Control Structures

Dr. H. B. Prajapati

Associate Professor
Department of Information Technology
Dharmsinh Desai University

2 July '20

Core Java Technology

Table of contents

Outline of Presentation

Program Control

- In real life, we take decisions for some tasks:
  - Should I become an Engineer or Doctor
  - I want to watch Matrix movie 3 times.
  - I want to eat Panipuris till my stomach gets full
  - I want to perform paragliding at least once, whether I like it or do not like it.
  - Should I travel by train or airplane? If by train, then 1st AC or 2nd AC? If by airplane, then Economic class or Executive class?

Program Control

- In a computer program also, we need to perform tasks based on decisions or choices
- In a Computer Program, program control specifies the order in which statements are executed in a computer program
- The control structures can be divided into two types:
  1. Decision constructs (Either this or that)
  2. Loop constructs (How many times or till what to perform some task)
- There are three forms of decision constructs:
  1. if
  2. if else
  3. switch
- There are four forms of Loop constructs:
  1. for
  2. while
  3. do while
  4. for each (added in Java later)

Outline of Presentation

if and if...else statements

- Both if and if...else statements include boolean condition
- The simple if statement executes an action only if the boolean condition is true.
- The if...else statement includes two actions:
  1. for the boolean condition is true
  2. for boolean condition is false

Simple if statement

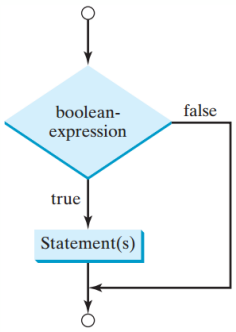

Figure : Flowchart of simple if

- The syntax for the simple if statement is as follows

```
1 if(booleanExpression){
2    statement(s)
3 }
```

- If booleanExpression evaluates as true, the statements inside the block are executed
- If the booleanExpression evaluates false, nothing happens

The if...else statement

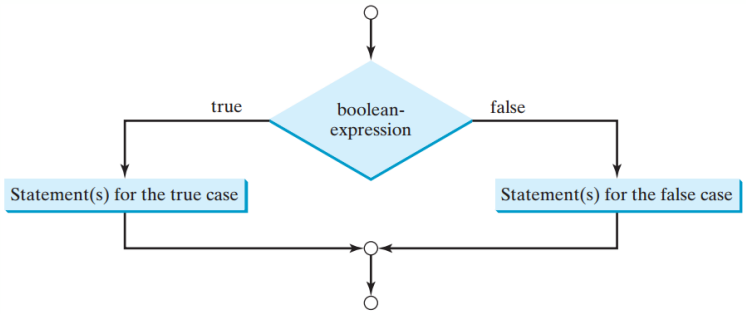

Figure : Flowchart of if...else

The if...else statement

- The if...else allows us to provide an alternative action.
- The syntax for if...else statements is as follows:

```
1 if(booleanExpression){
2    statement(s) for_true_case
3 }else{
4    statement(s) for_false_case
5 }
```


Program: Sessional Exam Status, Slide - I

```
1  class SessionalExamStatus{
2     public static void main(String[] args){
3        int sessional1 = 28;
4        int sessional2 = 2;
5        int sessional3 = 5;
6        int attendance = 3;
7        int average,total;
8        total = sessional1+sessional2+sessional3;
9        average = (int)(total/3);
10       System.out.println("Minimum 16 marks are
            required to pass sessional exam.");
11       if( ( (total/3.0)-(int)(total/3) ) >=
            0.5)
12          average +=1;
13       average += attendance;
14       if(average >= 16)
```


Program: Sessional Exam Status, Slide - II

```
15          System.out.println("Great! You passed
               sessional exam with "+average+"
               marks.");
16       else
17          System.out.println("Sorry! Your marks
               are "+average+". You did not pass
               sessional exam.");
18    }
19 }
```

```
D:\programs\CJT\programs\control>javac SessionalExamStatus.java

D:\programs\CJT\programs\control>java SessionalExamStatus
Minimum 16 marks are required to pass sessional exam.
Sorry! Your marks are 15. You did not pass sessional exam.
```

Nested if Statements

- The statements inside the if or the if...else statements can be any legal Java statements, including another if or if...else statements.
- The inner if statement is said to be nested inside the outer if statement.
- The inner if statement can contain another if statement. (there is no limit to the depth of the nesting)

