2.5.3 Vectors

Vector implements a dynamic array. It is similar to ArrayList but with two differences; vector is synchronized and it contains many legacy methods. The methods are as follows:

- `Vector ( )`
- `Vector (int Size)`
- `Vector (int size, int incr)`

The form creates a default vector which has an initial size of 10. The second form creates a vector whose initial capacity is specified by size. The third form creates a vector whose initial capacity is specified by size and whose increment is specified by incr.

Vector defines these protected data members:

```
int capacityIncrement;
int elementCount;
object elementData[ ]k;
```

Some important methods are:

- public void addElement (object Element)
- final int capacity( )
- final Object elementAt(int index)
- final int indexOf(object element)
- final object firstElement( )
- final void InsertElementAt(object element, int index)
- final void removeElementAt(int index)
- final void setSiz(int Size)
- String toString( )

2.5.4 Array Declaration Syntax

There is a second form that may be used to declare an array:

```
type [ ] varname;
int a1[ ] = new int [3];
int [ ] a2 = new int [3]
```

Check Your Progress

State whether the following statements are true or false:

1. Variables are not the identifiers that are used to store a data value.
2. The simplest method of giving value to a variable is through the assignment statement.
3. Vector does not implement a dynamic array.
4. A one dimensional array is essentially a list of like typed variables.

2.6 LET US SUM UP

Variables are the identifiers that are used to store a data value. A variable may take different values at different times during the execution of the program. Identifiers are the names of variables. They must be composed of letters, numbers, the underscore and the dollar sign ($). They cannot contain white spaces, identifiers may only begin with a letter, the underscore or a dollar sign. There are three kinds of variables in Java - instance variables, class variables, local variables. Java does not have global variables (which can be used in all parts of a program). Datatypes which are built with the help of primitive data types but are not actually primitive are known as non primitive datatypes. An int divided by an int is still an int and a double divided by a double is still a double. But what about an int divided by double or a double divided by an int? When doing arithmetic on unlike types, Java tends to widen the types involved so as to avoid loosing information.

2.7 KEYWORDS

Variables: The identifiers that are used to store a data value.

Literals: Pieces of Java source code that indicates explicit values.

String literal: Always enclosed in double quotes.

Characters: A special set. They can be treated as either a 16 bit unsigned integer with a value from 0-65535 or as a unicode character.

Array: An array is a group of like typed variables that are referred to by a common name.

2.8 QUESTIONS FOR DISCUSSION

1. Explain how variables can be declared in Java and the scope of variables.
2. Differentiate between primitive and non-primitive datatypes.
3. What is type conversion and casting?
4. Explain multi-dimensional arrays.

Check Your Progress: Model Answers

1. False
2. True
3. False
4. True

2.9 SUGGESTED READINGS

E. Balaguruswami, *Programming with Java*, Tata McGraw-Hill.

Davis, Stephen R., *Learn Java Now*, Microsoft Press.

Naughton, Patrick, *The Java Hand Book*, OSborne McGraw-Hill.

Sams.net, Java unleased.

Herbert Schildt, *The Complete Reference Java 2*, Tata McGraw-Hill.

LESSON

3

OPERATORS

CONTENTS

3.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Understand various operators and their uses
- Explain bitwise, relation, Boolean logic, ternary operations

3.1 INTRODUCTION

Java provides a rich operator environment. Most of its operators can be divided into the following four groups: arithmetic, bitwise, relational and logical. Java also defines some additional operators that handle certain special situations. Java also uses control statements to cause the flow of execution to advance and branch based on changes in the state of a program.

3.2 BIT-WISE OPERATORS

To manipulate the data at values of bit level, java supports bit-wise operators. Java defines several bit wise operators which can be applied to the integer types, long, int, short, char and byte. They are summarized in Table 3.1.

Table 3.1: Bit-wise Operators

Operator	Result
~	Bit wise unary NOT
&	Bit wise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill/ Unsigned Right Shift
<<	shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment

3.3 RELATIONAL OPERATORS

Relational operators are binary operators that require two operands to work with. The operands can be either constants or variables or expressions. The operators are as follows:

Table 3.2: Relational Operators

>	greater than
> =	greater than or equal to
<	less than
< =	less than or equal to
!	Boolean not
! =	not equal to

3.4 BOOLEAN LOGICAL OPERATORS

The Boolean logical operators shown here operate only on Boolean operands. All of the binary logical operators combine two Boolean values to form a resultant Boolean value.

Table 3.3: Boolean Logical Operators

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
| |	Short Circuit OR
&&	Short Circuit AND
!	Logical Unary Not
& =	AND ASSIGNMENT
| =	OR ASSIGNMENT

Contd....

^ =	XOR Assignment
= =	Equal To
! =	Not Equal To
? :	Ternary if then else

3.5 TERNARY OPERATORS

Java includes a special ternary (three-way) operator that can replace certain types of, if then else, statements. This operator works in Java much like it does in C and C++. It has this general form

```
expression1 ? expression2 ? : expression3
class Ternary {
public static void main (String args[ ])
  {
  int i, k;
  i = 10;
  k = i < 0 ? -i : i; // get absolute value of i
  System.out.println ("Absolute value of");
  System.out.println (i + "is" + u);
  i = -10;
  k = i < 0 ? -i : i;
  System.out.print ("Absolute value of");
        System.out.println (i + "is" + u);
  }
}
        class Boollogic {
  public static void main (String s[ ])
  {
  boolean a = true;
  boolean b = false;
boolean c = a | b;
```

3.6 ARITHMETIC OPERATORS

Arithmetic operators are used in mathematical expressions in the same way as they are used in Algebra. The Table 3.4 lists the Arithmetic operators.

Table 3.4: Arithmetic Operators

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
+ =	Addition Assignment
+ =	Subtraction Assignment
- =	Multiplication Assignment
* =	Division Assignment
% =	Modulus Assignment
- -	Decrement

Basic Assignment Operators are listed in the table below:

Table 3.5: Basic Assignment Operators

=	Assignment
^ =	bitwise XOR and assign
& =	bitwise AND and assign
% =	take remainder and assign
- =	Subtract and assign
* =	Multiply and assign
/ =	bitwise OR and assign
>> =	Shift bits right with sign extension and assign
<< =	Shift bits left and assign
>>> =	unsigned bit shift right and assign

Check Your Progress

Fill in the blanks:

1. To manipulate the data at values of bit level, java supports .....
2. The ..... operators shown here operate only on Boolean operands.
3. Java includes a special ternary operator that can replace certain types of, ..., statements.

3.7 LET US SUM UP

In this lesson we have discussed all the basic data types and operators available in Java and understand their use in expressions. Type conversion and order of precedence of operators during the evaluation

of expressions have been highlighted. Various mathematical functions are also discussed which are available in the Math class of Java. Note that the Java types have fixed sizes. There is no ambiguity and all Java types are machine-independent.

