Question 1
What are these primitive data types and what is each one used for?

Answer
Java has **eight** primitive data types:

- **byte** – Stores very small whole numbers from **-128 to 127**. Used when memory is limited.
- **short** – Stores numbers from **-32,768 to 32,767**.
- **int** – Default choice for integers, stores large whole numbers from **-2,147,483,648 to 2,147,483,647**.
- **long** – Stores very large whole numbers up to about **9 quintillion**.
- **float** – Stores decimal numbers with single precision (about 6–7 digits accuracy).
- **double** – Stores decimal numbers with double precision (about 15–16 digits accuracy).
- **char** – Stores a single Unicode character (like 'A', '9', or '@').
- **boolean** – Stores either **true** or **false**.

Question 2
"How are primitive data types in Java different from non-primitive ones?"

Answer

- **Primitive data types** are the most basic forms of data (like int, boolean, char). They store values directly in memory and are not objects.
- **Non-primitive data types** (like String, arrays, classes) are objects that store references to the actual data in memory.
- Primitives are faster and require less memory, while non-primitives are more flexible and can store complex data structures.

Question 3
"Can you write a Java program that uses all primitive data types?"

Answer

java

CopyEdit

```
public class PrimitiveTypesExample {

  public static void main(String[] args) {

    byte b = 100;
```

```
    short s = 20000;
    int i = 1000000;
    long l = 123456789L;
    float f = 5.75f;
    double d = 19.99;
    char c = 'A';
    boolean bool = true;

    System.out.println("byte: " + b);
    System.out.println("short: " + s);
    System.out.println("int: " + i);
    System.out.println("long: " + l);
    System.out.println("float: " + f);
    System.out.println("double: " + d);
    System.out.println("char: " + c);
    System.out.println("boolean: " + bool);
  }
}
```

Question 4
"What does type casting mean in Java? Can you give an example of both implicit and explicit casting?"

Answer
Type casting means **changing a variable from one data type to another**.

- **Implicit casting (widening)** – Happens automatically when converting from a smaller to a larger data type.
  Example:

java

CopyEdit

int num = 10;

double result = num; // int → double

- **Explicit casting (narrowing)** – Done manually when converting from a larger to a smaller data type.
  Example:

java

CopyEdit

double value = 9.78;

int num = (int) value; // double → int

Question 5
"What are the default values of each primitive data type in Java?"

Answer
If a primitive variable is declared but not initialized, Java assigns it a default value:

- byte → **0**
- short → **0**
- int → **0**
- long → **0L**
- float → **0.0f**
- double → **0.0d**
- char → **'\u0000'** (null character)
- boolean → **false**

Section 2: Java Control Statements

Question 1
"What are control statements in Java, and what are the different types with examples?"

Answer
Control statements are used to **control the flow of execution** in a Java program. They include:

1. **Decision-making statements** – e.g., if, if-else, switch.

java

CopyEdit

if (x > 0) {

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

2. **Looping statements** – e.g., for, while, do-while.

```
java
CopyEdit
for (int i = 1; i <= 5; i++) {
  System.out.println(i);
}
```

3. **Jump statements** – e.g., break, continue, return.

```
java
CopyEdit
for (int i = 1; i <= 5; i++) {
  if (i == 3) break;
  System.out.println(i);
}
```

Question 2
"Write a Java program that uses both if-else and switch-case statements."

Answer

```
java
CopyEdit
public class ControlExample {
  public static void main(String[] args) {
    int num = 3;

    // if-else
    if (num > 0) {
      System.out.println("Positive");
    } else if (num < 0) {
```

```
        System.out.println("Negative");
    } else {
        System.out.println("Zero");
    }

    // switch-case
    switch (num) {
        case 1:
            System.out.println("One");
            break;
        case 2:
            System.out.println("Two");
            break;
        case 3:
            System.out.println("Three");
            break;
        default:
            System.out.println("Other number");
    }
  }
}
```

Question 3
"How is the break statement different from the continue statement in Java?"

Answer

- **break** – Completely stops the loop and moves control out of it.
- **continue** – Skips the current iteration and moves to the next loop iteration.

Example:

java

CopyEdit

```
for (int i = 1; i <= 5; i++) {
   if (i == 3) break; // loop stops at 3
   System.out.println(i);
}


for (int i = 1; i <= 5; i++) {
   if (i == 3) continue; // skips 3
   System.out.println(i);
}
```

Question 4
"Write a Java program that prints even numbers between 1 and 50 using a for loop."

Answer

java

CopyEdit

```
public class EvenNumbers {
   public static void main(String[] args) {
      for (int i = 2; i <= 50; i += 2) {
         System.out.print(i + " ");
      }
   }
}
```

Question 5
"What's the difference between while and do-while loops in Java?"

