Decision making constructs

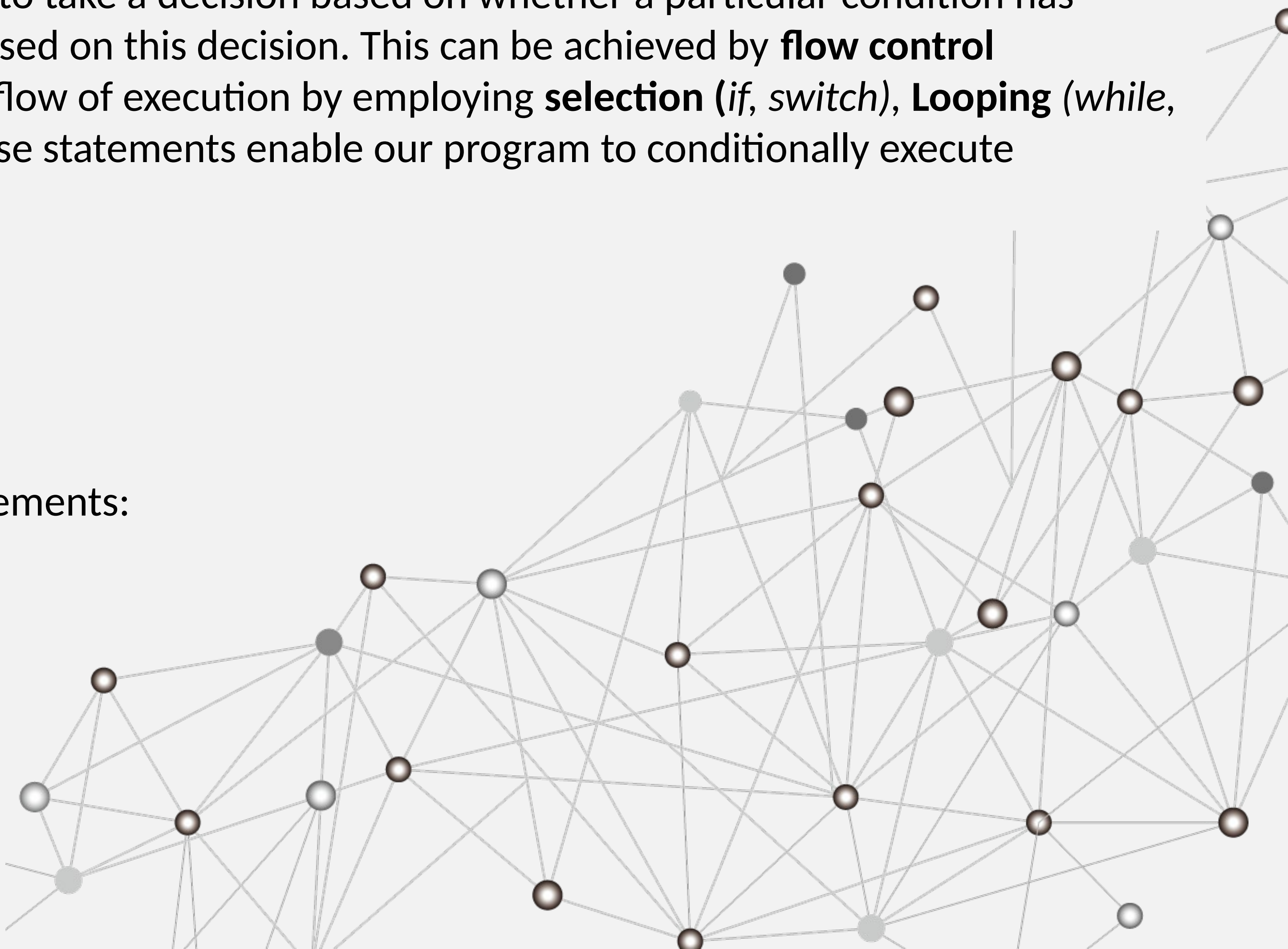
INTRODUCTION

In most of the Java program, generally the instructions are executed in order they are written within the program. Each instruction executes only once. These types of program are unrealistically simple, since they do not include any logical control statements. Sometimes our program needs to take a decision based on whether a particular condition has occurred or not and executes certain statements based on this decision. This can be achieved by **flow control statements.** *Flow control statements*, break up the flow of execution by employing **selection (***if*, *switch)*, **Looping** *(while*, *do while*, *for)*, and **Branching** *(break*, *continue)*. These statements enable our program to conditionally execute particular block of code.

SELECTION STATEMENTS

There are following types of different selection statements:

- if statement
- if … else statement
- if … else if ladder
- nested if statement
- switch statement