3.8 KEYWORDS

Argument: A value that is sent to a method when the method is called.

Assignment Expression: Assigns a value to a variable.

Expression: A line of program code that can be reduced to a value or that assigns a value.

Logical Expression: An expression that results in a value of true or false.

Logical Operators: Operators like && (AND) and ||(OR) that enables you to create logical expressions that yield true or false results.

Operator Precedence: Determines the order in which mathematical operations are performed.

3.9 QUESTIONS FOR DISCUSSION

1. Which of the following are valid arithmetic expressions?

   (a) 28/3%2 (b) +9/5+4

   (c) 9.3%3 (d) (5/3)*3+5%3

   (e) 14%8+7%2 (f) 28% (int) 3.5

2. Write Java assignment statements to evaluate the following equations:

   (a) Area = $\pi r^2 + 2\pi$ rh

   (b) Energy = mass (acceleration*height + velocity/2)

   (c) Torque = $(2m_1m_2/m_1+m_2)g$

3. Determine the value of each of the following logical expressions if a=5, b=10 and c = - 6

   (a) a>b && a<<

   (b) a= = c// b >a

   (c) (a/2.0 = = 0.0 && b/2.0!=0.0) // c<0.0

4. The total distance travelled by a vehicle in t seconds is given by:

   Distance = ut +1/2 at^2

   Where u = initial velocity (metres per second)

   a = acceleration (metres per $second^2$)

   Write a program to evaluate the distance travelled at regular intervals of time, given the value of u and a. The program should provide the flexibility to the user to select his own time intervals and repeat the calculations for different values of u and a.

Check Your Progress: Model Answers

1. Bit-wise operators
2. Boolean logical
3. If then else

3.10 SUGGESTED READINGS

E. Balaguruswamy, *Programming with Java*, Tata McGraw-Hill.

Davis, Stephen R., *Learn Java Now*, Microsoft Press.

Naught on, Patrick, *The Java Hand Book*, OSborne McGraw-Hill.

Sams.net, *Java unleased*.

Herbert Schildt, *The Complete Reference Java 2*, Tata McGraw-Hill.

UNIT II

LESSON

4

BRANCHING AND LOOPING STATEMENTS

CONTENTS

4.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Describe various decision-making statements
- Understand the usage of if and if .... else statements
- Understand the usage of nested if .... else statements
- Describe switch statement
- Describe looping statements

- Understand the usage of the while statement
- Understand the usage of the do-while statement
- Understand the usage of for loop

4.1 INTRODUCTION

Generally a program executes its statement from beginning to end. But not many programs execute all their statements in strict order from beginning to end. Most programs decide what to do in response to changing circumstances.

In a program, statements may be executed sequentially, selectively or iteratively. Every programming language provides constructs to support sequence, selection or iteration.

When a program breaks the sequential flow and jumps to another part of the code, it is called selection or branching. When the branching is based on a particular condition, it is known as conditional branching. If branching takes place without any condition, it is known as unconditional branching.

Java language supports two types of selections statements: if and switch. In addition, in certain circumstances '?' operator can be used as an alternative to if statements.

The iteration statements allow a set of instructions to be performed repeatedly until a certain condition is fulfilled. The iteration statements are also called loops or looping statements. Java provides three kinds of loops: while loop, do-while loop, and for loop.

All three constructs of Java repeat a set of statements as long as a specified condition remains true. This specified condition is generally referred to as a loop control. For all three loop statements, a true condition is any nonzero value. A zero value indicates a false condition.

4.2 IF STATEMENT

An if statement tests a particular condition; if the condition evaluates to true, a course-of-action is followed i.e. a statement or set-of-statements is executed. Otherwise (if the condition evaluates to false), the couse-of-action is ignored.

The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

1. Simple if statement
2. if .... else statement
3. Nested if ..... else statement
4. Else if ladder

4.2.1 Simple if Statement

The syntax of the if statement is as shown below:

```
if (expression)
statement;
```

where a statement may consist of a single statement, a compound statement, or nothing (in case of emply statement). The expression must be enclosed in parentheses. If the expression evaluates to true, the statement is executed, otherwise ignored.

Following Figure illustrates the if construct of Java.

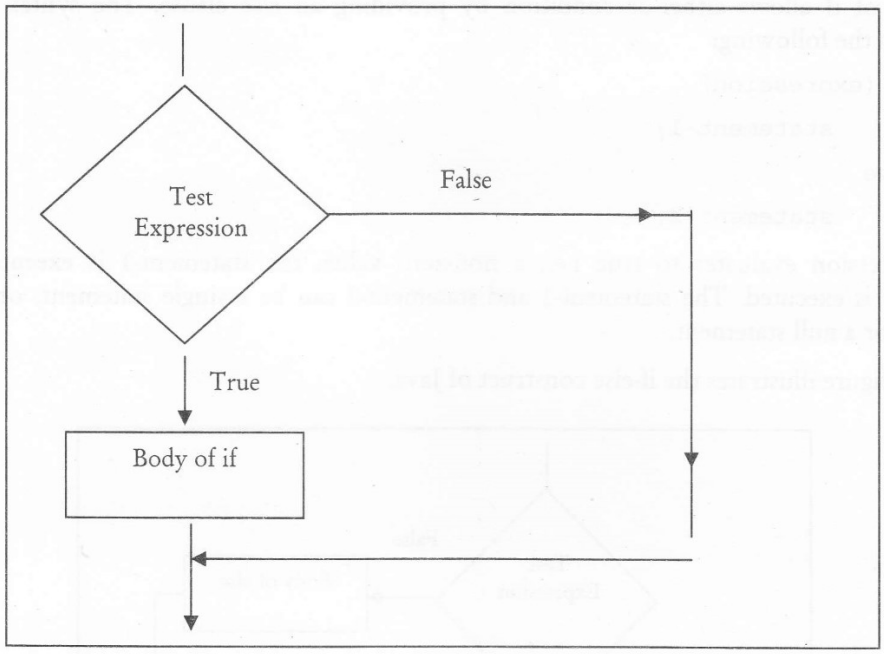


Figure 4.1: Flow Chart of if Control

For instance, the following code fragment,

```
if (ch = = ' ')
spaces ++;
```

Checks whether the character variable ch stores a space or not; if it does, the number of spaces are incremented by 1. The given example also makes use of an if statement.

```
int a, b, c;
    //...
    if (a> 10 && b < 15)
        {
            c=(a-b)* (a+b);
        }
```

This would have been equivalently done using two if statements as follows:

```
if (a>10)
if (b<15)
c= (a-b)*(a+b);
```

If the value of a is greater then 10, then the following statement is executed which in turn is another if statement. This if statement tests b and if b is less than 15, then c is calculated.

