Control Flow Statements

Sayed Ahmad Sahim

Benawa University

September 20, 2017

Road Map

1. Part 1
   - Operators in Java
2. Part 2
   - Control Flow Statements
3. Question?

Well known operators

- Arithmetic Operators
- Assignment Operators
- Unary Operators
- Relational Operators
- Logical Operators

Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

Example

```
class OperatorExampleArithmetic{
    public static void main(String args[]){
            int a=10;
            int b=5;
            System.out.println(a+b);
            System.out.println(a-b);
            System.out.println(a*b);
            System.out.println(a/b);
            System.out.println(a%b);
        }
}
```

Unary Operators

- Postfix
- Prefix

```
class OperatorExample{
    public static void main(String args[]){
        int x=10;
        System.out.println(x++);
        System.out.println(++x);
        System.out.println(x--);
        System.out.println(--x);
    }
}
```

Unary Operators

```
class OperatorExample{
    public static void main(String args[]){
        int a=10;
        int b=10;
        System.out.println(a++ + ++a);
        System.out.println(b++ + b++);

        }
}
```

Relational Operators:

The equality and relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand. The majority of these operators will probably look familiar to you as well. Keep in mind that you must use "==", not "=", when testing if two primitive values are equal.

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Bitwise Operators

The Java programming language also provides operators that perform bit-wise and bit shift operations on integral type.

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Example 1

```
class OperatorExample2{
    public static void main(String args[]){
        int a=10;
        int b=-10;
        boolean c=true;
        boolean d=false;
        System.out.println(~a);//-11 (minus of total positive value which starts from 0)
        System.out.println(~b);//9 (positive of total minus, positive starts from 0)
        System.out.println(!c);//false (opposite of boolean value)
        System.out.println(!d);//true
        }
}
```

Example 2

```
class OperatorExampleBitwise{
    public static void main(String args[]){
            System.out.println(10<<2);
            System.out.println(10<<3);
            System.out.println(20<<2);
            System.out.println(15<<4);

        }
}
```

Logical Operators

Logical Operators return boolean value of true or false.

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Assignment Operators

Assignment Operators are used to assign a value to a variable.

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Conditional Operator (? :)

- Conditional operator consists of three operands.
- After evaluating the boolean expression one of the value is assigned to the variable.
- variable x = (expression) ? value if true : value if false

```
public class Test {

   public static void main(String args[]){
      int a , b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

Type Conversion in Java

- When you assign value of one data type to another, the two types might not be compatible with each other.
- If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion.
- If not then they need to be casted or converted explicitly.
- For example, assigning an int value to a long variable.

Widening or Automatic Type Conversion

Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- When we assign value of a smaller data type to a bigger data type.
- For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean

Byte –> Short –> Int –> Long – > Float –> Double

Widening or Automatic Conversion

Example

```
class automaticTypeConversion
{
    public static void main(String[] args)
    {
        int i = 100;

        //automatic type conversion
        long l = i;

        //automatic type conversion
        float f = l;
        System.out.println("Int value "+i);
        System.out.println("Long value "+l);
        System.out.println("Float value "+f);
    }
}
```

Explicit Type Conversion

- If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing
  - This is useful for incompatible data types where automatic conversion cannot be done.
  - Here, target-type specifies the desired type to convert the specified value to.

Double –> Float –> Long –> Int –> Short –> Byte

Narrowing or Explicit Conversion

Example

```
//Java program to illustrate explicit type conversion
class ExplicitTypeConversion
{
    public static void main(String[] args)
    {
        double d = 100.04;

        //explicit type casting
        long l = (long)d;

        //explicit type casting
        int i = (int)l;
        System.out.println("Double value "+d);

        //fractional part lost
        System.out.println("Long value "+l);

        //fractional part lost
        System.out.println("Int value "+i);
    }
}
```

Decision Making Statements

Control Flow Statements

- Normally statements in Java source code is using top-bottom approach.
- Control flow statements breakup the flow of execution by using conditional statements.
- Control flow statements:
  - Decision Making
  - Looping
  - Branching

if-then Statement

- if-then statement is the most basic of all the control flow statements.
- It tells your program to execute a certain section of code only if a particular test evaluates to true.

