Week 5

Conditional and Iterative statements

Content:Decision making: if, if..else, switch ,Iterative: need of iterative statements; types of loops in java; how to use them; Break and continue statements;

Decision Making

✔ All the programs in Java have set of statements, which are executed sequentially in the order in which they appear.

✔ This happens when jumping of statements or repetition of certain calculations is not necessary

✔ There may arise some situations where programmers have to change the order of execution of statements based on certain conditions

✔ Java has such decision-making capabilities within its program by the use of **if** statement

If is a decision making statement and is used to control the flow of execution of statements.

It is a 2 way decision making statement.

It is used with an expression.

Expression is evaluated first and depending on whether the value is true or false, control is transferred to a particular statement.

Different forms of if statements are

- ❖ Simple if statement
- ❖ if else statements
- ❖ Nested if else statements
- ❖ else if ladder

1.SIMPLE IF STATEMENT:

General form of simple if statement

```
if (test expression)
{
        Statement - block;
}
  Statement - x
```

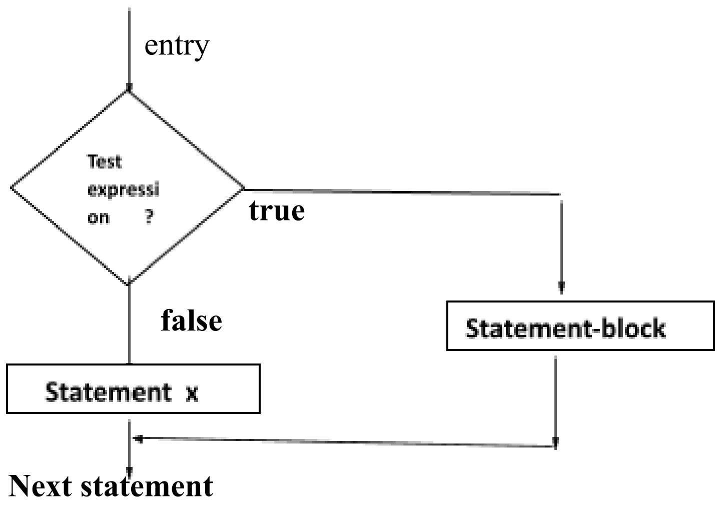


Fig: flowchart of simple if control

If the test expression is true, the statement-block will be executed, otherwise statement – block is skipped and execution will jump to the statement x. The statement- block may be a single statement or a group of statements.

Program to show simple if statement

```
class test
{
        public static void main(String arg[])
        {
             int age=10;
             if(age>=18)
                   System.out.println("You can vote");
        }
 }
```

2. IF....ELSE STATEMENT:

The if..else statement is an extension of the simple if statement.

general syntax

```
if (test –expression)
{
     true block statement;
}
else
```

```
    {
        false block statement
    }
    statement- x;
```

If test expression is true, then true block statements immediately following the if statement will be executed otherwise false block statement is executed.