4.2.2 The if... else Statement

This form of if allows either-or condition by providing an else clause. The syntax of the if-else statement is the following:

```
if (expression)
      statement-1;
else
      statement-2;
```

If the expression evaluates to true i.e., a non-zero value, the statement-1 is executed, otherwise statement-2 is executed. The statement-1 and statement-2 can be a single statement, or a compound statement, or a null statement.

Following figure illustrates the if-else construct of Java.

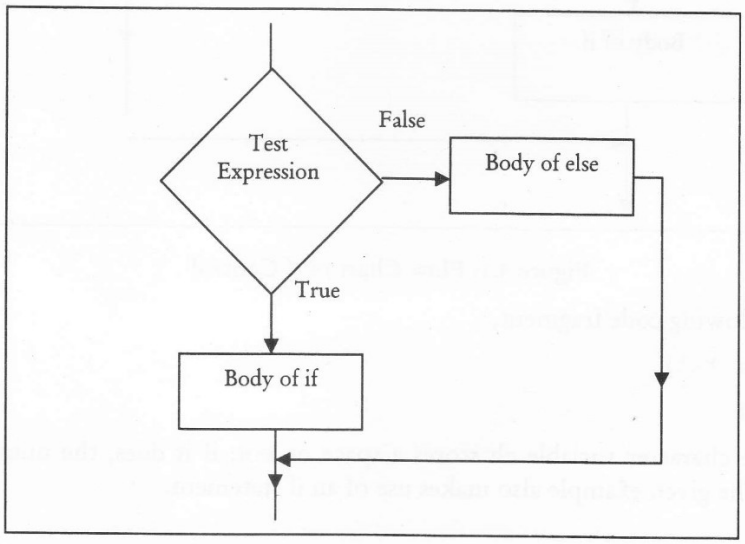


Figure 4.2: Flow Chart of if-else Control

Example

```
Int a, b;
If (a<b)
      a=0;
else
      b=0;
```

The program given below counts the even and odd numbers in a list of numbers using the if...else statement. Number [ ] is an array variable containing all the numbers and number. length gives the number of elements in the array.

```
//to illustrate if … else statement
Class IfElsetest
{
      public static void main(String args[ ])
      {
      int number[ ] = {50, 65, 56, 71, 80};
            int even = 0, odd =0;
      for (int i=0; i <number.length;i++)
      {
      if ((number [i]%2)= = 0)
            {
            even +=i;
            }
      else
            {
            add +=1;
            }
            }
System.out.println ("Even numbers: "+even +
      "odd Numbers : "+odd);
      }
}
```

Output: Even numbers : 3 odd numbers:2

4.2.3 Nesting of if... else Statements

A nested if is an if that has another if in its 'if's body or in its else's body. The nested if can have one of the following three forms:

1.
```
if (expression1)
{ :
if (expression 2)
   statement = 1;
[else
   statement = 2;]
```

```
}
else
body of else;
```

2.
```
if (expression 1)
body=of-if;
else
{ :
   if (expression 2)
   statement-1;
   [else
      statement-2;]
}
```

3.
```
if (expression 1)
[ :
if (expression 2)
statement-1;
[else
statement-2;]
:
}
else
{ :
if (expression 3)
statement-3;
[else
statement-4;]
}
```

In an if statement, either there can be if statement (s) in its body-of-if or in its body-of-else or in both.

The part in [ ] means, it is optional. The inner ifs can them selves be nested ifs, but the inner if must terminate before an outer if.

Here is an example:

```
If (i = = 10)
{
      if (j<20)
            a=b;
      if (k>100)
            c=d;
```

```
        else a=c;
}
else a=d;
```

The final else is not associated with if (j < 20), because it is not in the same block(even though it is the nearest if without an else). Rather, the final else is associated with if (i = =10). The inner else refers to if (k > 100), because it is the closest if within the same block.

4.2.4 The else-if Statement

A common programming construct in Java is the if-else-if ladder, which is often also called the if-else-if staircase because of its appearance.

It takes the following general form:

```
if (expression 1) statement 1;
else
        if (expression 2) statement 2;
        else
                if (expression 3) statement 3;
                :
                else statement n;
```

The expressions are evaluated from the top downward. As soon as an expression evaluates to true, the statement associated with it is executed and the rest of the ladder is bypassed. If none of the expressions are true, the final else gets executed. If the final else is missing, no action takes place if all other conditions are false.

Although it is technically correct to have indentation in the if-else-if ladder as shown above, however, it generally leads to overly deep indentation. For this reason, the if-else-if ladder is generally indented like this:

```
if (expression 2)
        statement 1;
elseif (expression 2)
        statement 2;
elseif (expression 3)
        statement 3;
        :
else
        statement n;
Here is an example for if-else-if ladder;
        int score = 65;
        char grade;
```

```
if (score >=90)
      grade = 'A';
elseif (score > = 70)
      grade = 'C';
elseif (score > = 60)
      grade = 'D';
else
      grade = 'F';
```

4.3 THE SWITCH CASE

Java provides a multiple-branch selection statement known as switch. This selection statement successively tests the value of an expression against a list of integer or character constants. When a match is found, the statements associated with that constant are executed.

The general form of a switch is

```
switch (expression)
{
case value1 :
      Codesegment1
case value2 :
      Codesegment2
. . .
case valuen:
      Codesegment n
default :
      defaultCodeSegment
}
```

A switch statement is used for multiple ways selection that will branch to different code segments based on the value of a variable or an expression. The optional default label is used to specify the code segment to be executed when the value of the variable or expression does not match with any of the case values. If there is no break statement as the last statement in the code segment for a certain case, the execution will continue on into the code segment for the next case clause without checking the case value.

An example of a switch statement

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

```
        int month = 4;
        String season;
        switch(month)
        {
              case 12 :
              case 1 :
              case 2:
              season = "Writer";
              break;
              case 3 :
              case 4 :
              case 5 :
              season = "Spring";
              break;
              case 6 :
              case 7 :
              case 8 :
              season = "Summer";
              break;
              case 9 :
              case 10 :
              case 11 :
              season = "Autumn
              break;
              default :
              season = "Bogus Month";
        }
        System.out.println("April is in the" + season + ".");
    }
}
```

Nested Switch

You can use a switch as part of the statement sequence of an outer switch. This is called a nested switch. Since a switch statement defines it own block, no conflicts arise between the case constants in the inner switch and those in the outer switch.

```
switch (count)
{
```

```
case 1 :
switch (target)
{
        case 0 :
        System.out.println("target is zero");
        break;
        case 1 :
        System.out.println("target is one");
        break;
}
break;
case 2 : // . . .
```

Here, the case 1 : statement in the inner switch does not conflict with the case 1 : statement in the outer switch. The count variable is only compared with the list of cases at the outer level. If count is 1, then target is compared with the inner list cases.

