Java Programming Assignment

Section 1: Java Data Types

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

Primitive Data Type are the basic building blocks that store simple values directly in memory. Examples of primitive data types are

byte: It is typically used to save memory in large arrays, primarily when storing data that doesn't require a large range, such as image processing or file I/O operations.

Size: 1 byte(8 bits)

Range: -128 to 127

Example: byte age = 25;

short: The short data type is used when memory savings are critical, and the range of values doesn't need to be large.

Size: 2bytes

Range: -32,768 to 32,767.
Example: short salary = 15000;

int: This data type is the most commonly used for integer values. It's used in counting, indexing, and arithmetic operations where large ranges of values are needed but not excessively large.

Size: 4 bytes (32 bits)

Range: -2,147,483,648 to 2,147,483,647

Example: int population = 100000;

float: Is used when dealing with floating-point numbers that require less precision but need to save memory. It's commonly used in scientific calculations or when memory efficiency is a priority.

Size: 4 bytes (32 bits)

Precision: Up to 7 decimal digits

long: It is used when integer values exceed the range of int. It's ideal for tracking large numbers, such as file sizes, timestamps, or large-scale financial calculations.

Size: 8 bytes

Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Example: long distance = 9876543210L;

double: It is the default data type for decimal values. It's used when higher precision is required, or when handling large or very small numbers.

Size: 8 bytes

Precision: Up to 15 decimal digits.

Example: double d=3.1412547891;

Char: char is used to store single characters or symbols. It can store letters, digits, or any Unicode character.

Size: 2 bytes

Example: char c='a';

Boolean: It is primarily used for logical operations such as conditional statements (if, while), to store truth values in expressions and flags. It has only two values either true or false.

Size: 1 byte.

boolean isJavaFun = true;

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

Primitive Data Types	Non-Primitive (Reference) Data Types
These are built-in data types provided by the Java language.	User-Defined or Built-in Objects. They are either created by the programmer
Primitive variables directly store the actual data value in memory.	Non-primitive variables store a reference (memory address) to the actual object data, which is typically stored in the heap memory.
Each primitive type has a predefined, fixed size in memory (e.g., int is 4 bytes).	The size of non-primitive types can vary depending on the content of the object they reference.
Primitive types do not have associated methods or behaviors.	Non-primitive types can have methods and fields, allowing for complex operations and data encapsulation.
Examples: byte, short, int, long, float, double, boolean, char.	Examples:String, Array, Class, Interface.

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

```
package day1_day2_Assessment;


public class PrimitiveDataTypes {
```

```
        public static void main(String[] args) {
            byte b = 100;
            short s = 20000;
            int i = 100000;
            long l = 15000000000L;
            float f = 5.75f;
            double d = 19.99;
            char c = 'A';
            boolean isJavaFun = 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: " + isJavaFun);
        }
    }
```

oputput:

```
========
byte: 100
short: 20000
int: 100000
long: 15000000000
float: 5.75
double: 19.99
char: A
```

boolean: true

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

Type casting in Java is the process of converting a value from one data type to another. This conversion can be either implicit (automatic) or explicit (manual).

Implicit casting occurs automatically when a smaller data type is converted to a larger data type. This is considered a widening conversion because the target data type can accommodate all possible values of the source data type, preventing data loss.

Example:

```
public class ImplicitCastingExample {
    public static void main(String[] args) {
        int myInt = 10;
        long myLong = myInt;
        double myDouble = myLong;
        System.out.println("int value: " + myInt);
        System.out.println("long value: " + myLong);
        System.out.println("double value: " + myDouble);
    }
}
```

Explicit casting, also known as narrowing conversion, requires manual intervention by the programmer. This type of casting is necessary when converting a larger data type to a smaller data type, as it may lead to data loss if the value of the larger type exceeds the range of the smaller type. The target type is specified in parentheses before the value being cast.