```
void applyBrakes() {
    // the "if" clause: bicycle must be moving
    if (isMoving){
        // the "then" clause: decrease current speed
        currentSpeed--;
    }
}
```

if-then-else Statement

- if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false.
- You could use an if-then-else statement in the applyBrakes method to take some action if the brakes are applied when the bicycle is not in motion.

```
void applyBrakes() {
    if (isMoving) {
        currentSpeed--;
    } else {
        System.err.println("The bicycle has already stopped!");
    }
}
```

if...else if...else Statement:

- An if statement can be followed by an optional else if...else statement.
- which is very useful to test various conditions using single if...else if statement.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

```
public class Test {

   public static void main(String args[]){
      int x = 30;

      if( x == 10 ){
         System.out.print("Value of X is 10");
      }else if( x == 20 ){
         System.out.print("Value of X is 20");
      }else if( x == 30 ){
         System.out.print("Value of X is 30");
      }else{
         System.out.print("This is else statement");
      }
   }
}
```

Nested if...else Statement

- It is always legal to nest if-else statements.
- which means you can use one if or else if statement inside another if or else if statement.

```
public class Test {

   public static void main(String args[]){
      int x = 30;
      int y = 10;

      if( x == 30 ){
         if( y == 10 ){
            System.out.print("X = 30 and Y = 10");
         }
      }
   }
}
```

Switch

- Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths.
- A switch works with the byte, short, char, and int primitive data types.

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

        int month = 8;
        String monthString;
        switch (month) {
            case 1:  monthString = "January";
                     break;
            case 2:  monthString = "February";
                     break;
            case 3:  monthString = "March";
                     break;
            case 4:  monthString = "April";
                     break;
            case 5:  monthString = "May";
                     break;
            case 6:  monthString = "June";
                     break;
            case 7:  monthString = "July";
                     break;
            case 8:  monthString = "August";
                     break;
            case 9:  monthString = "September";
                     break;
            case 10: monthString = "October";
                     break;
            case 11: monthString = "November";
                     break;
            case 12: monthString = "December";
                     break;
            default: monthString = "Invalid month";
                     break;
        }
        System.out.println(monthString);
    }
}
```

Switch Example 2

```
public class Test {

   public static void main(String args[]){
      //char grade = args[0].charAt(0);
      char grade = 'C';

      switch(grade)
      {
         case 'A' :
            System.out.println("Excellent!");
            break;
         case 'B' :
         case 'C' :
            System.out.println("Well done");
            break;
         case 'D' :
            System.out.println("You passed");
         case 'F' :
            System.out.println("Better try again");
            break;
         default :
            System.out.println("Invalid grade");
      }
      System.out.println("Your grade is " + grade);
   }
}
```

Looping

Looping

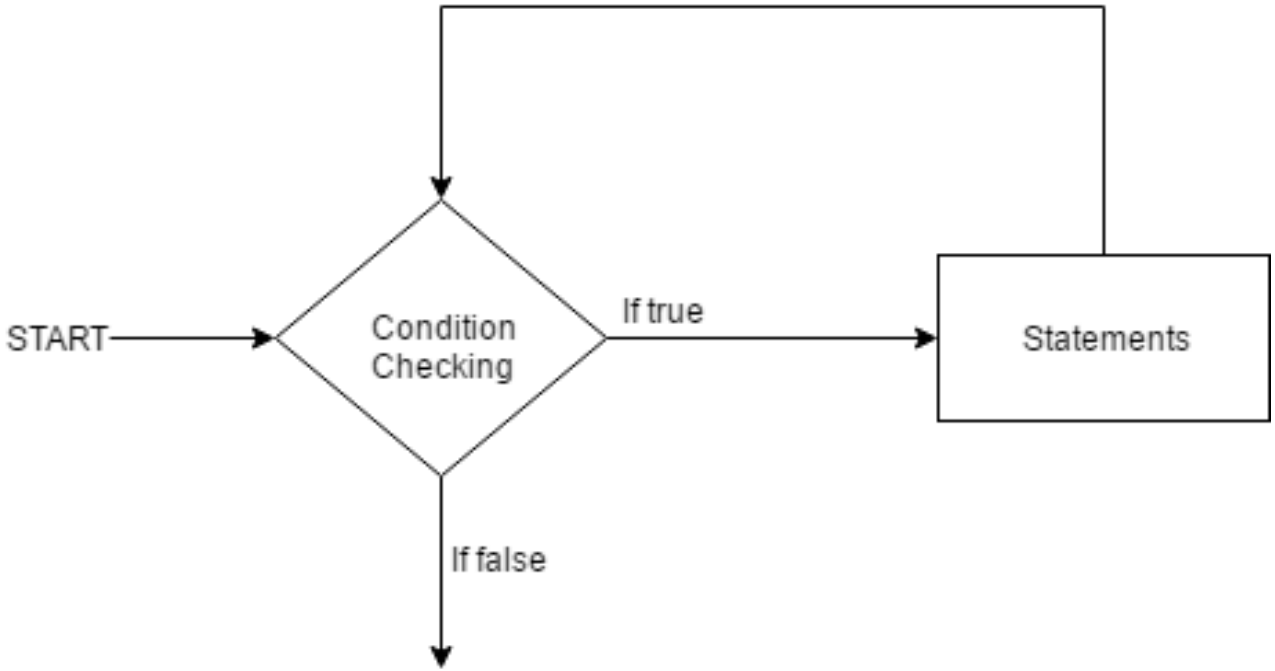
- Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.
- Java provides three ways for executing the loops:
  1. For
  2. while
  3. do-while