To summarize, there are three important features of the switch statement to note: the switch differs from the if, in that switch can only test for equality, whereas if can evaluate any type of boolean expression. That is, the switch looks only for a match between the value of the expression and one of its case constants.

No two case constants in the same switch can have identical values. A switch statement enclosed by an outer switch can have case constants in common.

A switch statement is usually more efficient than a set of nested ifs.

The last point is particularly interesting because it gives insight into how the Java compiler works. When it compiles a switch statement, the Java compiler will inspect case of the case constants and create a "jump table" that it will use for selecting the parts of execution depending on the value of the expression. Therefore, if you need to select among a large group of values, a switch statement will run much faster than the equivalent logic coded using a sequence of if-else's. The compiler can do this because it knows that the case constants are all of the same type and simply must be compared for equality with the switch expression.

4.4 THE WHILE STATEMENT

The most simple and general looping structure available in java is the while statement. The syntax of a while loop is

```
while (condition)
{
      // loop-body
}
```

where the loop body may contain a single statement, a compound statement or an empty statement. The loop iterates while the condition evaluates to true. When the expression becomes false, the program control passes to the line after the loop-body code.

In a while loop, a loop control variable should be initialized before the loop begins as an uniinitialized variable can be used in an expression. The loop variable should be updated inside the body-of-the-while. Following example program illustrates the working for a while loop.

```
// program for while loop
class whiletest
      {
            public static void main(String args[ ])
            {
                  int n=10;
                  while(n>0)
                  {
                        system.out.println("tick" +n)
                        n- -;
                  }
            }
      }
```

The body of the loop is executed 10 times for 10,9,8----| i.e. as long as n is nonzero, the loop-body iterates i.e. "tick" is printed on the screen along with the value of n followed by the decrement of n. Again the test-expression (n > 0) is evaluated: If it is true, the loop is repeated, otherwise terminated.

Example:

```
// program using a while loop
Class WhileCheck
{
public static void main(String args[ ])
{
      stringBuffer string = newstringBuffer ( );
      char c;
      system.out.println ("Enter a string");
      try
      {
            while ((c=char) system.in.read( ))!='10'
            {
                  string.append(c);
            }
```

```
        }
        catch (Exceptione)
        {
                System.out.println("Error in Input");
        }
        System.out.println("You have entered…");
        System.out.println(string);
        }
}
```

The output of the above program is

Enter a String

Java is a true object-oriented Language

You have entered...

Java is a true object-oriented Language

4.4.1 The do-while Statement

Unlike the while loop, the do-while is an exit-controlled loop i.e. it evaluates its text-expression at the bottom of the loop after executing its loop-body statements. This means that a do-while loop always executes at least once. The syntax of the do-while loop is:

```
do
{
        loop-body;
}
while (condition);
```

The braces are not necessary when the loop-body contains a single statement.

Example:

```
Class doWhileCheck
{
        public static void main (String args [ ] )
        {
                int n = 10;
                do
                {
                        System.out.println("tick" + n);
                        n--;
                }
```

```
                while (n>0);
            }
        }
```

The most common use of the do-while loop is in menu selection routine, where the menu is flashed atleast once. Then depending upon the user's response it is either repeated or terminated. The following example program is a menu selection program.

```
// Program to display a menu regarding rectangle operations
// and perform according to user's response
Class DowhileMenu
{
public static void main (String args [ ])
throws IO Exception
{
      char ch, ch1;
      float l, b, peri, area, diag;
      BufferedReader br=new BufferedReader (new InputStreamReader
      (System.in));
      System.out.prinln ("Rectangle Menu");
      System.out.println ("1.Area");
      System.out.println ("2. Perimeter");
      System.out.println ("3. Diagonal");
      System.out.println ("4. Exit");
      System.out.println ("Enter the choice :");
Do
{
      ch = (char) br.read( );
      if (ch = = '1' | | ch = = '2' | | ch = = '3')
      {
            System.out.println("Enter length and breadth:");
            l = (float) br.read( );
            b = (float) br.read( );
      }
Switch (ch)
{
      case '1' : area = l*b
      system.out.println ("Area =" +area);
            break;
```

```
        case '2' : peri = 2*(l+b);
                   System.out.println("Perimeter =" +peri);
                   Break;
        case '3' : diag = sqrt ((l*l) + (b*b));
                   System.out.prinln ("Diagonal = " +diag);
                   Break;
        case '4' : System.out.println("Breaking");
                   exit (0);
        default : System.out.println("Wrong choice !!!");
                   System.out.println("Enter a valid one");
                   Break;
}       //end of switch.
System.out.println("Want to enter more(Y/N)?");
ch1= (char)br.read( );
if (ch1 = = 'y' | | ch1 = = 'y')
        System.out.println ("Again enter choice (1-4):");
} while (ch1 = = 'y | | ch1 = = 'y');      //end of do-loop
}
}
```

4.5 THE FOR STATEMENT

The for loop is the easiest to understand of the Java loops. All its loop-control elements are gathered in one place (on the top of the loop), while in the other loop construction of C++, they (top-control elements) are scattered about the program. The syntax of the for loop statement is

```
for (initialization expression(s); test condition; update expression)
{
    loop-body
}
```

Figure 4.3 outlines the working of a for loop.