Nested if Statements

- The nested if statement allows us to implement multiple alternatives.

```
if(score >= 90.0)
   grade = 'A';
else
   if(score >= 80.0)
      grade = 'B';
   else
      if(score >= 70.0)
         grade = 'C';
      else
         if(score >= 60.0)
            grade = 'D';
         else
            grade = 'F';
```


Nested if Statements

- The nested if statement can be written in the following equivalent structure.

```
if(score >= 90.0)
   grade = 'A';
else if(score >= 80.0)
   grade = 'B';
else if(score >= 70.0)
   grade = 'C';
else if(score >= 60.0)
   grade = 'D';
else
   grade = 'F';
```

- The above code is easy to read and can be seen as else..if ladder.

Shortcut if statement

- Suppose to a variable, we want to assign one value if the condition is true and another value if the condition is false, such as shown below:

```
if(booleanExpression)
   variable = true_result_expression;
else
   variable = false_result_expression;
```

- There is a shortcut syntax–ternary operator
- This syntax does not include if and else keywords

```
variable = condition ? expr_true : expr_false;
```


Program: Status based on SPI

```
class SpiAndStatus{
   public static void main(String[] args){
      float spi;
      String status="";
      spi = 8.0f;
      status = spi>=5 ? "Pass": "Fail" ;
      System.out.println("SPI: "+spi+" ,
         status: "+status);
      spi = 4.5f;
      status = spi>=5 ? "Pass": "Fail" ;
      System.out.println("SPI: "+spi+",
         status: "+status);
   }
}
```


Program: Status based on SPI

```
D:\programs\CJT\programs\control>javac SpiAndStatus.java

D:\programs\CJT\programs\control>java SpiAndStatus
SPI: 8.0 , status: Pass
SPI: 4.5, status: Fail
```

Using switch Statements

- The if statement allows us to write decisions based on a single condition
- The nested if statements can allow us to write multiple conditions and associated actions (beyond some point it becomes difficult to understand)
- Java provides switch statement to handle multiple conditions efficiently and with less code.

The switch statement

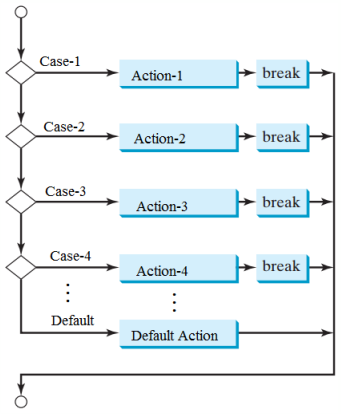

Figure : Flowchart of switch statement

The switch statement

- The syntax of switch statement is as follows:

```
switch(switch_expression){
   case value1: statement(s)1;
                break;
   case value2: statement(s)2;
                break;
   case value3: statement(s)3;
                break;
   case valueN: statement(s)N;
                break;
   default: statement(s) for_default;
}
```


The switch statement

- switch_expression must result in a value of char, byte, short, and int type
- The data type of case values must match with that of switch_expression and the values should be literals (cannot contain variable).
- Later Java added String data type for switch_expression.
- The break keyword is optional. If it is not present, the following case statement will be executed.
- The default case (must be the last) is optional. It will be executed if none of the cases is true.

Program on switch statement: Prepend Mr. or Ms.

```
class SwitchTest{
   public static void main(String[] args){
      String name="Harshad";
      String gender = "Male";
      switch(gender){
         case "Male":
            System.out.println("Mr. "+name);
            break;
         case "Female":
            System.out.println("Ms. "+name);
            break;
      }
   }
}
```


Program on switch statement: Prepend Mr. or Ms.

```
D:\programs\CJT\programs\control>javac SwitchTest.java

D:\programs\CJT\programs\control>java SwitchTest
Mr. Harshad
```

Program on switch statement: Test Odd or Even

```
1  class SwitchOddEven{
2     public static void main(String[] args){
3        int no=2;
4        switch(no){
5           case 1:
6           case 3: case 5:case 7: case 9:
7              System.out.println(no+" is odd");
8              break;
9           case 0: case 2: case 4: case 6:
10          case 8:
11             System.out.println(no+" is Even");
12             break;
13       }
14    }
15 }
```

- In this program, we have combined cases 1, 3, 5, 7, 9 and associated one action, same way an alternate action for cases 0, 2, 4, 6, and 8.

