Lecture Notes

Conditional Statements

Welcome to the session on conditional statements.

In the previous module on the basics of Java programming, you learnt about Boolean operators and logical statements.

You would recall that you wrote code such as this one here:

```
public class RelationalOperators {
	public static void main(String[] args) {
		System.out.println(6666 < 224*29);
	}
}
```

This code gave you false as outcome. But it was not followed up by any action. This was the shortcoming in this simple code. Because in a real scenario, each TRUE or FALSE is accompanied by an action. If a condition is met, an action is taken, and if it is not, another action is taken.

You saw the example of a Football World Cup final, where two teams competed for the ultimate reward. The final match is never a draw, someone has to be declared the winner.
So, you wrote a simple pseudocode to show this result:

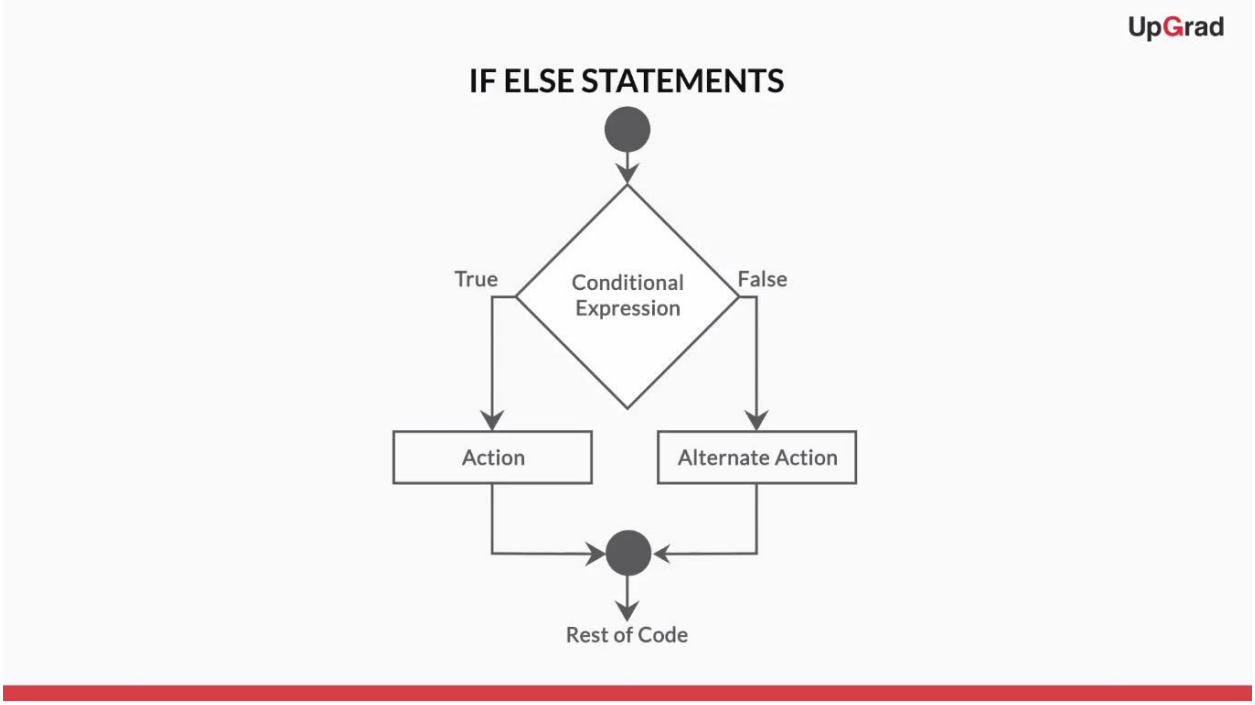
```
IF goalsA > goalsB THEN
  Display - A wins
ELSE
  Display - B wins
END IF
```

Then you saw a detailed example using more complex Boolean operations. You wanted to write a pseudocode that displayed a message based on the conditions. You saw this example where the user would be happy if the salary was more than or equal to $5000, or if the salary was half of that, i.e., $2500, and the workplace was Switzerland.

```
IF (salary >= $5000) OR
   (salary >= $2500 AND location.equals("Switzerland")) THEN
  Display - Yayy! Happy :)
ELSE
  Display - Ahh! Tough life
END IF
```

Through these cases, it was established that an IF-ELSE statement is designed to give either of the following two outputs:

1. When the condition is met, i.e., the expression is TRUE
2. Or, when the condition is not met, i.e., the expression is FALSE



Using this logic you created a program where you took two numbers from the user and returned the comparison results.