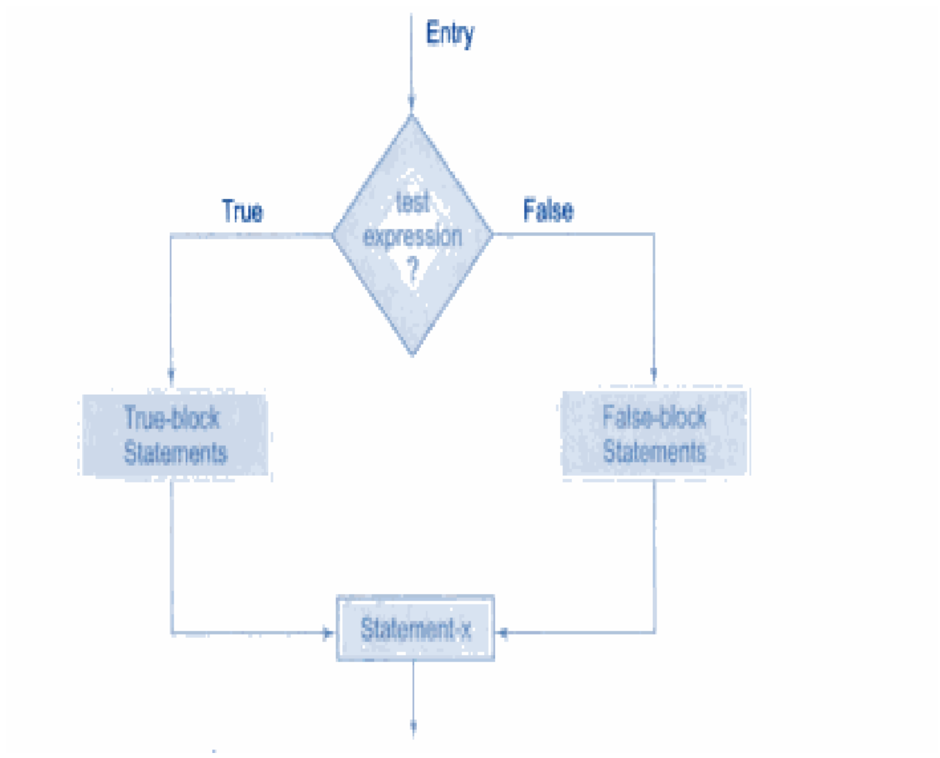


Fig: flowchart of if ..else control

Program to show if…else statement

```
class test
{
        public static void main(String arg[])
        {
        int a=10,b=50;
        if(a>b)
            System.out.println(a+"is greater\n");
        else
            System.out.println(b+"is greater\n");
        }
}
```

3.NESTED IF....ELSE STATEMENT:

When many decisions are involved, more than one if...else statement in nested form is needed. if condition1 is false, statement 3 is executed, otherwise it checks the second condition. If condition 2 is true, statement 1 is evaluated otherwise statement 2 will be evaluated, then control is transferred to statement x.

General syntax is:

```
    if (test condition1)
    {
          if (test condition 2)
          {
                statement 1;
          }
          else
          {
                statement 2;
          }
    }
   else
   {
      statement 3;
   }
  statement x ;
```