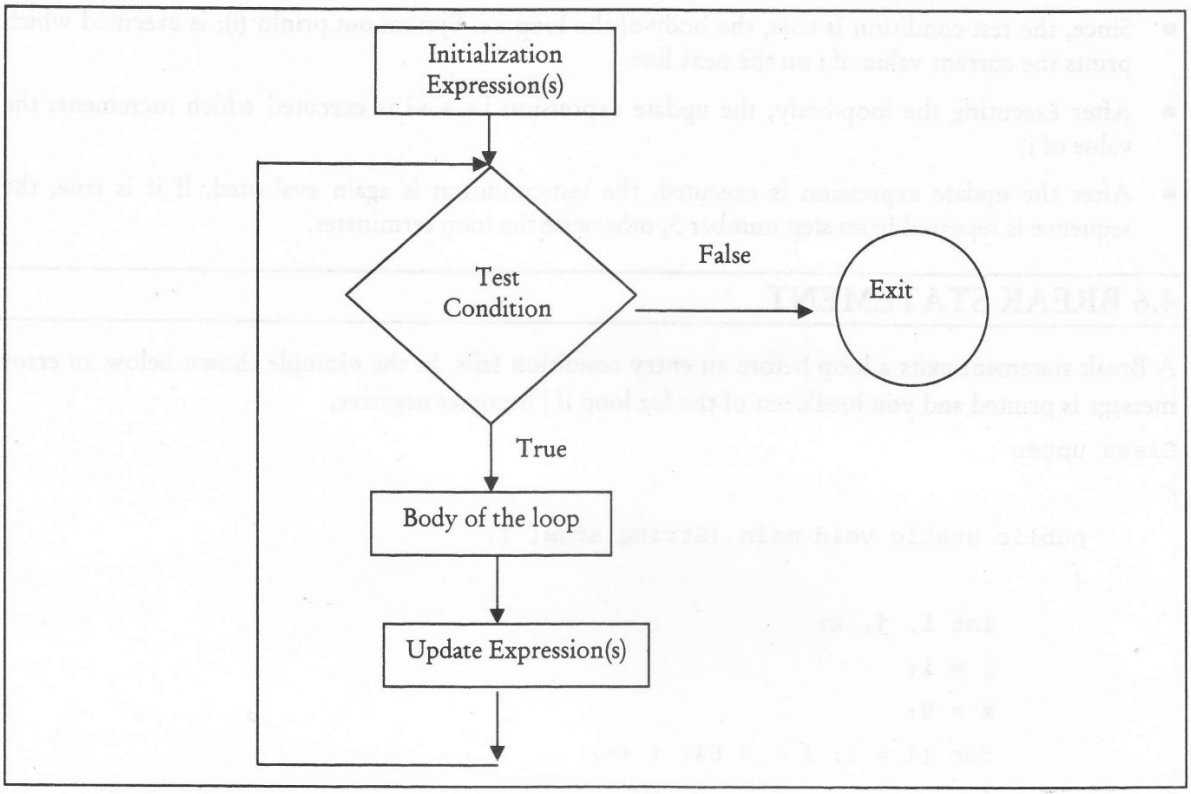


Figure 4.3: The Execution of a For Loop

The following example program illustrates the use of for statement.

```
//program to print numbers from 1 to 10
class ForTest
{
      public static void main(String args[ ])
{
      int i;
      for (i=1, i<=10; i++)

System.out.prinln (i);
}
}
```

The following lines explain the working of the above given for loop.

- Firstly, initialization expression is executed i.e. i=1 which gives the first value 1 to variable i.
- Then, the test condition is evaluated i.e. i< =10 which results into true i.e. 1.

- Since, the test-condition is true, the body-of-the loop i.e. System.out.prinln (i); is executed which prints the current value of i on the next line.
- After executing the loop-body, the update expression i.e.++i is executed which increments the value of i.
- After the update expression is executed, the test-condition is again evaluated. If it is true, the sequence is repeated from step number 3, otherwise the loop terminates.

4.6 BREAK STATEMENT

A Break statement exits a loop before an entry condition fails. In the example shown below an error message is printed and you break out of the for loop if j becomes negative.

```
Class uptec
{
      public static void main (String args[ ])
      {
            int i, j, k;
            j = i;
            k = 0;
            for (i = 1; i < = 64; i ++)
            {
                  j * 2 ;
                  if (j < = 0)
                  {
                        System.out.println ("Error :
                        overflow");
                        break;
                  }
                  k + = j;
                  System.out.print (k + "It");
                  if (i = = 0)
      System.out.println ( );
            }
            System.out.println ("All done!");
      }
}
```

The most common use of break is in switch statement. Another general form of break statement is break label; where the label is optional. Without a label, the break statement will transfer the program control to the statement just after the innermost enclosing loop or switch statement. With a label, it

will transfer the program control to the statement just after the enclosing statement or block of statements carrying the same label.

Example:

```
PrintWriter out = new PrintWriter (System.out);
Outerloop :
for (int i = 0, count = 0; i < dailyhigh.length; i++)
for (int j = 0; j < dailyhigh [i].length; j++)
      if ((dailyhigh [i] [j] > 70) & (++ count = = 3))
      {
            out.println ("The data is : month : " + (i +
            1) + , day =" + (j + 1));
      break Outerloop;
```

4.7 CONTINUE STATEMENT

The general form of a continue statement is:

```
continue Label;
```

where the label is optional. Without a label, it behaves exactly the same as in C and C++. The program control is transferred to the point right after the last statement in the enclosing loop body. In while and do statement, the condition at expression1 will not be retested.

In for loop, the increment statements will be executed next. With a label the program, control will be transferred to the end of the enclosing loop body with the same label, instead of the innermost one.

For example, the following code segment defines a method to return the offset position of the first occurrence of one string str2 in the other string str1.

```
int indexof (String Str1, String str2)
{
int len1 = str1.length ( );
      int len2 = str2.length( );
      char str2.firstChar = str2. CharAt(0);
      advanceOneCharAtStr1:
      for (int i = 0; i + len2 < = len1; i++)
      if (str1. CharAt (i) = = str2 FirstChar)
      {
            for (int j = 1; j < len2; j++)
            if (Str1.CharAt (i+j) ! = str2. CharAt (j))
            continue advanceOneCharAtStr1;
            return i;
}
```

```
    return - 1;
}
```

It is sometimes necessary to exit from the middle of a loop. Sometimes you will want to start over at the top of the loop. Sometimes you will want to leave the loop completely. For these purposes Java provides the break and continues statements. A continue statement returns to the beginning of the innermost enclosing loop without completing the rest of the statements in the body of the loop. If you are in a for loop, the counter is incremented.

```
for (int i = 0; i < m.length; i++)
    {
        if (m (i)%2 = = 0) continue;
        // process
    }
```

The continue statement is rarely used in practice because most of the instances where it is useful have simpler implementations. For instance, the above fragment could equally well have been written as:

```
for (int i = 0; i < m.length : i++)
{
    if (m[i] % 2 ! = 0)
    {
}
}
```

4.8 RETURN STATEMENT

The general form of return statement is

```
return Expression;
```

A return statement is used to return control to the caller from within a method or constructor. If the method is defined to return a value, the expression must be evaluated to the return type of that method. Otherwise only an unlabeled return statement can be used.

Check Your Progress

Explain the four forms of If statements with examples.

4.9 LET US SUM UP

The flow of control in a program can be in three ways: sequentially, selectively, and iteratively. The sequence construct means statements get executed sequentially. The selection construct means the execution of statements depending upon a condition-test. The iteration constructs mean repetition of a set-of-statements depending upon a condition-test. Java supports two types of selection statements: If and Switch. If if-else statement tests an expression and depending upon its truth value one of the two

sets-of-action is executed. In a nested if-else statement, an else goes with immediately preceding unmatched if.

Switch statement tests a value against a set of integer constants (that includes characters also). A switch statement can be nested also. An if-else statement is more flexible and versatile compared to switch but switch is more efficient in a situation when the same variable is compared against a set of values for equality.

