BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 4

DCA2202

JAVA PROGRAMMING

Unit 3

Operators and Control Statements

Table of Contents

SL No	Topic		Fig No / Table / Graph	SAQ / Activity	Page No
1	Introduction		-	-	3
	1.1	Objectives	-	-	
2	Operators		1, 2, 3, 4, 5, 6, 7, 8	1	4 - 8
	2.1	Arithmetic Operators	-	-	
	2.2	Increment and Decrement Operators	-	-	
	2.3	Comparison Operators	-	-	
	2.4	Logical Operators	-	-	
	2.5	Operator Precedence	9,10,11,12,13,14	-	
3	Control Flow Statements		-	2	9 - 26
	3.1	If-else Statement	-	-	
	3.2	Switch Statement	-	-	
	3.3	For Loop	-	-	
	3.4	While Loop	-	-	
	3.5	Do...While Loop	-	-	
	3.6	Break Statement	-	-	
	3.7	Continue Statement	-	-	
4	Summary		-	-	27
5	Terminal Questions		-	-	27
6	Answers		-	-	27

1. INTRODUCTION

In the last unit, you have learnt the basic program structure in Java. In this unit, we will explore various operators and control statements in Java. There are many constructs in Java to implement conditional execution. For example, flow of a program can be controlled using the ***if*** construct. The ***if*** construct allows selective execution of statements depending on the value of the expressions associated with the ***if*** construct. Similarly, repetitive tasks can be done using ***for*** constructs which would be discussed in detail in this unit.

1.1 Objectives:

After studying this unit, you should be able to:

- *Explain different operators in Java*
- *Discuss various control statements in Java*

2. OPERATORS

In Java, the operator plays an important role. Three different kinds of the operator in Java are as follow:

i) Arithmetic Operators
ii) Unary Operators
iii) Comparison / Relational Operators and
iv) Logical Operators
v) Bitwise Operators

2.1 Arithmetic Operators

Addition, Subtraction, Multiplication, Division and Modulus are the various arithmetic operations that can be performed in Java. Table 3.1 lists different Arithmetic Operators.

Table 3.1: List of Arithmetic Operators

Operator	Meaning	Use	Meaning
+	ADD	op1+op2	Adds two or more operators.
-	SUBTRACT	op1-op2	Performs subtraction of two or moreoperators.
*	PRODUCT (MULTIPLY)	op1*op2	Performs multiplication between two ormore operators.
/	DIVISION	op1/op2	Performs division between two operators.
%	MODULOUS	op1 % op2	Calculates the reminder left in divisionbetween two operators.

The Java program in the figure 3.1 adds two numbers and prints the result.

```java
/* Program to add two numbers */
public class BasicExample1 {
      public static void main( String arg[]){
            int a = 10 ;
            int b = 20;
            int c;

            c= a+b;
            System.out.println("The sum of two numbers is "+c);
      }
}
```

Figure 3.1: Java Program to add two numbers and printing the result

Output:

The Sum of Two Number Is : 30

2.2 Unary Operators - Increment and Decrement Operators

In Java, ++ is called the increment operator; its function is to increment the value by 1; and -- is called the decrement operator; and its function to decrease the value by 1. Figure 3.3 shows an example of increment operators in Java.

```
/* Program to demonstrate pre and post increment */
public class SimpleJava4 {
      public static void main( String arg[]) {
            int x1=5;
            int x2=5;
            int y1, y2;

            // The assignment of values of x1 to y1 happens
            // before the increment to the variable x1
            // since this is a post increment.
            y1 = x1++;
            System.out.println("x1=" + x1+" y1=" + y1);

            // The increment operation is performed and x2
            // is updated before the value is assigned to
            // variable y2 since it is a pre-increment.
            y2 = ++x2;
            System.out.println("x2="+x2+" y2="+y2);
      }
}
```