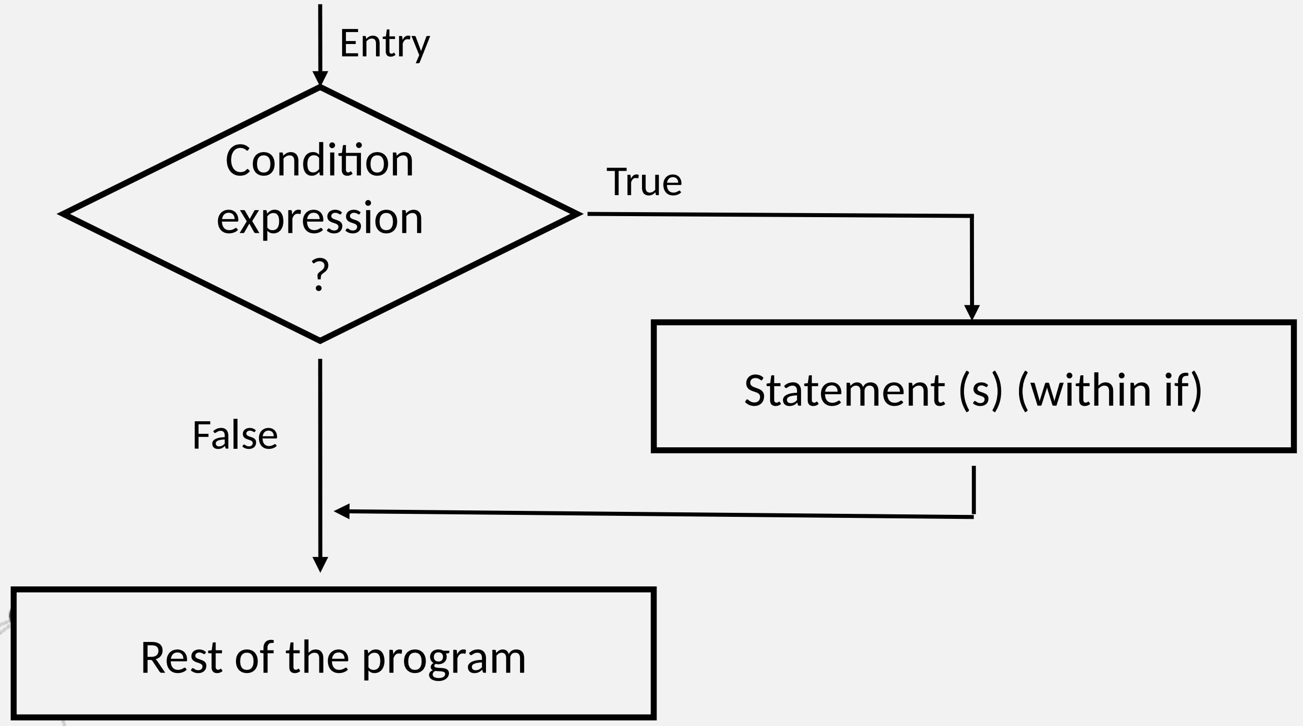


1. if statement

The **if statement** is a powerful decision making statement and is used to control the flow of execution of statements. It is basically a two way decision statement and used in conjunction with a conditional expression.

Syntax : `if(conditional expression)`

A simple *if statement* evaluates a conditional expression (*conditional expression must be a boolean expression that produce boolean results i.e., either true or false)* and if that evaluation is **true** then the specified action is taken, otherwise the *if block* is skipped and execution continues with the rest of the program.



Syntax: for multiple statements within if *(it can also be used for single statement within if)*

```
    if(Conditional expression)
    {
       Statement-1;
       Statement-2;
       .....
       Statement-N;
    }
    [Rest of the program]
```

(always execute whether conditional expression evaluates to true or false)

Example: Check for equality

```
    if(x==y)  //checking for equality
    {
       System.out.println("x and y are equal!");
    }
```

Here, if the value of **'x'** is equal to the value of variable **'y'**, only then it will display

```
    x and y are equal.
```

For multiple within if statements *opening and closing braces* are required. We do not require *opening and closing braces* for a single statements within if, so we can write the code as follows:

```
    if(x==y)
       System.out.println("x and y are equal");
```

PROGRAM 3-1

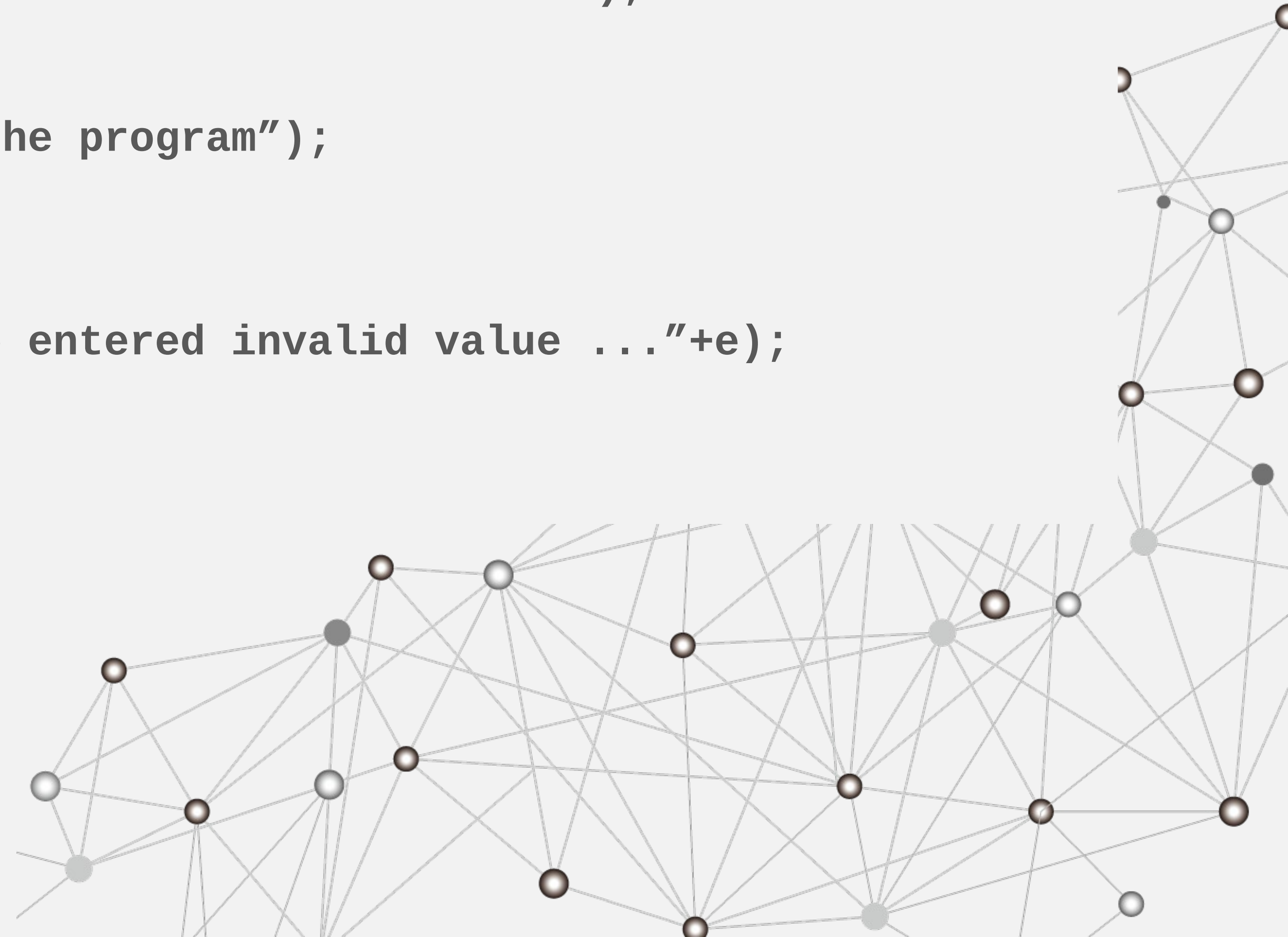
Description: *This program demonstrates the concept of simple 'if' statement. In this program we will take two numbers from the user as input and if second number is not 'zero' only then we divide the first number by second number.*

```
import java.io.*;
class SimpleIF
{
   public static void main(String args[])
   {
      int num1,num2; //To take user input
      int res;   //To store the result
      DataInputStream r=new DataInputStream(System.in);
      try
      {
         System.out.println(“Enter Number-1: “);
         num1=Intger.parseInt(r.readLine());
         System.out.println(“Enter Number-2: “);
         num2=Integer.parseInt(r.readLine());
```

```
        if(num2!=0)  //checking whether the num2 is ‘zero’ or not
        {
           //if num2 is not ‘zero’, only then we enter in this block
           res=num1/num2;
           System.out.println(“Result of division is: “+res);
        }
        //rest of the code
        System.out.println(“End of the program”);
        }
    catch(Exception e)
    {
        System.out.println(“You have entered invalid value ...”+e);
    }
}
}
```