Example:

```
public class ExplicitCastingExample {
    public static void main(String[] args) {
        double myDouble = 9.78;
        int myInt = (int) myDouble;
        long myLong = 10000000000L;
         int anotherInt = (int) myLong;
```

```
        System.out.println("double value: " + myDouble);
        System.out.println("int value after explicit cast: " + myInt);
    }
}
```

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

The default values for each primitive data type in Java are as follows:

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

Section 2: Java Control Statements

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

Control statements in Java control the flow of execution of a program based on conditions or loops. These statements allow you to make decisions, repeat code, or jump to different parts of the program.

Types of Control Statements in Java

1. Decision-Making Statements (Conditional Statements)

These are used to make decisions based on a condition.

a) if statement

```
int age = 20;
if (age >= 18) {
    System.out.println("You are eligible to vote.");
}
```

b) if-else statement

```
int age = 16;
if (age >= 18) {
    System.out.println("Eligible");
} else {
    System.out.println("Not eligible");
}
```

c) if-else-if ladder

```
int marks = 85;
if (marks >= 90) {
    System.out.println("Grade A");
} else if (marks >= 75) {
    System.out.println("Grade B");
} else {
    System.out.println("Grade C");
}
```

d) switch statement

```
int day = 3;
switch (day) {
    case 1: System.out.println("Monday"); break;
    case 2: System.out.println("Tuesday"); break;
    case 3: System.out.println("Wednesday"); break;
    default: System.out.println("Invalid day");
}
```

---

2. Looping Statements (Iteration Statements)

These are used to execute a block of code multiple times.

a) for loop

```
for (int i = 1; i <= 5; i++) {
```

```
    System.out.println("Count: " + i);
}
```

b) while loop

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

c) do-while loop

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

---

3. Jumping Statements

These are used to jump from one part of the code to another.

a) break statement

Used to exit a loop or switch.

java

Copy code

```
for (int i = 1; i <= 10; i++) {
    if (i == 5) break;
    System.out.println(i);
}
```

b) continue statement

Skips the current iteration and continues with the next.

java

Copy code

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

c) return statement

Exits from the method.

java

Copy code

```
public static void greet() {
   System.out.println("Hello");
   return;
}
```

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

```
package day1_day2_Assessment;

public class IfElseStatement {

	public static void main(String[] args) {
		int age = 16;
		if (age >= 18) {
		    System.out.println("Eligible for voting");
		} else {
		    System.out.println("Not eligible for voting");
		}

	}

}
output: Not eligible for voting
```

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

In Java, both break and continue are control flow statements used within loops and break in switch statements to alter the normal execution flow.

Break statement:- This statement is used to terminate the entire loop immediately. When a break statement is encountered, the loop is exited, and program control resumes at the statement immediately following the loop. It effectively "breaks out" of the loop.

Example :

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

Output will be: 0,1,2,3,4

Continue statement:- This statement is used to skip the current iteration of the loop and proceed to the next iteration. When a continue statement is encountered, the remaining code within the current iteration of the loop is bypassed, and the loop control jumps to the beginning of the next iteration.

Example:

```
  for (int i = 0; i < 10; i++) {
    if (i == 5) {
       continue;
    }
    System.out.println(i); 0, 1, 2, 3, 4, 6, 7, 8, 9
  }
```

Output: 0, 1, 2, 3, 4, 6, 7, 8, 9

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

```
package day1_day2_Assessment;
import java.util.Scanner;
public class Even_Numbers_in_Range {

	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter starting value: ");
		int starting_value=sc.nextInt();
		System.out.println("enter maximum limit: ");
		int last_value=sc.nextInt();
```

```
            System.out.println("even numbers are: ");
            for(int i=0;i<last_value;i++)
            {
                  if(i%2==0)
                  {
                        System.out.print(+i +" ");
                  }
            }
            sc.close();

      }

}
output:
 Enter starting value:
2
enter maximum limit:
50
even numbers are:
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48
```

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

While Loop (Entry-Controlled Loop):

A while loop checks its condition before executing the loop's body. If the condition is initially false, the loop's body will never execute.

Example:

```
int count = 0;

while (count < 5) {

   System.out.println("While loop iteration: " + count);

   count++;

}
```

Output:

While loop iteration: 0

While loop iteration: 1

While loop iteration: 2

While loop iteration: 3

While loop iteration: 4

Do-While Loop (Exit-Controlled Loop):

- The loop body executes at least once, and then the condition is checked after the first iteration.
- This guarantees that the code inside the do block will run irrespective of the initial state of the condition.