Figure 3.3: Example showing increment operators in Java

Output:

```
x1 = 6  y1 = 5

x2=6  y2 = 6
```

When the operator ++ is placed after the variable name, first the assignment of the value of the variable is done; then the value of the variable is incremented by 1. This operation is also called ***post-increment***. Therefore, the value of y1 will remain at 5 and the value of x1 will be 6. When the operator is placed before the variable, the increment of the variable takes place first and then the assignment occurs. Hence the value x2 and y2 both will be 6. This operation is also called ***pre-increment***. Similarly -- operator can be used to perform ***post-decrement*** and ***pre-decrement*** operations. If there is no assignment and only the value of the variable has to be incremented or decremented, then placing the operator after or before does not make difference.

2.3 Comparison Operators

In Java, to compare the values of two or more operators, comparison operators are used. Table 3.2 lists various Comparison Operators in Java.

Table 3.2: List of Comparison Operators in Java

Operator	Meaning	Example	Remarks
= =	Equal	op1 = = op2	Checks if both the operators are equal or not.
!=	Not Equal	op1 != op2	Checks if the operators are not equal to each other or not.
<	Less than	op1 < op2	Checks if the first operator is lesser than the other operator or not.
>	Greater than	op1 > op2	Checks if the first operator is greater than the other operator or not.
<=	Less than or equal	op1 <= op2	Checks if the operator is lesser than or equal to the other operator.
>=	Greater than or equal	op1 >= op2	Checks if the first operator is greater than or equal to the other operator.

2.4 Logical Operators

In Java, to perform ‘Boolean’ operations on operands we use Logical operators. Table 3.3 lists various Logical Operators in Java.

Table 3.3: List of Logical Operators in Java

Operator	Meaning	Example	Remarks
&&	Short-circuitAND	op1 &&op2	Returns TRUE if and only if both the operands are TRUE.
||	Short-circuit OR	op1 || op2	Returns true if any of the two operators are TRUE.
!	Logical unaryNOT	!op	Return compliment of the value of the operands. For example if the operand value is TRUE it will return FALSE.

2.5 Bitwise Operators

Operator	Meaning	Example	Remarks
~	Bitwise complement	~op1	Returns the bitwise complement of op1.
<<	Left Shift	op1<< op2	Shits the position of the bits in op1 to the left op2 times.
>>	Right Shift	op1>>op2	Shits the position of the bits in op1 to the right op2 times.
&	Bitwise AND	op1&op2	Returns the bitwise AND of op1 and op2
|	Bitwise OR	op1 | op2	Returns the bitwise OR of op1 and op2
^	Exclusive OR	Op1 ^op2	Returns bit wise exclusive OR of op1 and op2.

2.6 Operator Precedence

When more than one operator is used in an expression, Java will use the operator precedence rule to determine the order in which the operators will be evaluated. For example, consider the following expression:

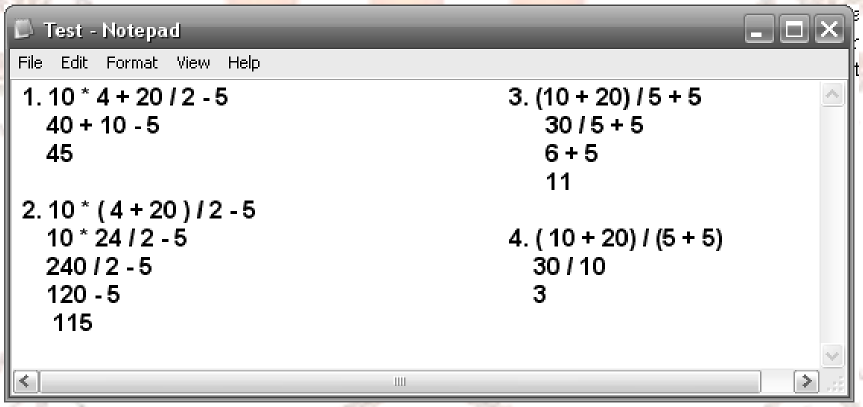
Result=10+5*8-15/5

In the above expression, multiplication and division operations have higher priority over addition and subtraction. Hence, they are performed first. Now, Result = 10+40-3.

