Pillars of OOP**

1. Encapsulation  
2. Inheritance  
3. Abstraction  
4. Polymorphism

**11. Student Class**

class Student {  
 String name, matricNo;  
 double score;  
  
 void display() {  
 System.out.println(name + " " + matricNo + " " + score);  
 }  
}

**12. Method Overloading**

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

**13. Inheritance Example**

class Person {  
 String name;  
 void greet() { System.out.println("Hello " + name); }  
}  
  
class Teacher extends Person {  
 String subject;  
 void teach() { System.out.println("Teaches " + subject); }  
}

# General Practices

**14. Clean Code Practices**

1. Meaningful variable names  
2. Keep methods short  
3. Use comments and consistent formatting

**15. Avoid Long Methods**

- Harder to read, debug, and test  
- Violates Single Responsibility Principle

**16. Naming Conventions**

- Class: StudentInfo  
- Variable: studentName  
- Method: calculateAverage()

class StudentInfo {  
 String studentName;  
  
 void calculateAverage() {  
 // logic  
 }  
}

**17. Importance of Methods**

- Reuse  
- Modularization  
- Easier testing and maintenance

**18. DRY Concept**

void printHeader() {  
 System.out.println("Welcome!");  
}  
// Call this method instead of repeating System.out.println()

**19. Benefits of Classes/Objects**

- Encapsulation  
- Reuse  
- Better code organization

# Testing & Debugging

**20. Importance of Testing**

- Ensures code works correctly  
- Prevents bugs  
- Saves time later

**21. Types of Errors**

- Syntax Error: e.g., missing semicolon  
- Runtime Error: e.g., divide by zero  
- Logic Error: wrong output but compiles and runs

**22. Test Average Method**

double average(int[] nums) {  
 int sum = 0;  
 for (int n : nums) sum += n;  
 return sum / 5.0;  
}  
// Test with: new int[]{10, 20, 30, 40, 50}

# Documentation & Comments

**23. Why Write Comments**

- Helps explain code  
- Aids future maintenance

**24. JavaDoc vs Regular Comments**

- // Single-line  
- /\* \*/ Multi-line  
- /\*\* \*/ JavaDoc for documentation tools

**25. JavaDoc Sample**

/\*\*  
 \* Calculates area of a rectangle.  
 \* @param length Length of rectangle  
 \* @param width Width of rectangle  
 \* @return Area  
 \*/  
public int area(int length, int width) {  
 return length \* width;  
}

# Versioning & Collaboration

**26. Importance of Version Control**

- Track changes  
- Collaborate easily  
- Revert to previous versions

**27. Explain Refactoring**

- Improve code structure without changing its behavior

**28. Collaboration Tools**

1. GitHub  
2. GitLab  
3. Bitbucket

# Good Practices Summary

**29. 5 Best Practices**

1. Use meaningful names  
2. Keep methods short  
3. Comment important logic  
4. Use version control  
5. Test often

**30. Code Readability vs Smart Code**

- Readable code is easier to maintain and collaborate on  
- Smart code is often confusing

# Advanced-Level Questions

**31. Student Grades App (Structure)**

class Student {  
 String name;  
 double grade;  
}  
  
class GradeApp {  
 static ArrayList<Student> students = new ArrayList<>();  
 // add, update, view methods here  
}

**32. Basic ATM Simulation**

double balance = 1000;  
  
void deposit(double amount) {  
 balance += amount;  
}  
  
void withdraw(double amount) {  
 if (balance >= amount) balance -= amount;  
 else System.out.println("Insufficient funds");  
}  
  
void checkBalance() {  
 System.out.println("Balance: " + balance);  
}