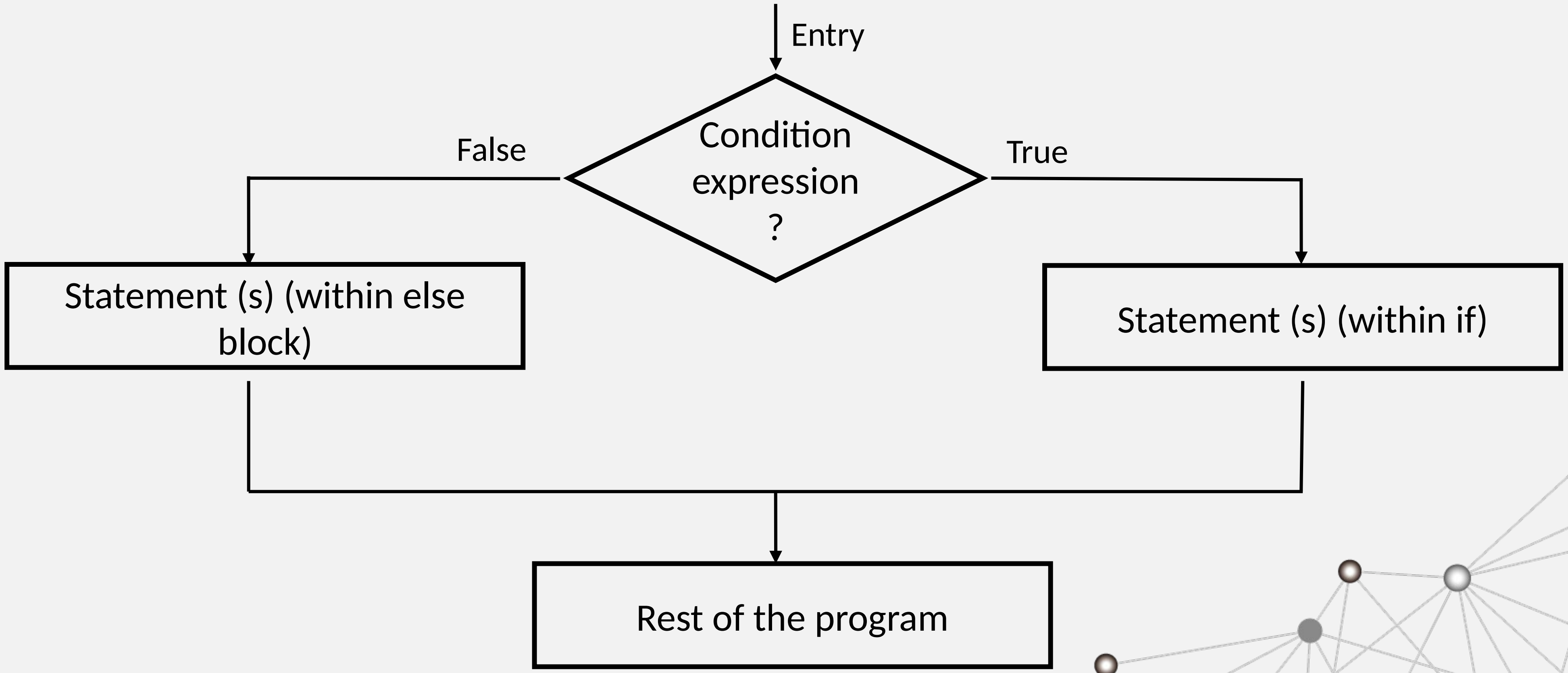
Output of Program 3-1

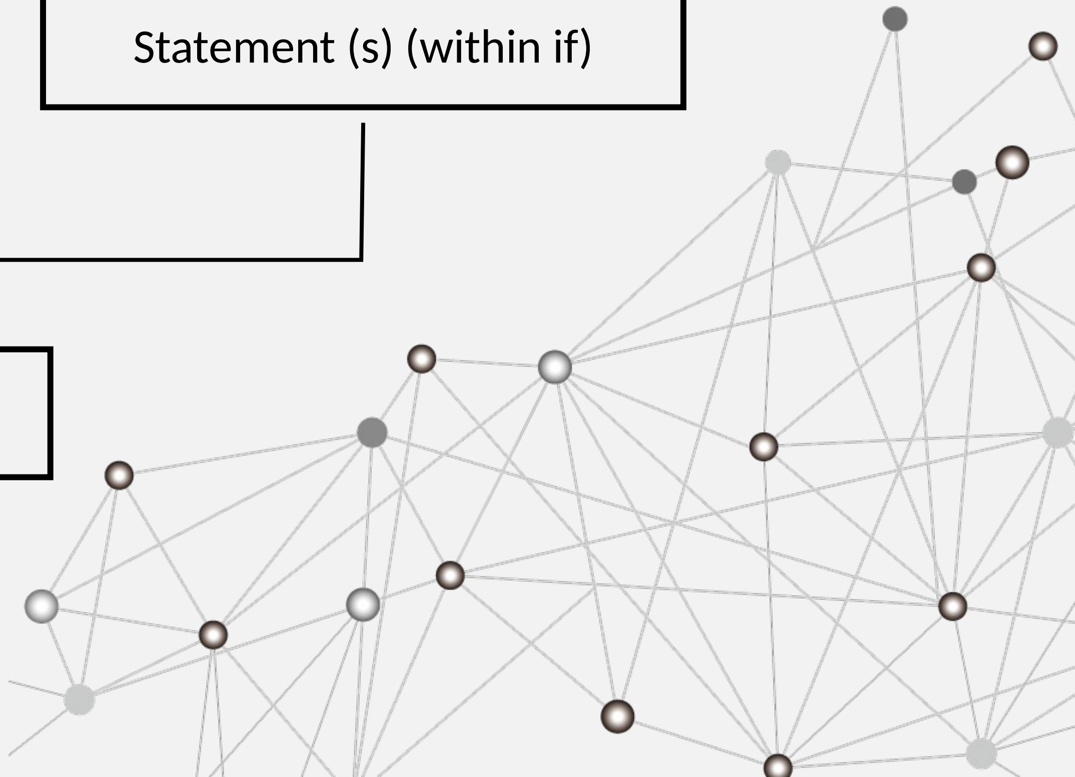
```
Enter Number-1: 34
Enter Number-2: 2
Result of division is: 17
End of the program
```