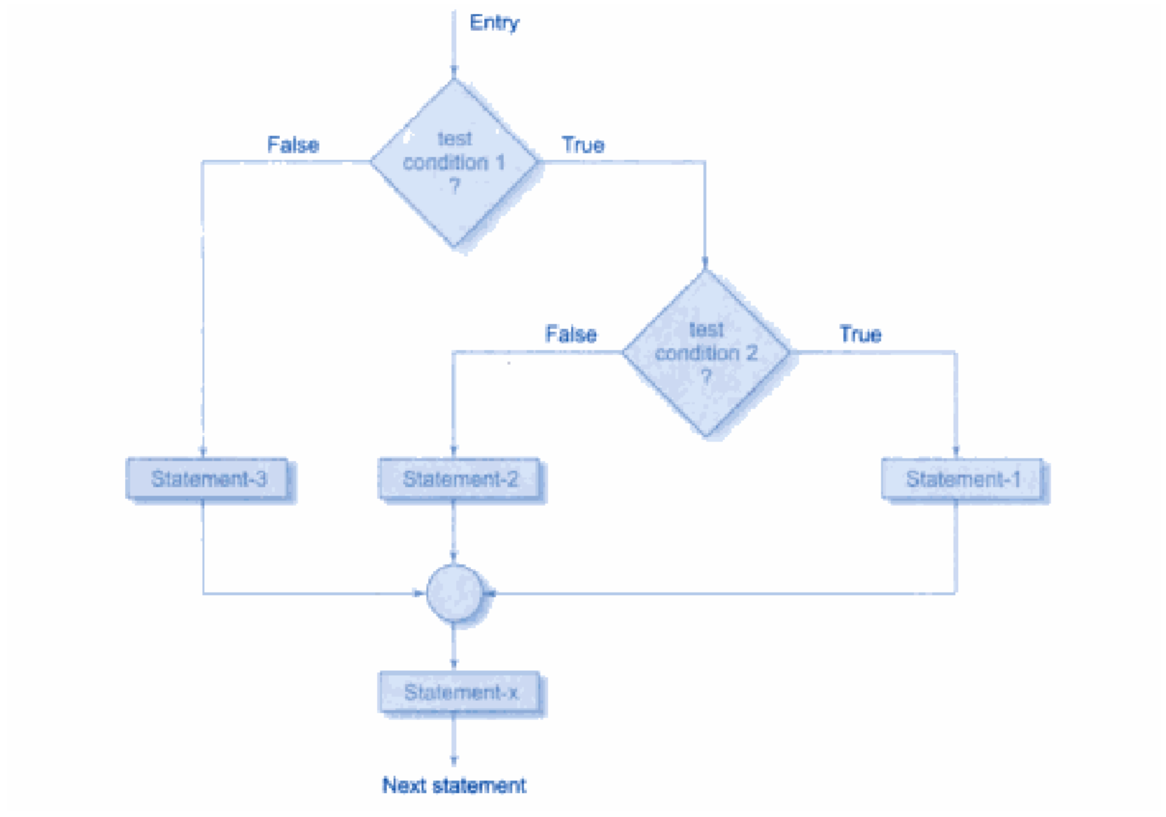


Fig: flowchart of nested if..else statements

Ex:

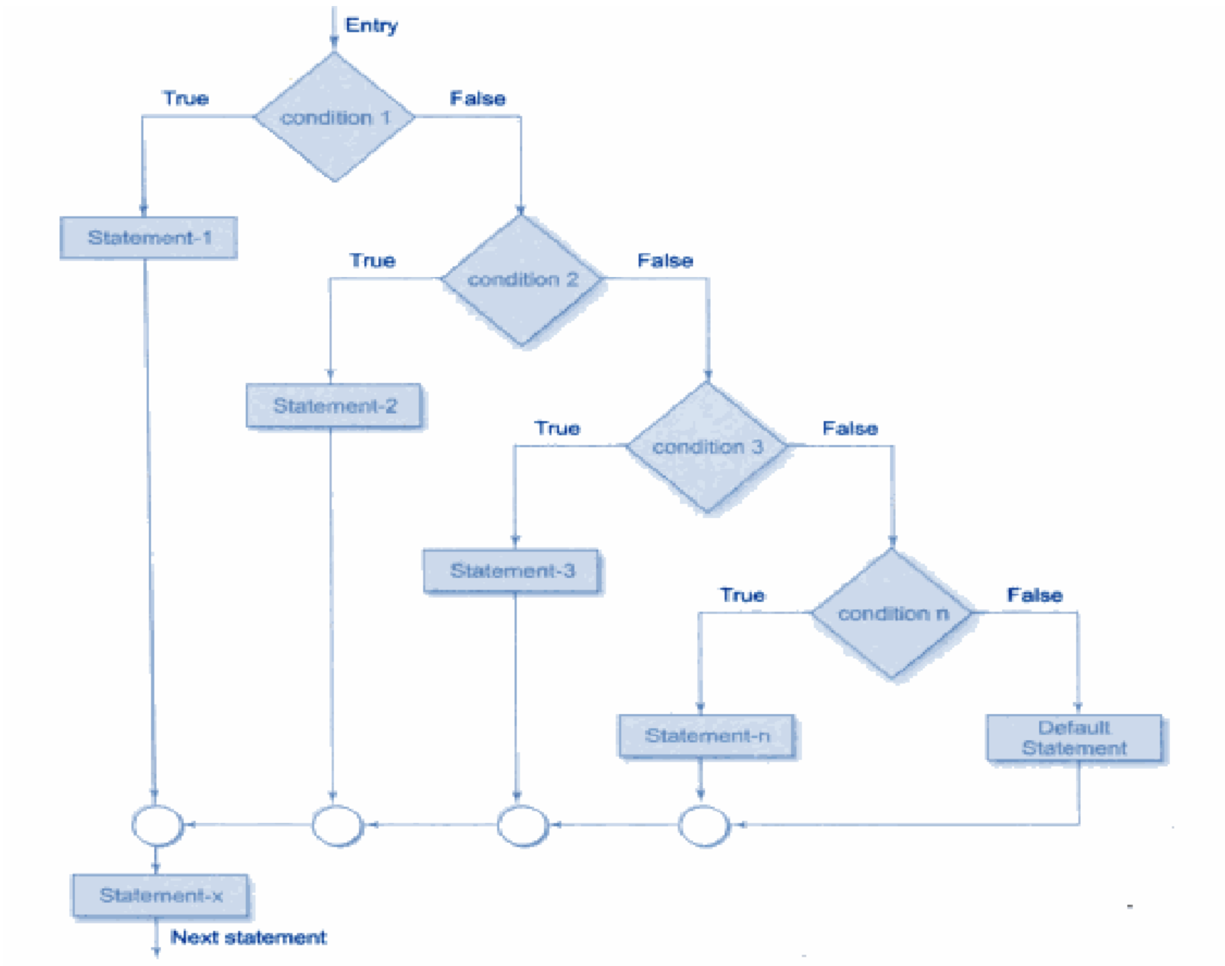
```
class test
{
                public static void main(String args[])
                {
      char gender='f';
      int balance=6000,bonus;
                if ( gender = = 'f')
                {
                          if (balance  >5000)
                                 Bonus=500;
                           else
                               Bonus = 200;
                }
                else
                {
                          Bonus=100;
}
balance = balance + bonus;
System.out.println("balance="+balance);
}
}
```

