Java Programming Assignment

Section 1: Java Data Types

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

A. There are 8 primitive data types available in Java. They are,

byte -> 8bits

short -> 16bits

int -> 32bits

long -> 64bits

float -> 32bits

double -> 64bits

char -> 16bit s

boolean -> 1bit true(1)/false (0)

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

1. **Definition**:
    - Primitive data types are the basic built-in data types provided by Java.
    - Non-primitive data types are created using classes and store references to objects.
2. **Examples**:
    - Primitive: byte, short, int, long, float, double, char, boolean
    - Non-primitive: String, Array, Class, Object, Interface
3. **Value Stored**:
    - Primitive types store the actual value.
    - Non-primitive types store a reference to the value.
4. **Null Assignment**:
    - Primitive types cannot be assigned null.
    - Non-primitive types can be assigned null.
5. **Memory Usage**:
    - Primitive types use less memory and are more efficient.
    - Non-primitive types use more memory as they involve object references.

6. **Operations**:
    - Primitive types support direct operations like arithmetic and logical operations.
    - Non-primitive types require method calls to perform operations.
7. **Default Values**:
    - Primitive types have default values like 0, 0.0, false, or '\u0000'.
    - Non-primitive types have null as the default value.
8. **Creation**:
    - Primitive types are predefined by Java.
    - Non-primitive types can be user-defined.

3. Write a Java program that demonstrates the use of all primitive data types.

A**. package** assignment;

```
public class Primitive_datatypes {
        public static void main(String[] args) {
                byte myByte = 10;
                System.out.println("byte value: " + myByte);
                short myShort = 1000;
                System.out.println("short value: " + myShort);
                int myInt = 50000;
                System.out.println("int value: " + myInt);
                long myLong = 100000L;
                System.out.println("long value: " + myLong);
                float myFloat = 5.75f;
                System.out.println("float value: " + myFloat);
                double myDouble = 19.99;
                System.out.println("double value: " + myDouble);
                char myChar = 'A';
                System.out.println("char value: " + myChar);
                boolean myBoolean = true;
                System.out.println("boolean value: " + myBoolean);
               }
    }
```

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

A. Type casting is the process of converting a variable from one data type to another. In

Java, this is useful when you want to assign a value of one type to a variable of another type.

Implicit Casting (Widening): Automatic conversion from a smaller data type to a larger data type without data loss.

Example: int num = 10;

```
System.out.println("Value of num: " + num);

float d = num; // int is automatically converted to float

System.out.println("Value of d: " + d);
```

Explicit Casting (Widening): Manual conversion from a larger data type to a smaller data type using a cast operator, which may cause data loss.

Example: float d = 10.5f;

```
System.out.println("Value of d: " + d);

int a = (int) d;  // float is explicitly cast to int

System.out.println("Value of a: " + a);
```

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

A. 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

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

A. Control statements in Java are used to control the flow of execution of the program. They decide which code block gets executed based on certain conditions or repetitions. Control statements help make decisions, loop through code, or jump to different parts of a program.

1.Condition statements:

1.If

Example:

```
if (age >= 18) {
   System.out.println("Eligible to vote");
```

2.If-else

Example:

```
if (num % 2 == 0)
   Even
else
   Odd
```

3.If-else-if Ladder

Example:

```
if (marks >= 90) A
else if (>= 75) B
else C
```

4.Nested if

Example:

```
if (age >= 18) {
   if (hasID) {
      Entry allowed
   } else {
      ID required
   }
} else {
   Not allowed
}
```

5.Switch case

Example:

```
switch(day) {
 case 1: Monday; break;
}
```

2.Looping statements

1.While loop

Example:

```
int i = 0;
while (i < 5) {
   System.out.println(i);
    i++;
}
```

2.Do-while loop

Example:

```
int i = 0;
do {
   System.out.println(i);
    i++;
} while (i < 5);
```

3.For loop

Example:

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

3.Jumping statements

1.Break

Example:

```
for (int i = 0; i < 10; i++) {
if (i == 5) break;
   System.out.println(i);
}
```