2. if .... else statement

This statement evaluates an expression and perform one action if that evaluation is **true** or different action if it **false.**





Syntax: if.....else statement

```
if(Conditional expression)
{
   True-block statement(s);      //if block
}
else
{
   False-block statement(s);     //else block
[Rest of the program]
```

(always execute whether conditional expression evaluates to true or false)

Example: Find greater among two numbers

```
if(num1>num2)
{
   System.out.println("Number-1 is greater");
}
else
{
   System.out.println("Number-2 is greater");
```

Here, if the value of **'num1'** is greater than the value of variable **'num2',** then the output will be displayed as `"Number-1 is greater"` and **else block** will be skipped, otherwise, **if block will be skipped** and the control will jump to the **else block** and the output will be displayed as `"Number-2 is greater"`

PROGRAM 3-2

Description: *This program demonstrates the concept of simple 'if …. else' statement. In this program we will take a number from the user as input and check whether that number is* ***'even'*** *or* ***'odd'****.*

```
import java.io.*;
class EvenOdd
{
   public static void main(String args[])
   {
      int num;  //To take user input
      DataInputStream r=new DataInputStream(System.in);
      try
      {
         System.out.println("Enter Number: ");
         num=Intger.parseInt(r.readLine());

         /* As we know a number is EVEN if it divisible by 2, which
means that after dividing a number 2 remainder will be 0. if          number is
not divisible by 2 then the number is ODD and there          will be remainder 1.
So, we have tested that whether the           remainder after dividing number by
2 is 0 or not. If                remainder is 0 then number is EVEN otherwise
ODD.
         */
```

```
            if(num%2==0)
            {
                //if remainder is 0
                System.out.println(num+” is an Even number”);
            }
            else
            {
                //if remainder is not 0
                System.out.println(num+” is an Odd number”);
            }
        }
        catch(Exception e)
        {
            System.out.println(“You have entered invalid value”+e);
        }
    }
}
```

Output of Program 3-2

```
Enter a Number : 45
45 is an Odd number
```

PROGRAM 3-3

Description: *This program demonstrates the concept of simple ‘if …. else’ statement. In this program we will take an alphabet from the user as input and check whether that alphabet is a* ***‘vowel’*** *or* ***‘consonant’.***

```
import java.io.*;
class VowelConsonant
{
   public static void main(String args[])
   {
      char ch;  //To take user input
      DataInputStream r=new DataInputStream(System.in);
      try
      {
         System.out.println(“Enter an Alphabet: “);
         num=(char)r.read(); // converting user input into a character

         /* An alphabet can be vowel only if it either ‘a’ or ‘e’ or
   ‘i’ or ‘u’ (both upper case and lower case, otherwise an            alphabet
is a consonant         */
```

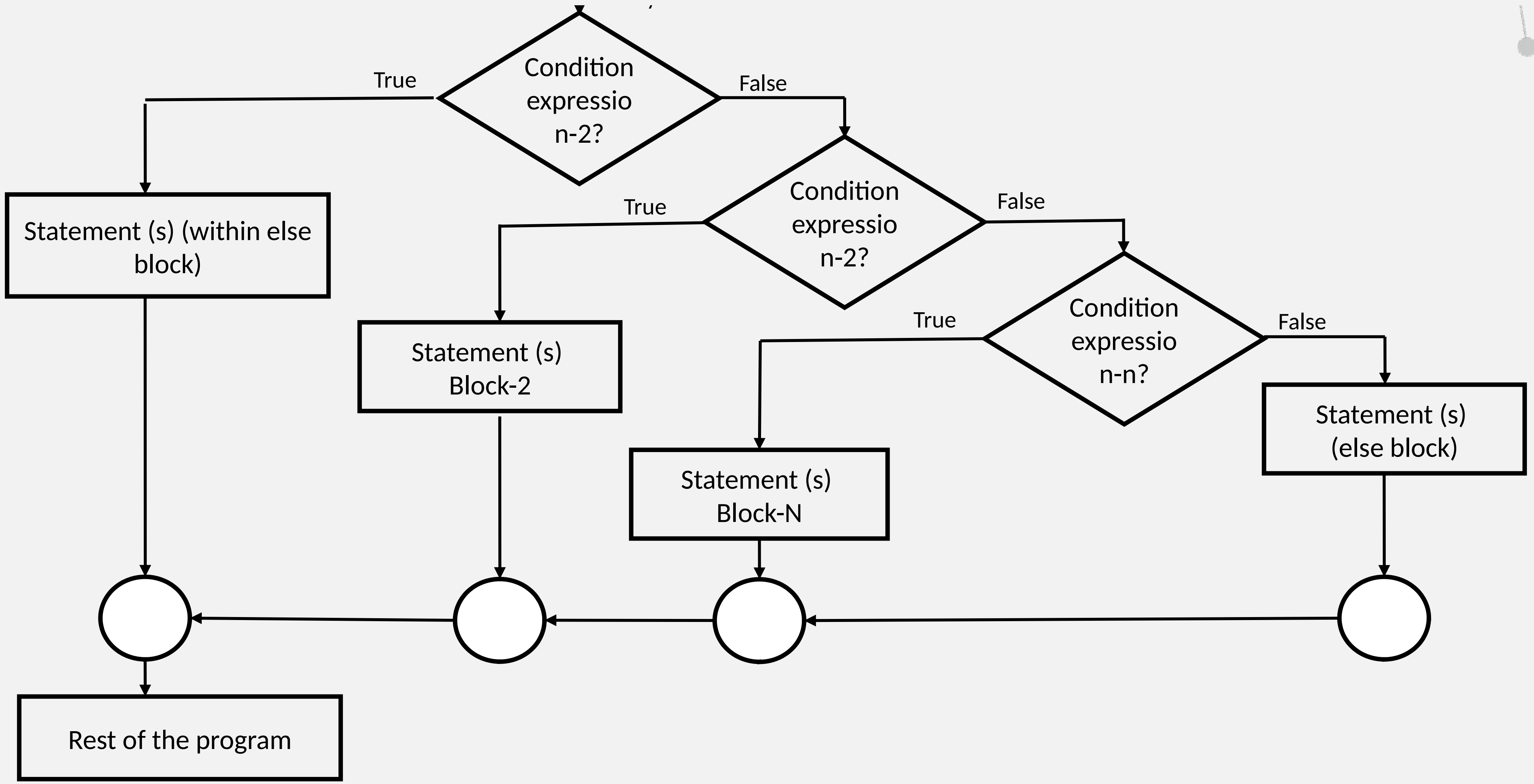
```
          if(ch==‘a’ || ch==‘e’ || ch==‘i’ || ch==‘o’ || ch==‘u’ ||
  ch==‘A’||ch == ‘E’ || ch==‘I’ || ch==‘O’ || ch==‘U’)
          {
             System.out.println(“You have entered a Vowel”);
          }
          else
          {
             System.out.println(“You have entered a consonant”);
          }
       catch(Exception e)
       {
          System.out.println(“You have entered invalid alphabet”);
       }
    }
}
```