while loop:

- A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
- The while loop can be thought of as a repeating if statement.

```
class whileLoopExample{
    public static void main(String[] args)
    {
        while (boolean condition){
            loop statements...
        }
    }
}
```

While loop flow chart



Example

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

        int x = 1;

        // Exit when x becomes greater than 4
        while (x <= 4){
            System.out.println("Value of x:" + x);

            //increment the value of x for next itera
            x++;
        }
    }
}
```

for loop

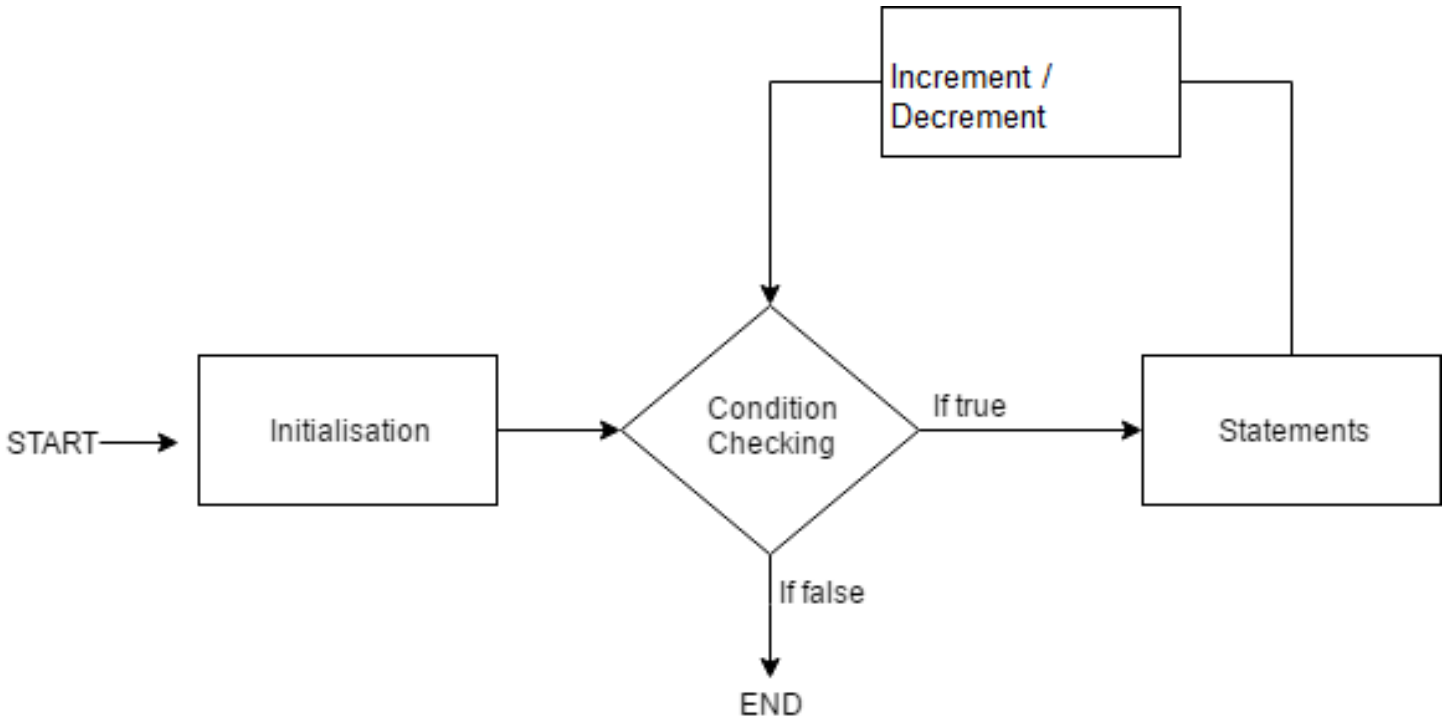
- for loop provides a concise way of writing the loop structure.
- Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

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

        for (initialization condition; testing condition; increment/decrement
            {
                statement(s)
            }
    }
}
```

