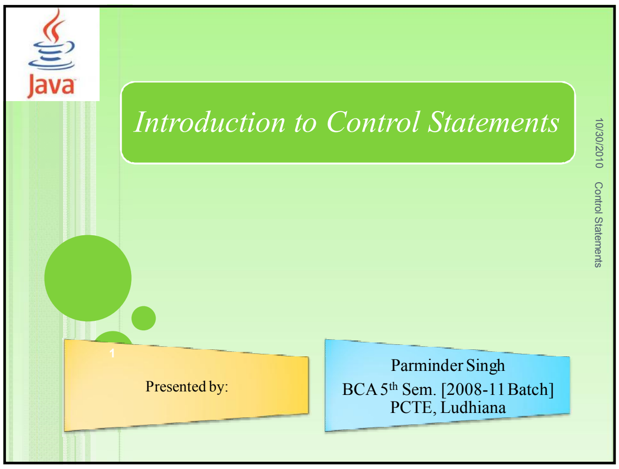


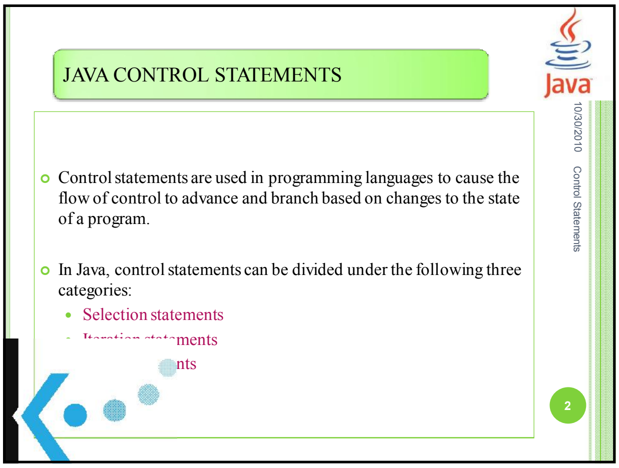
Introduction to Control Statements

Presented by:

Parminder Singh
BCA 5th Sem. [2008-11 Batch]
PCTE, Ludhiana

10/30/2010 Control Statements

1

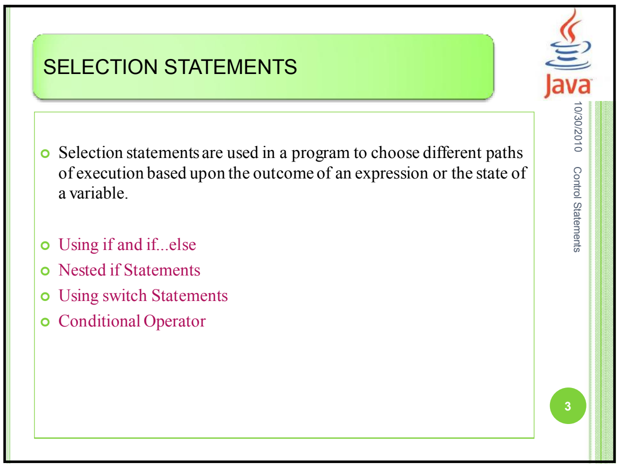


JAVA CONTROL STATEMENTS

- Control statements are used in programming languages to cause the flow of control to advance and branch based on changes to the state of a program.
- In Java, control statements can be divided under the following three categories:
  - Selection statements
  - Iteration statements
  - nts