The statements that allow a set of instructions to be performed repeatedly are iteration statements. They are also called loops or looping statements. Java provides three loops: for, while and do-while. Four elements control a loop: initialization expression(s), test expression, update expression and loop body.

A for loop can have multiple initialization and update expression separated by commas. The loop-control elements in a loop are optional. An infinite for loop has missing test-expression. An item declared in or while loop can be accessed after the loop is over. The loops can be nested also, if a loop contains another loop inside the body.

The statements that facilitate the unconditional transfer of program control are called jump statements. In nested structure, a break terminates the very statement it appears in. The continue statement abandons the current iteration of the loop by skipping over the rest of the statements in the loop-body. It immediately transfers control to the evaluation of the test-expression of the loop for the next iteration of the loop.

4.10 KEYWORDS

Conditional Branching: When a program jumps to a different part of a program based on a certain condition being met.

Nesting: When one program block is placed within another program block.

Infinite Loop: A loop that never ends.

Iteration Statement: Statement that allows a set of instructions to be performed repeatedly.

Jump Statement: Statement that unconditionally transfers program control within a function.

Nested Loop: A loop that contains another loop inside its body.

4.11 QUESTIONS FOR DISCUSSION

1. Write a program in Java to find out whether a year centered in 4-digit number representing it) is a leap year.
2. Given three numbers A, B and C, write a program to write their values in descending order. For example, if A=7, B=4, and C=10, your program should print out:

   10, 7, 4.
3. What is the effect of absence of break in a switch statement?
4. What is the significance of default clause in a switch statement?
5. Write the syntax and purpose of a switch statement.

6. Write a short program to check whether square root of a number is prime or not.
7. Write a program to find the number of and sum of all integers greater than 100 and less than 200 that are divisible by 7.
8. Given a list of marks ranging from 0 to 100, write a program to compute and print the number of students who have obtained marks
   (a) in the range 81 to 100,
   (b) in the range 61 to 80,
   (c) in the range 41 to 60, and
   (d) in the range 0 to 40.
9. The program should use a minimum number of if statements.
   (a) Write a program to print the Floyd's triangle given below:

   (i) 1
   (ii) 2 3
   (iii) 4 5 6
   (iv) 7 8 9 10
   (v) 11... 15
   (vi) :
   (vii) :
   (viii) 79..... 91

   (b) Modify the program to produce the following form of Floyd's triangle:

   1
   1
   1 0 1
   0 1 0 1
   1 0 1 0 1
   :
   :

Check Your Progress: Model Answers

(a) If
(b) If-else
(c) Nested-if-else
(d) Else-if

4.12 SUGGESTED READINGS

E. Balaguruswamy, *Programming with Java*, Tata McGraw-Hill.

Davis, Stephen R., *Learn Java Now*, Microsoft Press.

Naughton, Patrick, *The Java Hand Book*, OSborne McGraw- Hill.

Sams.net, *Java unleased*.

Herbert Schildt, *The Complete Reference Java 2*, Tata McGraw-Hill.

LESSON

5

CLASSES, METHODS AND OBJECTS

CONTENTS

5.0 AIMS AND OBJECTIVES

After studying this lesson, you will be able to:

- Define class, methods and objects
- Understand how to declare objects
- Describe various methods in classes

- Explain constructors
- Discuss This keyword
- Identify class structure

5.1 INTRODUCTION

Java gives you the capability to do object-oriented programming. Object-oriented programming enables the programmer to group together data and the code that uses data into discrete units. Objects in Java are defined using Java classes. Anything we wish to represent in a java program must be encapsulated in a class that defines the state and behavior of the basic program components known as object. Classes create objects and objects use methods to communicate between them.

In java, the data-items are called fields and the functions are called methods. Calling a specific method in an object is described as sending a message to the object. A class provides a sort of template for an object and behaves like a basic data type such as int.

5.2 WHAT IS A CLASS?

A class is a user defined data type with a template that serves to define its properties and operations. Once the class has been defined, we can create 'Variables' of that type. These variables are termed as instances of classes, which are the actual objects.

5.2.1 Class Structure

The general form of a class definition is

```
class classname
{
        [variable declarations;]
        [methods declarations;]
}
```

For e.g.;

- The data, or variables, defined within a class are called instance variables. The code is contained within methods. These are also called members of the class.
- In order to describe the characteristics of all pens, you might define a pen class and specify operations such as write and refill and attributes such as ink colour and ink remaining. When you create individual pen objects to represent my blue pen, teacher's red pen, you may initialize ink colour and ink amount.

5.2.2 Class as Datatype

Java has four kinds of named data types and two categories of variables. The data types are primitives, classes, interfaces and arrays. Variables can either contain primitive values or refer to objects.

A class creates a new data type that can be used to create objects. A class creates a logical framework that defines the relationship between its members.

When you declare an object of a class, you are creating an instance of that class. Thus, a class is a logical construct. An object has physical reality.

5.2.3 Simple Class

```
class Movie
{
        String title;
        int length;
        int cost;
        char type;
        void getdata (int x, int y, int z)
        {
              length = x;
              cost = y;
              type = z;
        }
        Movie (String s)
        {
              title = s;
        }
}
```

Class declaration only creates a template. It does not create an actual object, and hence does not occupy memory. To actually create movie object you will use a statement:

```
Movie mov1 = new Movie ("Mr. India");
Movie mov2 = new Movie( );
```

5.2.4 Defining a Class

A class is a user defined data type used to implement an abstract object, giving you the capability to use object-oriented programming with java.

Once the class-type has been defined, we can create 'variables' of that type. These variables are termed as instances of classes, which are the actual objects.

The general form of a class definition is

```
Class name [extends superclassname]
{
   [Variable declarations;]
   [methods declarations;]
```

```
}
```

Everything written inside the square brackets is optional. Thus, the following would be a valid class definition.

```
Class Empty
{
}
```

Classname and Superclass name are any valid java identifiers. The keywords extends indicates that the properties of the superclassname class are extended to the class classname. This concept is known as inheritance.

5.2.5 Adding Variables

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated. Instance variables can be declared exactly the same way as we declare local variables.

Example:

```
Class Movie
{
        string title;
        int length;
        int cost;
        char type;
}
```

The class movie contains four instance variables, out of which one is of type string, two are of type int and one is of type char. The two int variables can be declared in one line as

```
int length, cost;
```

Remember these variables are only declared and therefore no storage space has been created in the memory. Instance variables are also known as member variables.

5.3 WHAT ARE METHODS?

A Java method is equivalent to a function, procedure, or subroutine in other languages except that it must be defined inside a class definition. Instance methods form the foundation of encapsulation and are a key to providing a consistent interface to the class.

