Section 1: Java Data Types

1. What are the different primitive data types available in Java?

Java has 8 primitive data types:

- **byte (8-bit signed integer)**
- **short (16-bit signed integer)**
- **int (32-bit signed integer)**
- **long (64-bit signed integer)**
- **float (32-bit floating point)**
- **double (64-bit floating point)**
- **char (16-bit Unicode character)**
- **boolean (true or false)**

2. Explain the difference between primitive and non-primitive data types in Java.

- **Primitive data types are predefined by the language and represent simple values like numbers, characters, and boolean values. They store actual values and have a fixed size.**
- **Non-primitive data types (also called reference types) include classes, interfaces, and arrays. They store references (addresses) to objects in memory and can have methods and fields.**

Primitive types are faster and require less memory, while non-primitive types offer more functionality and flexibility.

4. What is type casting? Provide an example of implicit and explicit casting in Java.

Type casting is converting a variable from one data type to another.

- **Implicit casting (Widening conversion): Automatic conversion from smaller to larger data type, safe and no data loss.**

Example:

java

```
int i = 100;

long l = i;  // int to long (32-bit to 64-bit)
```

- **Explicit casting (Narrowing conversion): Manually converting from larger to smaller data type by specifying the target type, possible data loss.**

Example:

java

```
double d = 100.04;

int i = (int) d;  // double to int, fractional part lost
```

5. What is the default value of each primitive data type in Java?

Primitive Type	Default Value
byte	0
short	0
int	0
long	0L

Primitive Type	Default Value
float	0.0f
double	0.0d
char	'\u0000' (null char)
boolean	false

Section 2: Java Control Statements

1. What are control statements in Java? List the types with examples.

Control statements control the flow of execution in a program. Types:

- Decision-making statements: if, if-else, switch
- Looping statements: for, while, do-while
- Branching statements: break, continue, return

Examples:

java

```
if (a > b) { // if statement
  System.out.println("a is greater");
}


for (int i = 0; i < 5; i++) { // for loop
  System.out.println(i);
```

}

3. What is the difference between break and continue statements?

- **break: Exits the nearest enclosing loop or switch statement completely.**
- **continue: Skips the current iteration of the loop and continues with the next iteration.**

Example:

java

```
for (int i = 0; i < 5; i++) {
   if (i == 3) break;    // Exits loop when i=3
   if (i == 2) continue; // Skips printing 2
   System.out.println(i);
}
```

Output:

text

```
0
1
```

5. Explain the differences between while and do-while loops with examples.

- **while loop: Checks the condition before executing the loop body.**

Example:

java

```
int i = 1;
```

```
while (i <= 5) {
   System.out.println(i);
   i++;
}
```

If the condition is false initially, the loop won't run.

- do-while loop: Executes the loop body once before checking the condition.

Example:

java

```
int i = 1;
do {
   System.out.println(i);
   i++;
} while (i <= 5);
```

The body executes *at least once* regardless of condition.

Section 3: Java Keywords and Operators

1. What are keywords in Java? List 10 commonly used keywords.

Keywords are reserved words predefined by Java with special meaning and cannot be used as identifiers.

Examples of commonly used keywords:

- class, public, static, void, int, if, else, for, while, return

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

- **static: Denotes that a member belongs to the class, not instances. Shared among all objects.**
- **final: Used to declare constants (variables that cannot be changed), methods that cannot be overridden, or classes that cannot be subclassed.**
- **this: Refers to the current instance of the class, used to disambiguate variables or call constructors.**
- **super: Refers to the parent class, used to access parent methods or constructors.**

3. What are the types of operators in Java?

Java operators are classified into:

- **Arithmetic Operators: +, -, *, /, %**
- **Relational Operators: ==, !=, >, <, >=, <=**
- **Logical Operators: &&, ||, !**
- **Bitwise Operators: &, |, ^, ~, <<, >>, >>>**
- **Assignment Operators: =, +=, -=, etc.**
- **Unary Operators: ++, --, +, -**
- **Ternary Operator: ?:**

5. What is operator precedence? How does it affect the outcome of expressions?

Operator precedence determines the order in which operators are evaluated in an expression. Operators with higher precedence are evaluated before those with lower precedence.