10/30/2010 Control Statements

2

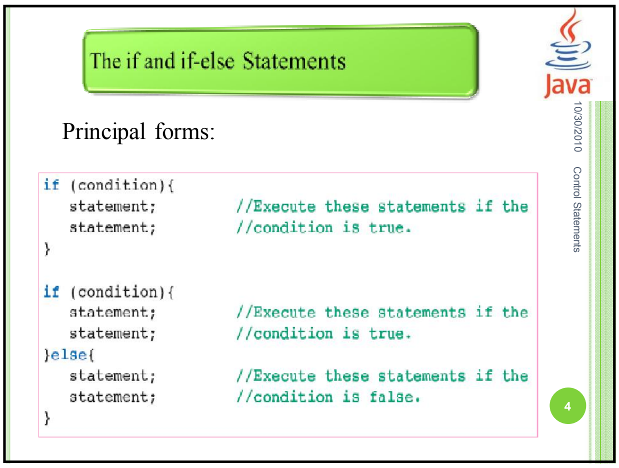


SELECTION STATEMENTS

- Selection statements are used in a program to choose different paths of execution based upon the outcome of an expression or the state of a variable.
- Using if and if...else
- Nested if Statements
- Using switch Statements
- Conditional Operator

10/30/2010 Control Statements

3



The if and if-else Statements

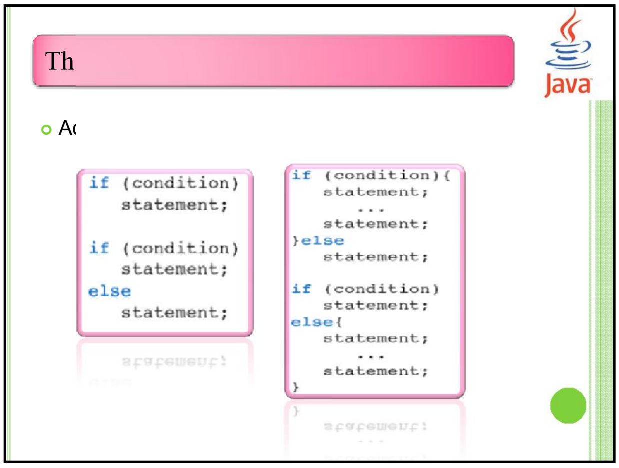
Principal forms:

```
if (condition){
   statement;          //Execute these statements if the
   statement;          //condition is true.
}

if (condition){
   statement;          //Execute these statements if the
   statement;          //condition is true.
}else{
   statement;          //Execute these statements if the
   statement;          //condition is false.
}
```

10/30/2010 Control Statements

4



Th

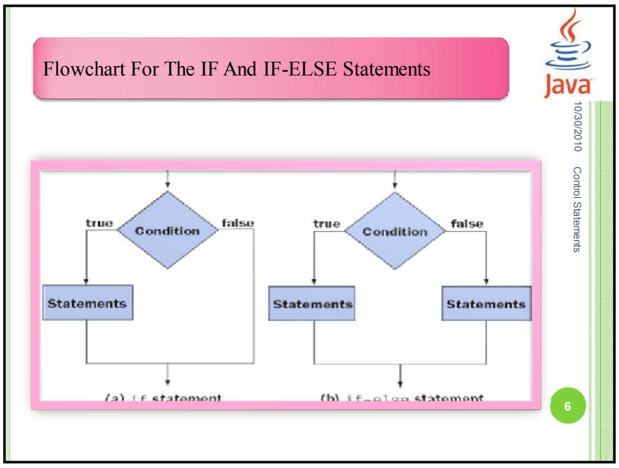
- A(

```
if (condition)
   statement;

if (condition)
   statement;
else
   statement;
```

```
if (condition){
   statement;
      ...
   statement;
}else
   statement;

if (condition)
   statement;
else{
   statement;
      ...
   statement;
}
```



Flowchart For The IF And IF-ELSE Statements

10/30/2010 Control Statements

6

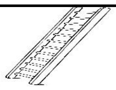
EXAMPLE OF IF-ELSE

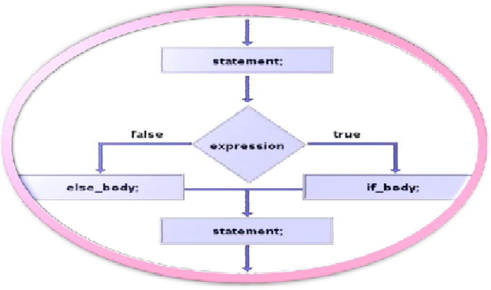
```
if( a > b)
{
  System.out.println("A = " + a + "\tB = "
    + b);
  System.out.println("A is greater than B");
}
else
{
System.out.println("A = " + a + "\tB = " +
    b);
System.out.println("Either both are equal
    or B is greater");
}
```