Program on switch statement: Test Odd or Even

```
D:\programs\CJT\programs\control>javac SwitchOddEven.java

D:\programs\CJT\programs\control>java SwitchOddEven
2 is Even
```


Outline of Presentation

Loop Constructs/Structures

- Loops are constructs or structures that control repeated execution of a statement or a block of statements.
- A loop construct has two structural parts:
  1. Loop head
  2. Loop body
- The part of the loop that contains the statements to be repeated is called the loop body.
- One-time execution of the loop body is called an iteration of the loop, which is controlled by loop continue condition.
- Loop continue-condition is a boolean expression and controls the execution of the body.
- After each iteration, the continue-condition is evaluated.
  - If the condition is true, the body is repeated
  - If the condition is false, the loop terminates.

Loop Constructs/Structures

- In general, any loop constructs have the following associated or essential parts:
  - Loop variable or variables
  - Initialization of Loop variable
  - Loop continue-condition
  - Modification in the value of Loop variable
- Depending upon where we can write these four associated parts, Java provides three types of loop constructs:
  - for loop
  - while loop
  - do-while loop

The for Loop

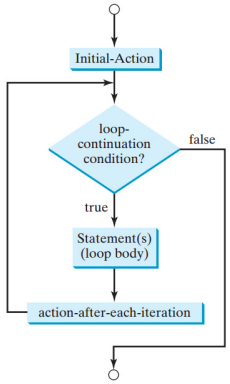

Figure : Flowchart of for loop

- The syntax for the for loop construct is as follows

```
1 for(initialization;
     continue_condition;
     loop variable update){
2    Loop body
3 }
```

- Example:

```
1 for(i=0; i<10; i++){
2    Loop body
3 }
```

Parts of the for Loop

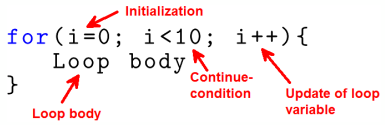

Figure : Four parts of for loop

- Generally, any for loop will have loop body, though it is optional.
- The other three parts are not required to be inside for loop head, i.e., inside round brackets of for.
- We can bring out Initialization and can place it before for loop.
- We can bring out update and can place it as part of Loop body
- We can even bring out condition and can place it as part of Loop body.
- Even, we can avoid all three parts: initialization, condition, and update.

Variations in Forming for Loop

An infinite for loop

```
1 for(;;){
2     Loop body
3 }
```

For loop with only condition in head

```
1 Initialization;
2 for(;condition;){
3     Loop body
4     Update (Loop Variable)
5 }
```

For loop with condition and update in head

```
1 Initialization;
2 for(;condition; update){
3     Loop body
4 }
```


Program: Separating random odd and even numbers

```
1  class RandomOddEvenNumbers{
2      public static void main(String[] args){
3          String oddNos="";
4          String evenNos="";
5          for(int i=0;i<10;i++){
6              int rNo = (int)(Math.random()*100);
7              if(rNo%2!=0)
8                  oddNos=oddNos+" "+rNo;
9              else
10                 evenNos=evenNos+" "+rNo;
11         }
12         System.out.println("Even Nos: "+evenNos);
13         System.out.println("Odd Nos: "+oddNos);
14     }
15 }
```


Program: Separating random odd and even numbers

```
D:\programs\CJT\programs\control>javac RandomOddEvenNumbers.java

D:\programs\CJT\programs\control>java RandomOddEvenNumbers
Even Nos:  76 60 78 60
Odd Nos:  63 89 67 85 63 25
```


Program: Nested for Loop

- It is possible to write one for loop as a statement inside another for loop
- It is called nesting. The first loop is called outer loop and the contained loop is called inner loop.

```
1  class TrianglePattern{
2      public static void main(String[] args){
3          int i=0,j=0;
4          for(i=0;i<5;i++){
5              for(j=0;j<=i;j++)
6                  System.out.print("* ");
7              System.out.println();
8          }
9      }
10 }
```


Program: Nested for Loop

```
D:\programs\CJT\programs\control>javac TrianglePattern.java

D:\programs\CJT\programs\control>java TrianglePattern
*
* *
* * *
* * * *
* * * * *
```

- We have used two loops:
  1. The outer loop is for repeating an action for multiple rows
  2. The inner for loop is the action for each row, which is to repeat another action for each column

The while Loop

- If we know how many times we need to repeat an operation, we can use for loop.
- If the number of repetitions is not known, the for loop is not useful.
- The while loop can handle unspecified number of repetitions.