Output of Program 3-3

```
Enter an Alphabet: i
You have entered a Vowel
```

3. if .... else if ladder

The ‘**if …. else if’ ladder** is a way of putting multiple if’s together when multipath decisions are involved. It is one of the types of decision making and branching statements. A multipath decision is a chain of ***if’s*** in a which the statement associated with each **‘else’** is an **‘if’.**



Syntax: if…..else if ladder

```
if(Conditional expression-1)
   statement(s) Block-1;
else if (Conditional expression-2)
   statement(s) Block-2;
.
.
.
else if (Conditional expression-n)
   statement(s) Block-n;
else
   Default statement(s) Block;
[Rest of the program]
```

(always execute whether a match is found or not)

Example: Check whether the number is positive, negative or zero.

```
if(num1>0)
{
   System.out.println(“Number is positive”);
}
else if(num<0)
{
   System.out.println(“Number is negative”);
}
```

```
    else
    {
        System.out.println(“Number is 0”);
    }
```

PROGRAM 3-

Description: *This program demonstrates the concept of simple ‘if .... else if ladder’ statement. In this program we will take a percentage marks of a student from the user as input and check whether that students got ‘First’, ‘Second’ or ‘Third’ division.*

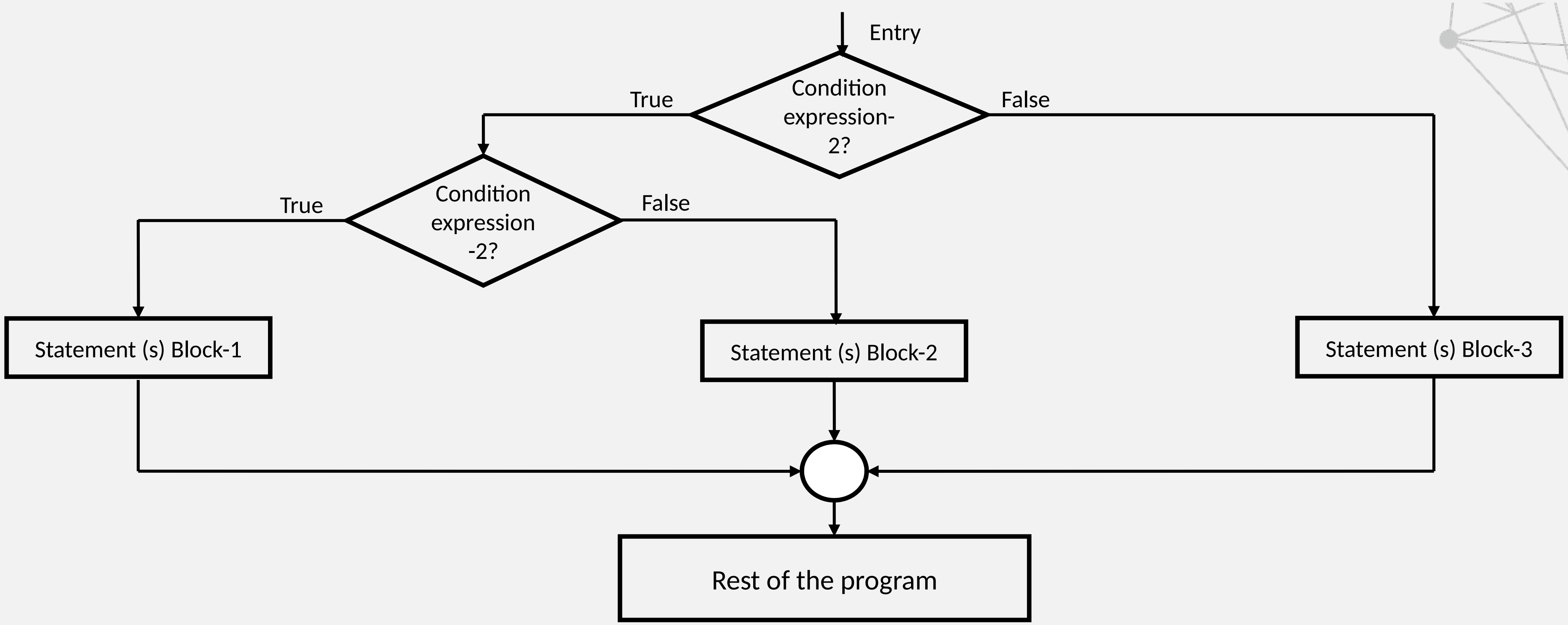
```
import java.io.*;
class Division
{
    public static void main(String args[])
    {
        double percentage;  //To take user input
        DataInputStream r=new DataInputStream(System.in);
        try
        {
            System.out.println(“Enter Percentage marks: “);
            percentage=Double.parseDoube(r.readLine());
            if(percentage>=60)
            {
                System.out.println(“You go first division”);
            }
            else if(percentage>=50 && percentage<60)
            {
                System.out.println(“You got second division”);
            }
            else if(“percentage>=33 && percentage<50)
            {
                System.out.printlnL(“You got third division”);
            }
            else
            {
                System.out.println(“You failed”);
            }
        }
        catch(Exception e)
        {
            System.out.println(“You have entered invalid value”);
        }
    }
}
```

Output of Program 3-4

```
Enter Percentage Marks :56
You got second division
```

4. Nested if statement

‘if’ within **‘if’** is called as **‘nested if’** statement. This type of selection is used when there are series of decisions involved.