2.continue

Example:

```
for (int i = 0; i < 5; i++) {
```

```
  if (i == 2) continue;
      System.out.println(i);
  }
```

3.return

Example:

```
   int add(int a, int b) {
  return a + b;
}
```

2. Write a Java program to demonstrate the use of if-else and switch-case statements.

A.

```
package controlstatement;
   public class IfElseSwitchDemo {
   public static void main(String[] args) {
      int age = 20;
     int wt = 20;
     if (age >= 18 && wt >= 30) {
        System.out.println("Eligible for Blood Donation");
     } else {
        System.out.println("Not Eligible for Blood Donation");
     }
     int bloodGroup = 2;
     switch (bloodGroup) {
        case 1:
           System.out.println("Blood Group: A+");
           break;
        case 2:
           System.out.println("Blood Group: B+");
           break;
        case 3:
           System.out.println("Blood Group: O+");
           break;
```

```
        case 4:
            System.out.println("Blood Group: AB+");
            break;
        default:
            System.out.println("Unknown Blood Group");
    }
    System.out.println("End of program");
  }
}
```

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

A. **break Statement:**

- Purpose: Immediately the loop or switch block where it is used.
- Use Case: When you want to stop the loop entirely.

continue statement:

- Purpose: Skips the current iteration and moves to the next one.
- Use Case: When you want to skip specific cases but keep looping.

4. Write a Java program to print even numbers between 1 to 50 using a for loop.

A.
```
public class EvenNumbers1To50 {
  public static void main(String[] args) {
    for (int i = 1; i <= 50; i++) {
      if (i % 2 == 0) {
        System.out.println(i);
      }
    }
  }
}
```

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

A. **while Loop:**

- Checks the condition first before running the loop body.
- If the condition is false at the start, the loop body may never execute.
- Used when you want to run the loop only if the condition is true initially.

Example:

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

do-while Loop:

- Executes the loop body first, then checks the condition.
- The loop body always executes at least once, even if the condition is false initially.
- Used when you want to run the loop body at least once regardless of the condition.

Example:

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

Section 3: Java Keywords and Operators

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

A. Java keywords are reserved words that have predefined meanings in Java.

- They cannot be used as identifiers (e.g., variable or method names).
- From the basic syntax and structure of Java programs.
- There are around 50+ keywords in Java.

List of 10 commonly used keywords:

- static
- final
- return
- void
- import
- package

- this
- super
- return
- new

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

A. **static:**

- Purpose: Used to define class-level members - variables or methods that belong to the class rather than any object.
- The main method is always static because it runs without creating an object.

Final:

- Purpose: Used to make constants, prevent method overriding, or stop inheritance.
- Once a final variable is assigned, it cannot be changed.

This:

- Purpose: Refers to the current object inside a class.
- Useful when method parameters or variables have the same name as instance variables.

Super:

- Purpose: Refers to the parent class (superclass).
- Used to call the parent constructor, Access parent class methods or variables

3. What are the types of operators in Java?

A. **1. Arithmetic Operators:**

- + : Addition (a + b)
- - : Subtraction (a - b)
- * : Multiplication (a * b)
- / : Division (a / b)
- % : Modulus (a % b)

2. Relational (Comparison) Operators:

- == : Equal to (a == b)
- != : Not equal to (a != b)
- > : Greater than (a > b)
- < : Less than (a < b)

- >= : Greater than or equal (a >= b)
- <= : Less than or equal (a <= b)

3. Logical Operators:

- && : Logical AND (a > 0 && b > 0)
- || : Logical OR (a > 0 || b > 0)
- ! : Logical NOT (!(a > b))

4. Assignment Operators:

- = : Assign (a = 5)
- += : Add and assign (a += 3)
- -= : Subtract and assign (a -= 2)
- *= : Multiply and assign (a *= 4)
- /= : Divide and assign (a /= 2)
- %= : Modulus and assign (a %= 3)

5. Unary Operators

- + : Unary plus (+a)
- - : Unary minus (-a)
- ++ : Increment (a++, ++a)
- -- : Decrement (a--, --a)
- ! : Logical complement (!true)

6. Bitwise Operators

- & : Bitwise AND (a & b)
- | : Bitwise OR (a | b)
- ^ : Bitwise XOR (a ^ b)
- ~ : Bitwise Complement (~a)

7. Shift Operators

- << : Left Shift (a << n)
- >> : Signed Right Shift (a >> n
- >>> : Unsigned Right Shift (a >>> n)

8. instanceof Operator

- Used to test whether an object is an instance of a class or subclass.