The while Loop

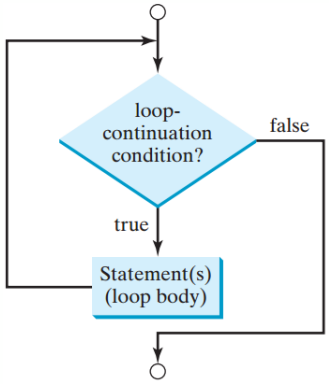

Figure : Flowchart of while loop

- The syntax of while loop is as follows:

```
while(condition){
   Loop body
}
```

- The condition is a boolean expression.
- If it evaluates to true, the body is executed/repeated.
- If it evaluates to false, the body is not executed and the control goes to the next statement that follows the while loop.

Program: Calculate Sum of Expenses

```
import java.util.*;
class SumExpenses{
   public static void main(String[] args){
      float dailyExpense=-1.0f;
      float totalExpense=0.0f;
      System.out.println("Enter expense amount
         or 0 (zero) to stop");
      Scanner input = new Scanner(System.in);
      while(dailyExpense!=0.0){
         System.out.print("Expense (float): ");
         dailyExpense = input.nextFloat();
         totalExpense += dailyExpense;
      }
      System.out.println("Your total expense
         is "+totalExpense);
   }
}
```


Program: Calculate Sum of Expenses

- We want to do summation of all daily expenses.
- But, we do not know in advance how many days expense occurred
- We use while loop, which we stop by entering 0 as expense amount (which is checked in the loop condition)

```
D:\programs\CJT\programs\control>javac SumExpenses.java

D:\programs\CJT\programs\control>java SumExpenses
Enter expense amount or 0 (zero) to stop
Expense (float): 50
Expense (float): 45.5
Expense (float): 15.50
Expense (float): 0
Your total expense is 111.0
```


The do-while Loop

- The do-while loop or do loop is a variation of the while loop

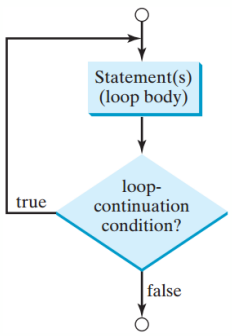

Figure : Flowchart of do-while loop

- The syntax of do-while loop is as follows:

```
do{
   Loop body
}while(condition);
```

- The body is executed first (at least once)
- If the condition evaluates to true, the body is executed/repeated again.
- If it evaluates to false, the do-while loop terminates.

Program: Do Sum of Expenses

```
import java.util.*;
class DoSumExpenses{
   public static void main(String[] args){
      float dailyExpense=0.0f;
      float totalExpense=0.0f;
      System.out.println("Enter expense amount
         or 0 (zero) to stop");
      Scanner input = new Scanner(System.in);
      do{
         System.out.print("Expense (float): ");
         dailyExpense = input.nextFloat();
         totalExpense += dailyExpense;
      }while(dailyExpense!=0.0);
      System.out.println("Your total expense
         is "+totalExpense);
   }
}
```

Program: Do Sum of Expenses

```
D:\programs\CJT\programs\control>javac DoSumExpenses.java

D:\programs\CJT\programs\control>java DoSumExpenses
Enter expense amount or 0 (zero) to stop
Expense (float): 50
Expense (float): 45.5
Expense (float): 15.50
Expense (float): 0
Your total expense is 111.0
```


Outline of Presentation

Use of break and continue

- Two statements: break; and continue; allow additional control to a loop construct.
  - **break** This keyword immediately ends the innermost loop that contains this break statement.
  - **continue** This keyword only ends the current iteration. Program control goes to the next cycle of the loop.

