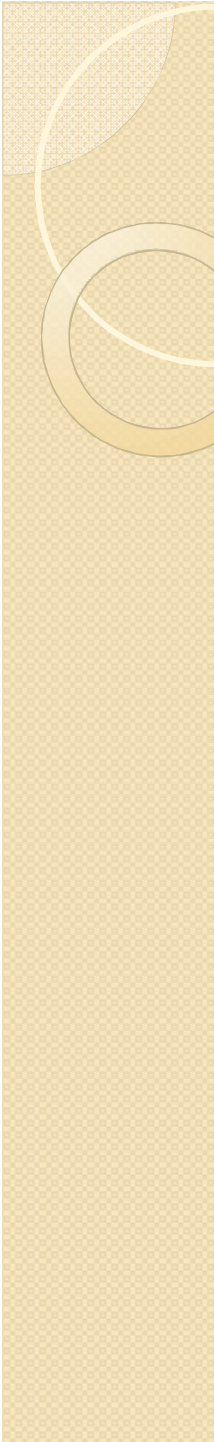




**Course Name:**  
**Advanced Java**



# Lecture 3

## Topics to be covered

- Control Statements

# Introduction

The control statement are used to control the flow of execution of the program . This execution order depends on the supplied data values and the conditional logic. Java contains the following types of control statements:

- Selection Statements
- Repetition Statements
- Branching Statements



# Selection Statement

Selection statements can be of three types:

- If statement
- If-else statement
- Switch statement

# if-statement

This is a control statement to execute a single statement or a block of code, when the given condition is true and if it is false then it skips **if** block and rest code of program is executed .

## Syntax:

```
if(conditional_expression)
{
    <statements>;
    ...;
    ...;
}
```

# Example of if-statement

## Example:

If  $n\%2$  evaluates to 0 then the "if" block is executed. Here it evaluates to 0 so if block is executed. Hence **"This is even number"** is printed on the screen.

```
int n = 10;
if(n%2 == 0)
{
    System.out.println("This is even number");
}
```

# if-else Statement

The "**if-else**" statement is an extension of if statement that provides another option when 'if' statement evaluates to "false" i.e. else block is executed if "if" statement is false.

## Syntax:

```
if(conditional_expression)
{
    <statements>;
    ...;
    ...;
}
else
{
    <statements>;
    ...;
    ...;
}
```

# Example of if-else statement

## Example:

If  $n\%2$  doesn't evaluate to 0 then else block is executed. Here  $n\%2$  evaluates to 1 that is not equal to 0 so else block is executed. So **"This is not even number"** is printed on the screen.

```
int n = 11;
if(n%2 == 0)
{
    System.out.println("This is even number");
}
Else
{
    System.out.println("This is not even number");
}
```



# Switch Statement

- This is an easier implementation to the if-else statements.
- The keyword "**switch**" is followed by an expression that should evaluate to byte, short, char or int primitive data types ,only.
- In a switch block there can be one or more labeled cases. The expression that creates labels for the case must be unique.
- The switch expression is matched with each case label. Only the matched case is executed ,if no case matches then the default statement (if present) is executed.

# Syntax for Switch Statement

## Syntax:

```
switch(control_expression)
{
    case expression 1: <statement>;
    case expression 2: <statement>;
    ...
    ...
    case expression n: <statement>;
    default:           <statement>;
} //end switch
```

# Example of Switch Statement

**Example:** Here expression "day" in switch statement evaluates to 5 which matches with a case labeled "5" so code in case 5 is executed that results to output "**Friday**" on the screen.

```
int day = 5;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    case 4:
        System.out.println("Thrusday");
        break;
    case 5:
        System.out.println("Friday");
        break;
    case 6:
        System.out.println("Saturday");
        break;
    case 7:
        System.out.println("Sunday");
        break;
    default:
        System.out.println("Invalid entry");
        break;
}
```



# Repetition Statements

In java, there are three types of Repetition statements:

- While loop statements
- Do-while loop statements
- For loop statements

# While loop statements

This is a looping or repeating statement. It executes a block of code or statements till the given condition is true. The expression must be evaluated to a boolean value. It continues testing the condition and executes the block of code. When the expression results to false control comes out of loop.