EXAMPLE OF NESTED IF

```
class Example4_2
{
  public static void main(String Args[])
  {
       int a = 3;
       if (a <= 10 && a > 0)
       {
        System.out.println("Number is valid.");
        if ( a < 5)
            System.out.println("From 1 to 5");
       else
           System.out.println("From 5 to 10");
       }
   else
   System.out.println("Number is not valid");
   }
}
```

Else-if ladder





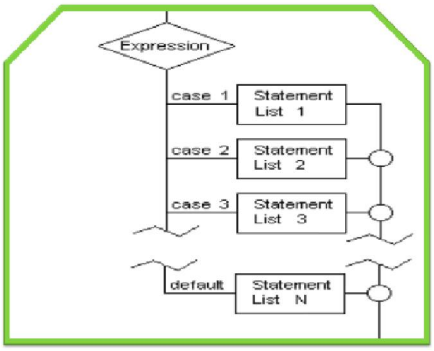
Example of else-if ladder

```
class Example4_1{
      public static void main (String Args[]){
         int a = 5;
         boolean val = false;
        if(val)
            System.out.println("val is false, so it won't execute";
        else if (a < 0 )
            System.out.println("A is a negative value");
            else if (a > 0)
                  System.out.println ("A is a positive value");
                  else
                   System.out.println ("A is equal to zero");
}
}
```

Switch Statement

- The switch statement of java is another selection statement that defines different paths of execution for a program.
- It is more efficient that the if statement
- The expression must be of type int, short, byte or char.
- The selection in the switch statement is determined by the values between the parenthesis after the keyword switch and the expression.
- The break statement is used in each sequence case value statements to terminate this sequence
- The break statement is optional in the switch statement

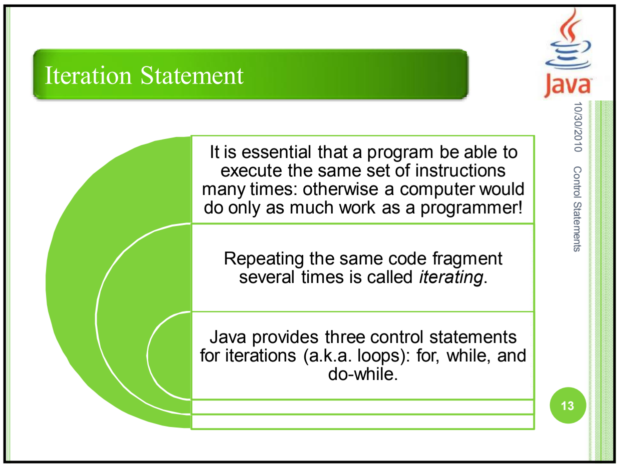
Sy



Example

```
class Example4_3{
public static void main(String Args[]){
int month = 3;
switch (month){
case 1:
System.out.println("The month of January");
break;
case 2:
System.out.println("The month of February");
break;
case 3:
System.out.println("The month of March");
break;
case 4:
System.out.println("The month of April");
break; case 5:
System.out.println("The month of May");
break;
```

```
case 6:
System.out.println("The month of June");
break;
case 7:
System.out.println("The month of July");
break;
case 8:
System.out.println("The month of August");
break;
case 9:
System.out.println("The month of September");
break;
case 10:
System.out.println("The month of October");
break;
case 11:
System.out.println("The month of November");
break;
case 12:
System.out.println("The month of December");
break;
default:
System.out.println("Invalid month");
}
}
}
```