Example:

```
if (obj instanceof String)
```

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

```
package day1_and_day2;
public class Arithmetic_relational_and_logical_operators {
	public static void main(String[] args) {
		int a = 10;
		int b = 5;
	// Arithmetic Operators
	System.out.println("Arithmetic Operators:");
	System.out.println("a + b = " + (a + b));
	System.out.println("a - b = " + (a - b));
	System.out.println("a * b = " + (a * b));
	System.out.println("a / b = " + (a / b));
	System.out.println("a % b = " + (a % b));
	// Relational Operators
	System.out.println("Relational Operators:");
	System.out.println("a == b: " + (a == b));
	System.out.println("a != b: " + (a != b));
	System.out.println("a > b : " + (a > b));
	System.out.println("a < b : " + (a < b));
	System.out.println("a >= b: " + (a >= b));
	System.out.println("a <= b: " + (a <= b));
	// Logical Operators
	boolean x = true;
	boolean y = false;
	System.out.println("Logical Operators:");
	System.out.println("x && y: " + (x && y));
	System.out.println("x || y: " + (x || y));
	System.out.println("!x: " + (!x));
	}
}
```

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

A. Operator precedence in Java defines the order in which operators are evaluated in an expression when multiple operators are used together. Each operator in Java has a specific precedence level, and operators with higher precedence are evaluated before those with lower precedence.

Effect on Outcome:

Operator precedence directly affects the result of expressions. If operators are not evaluated in the correct order, it can lead to unexpected results.To ensure correct evaluation, parentheses can be used to override the default precedence and control the order of execution.

Additional Questions

Java Data Types

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

A. **Primitive Data Types:** There are 8 basic types of primitive data types.

Integer Types:

Type	Size	Range
byte	1 byte	-128 to 127
short	2 bytes	-32,768 to 32,767
int	4 bytes	-2^{31} to $2^{31} - 1$
long	8 bytes	-2^{63} to $2^{63} - 1$

Floating-Point Types:

Type	Size	Range
float	4 bytes	3.4e−038 to 3.4e+038 (approx.)
double	8 bytes	3.4e−038 to 3.4e+038 (approx.)

Character Type:

Type	Size	Range
char	2 bytes	'\u0000' to '\uffff' (0 to 65,535)

Boolean Type:

Type	Size	Range
boolean	1 bit	true or false (JVM dependent)

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

A. In Java, overflow and underflow occur when the result of a calculation exceeds the range of the data type being used.

- For integer types (byte, short, int, long), Java handles overflow and underflow using wrap-around behavior. This means if a value exceeds the maximum limit, it wraps

around to the minimum value, and vice versa. Java does not throw an error or exception in such cases.

- For floating-point types (float, double), overflow results in positive or negative infinity, and underflow results in zero or a very small denormalized number. Java follows the IEEE 754 standard for floating-point arithmetic.

8. Write a program to convert a double value to an int without data loss.

```
package day1_and_day2;

public class ConvertDoubleInoInt {

	public static void main(String[] args) {

		double number = 100.0; // double value

		int result = (int) number; // convert to int

		System.out.println("Converted value: " + result);

	}

}
```

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

A. char is a primitive data type in Java.

It is used to store a single character like 'A', '9', or '@'.

String is a class in Java (not a primitive type).
It is used to store a sequence of characters like "Hello" or "Java123".

10. Explain wrapper classes and their use in Java.

A. Wrapper classes convert primitive types into objects. They are part of the `java. Lang` package. Useful when objects are required instead of primitives (e.g., in collections).