Example:

```
int count = 0;

do {

   System.out.println("Do-while loop iteration: " + count);

   count++;

} while (count < 5);
```

Output:

```
Do-while loop iteration: 0

Do-while loop iteration: 1

Do-while loop iteration: 2

Do-while loop iteration: 3

Do-while loop iteration: 4
```

Section 3: Java Keywords and Operators

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

   Keywords are also called as reserved words that have a special meaning in the language. These words cannot be used as variable names, class names, or method names, because they are part of Java syntax.

   They tell the compiler to perform specific operations and help define the structure and rules of the program.

   Commonly used Keywords are:

   Class, public, static, void, main, int string, if, if else, switch, for, while, return, new, short, float, long, double, char, etc.

2. **Explain the purpose of the following keywords: static, final, this, super.**
   Static:
   The static keyword is used to define class-level variables or methods. This means:
   It belongs to the class, not to any specific object.
   You can access it without creating an object.
   Final:

The final keyword is used to make something constant or unchangeable. It can be used with:
Variables – value cannot be changed
Methods – method cannot be overridden
Classes – class cannot be extended

This:

this keyword refers to the current object. It is used when:
Local variable name and instance variable name are the same.
You want to call another constructor of the same class.

Super:

The super keyword refers to the parent (superclass) of the current object. It is used to:
Call superclass constructor
Access superclass methods or variables

3. **What are the types of operators in Java?**

   Operators in Java are special symbols or keywords used to perform operations on variables and values. Operators work on operands.

   Types of operators in java:

   i) Arithmetic operators
   ii) Relational operators
   iii) Logical operators
   iv) Assignment operators
   v) Unary operators
   vi) Bitwise operators
   vii) Ternary operatorsoprators

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

```
package day1_day2_Assessment;

public class OperatorsDemo {
          public static void main(String[] args) {
               int a = 10, b = 5;
               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));
               System.out.println();
               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));
               System.out.println();
               boolean x = true, 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));
        }
    }
```

```
output
Arithmetic Operators:
a + b = 15
a - b = 5
a * b = 50
a / b = 2
a % b = 0

Relational Operators:
a == b: false
a != b: true
a > b: true
a < b: false
a >= b: true
a <= b: false

Logical Operators:
x && y: false
x || y: true
!x: false
```

5. **What is operator precedence? How does it affect the outcome of expressions?**
   Operator precedence determines the order in which operators are evaluated in a Java expression when there are multiple operators involved. Java follows a set of rules to decide which operation to perform first.
   Operator precedence affects the result of expressions. If you don't use it correctly or use parentheses to control it, your code may give unexpected results.
   For Example :
   If there is a statement like
   int result= 10+5*2;
   first 5*2=10 is executed and then 10+10=20
   because the * operator for has more precedence than +.

   Additional Questions
   Java Data Types
   7. How does Java handle overflow and underflow with numeric types?
   Java uses fixed-size data types like int, byte, short, long, which means each has a limited range of values it can store.
   When a value exceeds this range, Java does not throw an error — instead, it performs wrap-around behavior.
   Overflow:
   Overflow happens when a number is greater than the maximum value a data type can store.
   Example:

int max = Integer.MAX_VALUE;
System.out.println(max + 1);
Output: -2,147,483,648 (wraps around)
UnderFlow:
Underflow happens when a number is less than the minimum value a data type can store.
Example:
int min = Integer.MIN_VALUE;
System.out.println(min - 1);
Output: 2,147,483,647 (wraps around)

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

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

```
package day1_day2_Assessment;

public class DoubleToInt {
	public static void main(String args[])
	{
		double num=109.235;
		System.out.println("value of double="+num);
		int n=(int) (num);
		System.out.println("value of integer="+n);
	}
}
output
value of double=109.235
value of integer=109
```

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

Char:

It is a primitive data type holds single character.
It in enclosed with sing quotes(‘’).
It has fixed size i.e. 2 bytes.
Example: char letter=’a’;
String:

String is a non-primitive data type, it holds sequence of characters.
It is enclosed in double quotes(“ ”).
It’s size varies.
Example: String company = “wipro”;

10. Explain wrapper classes and their use in Java.

In Java, wrapper classes are used to convert primitive data types like int, char, Boolean into objects.

