**Operators And Loops**

**Assignment#2**

**COURSE- JAVA WITH DSA SYSTEM DESIGN**

****

**SUBMITTED BY:**

**SAMIM MONDAL**

**SUBMITTED TO:**

**HYDER ABBAS**

**1.What are the conditional operators in java?**

**Ans:** In Java, conditional operators, also known as ternary operators, are used to perform a conditional (if-else) operation in a concise form. The most common conditional operator in Java is the ternary operator **?**

variable = (condition) ? value\_if\_true : value\_if\_false;

Here's how it works:

* The **condition** is evaluated first.
* If the **condition** is **true**, the value of **value\_if\_true** is assigned to the variable If the **condition** is **false**, the value of **value\_if\_false** is assigned to the variable.Here's an example of using the ternary operator in Java:

int number = 10;

String result = (number > 5) ? "Greater than 5" : "Less than or equal to 5";

System.out.println(result);

in this example, if the **number** is greater than 5, the **result** will be set to **"Greater than 5"**, otherwise, it will be set to **"Less than or equal to 5"**.

Another important note is that the ternary operator is an expression, which means it can be used within other expressions or assignments, allowing for compact conditional logic.

Apart from the ternary operator, Java also has other conditional constructs such as **if**, **else if**, and **else** statements that allow you to write more complex conditional logic.

Example of if-else statement:

int number = 10;

if (number > 5) {

System.out.println("Greater than 5");

} else {

System.out.println("Less than or equal to 5");

}

**2.** **What are the types of operators based on the number of operands?**

**Ans:** Operators in programming languages can be categorized based on the number of operands they work with. The most common categories are:

1. **Unary Operators**: These operators work with a single operand.

Examples:

* + Unary minus/negation (-x)
  + Unary plus (typically just x, doesn't change the value)
  + Increment (x++)
  + Decrement (x--)
  + Logical NOT (!x)
  + Bitwise NOT (~x)

1. **Binary Operators**: These operators work with two operands.

Examples:

* + Addition (+)
  + Subtraction (-)
  + Multiplication (\*)
  + Division (/)
  + Modulus (%)
  + Assignment (=)
  + Equal to (==)
  + Not equal to (!=)
  + Greater than (>)
  + Less than (<)
  + Greater than or equal to (>=)
  + Less than or equal to (<=)
  + Logical AND (&&)
  + Logical OR (||)
  + Bitwise AND (&)
  + Bitwise OR (|)
  + Bitwise XOR (^)
  + Bitwise left shift (<<)
  + Bitwise right shift (>>)

1. **Ternary Operator**: This operator works with three operands and is often used for conditional expressions.

Example:

* + Conditional/ternary operator (condition ? value\_if\_true : value\_if\_false)

These are the primary categories of operators based on the number of operands they work with. Keep in mind that different programming languages might have additional operators or variations, so it's important to consult the documentation of the specific language you're working with for a comprehensive list.

1. **What is the use of switch case in java programming?**

**Ans:** In Java programming, the **switch** statement is a control flow statement that allows you to execute different code blocks based on the value of a specific expression. It's used to simplify the process of evaluating a single value against multiple possible cases and executing corresponding actions.

The basic syntax of the **switch** statement in Java is as follows

switch (expression) {

case value1:

// Code to be executed if expression matches value1

break;

case value2:

// Code to be executed if expression matches value2

break;

// More cases...

default:

// Code to be executed if expression doesn't match any case

}

Here's a breakdown of how the **switch** statement works:

1. An expression is evaluated, usually a variable or a value that you want to compare against different cases.
2. The **case** labels represent the possible values that the expression can take.
3. When the **switch** statement is executed, it evaluates the expression and then compares it with the values listed in the **case** labels.
4. If a match is found, the code block following that **case** label is executed. If there's no match, the code block under the **default** label (optional) is executed. The **break** statement is used to exit the **switch** block once a case is matched and executed.
5. If no **break** statement is encountered, the program will continue to execute the code for all subsequent cases until a **break** statement is encountered or the **switch** block ends.

Here's a simple example of how you might use the **switch** statement in Java:

int day = 3;

String dayName;

switch (day) {

case 1:

dayName = "Monday";

break;

case 2:

dayName = "Tuesday";

break;

case 3:

dayName = "Wednesday";

break;

// ... other cases ...

default:

dayName = "Invalid day";

break;

}

System.out.println("Today is " + dayName);

**5.What are the priority level of arithmetic operations in java?**

**Ans:** In Java, like in many programming languages, arithmetic operations follow a certain order of precedence, which determines the order in which they are evaluated when they appear in an expression. The priority level of arithmetic operations in Java follows the standard mathematical rules:

1. Parentheses: Expressions inside parentheses are evaluated first.
2. Exponentiation: Java doesn't have a built-in exponentiation operator, but you can use the **Math.pow()** function for exponentiation.
3. Multiplication and Division: Multiplication and division have the same priority level, and they are evaluated from left to right.
4. Addition and Subtraction: Addition and subtraction also have the same priority level, and they are evaluated from left to right.

It's important to note that if operations have the same priority level, they are evaluated from left to right. If you want to change the order of evaluation, you can use parentheses to explicitly specify the desired grouping.

Here's an example to illustrate the priority of arithmetic operations in Java:

int result = 10 + 5 \* 2; // Here, multiplication is performed first, then addition.

In this example, the result will be 20, not 30, because the multiplication is given higher priority.

int result = (10 + 5) \* 2; // Here, addition is performed first due to parentheses.

In this example, the result will be 30, as the addition is performed first due to the presence of parentheses.

Remember that understanding operator precedence is essential to write accurate and predictable expressions in Java and other programming languages.