## **Syntax:**

```
while(expression)
{
    <statement>;
    ...;
    ...;
}
```

# Example of while loop

Here expression  $i \leq 10$  is the condition which is checked before entering into the loop statements. When  $i$  is greater than value 10 control comes out of loop and next statement is executed. So here  $i$  contains value "1" which is less than number "10" so control goes inside of the loop and prints current value of  $i$  and increments value of  $i$ . Now again control comes back to the loop and condition is checked. This procedure continues until  $i$  becomes greater than value "10". So this loop prints values 1 to 10 on the screen.

```
int i = 1;
//print 1 to 10
while (i <= 10){
    System.out.println("Num " + i);
    i++;
}
```

# Do-while loop statements

This is another looping statement that tests the given condition past so you can say that the do-while looping statement is a past-test loop statement. First the **do** block statements are executed then the condition given in **while** statement is checked. So in this case, even the condition is false in the first attempt, do block of code is executed at least once.

## Syntax:

```
do{  
    <statement>;  
    ...;  
    ...;  
}while (expression);
```

# Example of do-while loop

Here first do block of code is executed and current value "1" is printed then the condition  $i \leq 10$  is checked. Here "1" is less than number "10" so the control comes back to do block. This process continues till value of  $i$  becomes greater than 10.

```
int i = 1;
do{
    System.out.println("Num: " + i);
    i++;
}while(i <= 10);
```



# for loop statements

This is also a loop statement that provides a compact way to iterate over a range of values. From a user point of view, this is reliable because it executes the statements within this block repeatedly till the specified conditions is true .

## Syntax:

```
for (initialization; condition; increment or decrement)
{
<statement>;
...;
...;
}
```

**initialization:** The loop is started with the value specified.

**condition:** It evaluates to either 'true' or 'false'. If it is false then the loop is terminated.

**increment or decrement:** After each iteration, value increments or decrements.

# Example of for loop

- Here num is initialized to value "1", condition is checked whether  $\text{num} \leq 10$ . If it is so then control goes into the loop and current value of num is printed. Now num is incremented and checked again whether  $\text{num} \leq 10$ . If it is so then again it enters into the loop. This process continues till  $\text{num} > 10$ . It prints values 1 to 10 on the screen.

```
for (int num = 1; num <= 10; num++)  
{  
    System.out.println("Num: " + num);  
}
```



# Branching Statements

In Java, there are three types of branching statements:

- Break statement
- Continue statement
- Return statement

# Break statement

The break statement is a branching statement that contains two forms: labeled and unlabeled. The break statement is used for breaking the execution of a loop (while, do-while and for) . It also terminates the switch statements.

## **Syntax:**

`break;` // breaks the innermost loop or switch statement.

`break label;` // breaks the outermost loop in a series of nested loops.

# Example of break statement

When if statement evaluates to true it prints "data is found" and comes out of the loop and executes the statements just following the loop.

```
int num[] = {2,9,1,4,25,50};  
  
int search = 4;  
  
for (int i = 1; i < num.length; i++){  
  
    if ( num[i] == search){  
  
        System.out.println("data is found!");  
  
        break;  
  
    }  
  
}
```

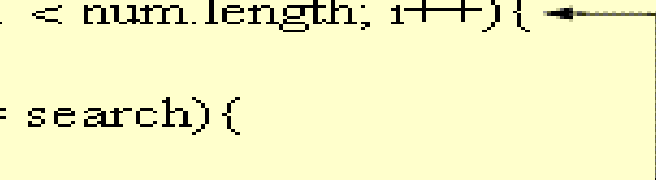
# Continue statement

- This is a branching statement that are used in the looping statements (while, do-while and for) to skip the current iteration of the loop and resume the next iteration .

## **Syntax:**

`continue;`

# Example of continue statement

```
int num[] = {2,9,1,4,25,50};  
  
int search = 4;  
  
.....  
for (int i = 1; i < num.length; i++) {  
    if (num[i] != search) {  
        continue;   
    }  
.....  
if (found == search) {  
    System.out.println("data is found!");  
    break;  
}  
}
```

# Return statement

It is a special branching statement that transfers the control to the caller of the method. This statement is used to return a value to the caller method and terminates execution of method. This has two forms: one that returns a value and the other that can not return. The returned value type must match the return type of method.

## Syntax:

`return;`

`return values;`

**return;** //This returns nothing. So this can be used when method is declared with void return type.

**return expression;** //It returns the value evaluated from the expression.



# Example of return statement

- Here Welcome() function is called within println() function which returns a String value "Welcome to roseIndia.net". This is printed to the screen.

```
public static void Hello(){  
    System.out.println("Hello " + Welcome());  
}  
static String Welcome(){  
    return "Welcome to RoseIndia.net";  
}
```

The diagram illustrates the flow of control between two methods. A bracket labeled '3rd' spans the entire Hello() method. An arrow labeled '1st' points from the Welcome() call inside Hello() to the start of the Welcome() method. Another arrow labeled '2nd' points from the return statement inside Welcome() back to the Welcome() call in Hello().