You wrote the pseudocode:

```
Read number1, number2
IF number1 >= number2 THEN
  Display - number1 is greater than or equal to number2
ELSE
  Display - number2 is greater
END IF
```

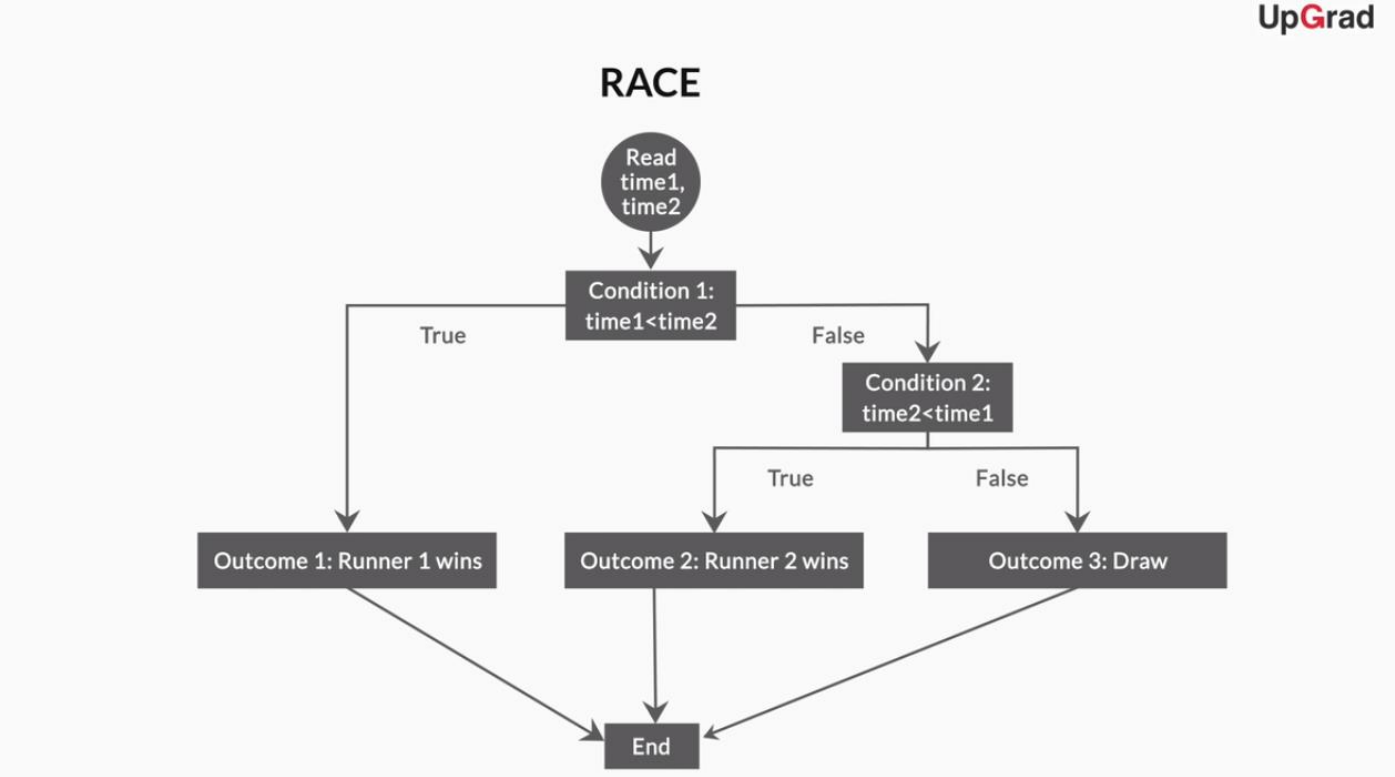
But there is an issue with this program. It checks a single condition and using the logic you created this program takes two numbers from the user and returns the comparison results. Since in an IF-ELSE condition you can only check a single condition, here you combined two conditions—**more than** and **equal to**—into one.

ELSE IF Statements

Then you saw the case of a race between the two runners. In this case, you couldn't simply combine the two conditions as above and say that runner 1 was better than or equal to runner 2. You could either proclaim both as winners or conduct another race to find the winner.

So now arose the need to check three separate conditions:

1. If runner A recorded less time than runner B, runner A wins.
2. If runner B recorded less time than runner A, runner B wins.
3. If runner A and runner B recorded the same time, both are joint winners.



Then based on the above logic, you wrote the following pseudocode:

```
Read time1, time2
IF time1 < time2 THEN
  Display - Runner 1 won the race
ELSE IF time2 < time1 THEN
  Display - Runner 2 won the race
ELSE
  Display - Race was a draw
END IF
```

Thus, using the ELSE-IF statement you learnt how to check for multiple conditions.

Then you wrote the program where you had to assign grades based on certain scores.

UpGrad

GRADES

Condition	Test Score (out of 100)	Grade
More than or equal to 90	>=90	A
Between 90 and 80 (incl. 80)	90 > Marks >= 80	B
Between 80 and 65 (incl. 65)	80 > Marks >= 65	C
Between 65 and 50 (incl. 50)	65 > Marks >= 50	D
Between 50 and 35 (incl. 35)	50 > Marks >= 35	E
Less than 35	Marks < 35	F

Then you wrote the pseudocode for this problem based on which you wrote –

```
Read testScore
IF testScore >= 90 THEN
  grade = "A"
ELSE IF testScore >= 80 AND testScore < 90 THEN
  grade = "B"
ELSE IF testScore >= 65 AND testScore < 80 THEN
  grade = "C"
ELSE IF testScore >= 50 AND testScore < 65 THEN
  grade = "D"
ELSE IF testScore >= 35 AND testScore < 50 THEN
  grade = "E"
ELSE
  grade = "F"
END IF
DISPLAY - grade
```