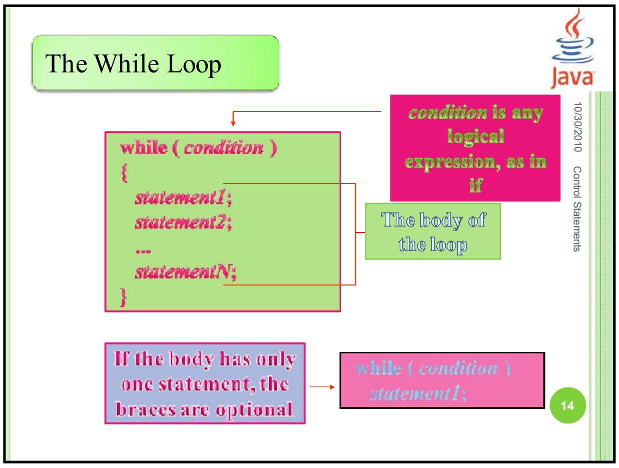


Iteration Statement

It is essential that a program be able to execute the same set of instructions many times: otherwise a computer would do only as much work as a programmer!

Repeating the same code fragment several times is called *iterating*.

Java provides three control statements for iterations (a.k.a. loops): for, while, and do-while.



The While Loop

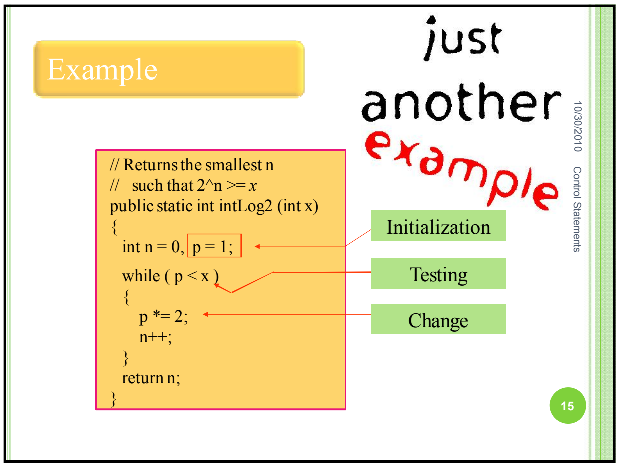
```
while ( condition )
{
   statement1;
   statement2;
   ...
   statementN;
}
```

condition is any logical expression, as in if

The body of the loop

If the body has only one statement, the braces are optional

```
while ( condition )
  statement1;
```

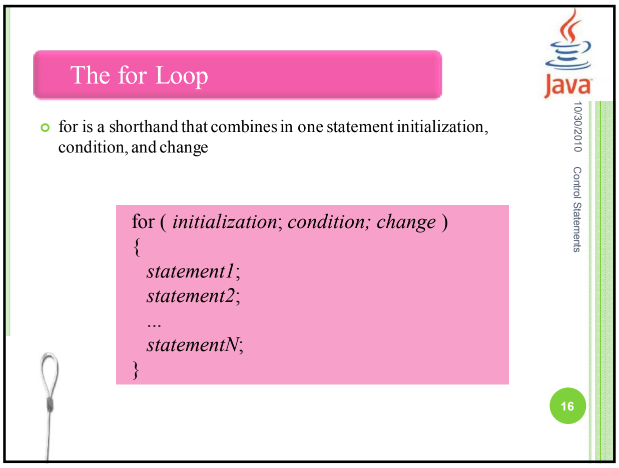


Example

just another example

```
// Returns the smallest n
//   such that 2^n >= x
public static int intLog2 (int x)
{
  int n = 0, p = 1;
  while ( p < x )
  {
     p *= 2;
     n++;
  }
  return n;
}
```