Below are the uses of the Wrapper class in Java:

- They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
- The classes in java.util package handle only objects and hence wrapper classes help in this case.
- Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
- An object is needed to support synchronization in multithreading.

Java Control Statements

6. Write a Java program using nested if statements.

```
package day1_day2_Assessment;

public class NestedIf {
      public static void main(String[] args) {
        int number = 8;
        if (number > 0) {
            System.out.println("The number is positive.");
            if (number % 2 == 0)
            {
                System.out.println("It is also even.");
            } else
            {
                System.out.println("It is also odd.");
            }
        }
        else
        {
            System.out.println("The number is not positive.");
        }
    }
}
output
The number is positive.
It is also even.
```

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

```
package day1_day2_Assessment;

public class Table {
       public static void main(String[] args) {
              int number = 17;
              for (int i = 1; i <= 10; i++) {
                  System.out.println(number + " x " + i + " = " + (number *
i));
              }
          }
```

```
}
output:
17 x 1 = 17
17 x 2 = 34
17 x 3 = 51
17 x 4 = 68
17 x 5 = 85
17 x 6 = 102
17 x 7 = 119
17 x 8 = 136
17 x 9 = 153
17 x 10 = 170
```

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

There are three way to exit from nested lops:

1. **Using a Labeled break (Most Direct Way:**

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

2. Using return (if inside a method)

If your loops are inside a method, return will exit from the method entirely, stopping all loops.

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        if (i == 2 && j == 2) {
            return; // exits method
        }
        System.out.println(i + " " + j);
    }
}
```

3. Using a Flag Variable:

```
boolean exit = false;
for (int i = 1; i <= 3 && !exit; i++) {
    for (int j = 1; j <= 3; j++) {
        if (i == 2 && j == 2) {
            exit = true;
            break;
        }
        System.out.println(i + " " + j);
    }
```

```
}
```

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

For Loop	While Loop	Do While Loop
Declared within the loop structure and executed once at the beginning.	Declared outside the loop; should be done explicitly before the loop.	Declared outside the loop structure
Condition is checked before each iteration.	Condition is checked before each iteration.	Condition is checked after each iteration.
For loop is suitable for a known number of iterations or when looping over ranges.	While loop is useful when the number of iterations is not known in advance or based on a condition.	Do while loop useful when the loop block must be executed at least once, regardless of the initial condition.
Scope is limited to the loop body.	Scope extends beyond the loop; needs to be handled explicitly.	Scope extends beyond the loop; needs to be handled explicitly.

Syntax:
for (initialization; condition; increment/decrement)
{
//statements
}
	Syntax:
	while (condition)
	{
	//statements
	}
		Syntax:
		do
		{
		Statements
		} while (condition);

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

```
package day1_day2_Assessment;
import java.util.*;
public class BasicCalculator {
          public static void main(String[] args) {
               Scanner scanner = new Scanner(System.in);
               System.out.print("Enter first number: ");
               double num1 = scanner.nextDouble();
               System.out.print("Enter second number: ");
               double num2 = scanner.nextDouble();
               System.out.println("Choose an operation: +, -, *, /");
               char operator = scanner.next().charAt(0);
               double result;
               switch (operator) {
                    case '+':
                         result = num1 + num2;
                         System.out.println("Result of Addition is: " +
result);
                         break;
                    case '-':
                         result = num1 - num2;
                         System.out.println("Result of Subtraction is: " +
result);
                         break;
                    case '*':
                         result = num1 * num2;
```

