Java Control Flow Statements

Control Flow Statements

- **Decision Making Statements**
  - if
  - if - else
  - if - else - if ladder
  - Nested - if
  - Switch Statement
  - Switch Expression (Introduced in Java 12)
- **Iterative Statements**
  - For Loop
  - While Loop
  - do - while loop
  - for - each loop
- **Branching Statements**
  - break Statement
  - continue Statement

⇒ Decision Making Statements

1) If then (Simple if)
   - if condition is true, then if - block will get executed

```
if (boolean condition) {
    // code here gets executed when condition is true
}
```

```java
public class Main {

    public static void main(String[] args) {

        int val = 10;

        if(val > 8){
            System.out.println("only get executes when val is greater than 8 ");
        }
        System.out.println("this code will executes, even if val is less than 8");
    }
}
```

Output:

```
only get executes when val is greater than 8
this code will executes, even if val is less than 8

Process finished with exit code 0
```

2) If - else :

- If condition is true, if - block will get executed else if condition is false, then else block will get executed.

```java
public class Main {

    public static void main(String[] args) {

        int val = 7;

        if(val > 8){
            System.out.println("only get executes when val is greater than 8 ");
        }
        else {
            System.out.println("this code will executes, if val is less than or equals to 8");

        }
        System.out.println("this code will executes, no matter condition is true or false");
    }
}
```

Output:

```
this code will executes, if val is less than or equals to 8
this code will executes, no matter condition is true or false

Process finished with exit code 0
```

3) If - else - if ladder :-

- It contains if - statement with multiple (chain of) else - if statements. It evaluate the condition from top and goes down and any condition gets true, its block will get executed.

For Eg:-

```java
public class Main {

    public static void main(String[] args) {

        int val = 13;

        if(val == 1){
            System.out.println("val is 1");
        }
        else if(val == 2) {
            System.out.println("val is 2");
        }
        else if(val == 3) {
            System.out.println("val is 3");
        }
        else {
            System.out.println("val is: " + val);
        }

        System.out.println("this code will executes anyhow");
    }
}
```

Output:

```
val is: 13
this code will executes anyhow
```

4) Nested – if Statement: If else Statement within if-block or else-block

```java
public class Main {

    public static void main(String[] args) {

        int val = 13;

        if(val > 8){
            System.out.println("value is greater than 8 ");

            if(val < 15){
                System.out.println("value is greater than 8 but else than 15");
            }
            else {
                System.out.println("value is greater than 15");
            }
        }
        else{
            System.out.println("value is less than equal to 8");
        }

        System.out.println("this code will executes anyhow");
    }
}
```

Output:

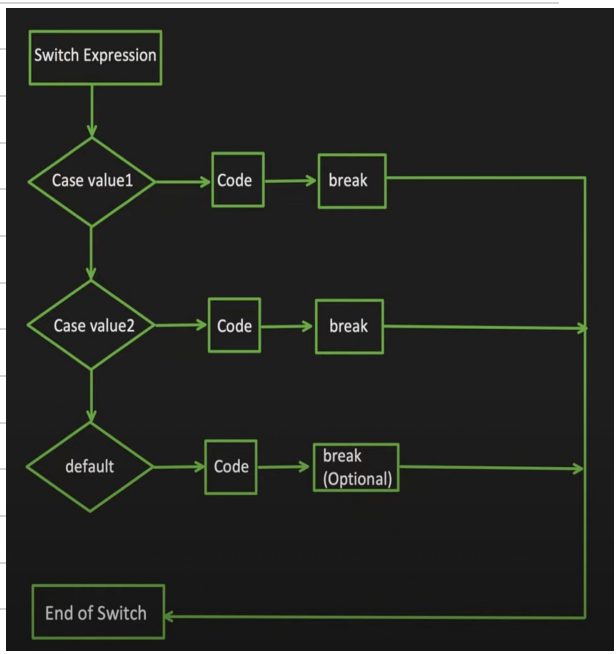
```
value is greater than 8
value is greater than 8 but else than 15
this code will executes anyhow

Process finished with exit code 0
```

5) SWITCH STATEMENT

- It is similar to if-else-if ladder based on condition particular block will get executed out of many alternatives.