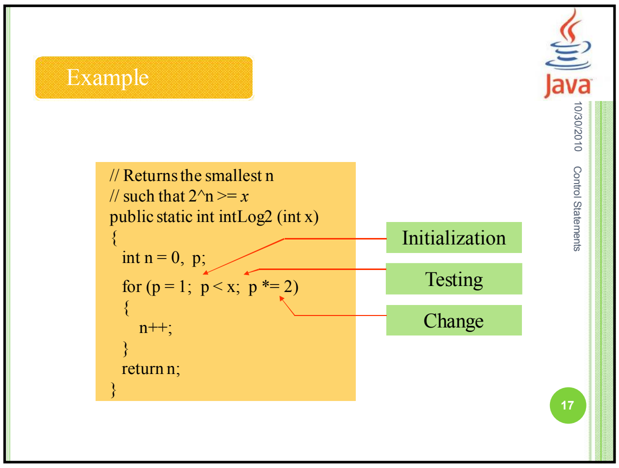
Initialization
Testing
Change



The for Loop

- for is a shorthand that combines in one statement initialization, condition, and change

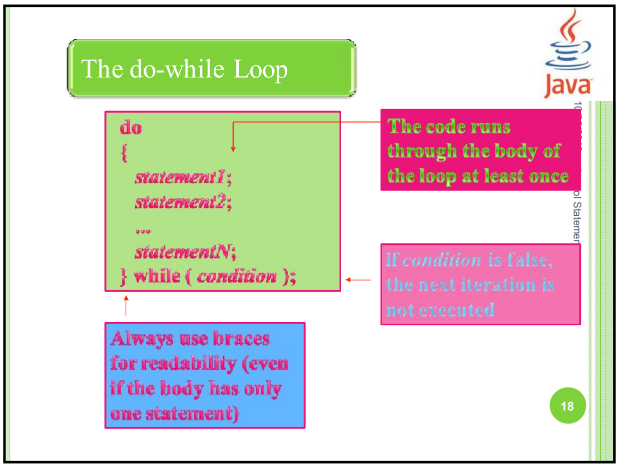
```
for ( initialization; condition; change )
{
   statement1;
   statement2;
   ...
   statementN;
}
```



Example

```
// Returns the smallest n
// such that 2^n >= x
public static int intLog2 (int x)
{
  int n = 0,  p;
  for (p = 1;  p < x;  p *= 2)
  {
     n++;
  }
  return n;
}
```

Initialization
Testing
Change



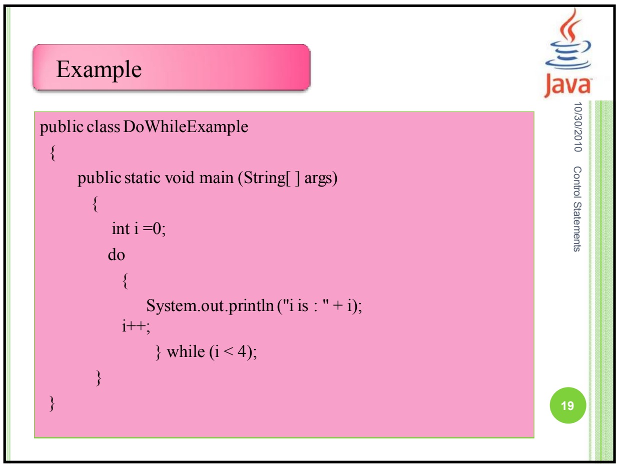
The do-while Loop

```
do
{
   statement1;
   statement2;
   ...
   statementN;
} while ( condition );
```

The code runs through the body of the loop at least once

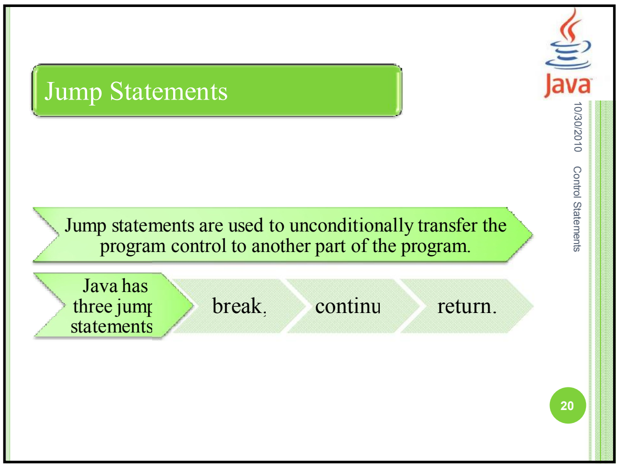
if *condition* is false, the next iteration is not executed