For example:

java

int result = 3 + 4 * 5; ***// Multiplication (*) has higher precedence than addition (+)***

result will be 23 because 4 * 5 is calculated first.

If you want to change the precedence, use parentheses.

Additional Questions

Java Data Types

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

Type	Size (bits)	Range
byte	8	-128 to 127
short	16	-32,768 to 32,767
int	32	-2,147,483,648 to 2,147,483,647
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	32	Approx ±3.40282347e+38F
double	64	Approx ±1.79769313486231570e+308
char	16	0 to 65,535 (Unicode characters)
boolean	1 (virtual)	true or false

7. How does Java handle overflow and underflow with numeric types?

- **Overflow occurs when a value exceeds the maximum limit of the data type; it wraps around to the minimum value (integer types).**
- **Underflow for floating-point types occurs when a value is too close to zero and is rounded to zero.**

Java does not throw exceptions on integer overflow or underflow; it silently wraps around.

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

- **char is a single 16-bit Unicode character (primitive type).**
- **String is a sequence (object) of characters.**

Example:

java

```
char c = 'A';     // single character
String s = "ABC";  // multiple characters (string)
```

10. Explain wrapper classes and their use in Java.

Wrapper classes provide an object representation for primitive types (e.g., Integer for int, Double for double). They allow primitives to be used where objects are required (collections, generics) and offer utility methods.

Example:

java

```
int i = 10;
Integer obj = Integer.valueOf(i);  // Boxing


int x = obj.intValue(); // Unboxing
```

Java Control Statements

8. How do you exit from nested loops in Java?

You can use labels with break to exit outer loops:

java

```
outer: for (int i=0; i<5; i++) {
   inner: for (int j=0; j<5; j++) {
      if (condition) break outer; // exits outer loop immediately
   }
}
```

9. Compare and contrast for, while, and do-while loops.

Loop Type	Condition Checked	Execution Guarantee
for	Before iteration	Zero or more times
while	Before iteration	Zero or more times
do-while	After iteration	At least once

Java Keywords and Operators

6. What is the use of the instanceof keyword in Java?

instanceof checks whether an object is an instance of a specified class/interface or its subclass.

Example:

java

if (obj instanceof String) {

System.out.println("obj is a String");

}

7. Explain the difference between == and .equals() in Java.

- **== compares reference equality (if two references point to the same object).**
- **.equals() compares logical equality (if two objects have the same content/value), but may be overridden for custom classes.**

For example:

java

String s1 = new String("hello");

String s2 = new String("hello");

s1 == s2; ***// false (different objects)***

s1.equals(s2); ***// true (same content)***

9. What is the use of this and super in method overriding?

- **this refers to the current object; used in the subclass to call its own methods or access variables.**
- **super refers to the parent class; used to invoke overridden methods or constructors from the parent class.**

Example in overriding:

java

class Parent {

```
    void display() {
        System.out.println("Parent");
    }
}
class Child extends Parent {
    void display() {
        super.display(); // calls Parent's display()
        System.out.println("Child");
    }
}
```

10. Explain bitwise operators with examples.

Bitwise operators work on individual bits of integer types.

- **& (AND), | (OR), ^ (XOR), ~ (NOT)**
- **<< (left shift), >> (right shift), >>> (unsigned right shift)**

Example:

java

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

System.out.println(a & b); // Output: 1 (binary 0001)
System.out.println(a | b); // Output: 7 (binary 0111)
System.out.println(a ^ b); // Output: 6 (binary 0110)
```

```
System.out.println(~a);    // Output: -6 (bitwise complement)
```