Answer

- **while loop** – Checks the condition first, and only then executes the block. If the condition is false from the start, the loop may never run.
- **do-while loop** – Executes the block at least once before checking the condition.

Example:

java

CopyEdit

```
int x = 5;


// while loop
while (x < 5) {
   System.out.println("While: " + x);
   x++;
}


// do-while loop
do {
   System.out.println("Do-While: " + x);
   x++;
} while (x < 5);
```

Section 3: Java Keywords and Operators

1. What are keywords in Java? List 10 commonly used keywords.
In Java, keywords are reserved words that have a predefined meaning in the language and cannot be used as identifiers (like variable names, method names, or class names). They are part of Java's syntax and are always written in lowercase.
Ten commonly used Java keywords are:
class, public, static, void, if, else, for, while, return, break.

---

2. Explain the purpose of the following keywords: static, final, this, super.

- **static** – Used to define variables and methods that belong to the class rather than any object instance.
- **final** – Used to declare constants, prevent method overriding, and inheritance of a class.
- **this** – Refers to the current instance of the class, often used to distinguish between instance variables and parameters.

- **super** – Refers to the immediate parent class object and can be used to access parent methods, constructors, and variables.

---

3. What are the types of operators in Java?
Java provides different categories of operators:

1. **Arithmetic Operators** – Perform basic mathematical operations (+, -, *, /, %).
2. **Relational Operators** – Compare values (==, !=, >, <, >=, <=).
3. **Logical Operators** – Combine boolean expressions (&&, ||, !).
4. **Assignment Operators** – Assign values to variables (=, +=, -=, *=).
5. **Unary Operators** – Operate on a single operand (++, --, +, -, !).
6. **Bitwise Operators** – Perform bit-level operations (&, |, ^, ~, <<, >>, >>>).
7. **Ternary Operator** – Conditional operator (?:).

---

4. Write a Java program demonstrating the use of arithmetic, relational, and logical operators.

java

CopyEdit

```
public class OperatorDemo {
   public static void main(String[] args) {
      int a = 10, b = 5;

      // Arithmetic Operators
      System.out.println("Addition: " + (a + b));
      System.out.println("Multiplication: " + (a * b));

      // Relational Operators
      System.out.println("Is a greater than b? " + (a > b));

      // Logical Operators
      System.out.println("Logical AND: " + (a > 0 && b > 0));
```

```
  }
}
```

---

5. What is operator precedence? How does it affect the outcome of expressions?
Operator precedence determines the order in which operations are performed in an expression. Higher precedence operators are evaluated before lower precedence ones. For example, in the expression 2 + 3 * 4, multiplication has higher precedence than addition, so it's evaluated first: 2 + (3 * 4) = 14.

Additional Questions – Java Data Types

6. What is the size and range of each primitive data type in Java?

- **byte** – 1 byte, range: -128 to 127
- **short** – 2 bytes, range: -32,768 to 32,767
- **int** – 4 bytes, range: -2,147,483,648 to 2,147,483,647
- **long** – 8 bytes, range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- **float** – 4 bytes, approx. ±3.4e38 (7 decimal digits precision)
- **double** – 8 bytes, approx. ±1.7e308 (15 decimal digits precision)
- **char** – 2 bytes, range: 0 to 65,535 (Unicode)
- **boolean** – 1 bit, values: true or false

---

7. How does Java handle overflow and underflow with numeric types?
Java does not throw an error for overflow or underflow in primitive numeric types. Instead, the value wraps around. For example:

java

CopyEdit

```
byte b = 127;

b++; // This causes overflow and wraps to -128
```

---

8. Write a program to convert a double value to an int without data loss.
It's not possible to convert from double to int without losing decimal data because int cannot store fractional values. However, we can convert safely for whole number doubles.

java

CopyEdit

```
public class DoubleToInt {
  public static void main(String[] args) {
    double d = 45.0;
    int i = (int) d; // Explicit casting
    System.out.println("Converted value: " + i);
  }
}
```

---

9. What is the difference between char and String in Java?

- **char** – A primitive data type that stores a single character using Unicode. Example: char c = 'A';
- **String** – A class in Java that represents a sequence of characters. Example: String s = "Hello";
  The main difference is that char is a single character, while String is a sequence of characters and is an object.

10: Explain wrapper classes and their use in Java.

In Java, wrapper classes are special classes that are used to convert primitive data types (like `int`, `char`, `double`, etc.) into objects. Every primitive type has a corresponding wrapper class in the `java.lang` package. For example, `int` has `Integer`, `char` has `Character`, `double` has `Double`, and so on.