Syntax: Nested if statement

```
    if(Conditional expression-1)
    {
        if(Conditional expression-2)
        {
            Statement(s) Block-1;
        }
        else
        {
            Statement(s) Block-2;
        }
    }
    else
    {
        Statement(s) Block-3;
    }
    [Rest of the program]
```

Example: Find Greater among three number

```
    if(a>b)   //First condition
    {
        if(a>c)   //Second condition
        {
            System.out.println(“a is greater”);
        }
```

```
    else      //else block-1
    {
        System.out.println(“c is greater”)
    }
else      //else block-2
{
    if(b>c)
    {
        System.out.println(“b is greater”);
    }
    else      //else block-3
    {
        System.out.println(“c is greater”);
    }
}
```

PROGRAM 3-5

Description: *This program demonstrates the concept of simple ‘nested if’ statement. In this program we will take a character from the user as input and check whether the entered character is an **‘alphabet’** (uppercase or lowercase), a **‘digit’** or a **‘special symbol’.***

```
import java.io.*;
class DemoNestedIF
{
    public static void main(String args[])
    {
        char value;  //To take user input
        DataInputStream r=new DataInputStream(System.in);
        try
        {
            System.out.println(“Enter a character: “);
            value=(char)r.read();

            //check for alphabet. It can be uppercase or lowercase
            if((value>=‘a’&&value<=‘z’)||(value>=‘A’&&value<=‘Z’))
            {
                if(value>=‘a’&&value<=‘z’) //check for lowercase
                {
                    System.out.println(“A lowercase alphabet”);
                }
                else
                {
                    System.out.println(“A uppercase alphabet”);
}
}
```

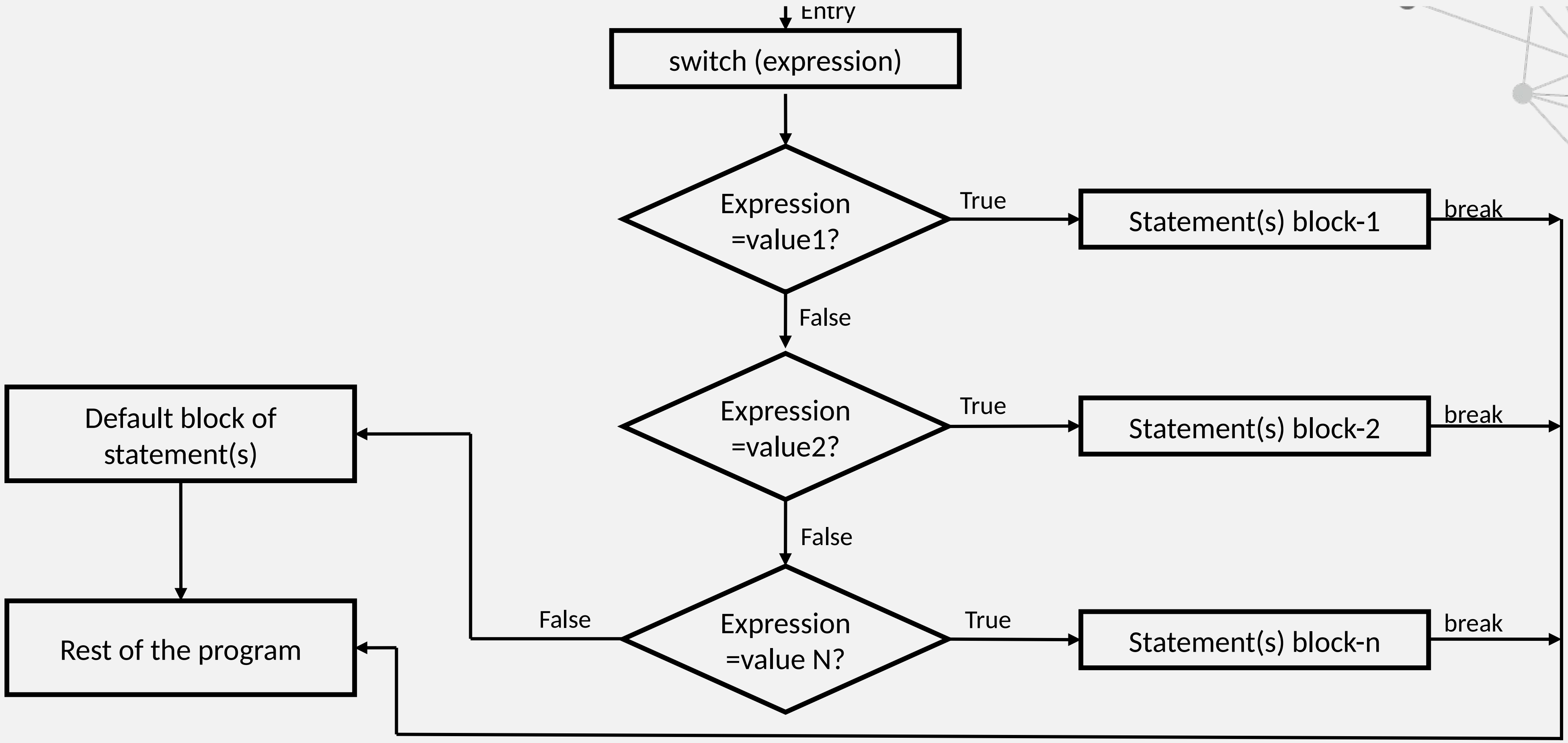
```
            else   //if character is not alphabet
            {
               if(value>=‘0’ && value<=‘9’)
               {
                  System.out.println(“A digit”);
               }
               else
               {
                  System.out.println(“A special symbol”);
               }
            }
         }
         catch(Exception e)
         {
            System.out.println(“You have entered an invalid value”);
         }
      }
   }
```

Output of Program 3-5

```
Enter a character: h
A lowercase alphabet
```

5. switch

Another way to control the flow of our programs is with something called a **'switch'** statement. Unlike simple *'if'* and *'if else'* statements, the switch statement can have a number of possible execution paths. That is why it is also called as ***multi-way decision making statement.*** A switch statement is useful when we need to select one of several alternatives based on the value. It also works enumerated types, the *String class*. A switch can be used for menu-driven programming.