Uses:

- To use primitives in Java Collections (which require objects).
- To use utility methods for conversions and parsing.
- For null values (which primitives cannot hold).

Java Control Statements

6. Write a Java program using nested if statements.

```
package day1_and_day2;

public class Nested_if_statements {

	public static void main(String[] args) {

		int number = 25;
```

```
    if (number > 0) {
      System.out.println("The number is positive.");
      if (number % 2 == 0) {
        System.out.println("It is an even number.");
      } else {
        System.out.println("It is an odd number.");
      }
    } else {
      System.out.println("The number is not positive.");
    }
      }
}
```

7. Write a Java program to display the multiplication table of a number using a loop.

```
package day1_and_day2;
public class MultiplicationTable {
        public static void main(String[] args) {
                int number = 5;
                System.out.println("Multiplication table of " + number + ":");
                for (int i = 1; i <= 10; i++) {
                        int result = number * i;
                        System.out.println(number + " x " + i + " = " + result);
    }
  }
}
```

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

A. In Java, we can exit from nested loops using a labeled break statement. A label is placed before the outer loop, and the break statement refers to that label to exit all the nested loops at once.

```
outer:
for (int i = 0; i < 5; i++) {
  for (int j = 0; j < 5; j++) {
    if (j == 2) break outer;
  }
}
```

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

A. **Compare:**

- **For loop:** Used to repeat code a specific number of times. It includes initialization, condition, and update in one line. The condition is checked before each iteration.
- **While loop:** Repeats code as long as a condition is true. The condition is checked before each iteration. Initialization and update are done separately.
- **Do-while loop:** Executes the code first, then checks the condition. It repeats the loop while the condition is true.

Contrast:

- **For loop:** Best when you know how many times you want to repeat the code. The loop may not run if the condition is false initially.
- **While loop:** Used when the number of iterations is not fixed and depends on the condition. Like the for loop, it may not run if the condition is false at the start.
- **Do-while loop:** The loop always executes at least once because the condition is checked after running the code inside the loop.

10. Write a program that uses a switch-case to simulate a basic calculator.

A.
```
package day1_and_day2;
public class SimpleCalculator {
        public static void main(String[] args) {
                char op = '+';
    int a = 10;
    int b = 5;
    switch (op) {
      case '+':
        System.out.println("Addition: " + (a + b));
        break;
      case '-':
        System.out.println("Subtraction: " + (a - b));
        break;
      case '*':
        System.out.println("Multiplication: " + (a * b));
        break;
      case '/':
        System.out.println("Division: " + (a / b));
```

```
            break;
        default:
            System.out.println("Invalid Operator");
    }
        }
}
```

Java Keywords and Operators

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

A. The instanceof keyword in Java is used to test whether an object is an instance of a specific class or implements a particular interface. It helps you check the type of an object at runtime before performing type-specific operations, which can prevent errors like ClassCastException.

- It returns true if the object is an instance of the given class or interface (including subclasses or implementing classes).
- It returns false if it is not.

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

A. == operator checks whether two reference variables point to the same object in memory. It compares the memory addresses of the objects.

.equals() method checks whether two objects are logically equal based on their content or state. It compares the values inside the objects.

8. Write a program using the ternary operator.

A.

```
package day1_and_day2;
public class MaxNumber {
        public static void main(String[] args) {
                int a = 10;
                int b = 20;
                int max = (a > b) ? a : b;
                System.out.println("Maximum number is " + max);
        }
}
```

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

A. this: refers to current class members, helps resolve ambiguity.

super: refers to parent class members, helps call superclass methods/constructors.

10. Explain bitwise operators with examples.

A. **Bitwise Operators:**

- & : Bitwise AND (a & b)
- | : Bitwise OR (a | b)
- ^ : Bitwise XOR (a ^ b)
- ~ : Bitwise Complement (~a)

Example:

```
int a = 5;

int b = 3;

System.out.println(a & b);

System.out.println(a | b);

System.out.println(a ^ b);

System.out.println(~a);

System.out.println(a << 1);

System.out.println(a >> 1);
```