- Syntax :-

```
switch (expression) {
    case value_1:
        // code statements
        break;
    case value_2 :
        // code statements
        break;
    ---
    ---
    case value_N :
        // code statements
        break;
    default :
        // default statements
        break; // (optional)
}
```



* If one of the condition is true & there is no break, then it'll execute the further statements irrespective of the value till it encounters the break statement.

* Let's see a few examples of switch statements

* Example - 1 : Normal Switch Statement.

```java
public class Main {
    public static void main(String[] args) {

        int a = 1;
        int b = 2;

        switch (a + b) {
            case 1:
                System.out.println("a+b is 1");
                break;
            case 2:
                System.out.println("a+b is 2");
                break;
            case 3:
                System.out.println("a+b is 3");
                break;
            default:
                System.out.println(a + b);
        }
    }
}
```

Output:

```
a+b is 3

Process finished with exit code 0
```

* Example - 2 : Without break statement

```java
public class Main {
    public static void main(String[] args) {

        int a = 1;
        int b = 2;

        switch (a + b) {
            case 1:
                System.out.println("a+b is 1");
                break;
            case 2:
                System.out.println("a+b is 2");
                break;
            case 3:
                System.out.println("a+b is 3");
            case 4:
                System.out.println("a+b is 4");
            default:
                System.out.println(a + b);
        }
    }
}
```

Output:

```
a+b is 3
a+b is 4
3

Process finished with exit code 0
```

* Example - 3 : Switch where default is in between cases :-

```java
public class Main {
    public static void main(String[] args) {

        int a = 1;
        int b = 2;

        switch (a + b) {
            case 1:
                System.out.println("a+b is 1");
                break;
            default:
                System.out.println(a + b);
            case 2:
                System.out.println("a+b is 2");
                break;
            case 3:
                System.out.println("a+b is 3");
            case 4:
                System.out.println("a+b is 4");

        }
    }
}
```

Output:

```
a+b is 3
a+b is 4

Process finished with exit code 0
```

* Example - 4 : Switch statement when default is in between 9 gets executed

```java
public class Main {
    public static void main(String[] args) {

        int a = 1;
        int b = 9;

        switch (a + b) {
            case 1:
                System.out.println("a+b is 1");
                break;
            default:
                System.out.println(a + b);
            case 2:
                System.out.println("a+b is 2");
                break;
            case 3:
                System.out.println("a+b is 3");
            case 4:
                System.out.println("a+b is 4");

        }
    }
}
```

Output:

```
10
a+b is 2

Process finished with exit code 0
```

* Example - 5 : Combining the cases in switch statement

```java
public class Main {
    public static void main(String[] args) {


        String month = "March";
        switch (month) {
            case "January":
            case "February":
            case "March":
                System.out.println("month value is in Q1");
                break;
            case "April":
            case "May":
            case "June":
                System.out.println("month value is in Q2");
                break;
            default:
                System.out.println("month value is in Q3 or Q4");
        }
    }
}
```

Output:

```
month value is in Q1

Process finished with exit
```

* We can write & combine the cases in a single line also:

```java
public class Main {
    public static void main(String[] args) {

        String month = "March";
        switch (month){
            case "January", "February", "March":
                System.out.println("month value is in Q1");
                break;
            case "April", "May", "June":
                System.out.println("month value is in Q2");
                break;
            default:
                System.out.println("month value is in Q3 or Q4");
        }
    }
}
```

Output:

```
month value is in Q1

Process finished with exit code 0
```

⇒ Few things we need to take care :

1) Two Cases can not have the same value

2) Switch expression data type and case values / constant data type should be same

3) Case value should be either LITERAL or CONSTANT

```java
public class Main {
    public static void main(String[] args) {

        int value = 1;

        switch (2+1-2) {
            case value:
                System.out.println("some code here");
            default:
                System.out.println("default code here");
        }
    }
}
```

This is wrong since value is not constant

```java
public class Main {
    public static void main(String[] args) {

        final int value = 1;

        switch (2+1-2) {
            case value:
                System.out.println("some code here");
            default:
                System.out.println("default code here");
        }
    }
}
```