1. **What are the conditional statements and use of conditional statements?**

**Ans:** Conditional statements, often referred to as control structures, are fundamental constructs in programming that allow you to execute different blocks of code based on whether certain conditions are true or false. These statements enable you to make decisions and control the flow of your program. The most common types of conditional statements are:

1. **if Statement**: The simplest form of a conditional statement. It allows you to execute a block of code only if a specific condition is true.

if condition:

# Code to execute if the condition is true

**2.if-else Statement**: This statement lets you execute one block of code if a condition is true, and another block if the condition is false.

if condition:

# Code to execute if the condition is true

else:

# Code to execute if the condition is false

**if-elif-else Statement**: Used when you have multiple conditions to check. The "elif" (short for "else if") allows you to test additional conditions in sequence.

if condition1:

# Code to execute if condition1 is true

elif condition2:

# Code to execute if condition2 is true

else:

# Code to execute if none of the conditions are true

**Nested if Statements**: You can place conditional statements within other conditional statements to create more complex decision structures.

if outer\_condition:

if inner\_condition:

# Code to execute if both outer and inner conditions are true

Here are some common use cases:

1. **Decision Making**: Conditional statements are used to make decisions in programs. For example, you might use an if-else statement to determine whether a user is old enough to access certain content.
2. **Validation**: They are often used to validate user input. For instance, you might check if an email address entered by a user follows a valid format.
3. **Iteration**: Conditional statements are used in loops to control how many times a loop iterates. The loop can continue as long as a certain condition is met.
4. **Menu Selection**: In menu-driven programs, conditional statements help determine which action to take based on user choices.
5. **Error Handling**: They can be used to handle exceptional cases or errors. If a certain condition is met, the program can respond accordingly.
6. **State Management**: In game development or simulations, conditional statements can be used to manage the state of the game or simulation based on certain conditions.
7. **Data Filtering**: Conditional statements can be used to filter and process data. For example, you might filter out specific data points from a dataset based on certain conditions
8. **What is the syntax of if else statement in java?**

**Ans:** The if-else statement in Java is used for conditional execution of code blocks. It allows you to execute different blocks of code based on whether a specified condition evaluates to true or false. The basic syntax of the if-else statement in Java is as follows:

if (condition) {

// Code to be executed if the condition is true

} else {

// Code to be executed if the condition is false

}

Here's a breakdown of each part of the syntax:

1. **if**: This keyword marks the beginning of the if statement.
2. **condition**: This is the expression that is evaluated. If the condition evaluates to true, the code block inside the if statement is executed; otherwise, the code block inside the else statement (if provided) is executed.
3. **{}**: These curly braces define the scope of the code blocks associated with the if and else statements. If the condition is true, the code within the first set of curly braces will be executed; if the condition is false, the code within the second set of curly braces (else block) will be executed.
4. **else**: This keyword indicates the beginning of the else block, which is executed when the condition in the if statement evaluates to false. The else block is optional; you can omit it if you only want to execute code when the condition is true.

Here's an example to illustrate the usage of if-else statement in Java:

public class IfElseExample {

public static void main(String[] args) {

int number = 10;

if (number > 0) {

System.out.println("The number is positive.");

} else {

System.out.println("The number is not positive.");

}

}

}

1. **What are the three types of iterative statement in java?**

**Ans:** **for loop:** The **for** loop is a widely used iterative statement in Java. It allows you to specify an initialization step, a condition for continuation, and an update step. The loop continues to execute as long as the condition is true. Here's the basic structure of a **for** loop:

for (initialization; condition; update) {

// Code to be executed repeatedly

}

**while loop:** The **while** loop repeatedly executes a block of code as long as the specified condition evaluates to **true**. It does not have an explicit initialization or update step; those must be handled manually within the loop. Here's the basic structure of a **while** loop:

while (condition) {

// Code to be executed repeatedly

// Don't forget to update the condition appropriately

}

**do-while loop:** The **do-while** loop is similar to the **while** loop, but the difference is that the block of code is executed at least once before the condition is checked. This ensures that the code within the loop executes at least once, regardless of the initial condition. Here's the basic structure of a **do-while** loop:

do {

// Code to be executed repeatedly

// Don't forget to update the condition appropriately

} while (condition);

1. **Difference between the for loop and do while loop?**

**Ans.** Both the **for** loop and **do-while** loop are control structures in programming that allow you to repeat a set of statements multiple times. However, they have different structures and behaviors:

1. **For Loop**: The **for** loop is used when you know the number of iterations you want to perform beforehand. It has a specific structure that includes three parts:
   * Initialization: Initializing the loop variable (usually a counter) before the loop starts.
   * Condition: Checking a condition before each iteration. If the condition is true, the loop continues; if false, the loop terminates.
   * Iteration: Executing a set of statements and updating the loop variable after each iteration.

Example:

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

// code to be repeated 5 times

}

**Do-While Loop**: The **do-while** loop is used when you want to execute a block of code at least once, regardless of whether the loop condition is true or false. After the first iteration, the loop condition is checked, and if it's true, the loop continues; otherwise, it terminates.

**Example**:

int i = 0;

do {

// code to be repeated at least once

i++;

} while (i < 5);

In summary:

* The **for** loop is suitable when you know the exact number of iterations you need and want to control initialization, condition checking, and iteration in a single place.
* The **do-while** loop is useful when you want to ensure that a certain block of code is executed at least once before the loop condition is checked.

1. **Write a program to print the numbers from 1 to 10 ?**

**Ans:** public class NumberPrinter {

public static void main(String[] args) {

for (int i = 1; i <= 10; i++) {

System.out.println(i);

}

}

}

**THANK YOU!!**