In Java the operators are evaluated from left to right in the order they appear in the given expression if in case they have the same priority. Hence the value of the result will become 47. In general, the following priority order is followed when evaluating an expression:

- Increment and decrement operations.
- Arithmetic operations.
- Comparisons.
- Logical operations.
- Assignment operations.

To change the order of evaluation of any given expression we use parenthesis. The expression within the parenthesis will be evaluated first. When the parentheses are nested together, the expressions in the innermost parentheses are evaluated first. Parentheses also improve the readability of the expressions. When the operator precedence is not clear, parentheses can be used to avoid any confusion. The examples of operator precedence have been shown in the figure 3.5.



```
1. 10 * 4 + 20 / 2 - 5                  3. (10 + 20) / 5 + 5
   40 + 10 - 5                              30 / 5 + 5
   45                                       6 + 5
                                            11
2. 10 * ( 4 + 20 ) / 2 - 5
   10 * 24 / 2 - 5                      4. ( 10 + 20) / (5 + 5)
   240 / 2 - 5                             30 / 10
   120 - 5                                 3
    115
```

Figure 3.5: Operator Precedence Example

Self-Assessment Questions - 1

1. Give the symbol for the modulus operator.
2. Give the symbol for the logical AND operator.

3. CONTROL FLOW STATEMENTS

To control the flow of execution in a program the following statements are used:

1. Decision-Making Statements
   - If-else statement
   - Switch – case statement
2. Looping Statements
   - For loop
   - While loop
   - Do-while loop
3. Other statements
   - Break
   - Continue

3.1 If-Else Statement

IF statement is a conditional branch statement that can be used to route the program execution through two different paths. The syntax for the 'if' statement is as follows:

if (condition) statement1;

else statement2;

In the above syntax statement1 and statement2 can be a single statement or a group of statements, in the case of group statements, it is written within curly brackets { }. The ***condition*** is an expression that returns a Boolean value based on which the choice between the two statements are made. The ***else*** part is optional and can be left also in case the program doesn't need it.

Working of ***If statement*** is as follow: if the condition is TRUE then in the above syntax, statement1 is executed; the statement1 part is called the TRUE part, and that block is executed only when the condition is TRUE i.e. value 1 is returned; and if the condition is FALSE, i.e. condition returns 0, then else part is executed, i.e. statement2 is executed. The else part is the False block of if-else statement which is executed when the condition is FALSE. An example of If… else statement shown in the figure 3.6.

Figure 3.6: If… else statement example

```
/* This is a simple program to demo the working of if-else */
public class IfElseExample1 {
       public static void main( String arg[]) {
              int mark=45;
              if( mark < 35) {
                     System.out.println ("The student has failed");
              } else {
                     System.out.println("The student has passed");
              }
       }
}
```

Relational operators are mostly used to specify the conditions but we can also use a single Boolean variable also. The code given below shows the working of an ***if*** statement:

```
boolean singleValue;
// ...
if (singleValue)
        Perform1();
else
     WaitForCorrectData();
```

In the above example, we have only used a single value (Boolean type) for the condition check and each of the TRUE and FALSE blocks has just one statement.

We will see one more example where we will see how to include more than one statement in the TRUE and FALSE block.

```
int checkCondition;
// ...
if (checkCondition > 0) {
            Perform1();
```

```
                    Perform2();
                    Perform3();
        } else
        {
                    Perform4();
                    Perform5();
        }
```

Here, both statements within the **if** block will execute more than one statement. The statements are included with curly brackets { }. One set of

{ } brackets forms one block. Here if we forget to define the block then it will cause error.