This is right since value is constant

4) All use case need not to be handled

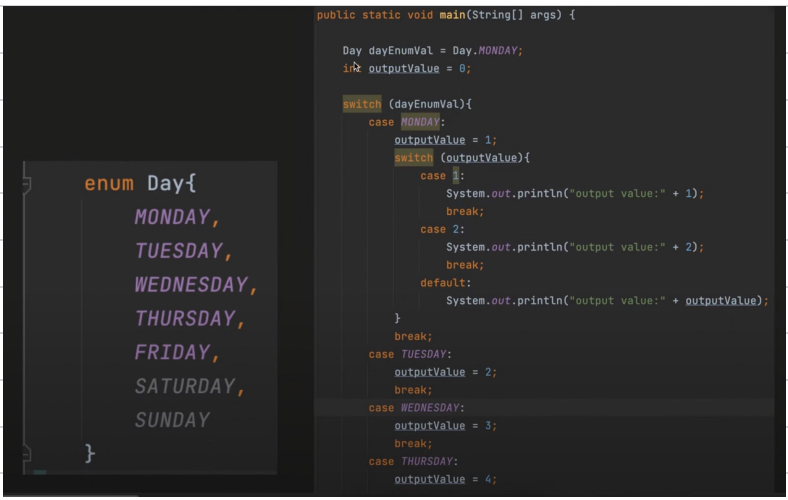
```java
enum Day{
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
}
```

```java
public static void main(String[] args) {

    Day dayEnumVal = Day.FRIDAY;
    int outputValue = 0;

    switch (dayEnumVal){
        case MONDAY:
            outputValue = 1;
            break;
        case TUESDAY:
            outputValue = 2;
            break;
        case WEDNESDAY:
            outputValue = 3;
            break;
        case THURSDAY:
            outputValue = 4;
            break;
    }

    System.out.println(outputValue);
}
```

Output:

```
0

Process finished with exit code
```

5.) Nested Switch statement is possible:



```
enum Day{
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
}
```

```
public static void main(String[] args) {

    Day dayEnumVal = Day.MONDAY;
    int outputValue = 0;

    switch (dayEnumVal){
        case MONDAY:
            outputValue = 1;
            switch (outputValue){
                case 1:
                    System.out.println("output value:" + 1);
                    break;
                case 2:
                    System.out.println("output value:" + 2);
                    break;
                default:
                    System.out.println("output value:" + outputValue);
            }
            break;
        case TUESDAY:
            outputValue = 2;
            break;
        case WEDNESDAY:
            outputValue = 3;
            break;
        case THURSDAY:
            outputValue = 4;
```

```
Output:

output value:1
1


Process finished with ex
```

6) Supported data types:

- 4 primitive types:
  - int
  - short
  - byte
  - char
- Wrapper Types of above primitive data types
  - Integer
  - Short
  - Byte
  - Character

7) Return is not possible within switch case

```java
public class Main {
    public static void main(String[] args) {

        String day = "";
        int val = 1;
         String outputVal = switch (val){

            case 1 :
                return "One";
        }
        System.out.println(day);
    }
}
```

Since return is not possible in switch, there are 2 other ways to do it:

1) Using "case N ->" label
2) Using "yield" statement

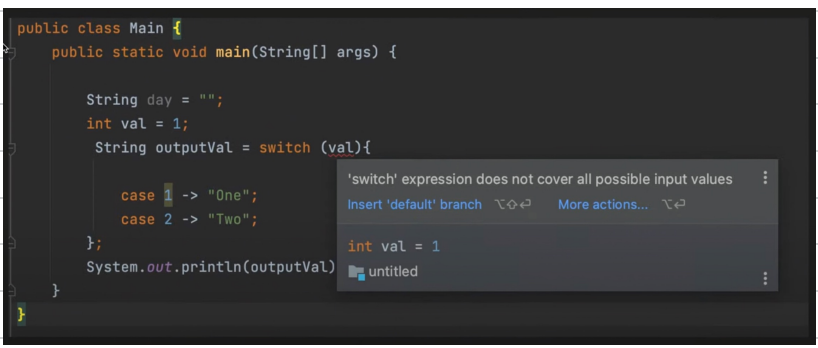
* These are available after Java 12

* Use of "case N ->":

```java
public class Main {
    public static void main(String[] args) {

        String day = "";
        int val = 1;
         String outputVal = switch (val){

            case 1 -> "One";
            case 2 -> "Two";
            default -> "None";
        };
        System.out.println(outputVal);
    }
}
```