Program: Place Order of Items, Slide - I

```
1  class PlaceOrder{
2     public static void main(String[] args){
3        int itemCount=0;
4        int price=0;
5        int totalCost=0;
6        String itemPrices="";
7        System.out.println("Placing an order for
            maximum 10 items");
8        do{
9           price = (int)(Math.random()*5)*100;
10          if(totalCost+price <= 1000){
11             totalCost +=price;
12             itemPrices = itemPrices+price+" ";
13             itemCount++;
14          }else{
15             break;
16          }
17       }while(itemCount<10);
```


Program: Place Order of Items, Slide - II

```
18       System.out.println("You placed an order
            for "+itemCount+" items");
19       System.out.println("Individual prices of
            items are ");
20       System.out.println("     "+itemPrices);
21       System.out.println("Total Cost of items
            is "+totalCost);
22    }
23 }
```


Program: Place Order of Items, Slide - III

```
D:\programs\CJT\programs\control>java PlaceOrder
Placing an order for maximum 10 items
You placed an order for 7 items
Individual prices of items are
    100 0 300 0 100 0 300
Total Cost of items is 800

D:\programs\CJT\programs\control>java PlaceOrder
Placing an order for maximum 10 items
You placed an order for 4 items
Individual prices of items are
    0 400 0 400
Total Cost of items is 800

D:\programs\CJT\programs\control>java PlaceOrder
Placing an order for maximum 10 items
You placed an order for 4 items
Individual prices of items are
    100 200 300 400
Total Cost of items is 1000
```

Program: Order of Items of Non-zero Prices, Slide - I

- In our earlier program, when Math.random() method generates 0.0 value, we want to skip it.
- We can do that using continue; statement.

```
1  class PlaceOrderNonZeroItems{
2     public static void main(String[] args){
3        int itemCount=0;
4        int price=0;
5        int totalCost=0;
6        String itemPrices="";
7        System.out.println("Placing an order for
            maximum 10 items");
8        do{
9           price = (int)(Math.random()*5)*100;
10          if(price == 0)
11             continue;
12          if(totalCost+price <= 1000){
```


Program: Order of Items of Non-zero Prices, Slide - II

```
13             totalCost +=price;
14             itemPrices = itemPrices+price+" ";
15             itemCount++;
16          }else{
17             break;
18          }
19       }while(itemCount<10);
20       System.out.println("You placed an order
            for "+itemCount+" items");
21       System.out.println("Individual prices of
            items are ");
22       System.out.println("      "+itemPrices);
23       System.out.println("Total Cost of items
            is "+totalCost);
24    }
25 }
```


Program: Order of Items of Non-zero Prices, Slide - III

```
D:\programs\CJT\programs\control>java PlaceOrderNonZeroItems
Placing an order for maximum 10 items
You placed an order for 3 items
Individual prices of items are
     300 400 200
Total Cost of items is 900

D:\programs\CJT\programs\control>java PlaceOrderNonZeroItems
Placing an order for maximum 10 items
You placed an order for 5 items
Individual prices of items are
     200 200 100 200 200
Total Cost of items is 900

D:\programs\CJT\programs\control>java PlaceOrderNonZeroItems
Placing an order for maximum 10 items
You placed an order for 4 items
Individual prices of items are
     300 100 200 200
Total Cost of items is 800
```


Outline of Presentation

Comparison of Loops

- Java has three loops: for, while, and do-while.
- All these three loops can solve the given looping problem.
- Therefore, how to decide, which loop to use?
- There are two types of loops:

  Pretest loop Continuation condition is checked before the loop body is executed. Examples: for and while.

  Posttest loop Continuation condition is checked after the loop body is executed. Example: do-while.

Equivalent pretest loops

- Since for and while loops are of same type, i.e., pretest, one can be converted into another.
- In the following, while loop (first) can be converted into for loop (second)

```
1 while(continue_condition){
2    Loop body
3 }
```

```
1 for(; continue_condition ; ){
2    Loop body
3 }
```

Equivalent pretest loops

- In the following, for loop (first) can be converted into while loop (second)

```
for(initialization; continue_condition; loop
    variable update){
    Loop body
}
```

```
initialization;
while(continue_condition){
    Loop body
    loop variable update;
}
```


Equivalent pretest loops

- In the following three equivalent loops, while is better.

```
for( ; ; ){
    //Loop body
}
```

```
for(;true;){
    //Loop body
}
```

```
while(true){
    //Loop body
}
```


Which Loop to Use?