```
                        System.out.println("Result of Multiplication is: " +
result);
                        break;
                    case '/':
                        if (num2 != 0) {
                            result = num1 / num2;
                            System.out.println("Result of Division is: " +
result);
                        } else {
                            System.out.println("Error: Cannot divide by
zero.");
                        }
                        break;
                    default:
                        System.out.println("Invalid operator!");
                }
                scanner.close();
            }
        }
output
for addition
Enter first number: 20
Enter second number: 40
Choose an operation: +, -, *, /
+
Result of Addition is: 60.0

for subtraction
Enter first number: 40
Enter second number: 15
Choose an operation: +, -, *, /
-
Result of Subtraction is: 25.0
for multiplication
Enter first number: 20
Enter second number: 40
Choose an operation: +, -, *, /
*
Result of Multiplication is: 800.0

for division
Enter first number: 80
Enter second number: 12
Choose an operation: +, -, *, /
/
Result of Division is: 6.666666666666667
```

Java Keywords and Operators

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

In Java, instanceof is a keyword used for checking if a reference variable contains a given type of object reference or not.

It helps to avoid ClassCastException by ensuring that an object is of a certain type before performing a type cast or calling methods specific to that class.

Syntax:

Object instacnceOf ClassName

- It returns true if the object is an instance of the class or subclass or interface
- Otherwise it returns false

Example program:

```
Import java.util.*

class Animal {}

class Dog extends Animal {}

public class InstanceofExample {
   public static void main(String[] args) {
      Animal a = new Dog();
      if (a instanceof Dog) {
         System.out.println("a is an instance of Dog");
      }
      if (a instanceof Animal) {
         System.out.println("a is also an instance of Animal");
      }
     }
   }
}
```

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

Equality (==) Operator	**.equals() Method**
Compares if two references point to the same memory location.	Compares the content of objects.

It works with Primitives and object references.	It works with only objects
Operator cannot be overridden.	Can be cannot be overridden in custom classes.
It compares memory addresses.	It compares references unless overridden.

8. Write a program using the ternary operator.

```
package day1_day2_Assessment;

public class TernaryOperator {
          public static void main(String[] args)
          {
              int n1 = 5, n2 = 10, max;
              System.out.println("First num: " + n1);
              System.out.println("Second num: " + n2);
              max = (n1 > n2) ? n1 : n2;
              System.out.println("Maximum is = " + max);
          }
      }
```

```
Output:
First num: 5
Second num: 10
Maximum is = 10
```

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

super in Method Overriding:

*. Super is Used to call parent class methods (including the overridden method) or access parent class variables.

*. Sometimes you override a method but still want to use the parent's version inside the child's method.

this in Method Overriding

*. This refers to the current object of the class where it's used.

*. We can use it to call another method from the same class (including the overridden method in the same class if not shadowed by a parameter).

```
package day1_day2_Assessment;
class Animal {
    String name = "Animal";
    void speak() {
        System.out.println("Animal speaks...");
    }
    void displayName() {
        System.out.println("Name from Animal: " + name);
    }
}
class Dog extends Animal {
    String name = "Dog";
    void speak() {
        System.out.println("Dog barks...");
    }
    void show() {
```

```
        System.out.println("Calling this.speak():");
        this.speak();
        System.out.println("Calling super.speak():");
        super.speak();
        System.out.println("this.name: " + this.name);
        System.out.println("super.name: " + super.name);
    }
}
public class ThisAndSuper {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.show();
    }
}
```

```
output
Calling this.speak():
Dog barks...
Calling super.speak():
Animal speaks...
this.name: Dog
super.name: Animal
```

10. Explain bitwise operators with examples.

```
package day1_day2_Assessment;

public class BitWiseOperator {
          public static void main(String[] args) {
              int a = 5;
              int b = 3;
              int andResult = a & b;
              System.out.println("a & b = " + andResult);
              int orResult = a | b;
              System.out.println("a | b = " + orResult);
              int xorResult = a ^ b;
              System.out.println("a ^ b = " + xorResult);
              int notA = ~a;
              System.out.println("~a = " + notA);
              int leftShift = a << 1;
              System.out.println("a << 1 = " + leftShift);
              int rightShift = a >> 1;
              System.out.println("a >> 1 = " + rightShift);
          }
      }
```

```
output
a & b = 1
a | b = 7
a ^ b = 6
~a = -6
a << 1 = 10
a >> 1 = 2
```