Syntax: switch statement

```
switch (expression)
{
    case value1:
        Statement(s) block-1;
        break;
    case value2:
        Statement(s) block-2;
        break;
    case valueN:
        Statement(s) block-N;
        break;
    default:
        Default-block of statement(s);   //if no match is found
        break;

[Rest of the program]
```

(always execute whether a match is found or not)

Example: Display colour name: 1-Red, 2-blue, 3-Green, Otherwise, invalid colour

```
int colour=2;
switch(colour)
{
    case 1:
        System.out.println(“Red”);
```

```
        break;
    case 2:
        System.out.println(“Blue”);
        break;
    case 3:
        System.out.println(“Green”);
    default:
        System.out.println(“Invalid colour”);
        break;
}
System.out.println(“End of Program”);
```

PROGRAM 3-6

Description: *This program demonstrates the concept of simple ‘switch’ statement. In this program we will take a day number from the user and display the day name*

```
import java.io.*;
class Day
{
    public static void main(String args[])
    {
        int day;  //To take user input
        DataInputStream r=new DataInputStream(System.in);
```

```
try
{
   System.out.println(“Enter a Day Number: “);
   day=Integer.parseInt(r.readLine());
   switch(day)
   {
      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;
      case 7:System.out.println(“Saturday”);
          break;
      default:System.out.println(“Invalid Day Number”);
   }
   System.out.println(“End of program”);
}
```

```
        catch(Exception e)
        {
           System.out.println(“You have entered invalid value”);
        }
    }
}
```

Output of Program 3-6

```
Enter a Day Number : 4
Wednesday
End of program
```

PROGRAM 3-7

Description: *This program demonstrates the concept of* ***Menu-driven programming with switch statement****. In this program number of options will be given in menu. We will take an input from the user to choose one of the options of the menu. Depending upon the user’s choice, further actions will be taken*

```
import java.io.*;
class SwitchStatement
{
    public static void main(String args[])
    {
        int num1, num2; //to take two numbers from the user
```

```
char choice;  //to store user choice among +,-,*,/
int add,subtract,mul,div;  //to store results
DataInputStream r=new DataInputStream(System.in);
try
{
    System.out.println(“Enter Number-1: “);
    num1=Integer.parseInt(r.readLine());
    System.out.println(“Enter Number-2: “);
    num2=Integer.parseInt(r.readLine());

    System.out.println(“\nEnter + for Addition”);
    System.out.println(“Enter – for Subtraction”);
    System.out.println(“Enter * for Multiplication”);
    System.out.println(“Enter / for Division”);

    System.out.println(“Enter you Choice: “);
    choice=(char)r.read();

    switch(choice)
    {
        case ‘+’:
            add=num1+num2;
            System.out.println(“Sum is : “+add);
            break;
```

```
                case ‘-’:
                    subtract=num1-num2;
                    System.out.println(“Result of Subtraction:
                “+subtract);
                    break;
                case ‘*’:
                    mul=num1*num2;
                    System.out.println(“Result of Multiplication:
                    “+mul);
                    break;
                case ‘/’:
                    div=num1/num2;
                    System.out.println(“Result of Division: “+div);
                    break;
                default:
                    System.out.println(“Invalid Choice”);
            }
            System.out.println(“End of program!”);
        }
        catch(Exception e)
        {
            System.out.println(“You have entered a wrong value”);
        }
}}
```

Output of Program 3-7

```
Enter Number-1: 65
Enter Number-2: 5

Enter + for Addition
Enter – for Subtraction
Enter * for Multiplication
Enter / for Division
Enter you Choice: /
Result of Division: 13
End of Program!
```

Rules to be remembered while using ‘switch’

1. The switch statement, ‘expression’ must be of an int, char, byte or short type. Switch also works with enumerated types, the ***String class***, and a few special classes that wrap certain primitive type such as *Character*, *Byte*, *Short and Integer*. Lets take an example to demonstrate how switch work with *String class*.

```
String str=“Red”;
switch(str)
{
    case “Red”:
        System.out.println(“You choose Red colour”);
        break;
    case “Blue”:
        System.out.println(“You choose Blue colour”);
        break;
    default: System.out.println(“Invalid Choice”);
        break;
```

2. Cases value must be constants or consonant expression, not variable, For example:

```
int value=15;
int choice=10;
switch(choice)
{
    case 2*5:    //valid constant expression equivalent to 10
        System.out.println(“First block”);
        break;
    case value:  //Error: constant expression required
        System.out.println(“Second block”);
        break;
    default: System.out.println(“invalid choice”);
        break;
}
```

3. All the case values must be ***unique*** which means that duplicate cases are not allowed/

```
int choice=1;
switch(choice)
{
   case 1:   System.out.println(“First block”);
         break;
   case 2:   System.out.println(“Second block”);
         break;
   case 2:   //Duplicate case label
         System.out.println(“Third block”);
         break;
   case 3:   System.out.println(“Fourth block”);
         break;
   default:  System.out.println(“Invalid choice”);
         break;
}
```

4. Multiple cases are allowed with a single block of code.

```
int choice=5;
switch(choice)
{
   case 1:
   case 5:
    case 2*5: //constant expression, equivalent to 10
    //this block is executed for multiple case labels ,i.e., 1,5 and 10
        System.out.println(“First block”);
        break;
    case 2:
    case 6:
    //this block is executed for multiple case labels, i.e., 2 and 6
        System.out.println(“Second block”);
        break;
    case 3: System.out.println(“Third block”);
        break;
    default: System.out.println(“Invalid choice”);
        break;
}
```