This is switch expression

We don't need break in switch expression because it returns the value. Since we're returning the value & that is assigned to a variable. So, all possible use cases need to be handled for the expression



So it is giving compilation error because all use cases are not being handled. We can fix it using default.

```java
public class Main {
    public static void main(String[] args) {

        String day = "";
        int val = 1;
         String outputVal = switch (val){

            case 1 -> "One";
            case 2 -> "Two";
            default -> "None";
        };
        System.out.println(outputVal);
    }
}
```

* Using "->" we can not have block of statements. If we want block of statements and then return the value, we need to use "yield"

* Use of "yield":

```java
public class Main {
    public static void main(String[] args) {

        String day = "";
        int val = 1;
        String outputVal = switch (val){
            case 1 -> {
                //some code logic here
                yield "One";
            }
            case 2 -> {
                //some code logic here
                yield "Two";
            }
            default -> "None";
        };
```

⇒ ITERATIVE STATEMENTS:

1) For Loop:

- for (initialization of variable; condition check; increment/decrement of variable) {
  // statement block
  }

For Example:

```java
public class Main {
    public static void main(String[] args) {

      for(int val=1; val<=10; val++){
          System.out.println(val);
      }
    }
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10

Process finished with exit code 0
```

* We can also have nested for loops :

```java
public class Main {
    public static void main(String[] args) {

      for(int x=1; x<=3; x++){
          for(int y=1; y<=3; y++){
              System.out.println("x="+x + " : y=" +y);
          }
      }
    }
}
```

```
x=1 : y=1
x=1 : y=2
x=1 : y=3
x=2 : y=1
x=2 : y=2
x=2 : y=3
x=3 : y=1
x=3 : y=2
x=3 : y=3

Process finished with exit code 0
```

2) While loop :

- Initialize variable;

```
While (condition check) {
    // block of statements
 increment / decrement variable
}
```

For Example :

```java
public class Main {
    public static void main(String[] args) {

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

Output:

```
1
2
3
4
5

Process finished with exit code 0
```

3) Do - While Loop :

- initialize variable

```
do {
    // block of statements
 increment / decrement variable
} while (condition check);
```

- It is used when we require a loop to run atleast one time.

For example:

```java
public class Main {
    public static void main(String[] args) {

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

Output:

```
1
2
3
4
5

Process finished with exit code 0
```

4) For each loop:

- This is generally used to iterate over array or collections

```java
public class Main {
    public static void main(String[] args) {

      int valArray[] = {1,2,3,4,5};

        for(int val : valArray){
            System.out.println(val);
        }
    }
}
```

Output:

```
1
2
3
4
5

Process finished with exit code
```

⇒ BRANCHING STATEMENTS

1) break statement: It will break the loop & rest of the condition won't run

For example:

```java
public class Main {
    public static void main(String[] args) {

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

            if(val == 3){
                break;
            }
            System.out.println(val);
        }
    }
}
```

Output:

```
1
2

Process finished with exit code 0
```

Example - 2 : break is used to break the internal loop.

```java
public class Main {
    public static void main(String[] args) {

        for (int outerLoop = 1; outerLoop <= 5; outerLoop++)
            for (int innerLoop = 1; innerLoop <= 5; innerLoop++) {

                if (innerLoop == 2) {
                    break;
                }
                System.out.println(outerLoop + "," + innerLoop);
            }
    }
}
```

Output:

```
1,1
2,1
3,1
4,1
5,1

Process finished with
```

2) Continue statement :

- It is opposite of break . It will skip the particular iteration of the loop.

For Example :

```java
public class Main {
    public static void main(String[] args) {

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

            if(val == 3){
                continue;
            }
            System.out.println(val);
        }
    }
}
```

Output:

```
1
2
4
5
6
7
8
9
10

Process finished with exit code 0
```