- We can see that both for and while are equivalent.
- One can be converted into another
- Which one to use for or while?
  - If the number of repetitions is known in advance, we should use for loop.
  - If the number of repetitions is not known in advance, we should use while loop.
- Which one to use while or do-while?
  - The while (pretest) and do-while (posttest) are of different type
  - The while loop can be replaced by do-while, if the loop body can be executed before continue_condition is evaluated.

Avoid Common Errors in Loops

- Do not put semicolon (;) at the end of for clause.
- The semicolon specifies that the loop has no body

```
for(i=1;i<=5;i++);
{
    Loop Body
}
```

- The above for loop is equivalent to the following for loop

```
for(i=1;i<=5;i++){ }
{
    Loop Body
}
```


Program: Promotion of Students

```
class ErroneousForLoop{
    public static void main(String[] args){
        int i=0;
        for(i=1;i<=5;i++);
        {
            System.out.println("Roll No "+i+" was
                regular during study");
        }
        for(i=1;i<=5;i++){}
        {
            System.out.println("Roll No "+i+" is
                promoted to next semester");
        }
    }
}
```


Program: Promotion of Students

- We want to print that Roll No 1 to 5 were regular in their studies and they are promoted to the next semester.
- But due to mistakes in the loop, we get erroneous output, as shown below.

```
D:\programs\CJT\programs\control>javac ErroneousForLoop.java

D:\programs\CJT\programs\control>java ErroneousForLoop
Roll No 6 was regular during study
Roll No 6 is promoted to next semester
```

Avoid Common Errors in Loops

- Similarly, do not put semicolon (;) at the end of while clause.
- The semicolon specifies that the loop has no body

```
i=1;
while(i<=5);
{
    Loop Body
}
```

- The above while loop is equivalent to the following while loop

```
i=1;
while(i<=5){}
{
    Loop Body
}
```


Program: Promotion of Students

```
class ErroneousWhileLoop{
   public static void main(String[] args){
      int i=1;
      while(i<=5);
      {
         System.out.println("Roll No "+i+" was
            regular during study");
         i++;
      }
      i=1;
      while(i<=5){}
      {
         System.out.println("Roll No "+i+" is
            promoted to next semester");
         i++;
      }
   }
}
```


Program: Promotion of Students

- We want to print that Roll No 1 to 5 were regular in their studies and they are promoted to the next semester.
- But due to mistakes in the loop, we do not get any output, as shown below.

```
D:\programs\CJT\programs\control>javac ErroneousWhileLoop.java

D:\programs\CJT\programs\control>java ErroneousWhileLoop
```

- In the while loop, we update loop variable inside loop body.
- As the loop body is never executed, the loop variable does not get updated
- Therefore condition ($i <= 5$) remains true forever, thus results in infinite loop.

The do while Loop needs Semicolon

- In for loop and while loop, semicolon is not needed after the clause.
- However, semicolon (;) is required at the end of while clause.

```
i=1;
do{
    Loop Body
}while(i<=5);
```

- If we forget to put semicolon (;) at the end of do-while loop, the compiler will detect it.
- However, if we forget to put semicolon (;) at the end of for loop or while loop, the compiler will not give any error.

The do while Loop needs Semicolon

```
class ErroneousDoWhileLoop{
   public static void main(String[] args){
      int i=1;
      do {
         System.out.println("Roll No "+i+" is
            promoted to next semester");
         i++;
      }while(i<=5)
   }
}
```

```
D:\programs\CJT\programs\control>javac ErroneousDoWhileLoop.java
ErroneousDoWhileLoop.java:7: error: ';' expected
            }while(i<=5)
                        ^
1 error
```


Summary of key terms

- Program control, decision, loop
- Decision constructs: if, if...else, nested if, switch
- Loop constructs: for, while, do-while, nested loop
- Additional loop control: break, continue
- Comparison of loops, equivalent loops, loop selection decision
- Errors in for loop, while loop, and do-while loop

References

- An Introduction to Java Programming, Y. Daniel Liang, PHI
- An Introduction to Java Programming, Y. Daniel Liang, Eigth Edition, Prentice Hall