For example, consider the following code fragment:

```
int bytesAvailable;
// ...
if (bytesAvailable > 0) {
            ProcessData();
           bytesAvailable -= n;
} else
          waitForMoreData();
         bytesAvailable = n;
```

It seems clear that the statement **bytesAvailable = n**; was intended to be executed inside the **else** clause, because of the indentation level. However, as you recall, whitespace is insignificant to Java, and there is no way for the compiler to know what was intended. This code will compile without complaint, but it will behave incorrectly when run.

The preceding example should have been written as follows:

```
  int bytesAvailable;
  // ...
if (bytesAvailable > 0) {
        ProcessData();
        bytesAvailable -= n;
 } else {
        waitForMoreData();
        bytesAvailable = n;
}
```

The if-else-if Ladder

When there are more than one condition checks needed and the condition occurs in a sequence or based on the result of one condition there are other sets of conditions or condition that needs to be checked for then we use if-else-if ladder. It is basically nesting of if-else statement where we use an if statement within another if statement. The example given below explains further the use of nested if-else statement:

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

The **if** statements follow top-down approach. In the above demonstration of nested if-else as soon as the condition of the first if statement is TRUE then the statement in the TRUE part is executed. In case the statement is FALSE it goes to 'else' part where it encounters another if statement and the whole process is repeated.

The below program demonstrates the use of nested if-else statement. The following program prints the type of month it is based on the month number:

```
/* Demonstration nested if-else statements. */

class IFElseExample2 {
      public static void main(String args[]) {
            int month_number = 4; // April
            String season;
            if (month_number == 12 ||
                        month_number == 1 ||
                        month_number == 2)
                  season = "Winter";
            else if (month_number == 3 ||
                        month_number == 4 ||
                        month_number == 5)
                  season = "Spring";
            else if (month_number == 6 ||
                        month_number == 7 ||
                        month_number == 8)
                  season = "Summer";
            else if (month_number == 9 ||
                        month_number == 10 ||
                        month_number == 11)
                  season = "Autumn";
            else
                  season = "Bogus month_number ";
            System.out.println("April is in the "
                  + season + ".");
      }
}
```

Output:

```
April is in the Spring.
```

In the following program you can change the month_number and check the output for the purpose of learning and better understanding of the nested if else statement.

3.2 Switch Statement

The **switch** statement in Java provides multi-way branching. Multi-way means that by using one condition it can transfer the program control to a different set of instructions for execution. Switch statement proves to be a better alternative to nested if-else, we do not have to write the if statement multiple times.

Syntax of switch statement:

```
switch (expression) {
        case value1:
            // statement sequence
            break;
        case value2:
            // statement sequence
            break;
        .
        .
        .
        case valueN:
            // statement sequence
            break;
        default:
            // default statement sequence
}
```

The *expression* in the above-mentioned syntax for the switch statement must be of a valid type: int, float, etc. The data type of values of case (value1, value2,..) should be the same as that of the expression. The value of cases must be unique, these values must be constant as we cannot use variables. The break statements are not compulsory. However, if we do not give a break statement then once the execution command is switched to the case which holds true, the cases falling after it will also be executed. So, to avoid this we use a break statement.

The switch statement works as follows: once the command enters the switch statement it checks the condition. After it checks the condition the value is matched with values following the keyword 'case'. If matched, the execution command is shifted to that case. The statements within the case are executed. If you use a 'break' statement, then after execution of the statements in the matching case, the command jumps out of the switch statement. In case you do not use a break statement then the statements within the next case are also executed. An example of switch…case statement is shown in the figure 3.7.