Always use braces for readability (even if the body has only one statement)



Example

```
public class DoWhileExample
 {
      public static void main (String[ ] args)
        {
           int i =0;
           do
             {
                System.out.println ("i is : " + i);
             i++;
                  } while (i < 4);
        }
 }
```



Jump Statements

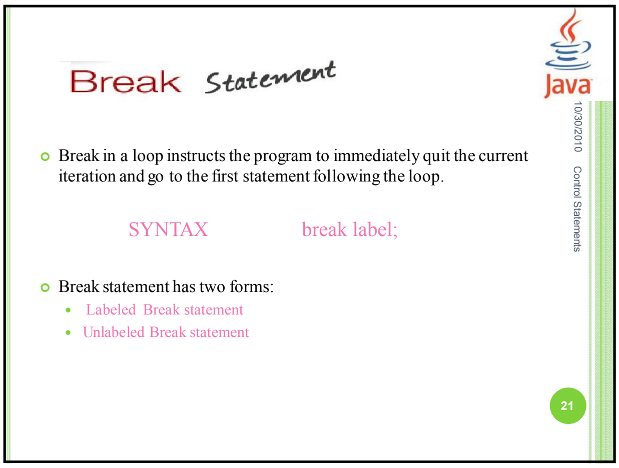
Jump statements are used to unconditionally transfer the program control to another part of the program.

Java has three jump statements

break.

continu

return.

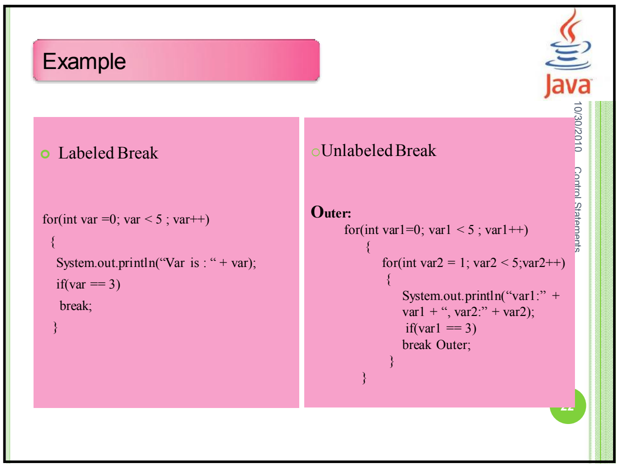


Break Statement

- Break in a loop instructs the program to immediately quit the current iteration and go to the first statement following the loop.

SYNTAX break label;

- Break statement has two forms:
  - Labeled Break statement
  - Unlabeled Break statement



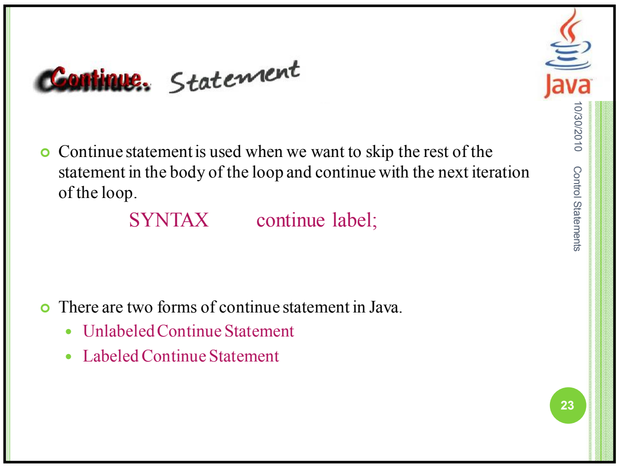
Example

- Labeled Break

```
for(int var =0; var < 5 ; var++)
 {
   System.out.println(“Var is : “ + var);
   if(var == 3)
    break;
  }
```