5.3.1 Adding Methods to Classes

Methods are declared inside the body of the class but immediately after the declaration of instance variables. The general form of a method declaration is

```
Returntype methodName (parameter list)
        {
```

```
        Method-body;
}
```

A Returntype can be a primitive type such as int, a class type such as string or void.

A MethodName begins with a lowercase letter and compound words in the method name should begin with uppercase letter according to Java convention.

An optional parameter list/argument list must be inside parentheses, separated by commas.

The method-body must be enclosed in braces.

5.3.2 Return Value

The return type specifies the type of value the method would return. This could be a simple data type such as int as well as any class type or it could even be void type if the method does not return any value.

e.g.: class Box

```
{
        double width;
  double height;
double depth;
        double volume( )
        {
              return width * height * depth;
        }
    class ABC
    {
        public static void main (String args[ ])
        {
              Box mybox1 = new Box( );
              double vol;
              mybox1.width = 10;
              mybox1.height = 20;
              mybox1.depth = 15;
              vol = mybox1.volume( );
              System.out.println ("Volume is" + vol);
        }
}
```

The type of the data returned by a method must be compatible with the return type specified by the method.

The variable receiving the value returned by a method must also be compatible with the return type specified for the method.

5.3.3 Method Accepting Parameters

The parameter list is always enclosed in parentheses. This list contains variable names and types of all the values we want to give to the method as input. If the method does not take any argument, simply leave the parentheses empty.

```
class Rectangle
        {
              int length; int width;
              void getdata (int x, int y)
                    {
                          length = x;
                          width = y;
                    }
              int rectArea( )
                    {
                          int area = length * width;
                          return (area);
                    }
        }
```

In the above example, length and width are instance variables.

- x and y are parameters.
- void indicates method, does not return a value.

When a primitive is passed to a method, a copy of the value is generated. If the method changes the value of the argument in any way, only local argument will be affected. When the method terminates, this local argument is discarded and the original variable in the calling method is unchanged.

```
class Rectangle
{
        int length;
        int width;
        void change (int l, int w)
        {
              l = l + 10;
              w = w + 10;
              System.out.println ("l =" + l + "w = " +w);
        }
        public static void main (String args[ ])
        {
              int len = 20;
              int thick = 20;
              Rectangle rect = new Rectangle( );
              rect.change (len, thick);
```

```
                System.out.println ("len" + len + "thickness" + thick);
        }
}
Accessing Class members
        class Rectangle
        {
                int length, width;
                void getData (int x, int y)
                {
                        length = x;
                        width = y;
                }
                int rectarea( )
                {
                        int area = length * width;
                        return (area);
                        }
                }
        class RectArea
        {
                public static void main (String args[ ])
                {
                        int area1, area2;
                        Rectangle rect1 = new Rectangle ( );
                        Rectangle rect2 = new Rectangle( );
                        rect1.length = 15 // Accessing variables
                        rect1. width = 10
                        area1 = rect1.length * rect1.width
                        rect2.getData (20, 12);
                        area2 = rect2.rectArea( );
                        System.out.println ("Area1 = "+ area1);
                        System.out.println ("Area2 = "+ area2);
        }
}
```

Use the dot operator with instance variable and to call instance methods. The general syntax is

```
Objref.instance variable;
Objref.MethodName (arguments);
```

5.4 THIS KEYWORD

All instance methods receive an implicit argument called "this", which may be used inside any method to refer to the current object i.e., the object on which method was called. Inside an instance method, any unqualified reference is implicitly associated with the this reference.

```
public void setRating (String st)
        {
              rating = st;
              this.rating = st;
        }
```

Usage of this keyword:

- When you need to pass a reference to the current object as an argument to another method.
- When the name of the instance variable and parameter is the same, parameter hides the instance variable.

```
class Rectangle
{
        int length, width;
        Rectangle (int length, int w);
        {
              length = length;          // incorrect
              width = w;                // correct
              this.length = length;     // incorrect
        }
}
```

5.5 CREATING OBJECTS

In java, objects are essentially a block of memory that contains space to store all the instance variables. The other name of creating an object is instantiating an object.

To create an object a new keyword is used as an operator. This operator creates an object of the specified class and returns a reference to that objects.

The syntax of creating an object is as follows:

```
Class-name object-name;          //declare
Object-name = new class-name ( );   //instantiate
```

For example:

```
Movie mov1;
Mov1 = new Movie( );
```

The first statement declares a variable to hold the object reference and the second one actually assigns the object reference to the variable. Both statements can be combined into one as shown below:

```
Movie mov1 = new Movie( );
```

where Movie is the class name, mov1 is an object created by the default empty constructor of the class. We can create any number of objects of a class. Each object has its own copy of instance variables of its class. Thus, any changes to the variables of one object have no effect on the variables of another. Two or more references can be created to the same object just by assigning one object reference variable to another.

Example:

```
Movie mov2 = new Movie( );
Movie mov3 = mov2;
```

Here, both mov3 and mov2 refer to the same object

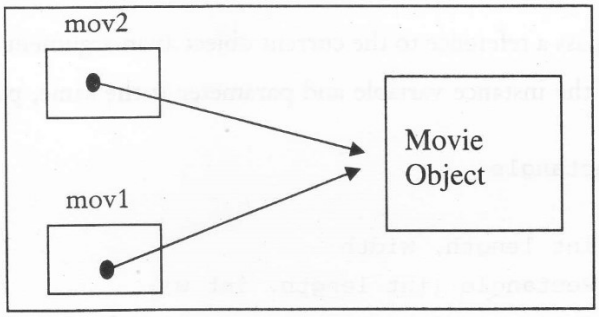


Figure 5.1: Two References of Same Object

5.5.1 Accessing Class Members

After creating objects, we are discussing about the accessing class members using objects. Each objects containing its own set of variables, we should assign values to these variables in order to use them in our program. Since we are outside the class, we cannot access the instance variables and the methods directly. Thus, to access them, we use the concerned object with the dot operator. The syntax is as follows:

```
Object_name.Variable_name;
Object_name.Method_name (parameter_list);
```

Here object_name is the name of the object of the class and variable-name is the name of the instance variable inside the object.

Similarly, method_name is the method inside the object and parameter_list is the comma-separated list of the actual values that must match in type and number with the parameter list of the methodname declared in the class.

The instance variable and method of Movie class may be accessed and assigned values as follows:

```
mov1.length = 15;
mov1.cost=150;
mov2.length=25;
mov2.cost=250;
```

Both objects store different values, and any changes in one have no effect on other.

This is way of assigning values to the variables in the object directly calling by their object using dot operator.

The other and more convenient way of assigning values to the instance variables is to use a method that is declared inside the class.