For example, consider the following program:

```
public class SwitchCase {
	public static void main(String args[]) {
		int weekday = 3;
		switch(weekday) {
			case 1: System.out.println("Sunday");break;
			case 2: System.out.println("Monday");break;
			case 3: System.out.println("Tuesday");break;
			case 4: System.out.println("Wednesday");break;
			case 5: System.out.println("Thursday");break;
			case 6: System.out.println("Friday");break;
			default: System.out.println("Saturday");
		}
	}
}
```

Figure 3.7: The switch...case statement example

```
// In a switch, break statements are optional.
class NoBreak {
	public static void main(String args[]) {
		for (int i = 0; i < 12; i++)
			switch (i) {
			case 0:
			case 1:
			case 2:
			case 3:
			case 4:
				System.out.println("i is less than 5");
				break;
			case 5:
			case 6:
			case 7:
			case 8:
			case 9:
				System.out.println("i is less than 10");
				break;
			default:
				System.out.println("i is 10 or more");
			}
	}
}
```

Output:

```
i is less than 5
i is less than 5
i is less than 5
i is less than 5
i is less than 5
i is less than 10
i is less than 10
 i is less than 10
i is less than 10
i is less than 10
 i is 10 or more
i is 10 or more
```

Nested switch Statements

A switch statement can be used within another switch statement. This procedure is called **nested switch statement**. The following example shows the working of nested switch statement:

```
switch(count) {
case 1:
switch(target) {
                // nested switch
        case 0:
                System.out.println("target is zero");
                break;
        case 1: // no conflicts with outer switch
                System.out.println("target is one");
                 break;
   }
   break;
case 2: // ...
```

In the above example, the case 1 of inner switch do not conflict with the case 1 of outer switch. Once the outer switch shifts the command control to case 1 then inner switch is executed. You must not get confused with the case 1 of inner switch with the case 1 of outer switch.

In summary, there are three important features of the **switch** statement to note:

- The **switch** statement only checks equality compared to if else statement where other relational operations can also be performed.
- In same **switch** you cannot have two cases with similar value, however you must not get confused with inner and outer switch as they different and work differently, their cases can have same value as they are part of two distinct switch statements.
- A **switch** statement are often more efficient compared to nested if statements.

When the Java compiler encounters a **switch** statement it creates a jump table and based on this table and expression of the switch statement compiler jumps the execution to appropriate cases that comply with the expression. Therefore, if we have to select among a large pool of values, switch statements prove to be faster and more efficient compared to **nested** if statements. In the case of nested-if, the compiler does not have any pre-information of the long set of relational operations formed in a nested **if-else** statement.

3.3 For Loop

Syntax of **for** loop is as follow:
for (initial statement; termination condition; increment/decrement instruction) statement;

In a for loop, the initialization of the variable used for condition checking, the termination condition, and the increment or decrement statement needed to change the value of the variable initialized are included in the same statement. They are separated by the semicolon ';'. When the command enters a **for loop** the initialization statement is executed first and then the command shift to the condition part. If the condition is **TRUE**, then the statement included within the 'for loop' is executed. Once execution is completed, the increment/decrement statement in the for-loop statement is executed. The value is now updated. The condition statement in the for-loop statement is checked again. If the condition is **TRUE**, execution is repeated and if the condition becomes **FALSE**, then the control exits the loop. Remember that the initialization statement is executed only once at the start of the **for-loop**.

When multiple statements are to be included in the **for** loop, the statements are included inside flower braces as below:

```
for (initial statement; termination condition; increment instruction)
{
      statement1;
       statement2;
}
```

The example in figure 3.8 prints numbers from 1 to 10.

```
/* This is a simple program to demo the working of for-loop */
public class ForLoopExample1 {
        public static void main( String arg[]) {
                int i;
                for(i=1;i<=10;i++) {
                        System.out.println(i);
                }
        }
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

```
/* This program demonstrates the working of Nested Loop. */
class Nested {
      public static void main(String args[]) {
            int i, k;
            for (i = 0; i < 5; i++) {
                  for (k = i; k < 5; k++)
                        System.out.print("*");

                  System.out.println();
            }
      }
}
```

In Java **for** loop can be nested too, i.e. you can use a for loop within another for loop. Following example shows the working of a **nested for** loop:

Output:

```
*****
****
***
**
*
```

Enhanced for statement: 'Enhanced' for loops can be used to iterate through collections and arrays. The use of 'Enhanced for statement' makes loops more compact and easier to understand. Though we will study arrays in the next unit,for demonstrating the use of enhanced for statement, we have used the same in the example given below:

Example:

```
class EnhancedForDemo {
	public static void main(String[] args) {
		int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
		for (int item : numbers) {
			System.out.println("Count is: " + item);
		}
	}
}
```

In this example, the variable item holds the current value from the numbers array.

Output:

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```

It is recommended to use this enhanced form of **for** statements wherever it is possible to use.

3.4 While Loop

While statement, like the **for** statement, is used to create a loop based on condition. As long as the condition is **TRUE** the block of the statement (body of loop) within the **while** statement is executed repeatedly. Unlike **for** we just have the condition expression or statement associated with the while statement. Here is its general form:

```
 while (condition) {
// body of loop or while block
 }
```

```java
/* This program calculates the factorial of a number */
public class Factorial {
	public static void main ( String args[] ){
		int n=5;
		int fact=1;
		int i=1;

		while (i<=n) {
			fact *= i;
			i++;
		}
		System.out.println("Factorial of "+n+" is "+fact);
	}
}
```

Output:

```
Factorial of 5 is 120
```

3.5 Do…. While Loop

Do While Loop serve the same purpose as other loop statement but the operation is a bit different than the other loops statements mentioned earlier. Here the statement is executed first and then the condition is checked, unlike other two loop where the condition was checked during the beginning.

do While loop is exit control loop where as other are entry control loop. Sometimes it is required the loop's body has to be executed at least once before the command exits the loops, in such cases do While loop is very useful. Syntax of **do While loop**:

```
do {
 // body of loop
} while (condition);
```

It’s clear from the syntax that once the program control enters a **do while** loop then the body of the loop is executed before it checks the condition. If the condition is **TRUE**, then the body of the loops is executed and then the condition is checked. This process is repeated as long as the condition is **TRUE**. Program control exits the loop if the condition in the **while** part becomes **FALSE**. Following example in the figure 3.11 shows the working of **do while** loop:

```
/* This program calculates the factorial of a number */
public class Factorial {
	public static void main ( String args[] ){
		int n=5;
		int fact=1;
		int i=1;

		do {
			fact *= i;
			i++;
		} while (i<=n)
		System.out.println("Factorial of "+n+" is "+fact);
	}
}
```

Figure 3.11: Do...while loop example

Output:

Factorial of 5 is 120

The **do-while** loop is useful for programs where we have to create some Menu Selection List, as in a Menu Selection the Menu has to be executed at least once before the user can enter choice. Consider the following program which implements a very simple help system for Java's selection and iteration statements:

```
// Using a do-while to process a menu selection
class Menu {
        public static void main(String args[]) throws java.io.IOException {
                char choice;
                do {
                        System.out.println("Help on:");
                        System.out.println(" 1. if");
                        System.out.println(" 2. switch");
                        System.out.println(" 3. while");
                        System.out.println(" 4. do-while");
                        System.out.println(" 5. for\\n");
                        System.out.println("Choose one:");
                        choice = (char) System.in.read();
                } while (choice < '1' || choice > '5');
                System.out.println("\\n");
                switch (choice) {
                case '1':
                        System.out.println("The if:\\n");
                        System.out.println("if(condition) statement;");
                        System.out.println("else statement;");
                        break;
                case '2':
                        System.out.println("The switch:\\n");
                        System.out.println("switch(expression) {");
                        System.out.println(" case constant:");
                        System.out.println(" statement sequence");
                        System.out.println(" break;");
                        System.out.println(" // ...");
                        System.out.println("}");
                        break;
                case '3':
                        System.out.println("The while:\\n");
                        System.out.println("while(condition) statement;");
                        break;
                case '4':
                        System.out.println("The do-while:\\n");
                        System.out.println("do {");
                        System.out.println(" statement;");
                        System.out.println("} while (condition);");
                        break;
```

