

Bule Hora University

College of Informatics

Department of Information Technology

Object oriented Programming in Java (ITec2052)

Chapter-3

Decision and Repetition Statements

Organized by: Negessa W and Sadik A

October, 2020

Content of chapter-3

- **Decision and Repetition Statements**
- Overview of Java statements
- If statement
- Switch statement
- For loop
- While, Do while loop

Decision Statements

- Java provides several different types of control flow structures.
- These structures provide your application with direction.
- They take an input, decide what to do with it and how long to do it, and then let expressions handle the rest.
- Let us see them one by one how they work in Java.

If statement

- **if**: if statement is the most simple decision making statement.
- It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed

Syntax:

if(condition)

{

// Statements to exec

// condition is true

}

```
package it2;

public class IT2 {

    public static void main(String[] args) {

        int i = 1;

        if (i < 3) {

            System.out.println("yes 1 is less than 3");

        }

    }

}
```

If...else statement

- The **if..else** structure performs this operation:
if *this* is true then do *that* otherwise do
something else.
- Of course, the “otherwise” portion is optional.
- Here is the structure, labeled with standard terms:

if (***if(condition)***
statement;
else
statement;

```
package it2;  
  
public class IT2 {  
  
    public static void main(String[] args) {  
        int i = 1;  
        if (i > 3) {  
            System.out.println("yes 1 is less than 3");  
        }  
        else  
        {  
            System.out.println("No 1 is not greater than 3");  
        }  
    }  
}
```

If...else..if statement...

- Here, a user can decide among multiple options

- Syntax:

***if (condition)
statement;
else if (condition)
statement; .
Else
statement;***

```
package it2;  
public class IT2 {  
    public static void main(String[] args) {  
        int i = 1;  
        if (i == 3) {  
            System.out.println(" i is 3");  
        }  
        else if(i==1)  
        {  
            System.out.println("i is 1");  
        }  
        else  
        {  
            System.out.println("i is 2");  
        }  
    }  
}
```

nested..if

- Nested if statements means an if statement inside an if statement
- Syntax:

```
if (condition1) {  
// Executes when condition1 is  
true  
  
    if (condition2) {  
        // Executes when  
condition2 is true  
    }
```

Example of nested if

```
package it2;

public class IT2 {

    public static void main(String[] args) {
        int i = 1;
        if (i < 3) {
            System.out.println(" i is less than 3");
            if (i<2){
                System.out.println(" i is also less than 2");
            }
        }
        else
        {
            System.out.println("i is greater than 4");
        }
    }
}
```

switch statement

- The **switch** control flow structure is useful when you have a single expression with many possible options.
- The **switch** statement is executed by comparing the value of an initial expression or variable with other variables or expressions.

```
switch(expression) {  
    case expression: statement;  
        break;  
    case expression: statement;  
        break;  
    case expression: statement;  
        break;  
    default: statement;  
        break;  
}
```

Example of switch

```
public class switch_class {  
    public static void main(String args[])  
    {  
        int i = 9;  
        switch (i)  
        {  
            case 0:  
                System.out.println("i is zero.");  
                break;  
            case 1:  
                System.out.println("i is one.");  
                break;  
            case 2:  
                System.out.println("i is two.");  
                break;  
            default:  
                System.out.println("i is nine.");  
        }  
    }  
}
```

- **while and do...while statements**
- The **while** and **do..while** loops perform the same function. The only difference is that:
 - The **while** loop verifies the expression *before* executing the statement, and
 - The **do..while** loop *after* executing the s
- Syntax of **while**:
***while (condition) {
statement;
}***

```
public class loop_class {  
    public static void main(String args[])  
    {  
        int n = 0;  
        while(n <= 5) {  
            System.out.println(n);  
            n++;  
        }  
    }  
}
```

Syntax of do...while

do {
 statement
} while(condition)

```
public class loop_class {  
    public static void main(String args[])  
    {  
        int n = 0;  
        do{  
            System.out.println(n);  
            n++;  
        }while(n <= 5);  
    }  
}
```

for loop

- The idea behind a **for** loop is that we want to step through a sequence of numbers until a limit is reached.
 - The loop steps through our range in whatever step increment we want, checking at the beginning of each loop to see if we have caused our “quit” expression to become true.

- Syntax:

```
for (variable ; expression1 ; exp  
{  
//statement  
}
```

```
public class loop_class {  
    public static void main(String args[])  
    {  
        for (int n = 0; n <=5; n++){  
            System.out.println(n);  
        }  
    }  
}
```

Continue Statement in Java

- **Continue statement** is mostly used inside loops. Whenever it is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements for the current iteration.

```
public class ContinueExample {  
  
    public static void main(String args[]){  
        for (int j=0; j<=6; j++){  
            {  
                if (j==4)  
                {  
                    continue;  
                }  
  
                System.out.print(j+" ");  
            }  
        }  
    }  
}
```

Output:

```
0 1 2 3 5 6
```

Break statement in Java

- Use break statement to come out of the loop instantly. Whenever a break statement is encountered inside a loop, the control directly comes out of loop and the loop iteration

```
public class BreakExample1 {  
    public static void main(String args[]){  
        int num =0;  
        while(num<=100)  
        {  
            System.out.println("Value of variable is: "+num);  
            if (num==2)  
            {  
                break;  
            }  
            num++;  
        }  
        System.out.println("Out of while-loop");  
    }  
}
```

Output:

```
Value of variable is: 0  
Value of variable is: 1  
Value of variable is: 2  
Out of while-loop
```