Q6. Write a Java program using nested if statements.
Answer:
Nested if statements mean placing one if inside another. This is useful when we need to check multiple conditions step by step.
Example:

java

CopyEdit

```
public class NestedIfExample {
  public static void main(String[] args) {
    int age = 25;
```

```
    boolean hasID = true;

    if (age >= 18) {
      if (hasID) {
        System.out.println("You are allowed to enter.");
      } else {
        System.out.println("You need to show your ID.");
      }
    } else {
      System.out.println("You are too young to enter.");
    }
  }
}
```

Q7. Write a Java program to display the multiplication table of a number using a loop.
Answer:
A loop can be used to print the multiplication table of any number easily.
Example:

```
java

CopyEdit

public class MultiplicationTable {
  public static void main(String[] args) {
    int num = 5;

    for (int i = 1; i <= 10; i++) {
      System.out.println(num + " x " + i + " = " + (num * i));
    }
  }
}
```

Q8. How do you exit from nested loops in Java?
Answer:
You can exit from nested loops using **break with a label**. This stops the execution of all loops up to the labelled one.
Example:

java

CopyEdit

```
public class ExitNestedLoops {
  public static void main(String[] args) {
    outerLoop:
    for (int i = 1; i <= 3; i++) {
      for (int j = 1; j <= 3; j++) {
        if (i == 2 && j == 2) {
          break outerLoop; // exits both loops
        }
        System.out.println(i + " " + j);
      }
    }
  }
}
```

Q9. Compare and contrast for, while, and do-while loops.
Answer:

- **for loop** – Best when you know in advance how many times to repeat.
- **while loop** – Best when you want to run until a condition becomes false (may run zero times).
- **do-while loop** – Similar to while, but always runs at least once since the condition is checked after execution.

Example difference:

java

CopyEdit

```
// for loop example
```

```
for (int i = 0; i < 5; i++) {
   System.out.println("For: " + i);
}

// while loop example
int i = 0;
while (i < 5) {
   System.out.println("While: " + i);
   i++;
}

// do-while loop example
int j = 0;
do {
   System.out.println("Do-While: " + j);
   j++;
} while (j < 5);
```

Q10. Write a program that uses a switch-case to simulate a basic calculator.
Answer:

```
java
CopyEdit
import java.util.Scanner;

public class Calculator {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter first number: ");
      double num1 = sc.nextDouble();
```

```
        System.out.print("Enter second number: ");
        double num2 = sc.nextDouble();
        System.out.print("Enter operator (+, -, *, /): ");
        char op = sc.next().charAt(0);

        switch (op) {
            case '+': System.out.println("Result: " + (num1 + num2)); break;
            case '-': System.out.println("Result: " + (num1 - num2)); break;
            case '*': System.out.println("Result: " + (num1 * num2)); break;
            case '/':
                if (num2 != 0) {
                    System.out.println("Result: " + (num1 / num2));
                } else {
                    System.out.println("Cannot divide by zero.");
                }
                break;
            default: System.out.println("Invalid operator!");
        }
        sc.close();
    }
}
```

Java Keywords and Operators

Q6. What is the use of the instanceof keyword in Java?
Answer:
The instanceof keyword checks if an object belongs to a particular class or subclass. It returns true or false.
Example:

java

CopyEdit

```
String str = "Hello";
System.out.println(str instanceof String); // true
```

Q7. Explain the difference between == and .equals() in Java.
Answer:

- == checks if two references point to the **same memory location** (address comparison).
- .equals() checks if two objects have **the same value/content** (value comparison).

Example:

java

CopyEdit

```
String a = new String("Hello");
String b = new String("Hello");

System.out.println(a == b);      // false (different objects)
System.out.println(a.equals(b));  // true (same content)
```

Q8. Write a program using the ternary operator.
Answer:

java

CopyEdit

```
public class TernaryExample {
   public static void main(String[] args) {
      int num = 10;
      String result = (num % 2 == 0) ? "Even" : "Odd";
      System.out.println("The number is " + result);
   }
}
```

Q9. What is the use of this and super in method overriding?
Answer:

- this – Refers to the current object. It can be used to call current class methods or variables.
- super – Refers to the parent class. It is often used to call the parent's version of a method when overridden.

Example:

java

CopyEdit

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

class Child extends Parent {
  void show() {
    super.show(); // call parent method
    System.out.println("Child method");
  }
}

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

Q10. Explain bitwise operators with example.
Answer:
Bitwise operators work on binary representations of numbers. Common ones:

- & (AND)

- | (OR)
- ^ (XOR)
- ~ (NOT)
- << (left shift)
- >> (right shift)

Example:

java

CopyEdit

```
int a = 5;  // 0101
int b = 3;  // 0011

System.out.println(a & b); // 1  (0001)
System.out.println(a | b); // 7  (0111)
System.out.println(a ^ b); // 6  (0110)
System.out.println(~a);    // -6
System.out.println(a << 1);// 10 (1010)
System.out.println(a >> 1);// 2  (0010)
```