```
            case '5':
                System.out.println("The for:\\n");
                System.out.print("for(init; condition; iteration)");
                System.out.println(" statement;");
                break;
            }
        }
}
```

Output:

Help on:

1. if
2. switch
3. while
4. do-while
5. for

Choose one:

The do-while:

do {

statement;

} while (condition);

In the above program, the first **do-while loop** is used to check for the correct input. Check the condition part, until the given input "choice" is not valid the program control doesn't exit the loop. Once the correct input is provided, we use a **switch** statement to perform the action based on the choice made.

3.6 Break Statement

Break statement is used to immediately terminate the loop. When the program control encounters a **break** statement it jumps out of the loop. Here is a simple example:

```
/* Program to demonstrate break statement */
class BreakStmtDemo {
      public static void main(String args[]) {
            for (int i = 0; i < 50; i++) {
                  if (i == 5)
                        break; // terminate loop if i is 10
                  System.out.println("i: " +  i);
            }
            System.out.println("Loop complete.");
      }
}
```

Output:

```
i: 0
i: 1
i: 2
i: 3
i: 4
Loop complete.
```

The **for** loop is made to execute until the value of **i** reaches **49** however the loop breaks when the value of **i** is equal to **5**. Use of **break** statement leads to the termination of the loop when **i** is equal to **5**.

3.7 Continue Statement

Sometimes in some programs, we need to bypass immediate code and jump back to the loop i.e control should be transferred to the condition statement of the **while** and **do while** loop and increment/decrement part in **for** loop; in such cases, we use **continue** statement.

The following program demonstrate the use of **continue** statement:

```
/* Program to demonstrate the working of
 * continue statement */

class ContinueStmtDemo {
	public static void main(String args[]) {
		for (int j = 0; j < 10; j++) {
			System.out.print(j + " ");
			if (j % 2 == 0)
				continue;
			System.out.println("");
		}
	}
}
```

This code uses the % operator to check if j is even. If it is, the loop continues without printing a newline.

Output:

```
0 1
2 3
4 5
6 7
8 9
```

As with the **break** statement, **continue** may specify a label to describe which enclosing loops to continue.

Self-Assessment Questions - 2

3. If-else is a looping statement. True or False.
4. Do…While is a decision making statement. True or False.

4. SUMMARY

This unit has given a brief overview of various operators and conditional statements in Java. Let us summarize the concepts:

- **Implementing Conditional Execution**

  You can control the flow of a program using the **if** construct. The **if** construct allows selective execution of statements depending on the value of the expressions associated with the **if** construct.

- **Performing Repetitive Tasks Using Loop Construct**

  You can perform repetitive tasks by making use of the for construct.

5. TERMINAL QUESTIONS

1. What are the different types of operators used in Java?
2. What do you understand by operator precedence?
3. What are the different types of control statements?
4. Write Java program to print the address of the study center.
5. Write Java program to convert the Rupees to Dollars.
6. Write a Java program to compare whether your height is equal to your friends height.
7. Write a Java program to find the sum of 1+3+5+…. for 10 terms in the series.

6. ANSWERS

Self-Assessment Questions:

1. %
2. &
3. False
4. False

Terminal Questions

1. Arithmetic, Comparison and Logical Operators. (Refer Section 2.1 to 2.4)
2. When more than one operator is used in an expression, Java will use operator precedence rule to determine the order in which the operators will be evaluated. (Refer Section 2)
3. If…else, Switch…Case, For, While, Do…While, Break, and Continue.(Refer Section 3.1 to 3.7)

   (Terminal Questions 4 to 7 are programming exercises. For this, you should go through this unit thoroughly.)