4.Else – if ladder

Used for multipath decisions. It is a chain of if"s where each else is associated with an if.

General syntax is :

```
if (condition1)
            statement 1;
 else if (condition 2)
             statement 2;
else if (condition 3)
            statement 3;
            ....
else if (condition n)
             statement n;
statement x ;
```



Flowchart of else if ladder

Ex

```
class test
{
            public static void main(String arg[])
            {
     int marks=81;
       if (marks>=75)
               System.out.println(”distinction”);
       else if (marks >=60)
                        System.out.println(”first class”);
       else if (marks >=35)
               System.out.println(” second class”);
     else
        System.out.println(“fail”);
}
}
```

SWITCH STATEMENT

- ✔ Switch is used in place of multiple if… else statements which makes a program complicated.
- ✔ switch is a multiway decision.
- ✔ It tests the value of a given variable with a list of case values. When a match is found, the statement of that case is executed

General Syntax:

```
switch (expression)
{
   case value -1:
               block 1;
               break;
   case value -2:
               block2;
                break;
   …….
   …….
   default:
           default block
           break;
}
statement x;
```

where value-1, value -2 are constants.

- ✔ Each value should be unique in a switch statement. The expression may be integer or character
- ✔ The break after each block is the end of a case .
- ✔ Break causes an exit from switch.
- ✔ The statement x after the switch is executed.
- ✔ Default statement is executed if no case value matches the expression