Then you saw how to create a basic **calculator** using ELSE-IF statements. In this case, you had to make a special accommodation in the code for the case where the denominator input by the user was zero. Also, in the last condition, we inserted another expression to check whether the denominator was zero. You later learnt that this was an instance of nested IF-ELSE statements.

```
import java.util.Scanner;

public class Calculator {
    public static void main(String args[]) {
        String operation;
        Double num1, num2, result;
        Scanner in = new Scanner(System.in);

        System.out.println("Enter first number  ");
        num1 = in.nextDouble();
        System.out.println("Enter second number  ");
        num2 = in.nextDouble();

        System.out.println("Enter operation:");
        System.out.println("Enter Sum, Multiply, Subtract or Divide" + "\n");
        Scanner op = new Scanner(System.in);
        operation = op.next();
        String sum = "Sum";
        String multiply = "Multiply";
        String subtract = "Subtract";
        String divide = "Divide";

        if (operation.equals("Sum")) {
            result = num1 + num2;
            System.out.println("Sum of entered numbers = " + result);
        } else if (operation.equals("Multiply")) {
            result = num1 * num2;
            System.out.println("Product of entered numbers = " + result);
        } else if (operation.equals("Subtract")) {
            result = num1 - num2;
            System.out.println("Difference of numbers = " + result);
        } else if (operation.equals("Divide")) {
            if (num2 == 0) {
                System.out.println("Division not defined");
            } else {
                result = num1 / num2;
                System.out.println("Dividend of entered numbers = " + z);
            }
        }
    }
}
```

Nested If Else Statements

A nested IF-ELSE condition basically means an IF (or IF ELSE) loop inside another IF loop. This is very useful when a condition also has some sub-conditions that need to be checked.

So, this was evident throughout the Ticket Pricing example.

As you noticed here, after the age condition was checked, other specific “sub-conditions” such as suffering from a chronic disease or being a student needed to be checked to arrive at the ticket price.

UpGrad

TICKET PRICES

Normal Ticket - $20

Age	Student	Chronic Disease	Discount
<= 3			100%
3 < age <= 18		Yes	100%
3 < age <= 18		No	50%
> 65			50%
18 < age <= 65	Yes		25%
18 < age <= 65	No		0%

Thus, you framed the pseudocode to solve the problem:

```
Input age
IF age < 3 THEN
  ticketPrice = 0
  Display - Free Entry
ELSE IF age > 3 AND age < 18 THEN
  IF chronicDisease == TRUE  THEN
    ticketPrice = 0
    Display - Free Entry
  ELSE
    ticketPrice = 20 * 0.5
    Display - ticketPrice
  END IF
```

```
ELSE IF age > 18 AND age < 65 THEN
  IF enrolledAsStudent == TRUE  THEN
    ticketPrice = 20 * 0.75
    Display - ticketPrice
  ELSE
    ticketPrice = 20
    Display - ticketPrice
  END IF

ELSE
  ticketPrice = 20 * 0.5
  Display - ticketPrice
END IF
```

Switch Statements

After learning the IF-ELSE statements, ELSE-IF statements, nested IF-ELSE statements, you saw the Switch statements.

Just like ELSE-IF, a Switch statement is commonly used where you need to check multiple cases.
A switch statement looks like:

```
Read choice
Switch(choice)
 Case x:
   Statement
   break;
 Case y:
   Statement
   break;
 Default:
   Display - Incorrect input
END SWITCH
```

You saw that Switch has a default case, which is equivalent to the ELSE condition in the case of IF-ELSE statements. The default case is executed if no other case is met. The break statement tells the Switch operator when to stop.

Then, you created a program that grants access to a website if the user enters y or Y. The code looked like:

```
import java.util.Scanner;

public class as {
    public static void main(String[] args) {
        char choice;
        Scanner scan = new Scanner(System.in);
        // Ask the user to enter y or n.
        System.out.print(" Are you over 18. Enter Y or N: ");
        choice = scan.next().charAt(0);
        // the first character, y or n,  you input here is picked up by the
          variable choice.

        switch (choice) {
            case 'y':
                System.out.println("You may access this website.");
                break;
            case 'Y':
                System.out.println("You may access this website.");
                break;
          // break: The break statement tells the code to quit once a case is
          TRUE and has been executed. If there is no break, execution flows
          through into the next case, which is not desired.

            case 'N': // if cases have common output
            case 'n':

                System.out.println("You don't have access to this material");
                break;

            default: // executed if none of the above case is satisfied
                System.out.println("Incorrect Input!");
        }
    }
}
```

A Switch condition is similar in functionality to an ELSE-IF condition as it also checks multiple conditions and then gives the corresponding output. Although it has more readability, a Switch condition has limited use cases because it cannot evaluate the conditions within each case.

Switch compares the input against predefined cases. For earlier versions of Java, these predefined cases could only be constants such as an integer, boolean (true/false), or a character. But with Java version 7 and 8, you can also use strings as cases.