for-loop flow chart



Example

```
class forLoopExample{
    public static void main(String args[])
    {
        // for loop begins when x=2
        // and runs till x <=4
        for (int x = 2; x <= 4; x++)
            System.out.println("Value of x:" + x);
    }
}
```

Enhanced for loop

- Java also includes another version of for loop introduced in Java 5.
- Enhanced for loop provides a simpler way to iterate through the elements of a collection or array.
- It is inflexible and should be used only when there is a need to iterate through the elements in sequential manner without knowing the index of currently processed element.
- Syntax:

```
for (T element:Collection obj/array)
{
    statement(s)
}
```

Example

- Print all the elements of an array with for and enhanced for loop.

```java
// Java program to illustrate enhanced for loop
public class enhancedForLoop
{
    public static void main(String args[])
    {
        String array[] = {"Ron", "Harry", "Hermoine"};

        //enhanced for loop
        for (String x:array)
        {
            System.out.println(x);
        }

        /*for loop for same function
        for (int i = 0; i < array.length; i++)
        {
            System.out.println(array[i]);
        }
        */
    }
}
```

do while

- do while loop is similar to while loop with only difference that it checks for condition after executing the statements.
- It is an example of Exit Control Loop.
- It is important to note that the do-while loop will execute its statements at least once before any condition is checked, and therefore is an example of exit control loop.
- Syntax:

```
do
{
    statements..
}
while (condition);
```

Example

- do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false

```
// Java program to illustrate do-while loop
class dowhileloopDemo
{
    public static void main(String args[])
    {
        int x = 21;
        do
        {
            //The line while be printer even
            //if the condition is false
            System.out.println("Value of x:" + x);
            x++;
        }
        while (x < 20);
    }
}
```

Pitfalls of Loops

- Infinite loop: One of the most common mistakes while implementing any sort of looping is that that it may not ever exit, that is the loop runs for infinite time.
- This happens when the condition fails for some reason.
- Syntax:

```
do
{
    statements..
}
while (condition);
```

Example 1

```java
//Java program to illustrate various pitfalls.
public class LooppitfallsDemo
{
    public static void main(String[] args)
    {

        // infinite loop because condition is not apt
        // condition should have been i>0.
        for (int i = 5; i != 0; i -= 2)
        {
            System.out.println(i);
        }
        int x = 5;

        // infinite loop because update statement
        // is not provided.
        while (x == 5)
        {
            System.out.println("In the loop");
        }
    }
}
```

Example 2

- Another pitfall is that you might be adding something into you collection object through loop and you can run out of memory.
- If you try and execute the below program, after some time, out of memory exception will be thrown.
- Syntax:

```java
//Java program for out of memory exception.
import java.util.ArrayList;
public class Integer1
{
    public static void main(String[] args)
    {
        ArrayList<Integer> ar = new ArrayList<>();
        for (int i = 0; i < Integer.MAX_VALUE; i++)
        {
            ar.add(i);
        }
    }
}
```

The continue Keyword

- The continue keyword can be used in any of the loop control structures.
- It causes the loop to immediately jump to the next iteration of the loop.

```
public class test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
          continue;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

The break Keyword

- The break keyword is used to stop the entire loop.
- It is always used inside any loop or a switch statement.
- The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

```
public class test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
          break;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```