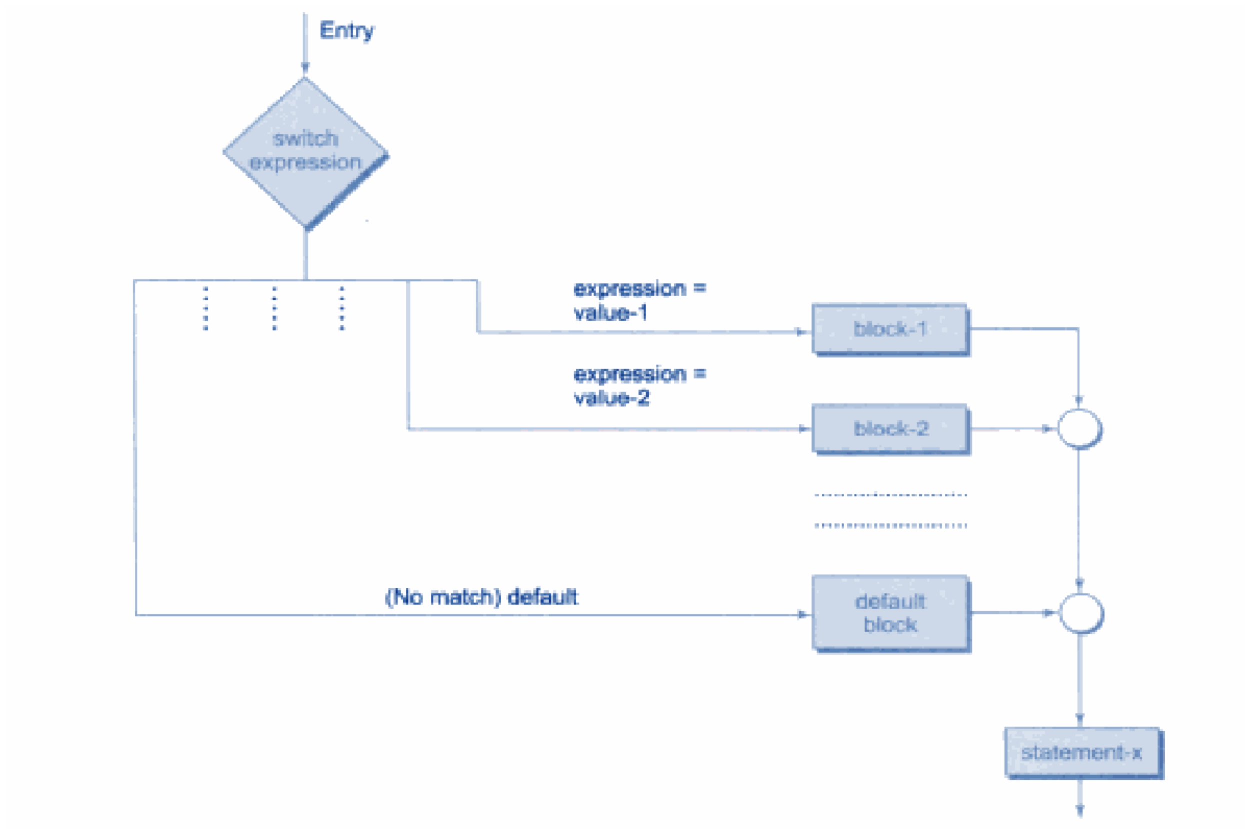


Fig. 6.6 *Selection process of the switch statement*

Ex :

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

        int choice =3;
         switch (choice)
         {
              case 1:
                   System.out.println ("choice is one");
                       break;
              case 2:
                   System.out.println ("choice is two");
                       break;
```

```
                    default:
                              System.out.println ("choice is  not one or two");
              }
      }
      }
```

✔ A **switch** statement is used for multiway 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 of case block, the execution will continue on into the code segment for the next case without checking the case value.

Iterative Statements

✔ The java programming language provides a set of iterative statements that are used to execute a statement or a block of statements repeatedly as long as the given condition is true.

✔ The iterative statements are also known as looping statements or repetitive statements.

LOOPING

✔ The process of repeatedly executing a set of statements is known as looping.

✔ The statements inside a loop are executed repeatedly until some condition for ending the loop is satisfied.

✔ A loop has 2 parts: a) body of the loop - It has the statements which are repeated.
- b) control statement - tests the condition

✔ A looping process has 4 steps:

1.Initializing a counter

2.Execution of statements in the loop

3.Test for a specified condition

4.incrementing the counter

Java has 4 loop constructs:-

1.while loop construct

2.do while loop construct

3.for loop construct

4.foreach loop construct

1.while construct

It is the simplest loop. It is an entry controlled. The test condition is evaluated first and if the condition is true, the body of the loop is executed. After execution of the body, the test condition is again tested and if it is true, body is again executed. If the condition is false at the first attempt, the body is not executed at all.

General syntax:

```
initialization;
while (test condition)
{
      Body of loop
 }
```

Ex:

```
int i=1;
while (i<=5)
{
          System.out.println( i);
}
```

2.do while construct

The body of the loop is executed at least once even if condition is false. The condition is checked at the end of the loop

General Syntax:

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

ex:

```
  int i=1;
    do
```

```
{
        System.out.println( i);
        i ++;
}
while (i<=5);
```

3.for statement

It is an entry controlled loop.

general syntax:

```
for (initialization test condition ; increment )
{
        body of the loop
}
```

Ex:

```
for (int i = 1; i < 10; i++)
{
        System.out.println( i );
}
```

Steps in execution of for loop is:

1.Initialization of the control variable is done first(i=1)

2.value of control variable is tested using test condition (i<3)

3.body of the loop is executed

4. Then control variable is incremented (i++)

5.The condition is tested again with new value of control variable

4.Enhanced for loop [for-each loop]

This special loop in java is used to access array values efficiently, without using index .It can also be used with collection.

General Syntax:

```
for (type identifier: expression)
{
        Statements
}
```

Where type - the data type of array or object

identifier - variable

expression -array or a collection

ex:

```
int arr [ ] = {10,20,30};
for ( int k : arr)
        System .out .println (k);
```

ex:

```
String [  ] str = {“ mysore “,”aaa”, “bbb”};
for ( string i : str)
      System .out . println (i);
```

Break and continue statement

- ✔ Java allows a jump from one statement to the end or beginning of a loop or jump out of a loop. Sometimes we would like to skip a part of the loop or leave the loop.
- ✔ Jumps in loops is done using break and continue statements.

Break statement

- ✔ Break is used to jump out of a loop
- ✔ An early exit from a loop is done by using break statement.
- ✔ When a break is written in a loop, the loop is exited immediately and continue with the statement after the for statement.
- ✔ In a nested loop, break causes   exit from the inner loop only.

Ex: break from for loop

```
 for(……..)
 {   if ( condition)
        break;`
        ……..
      ….
}
…….
exit from loop
```

ex: break from inner for loop

```
for (……...)
```

```
    {
         ………
         ………
         for (……)
         {
            if (condition)
            break;
         }
         ……….. Exit from Inner loop
    }
```

continue statement

- ✔ continue statement is used to skip a part of a loop
- ✔ A part of the body of a loop can be skipped under certain conditions.
- ✔ Continue statement causes the loop to be continued with the next iteration by skipping statements in between.
- ✔ It can be used with while, do or for loop
- ✔ In while and do while loop, continue causes control to go directly to the test condition and then to continue the iteration process.
- ✔ In for loop, it goes to the increment section, then to the condition testing.

General syntax

```
    for (          )
    {
      if ( condition )
         continue;
    ……..
    }

  while (condition )
  {
     if (condition)
    continue;
    …….
  }
```

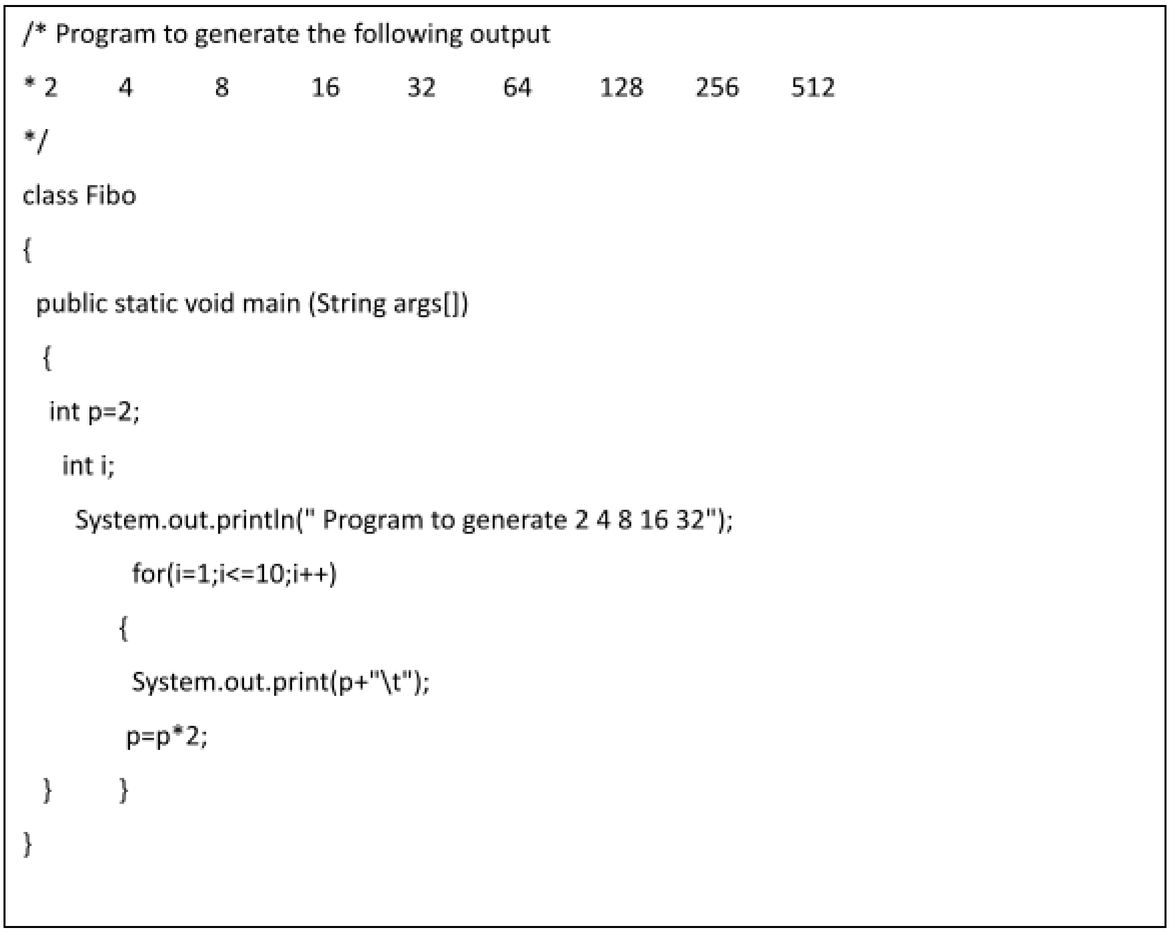
Ex