- Unlabeled Break

```
Outer:
        for(int var1=0; var1 < 5 ; var1++)
              {
                   for(int var2 = 1; var2 < 5;var2++)
                   {
                       System.out.println(“var1:” +
                       var1 + “, var2:” + var2);
                        if(var1 == 3)
                       break Outer;
                   }
              }
```

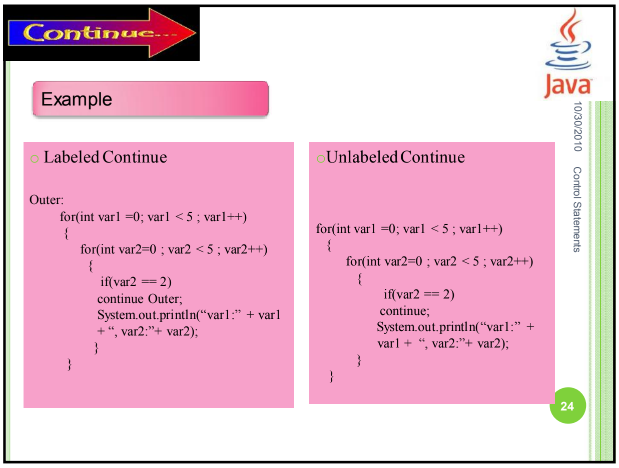


Continue. Statement

- Continue statement is used when we want to skip the rest of the statement in the body of the loop and continue with the next iteration of the loop.

SYNTAX continue label;

- There are two forms of continue statement in Java.
  - Unlabeled Continue Statement
  - Labeled Continue Statement



Continue...

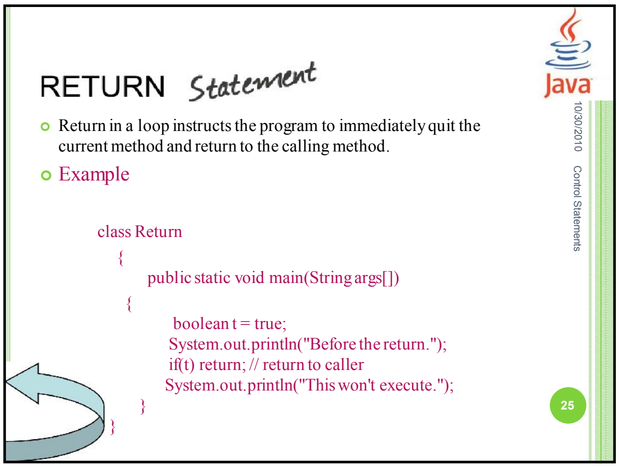
Example

- Labeled Continue

```
Outer:
       for(int var1 =0; var1 < 5 ; var1++)
        {
            for(int var2=0 ; var2 < 5 ; var2++)
              {
                if(var2 == 2)
               continue Outer;
               System.out.println(“var1:” + var1
               + “, var2:”+ var2);
              }
         }
```

- Unlabeled Continue

```
for(int var1 =0; var1 < 5 ; var1++)
  {
       for(int var2=0 ; var2 < 5 ; var2++)
         {
               if(var2 == 2)
              continue;
             System.out.println(“var1:” +
             var1 +  “, var2:”+ var2);
         }
  }
```



RETURN *Statement*

- Return in a loop instructs the program to immediately quit the current method and return to the calling method.
- Example

```
class Return
    {
        public static void main(String args[])
     {
          boolean t = true;
         System.out.println("Before the return.");
         if(t) return; // return to caller
        System.out.println("This won't execute.");
      }
   }
```