As discussed in Adding Method topic, the getData method is basically added to provide values to the instance variables. For example:

```
Movie mov1=new Movie( );
Mov1.getData("DON", 180, 250, 's');
```

This code creates mov1 object and then passes in the values to the temporary variables s,x,y & z and the assigns these values to title, length, cost and type respectively.

Let us consider a complete example

```
Class Rectangle
{
        int length, width;
        void getData(int a, int b)
        {
                length = a;
                width = b;
        }
        int area( )
        {
                int area = length * width;
                return (area);
        }
}
class AreaofRect
{
public static void main(String args[ ])
{
        int area1, area 2;
        Rectangle R1 = new Rectangle( );
        Rectangle R2 = new Rectangle( );
        R1.length=15;
        R1.width=20;
        area 1 =R1.length*R1.width;
```

```
        R2.getData (30, 15);
        area 2 = R2.area1 ( );
        System.out println ("AREA1=" +area1);
        System.out.println ("AREA2="+area 2);
        }
}
```

The output of above stated example is as follows:

AREA 1 = 300

AREA 2 =450

5.5.2 Declaring Objects

Class is a static definition that enables us to understand all the objects of that class. Objects exist at run time. They hold attributes and they send messages to each other Classes, however, do not exist at run time.

Each instance of the class (that is, each object of the class) contains its own copy of the variables.

In case of the movie class, each individual movie is an instance of movie. "Gone with the wind" is one distinct instance of movie while "Last action her" is another. Each has its own set of variables. For e.g. mov1 and mov2 objects have four instance variables each - mov1.title, mov1.length, mov2.title, mov2.length.

5.6 CONSTRUCTORS

When an object is created, Java performs default initialization, char variables are set to '\u0000', byte, short, int, long to zero, boolean to false, float to 0.0

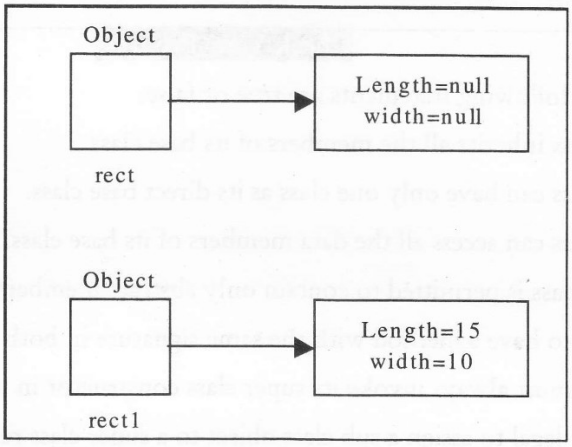
All objects that are created must be given initial values. We can do it,

1. By assigning the value to instance variable using dot operator.
2. Through instance method for e.g. get data( )
3. By using constructor method to initialize an object when it is first created.
   (a) Constructors have the same name as class itself.
   (b) Constructors do not specify a return type.
   (c) Constructor is called automatically by the run time system.

For example

```
class Rectangle
    {
        int length, width;
        Rectangle(int x, int y) // constructor
        {
```

```
            length = x;
            width = y;
        }
        int rectArea( )
        {
            return (length * width);
        }
    }
class ABC
{
    public static void main (String args[ ])
    {   Rectangle rect = new Rectangle( )
        Rectang lerect1= new Rectangle(15,10)// calling constructor
        int area1 = rect1.rectArea( );
        System.out.println("Area1 = "+ area1);
    }
}
```



If you do not provide any constructors, a default no-arg constructor is provided for you. This constructor takes no arguments and does nothing. If you want a specific no argument constructor as well as constructors that take arguments, you must explicitly provide your own no argument constructor.

Default Constructor

If no constructors are declared in a class, the compiler will create a default constructor that takes no parameter.

When you create new Movie object, the compiler decides which constructor to call, based on the argument specified in parentheses in the new statement. Constructor can call another constructor of

the same class by using this( ) syntax. By using this( ), you avoid duplicate code in multiple constructors. This technique is used when Initialization routine is complex.

```
public class Movie
    {
        private String title;
        private String rating;
        public Movie( )
        {
            this("G");
        }
        public Movie (String newRating)
        {
        rating = newRating;
        }
    }
```

Rules: The call to this( ) must be the first statement in the constructor. The argument to this( ) must match those of the target constructor.

Check Your Progress

State whether the following statements are true or false:

1. A derived class inherits all the members of its base class.
2. A derived class can have only one class as its direct base class.
3. A derived class can access all the data members of its base class.
4. An abstract class is permitted to contain only abstract members.
5. It is an error to have a method with the same signature in both super class and its subclass.
6. Constructor must always invoke its super class constructor in its first statement.
7. It is perfectly legal to assign a sub class object to a super class reference.

5.7 LET US SUM UP

Classes, objects ad Methods are the basic components used in java programming. As java is fully object oriented language, every method even main ( ) is also defined in a class, therefore the concept of classes is at the root of java's design. We have discussed the following in this lesson: How to define a class; How to add variables and Methods in a class; How to create objects of the class by using new keyword; How to access the class members by using dot operator; Usability of constructor; How to extend or reuse a class.

We have also discussed various features that could be used to restrict the access to certain variables and methods from outside the class.

5.8 KEYWORDS

Classes: A collection of variables and methods that an object can have, or a template for building objects.

Objects: An instantiation of a class.

Methods: A routine that belongs to a class.

Fields: A data object encapsulated in a class.

Instance: A concrete representation of a class or object. A class can have many instances.

Inheritance: A property of object-oriented languages where a class inherits the methods and variables of more general classes.

Subclass: A class that inherits methods and variables from another class. The statement class subclass extends super class means that sub class is a sub class of super class.

Super Class: A generalization of another class. X is a super class of Y if y inherits variables and Methods from X.

Constructor: A method that is used to create an instantiation of a class.

Package: A Java keyword used to assign the contents of a file to a package. Packages are java's mechanism for grouping classes. Packages simplify reuse and they are very useful for large projects.

5.9 QUESTIONS FOR DISCUSSION

1. What is a constructor? What are its special properties and how do we invoke it?
2. Write a short note on classes and objects.
3. When do we declare method or class final?
4. When do we declare code in constructor of class A (intx)?
5. When do we declare a method or class abstract?
6. Describe the different form of inheritance with example.
7. Discuss the different levels of access protection available in java.

Check Your Progress: Model Answers

1. False
2. True
3. False
4. True
5. False
6. True
7. True

5.10 SUGGESTED READINGS

E. Balaguruswami, *Programming with Java*, Tata McGraw-Hill.

Davis, Stephen R., *Learn Java Now*, Microsoft Press.

Naughton, Patrick, *The Java Hand Book*, OSborne McGraw- Hill.

Sams.net, *Java unleased*.

Herbert Schildt, *The Complete Reference Java 2*, Tata McGraw-Hill.

UNIT III