```
for (i=1;i<=5;i++)
{
   if (i==3)
       continue;
   System.out.println(i);
}
```

Output

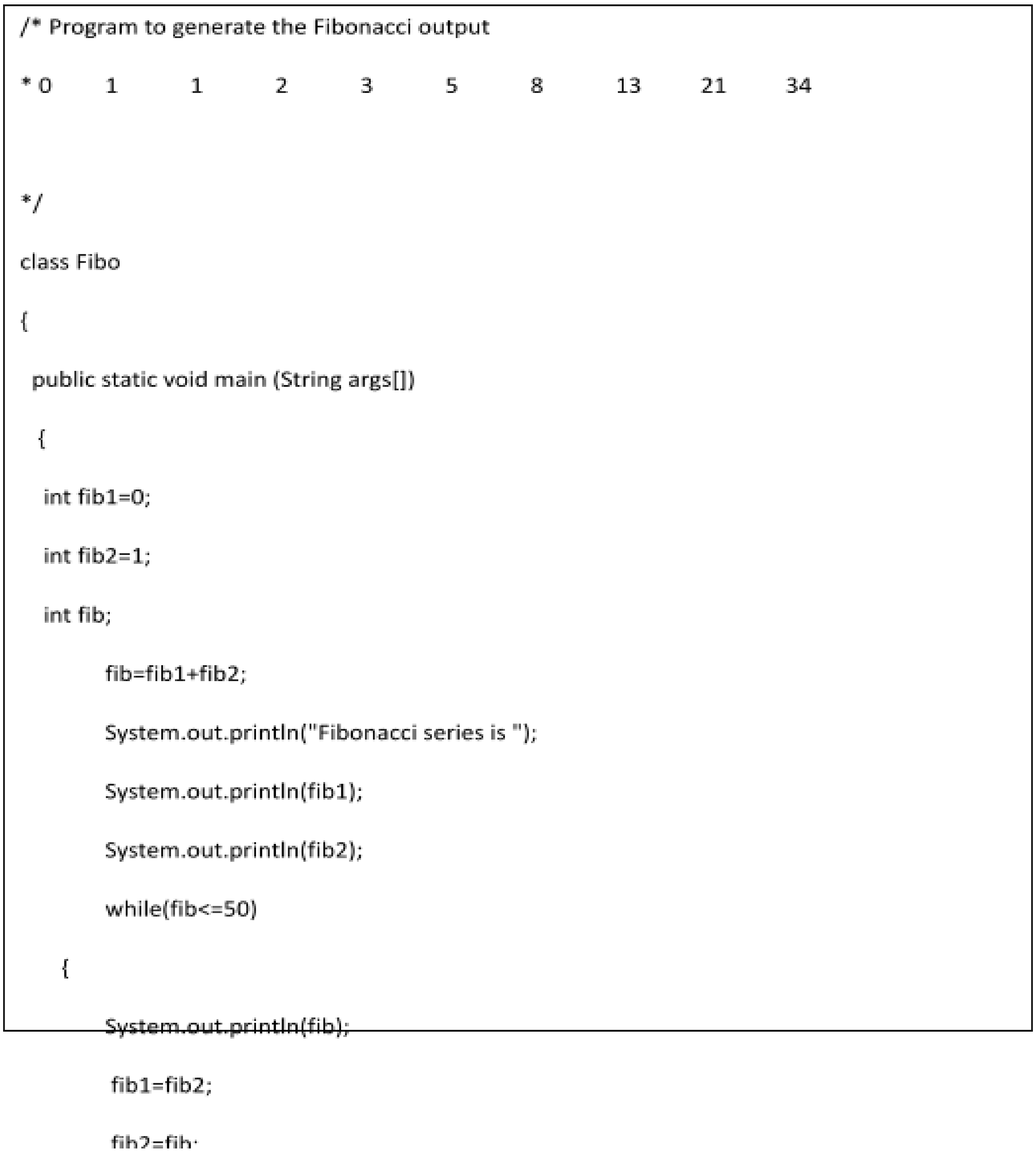
```
1
2
4
5
```

Write a Java method to print the sequence 2,4,8,16,..... using for loop.



```
/* Program to generate the following output
* 2     4     8     16     32     64     128     256     512
*/
class Fibo
{
  public static void main (String args[])
   {
    int p=2;
     int i;
      System.out.println(" Program to generate 2 4 8 16 32");
              for(i=1;i<=10;i++)
            {
              System.out.print(p+"\t");
             p=p*2;
   }        }
}
```

Program to print fobonacci series



```
/* Program to generate the Fibonacci output

* 0      1      1      2      3      5      8      13      21      34



*/

class Fibo

{

  public static void main (String args[])

   {

    int fib1=0;

    int fib2=1;

    int fib;

           fib=fib1+fib2;

           System.out.println("Fibonacci series is ");

           System.out.println(fib1);

           System.out.println(fib2);

           while(fib<=50)

      {

           System.out.println(fib);

            fib1=fib2;

            fib2=fib
```

Labelled loops:

- ✔ In java, we can give a label to a block of statements. A label is any valid java variable name.
- ✔ To give a label to a loop, place it before the loop with a colon at the end.
- ✔ A block of statements can be labelled as shown below.

```
loop1: for ( . . . . . . . .)
                         {
                          ..............
                         ...............
                         }

                         ..............



block1:        for ( . . . . . . . .)
                         {
                         ...............
                         }
block2:        for ( . . . . . . . .)
                         {
                         break block1;
                         ..............
                         }
```

Note that a single break statement causes the control to jump outside the nearest loop and a simple continue statement restarts the current loop. If we want to jump outside a nested loops or to continue a loop that is outside the current one, then we may have to use the labeled break and labeled continue statements.

<u>Example:</u>

```
outer:  for (int i=1;i<=10 i++)
                 {
                 For( int j=1;j<=10 j++)
                 {

                System.out.println("   "+ i*j );

                  if (i == j)

                 continue outer;

                 }
         }
```

Here the continue statement terminates the inner loop when i==j and continue with the next iteration of the outer loop.

```
Loop1:          for (int i=1;i<=10 i++)
                {
Loop2:          while (count<100)
                        {          y= I * x;
                        if ( y> 50)
               break loop1;
                 ……………..
                 ………………
                ………..…….
                }
           ……………..
          }
```

Here, the label loop1 labels the outer loop and therefore the statement

```
Break   loop1;
```

It causes the execution to break out of both the loops.