5. Each block of statements should be ended with a ‘break’ statement. The break statements are necessary because without them, statements in switch blocks ***‘fall through’***, which means that all statements after the matching *‘case label’* are executed in sequence until a **break** statement is encountered.

```
int month=3;
switch(month)
{
    case 1: System.out.println(“January”);
    case 2: System.out.println(“February”);
    case 3: System.out.println(“March”);
```

```
       case 4: System.out.println(“April”);
       case 5: System.out.println(“May”);
       case 6: System.out.println(“June”);
       case 7: System.out.println(“July”);
          break;
       case 8: System.out.println(“August”);
       case 9: System.out.println(“September”);
       case 10: System.out.println(“October”);
       case 11: System.out.println(“November”);
       case 12: System.out.println(“December”);
       default: System.out.println(“invalid choice”);
    }
```

Output of above code snippet will be as follows:

```
    March
    April
    May
    June
    July
```

6. The ***‘default’*** is an optional case which means that this is the programmer’s wish whether to use a default case in the program while using switch statement or not. If a default case is there, then it will be executed only if the value of the expression does not match with any of the case values.

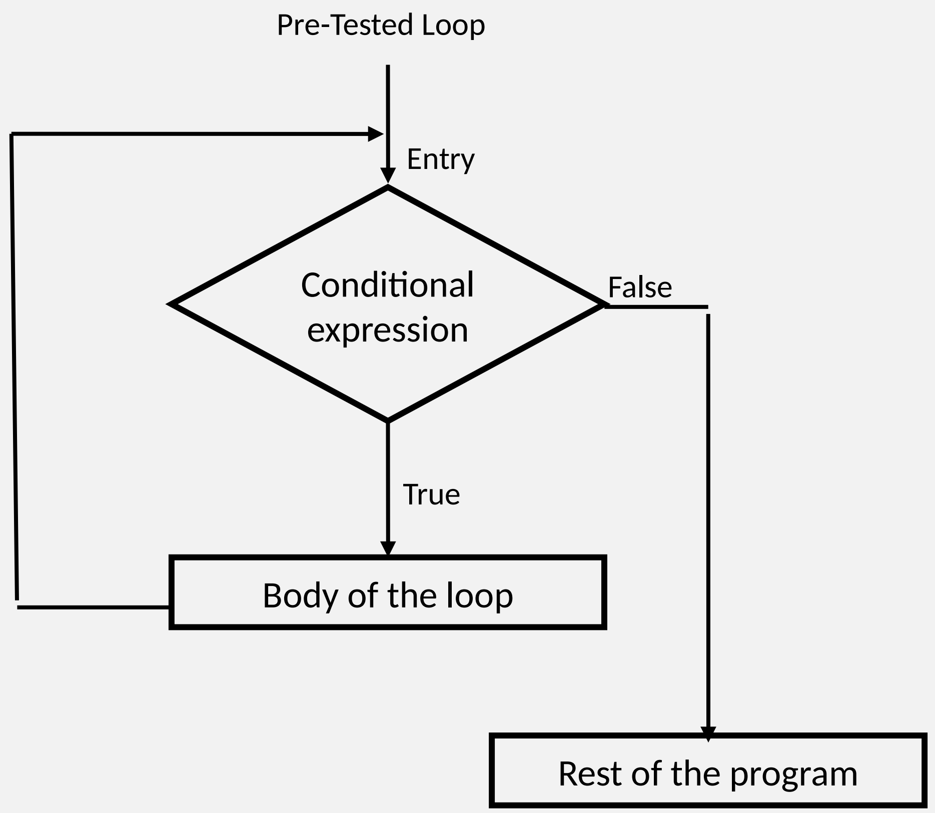
7. In ‘**switch**’ statement, expression must not be **null.** Otherwise a ***‘NullPointerExecption’*** will occur.

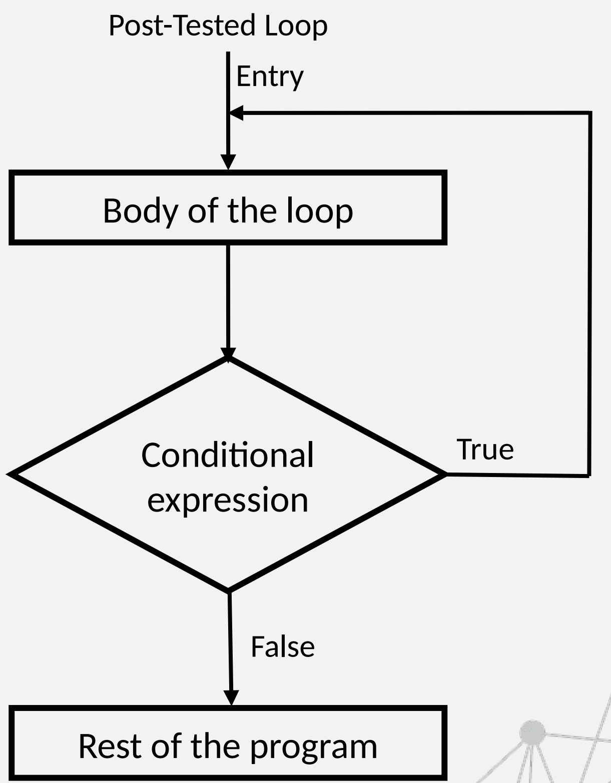
LOOPING STATEMENTS

The process of repeatedly executing a block of statements while a specified condition evaluates to ***‘true’*** is called as **Looping.** The block of the loop that contains the statement to be repeated is called the ***body of the loop.*** A one-time execution of a body of the loop is referred to as an *iteration of the loop*. A condition (*conditional expression*), which is used to control the repeatedly execution of the block, must return a boolean value *(either true or false)*. Loops are used to concise programs. It includes for steps:

1. Setting and initialization of a counter
2. Test for a specified condition for execution of the loop
3. execution of the block of statements in the loop
4. Increment or Decrement the counter

A loop can be a ***pre-tested*** loop or ***post-tested*** loop. A pre-tested loop is also known as ***entry controlled*** loop and a post-tested loop is known as ***exit controlled*** loop. In pre-tested loops, a *conditional expression* is evaluated first and if that evaluation is *true* only then the *body of the loop* is executed, otherwise, the loop terminated and control will directly jump to the *statement*, immediate to the body of the loop, whereas, in post-tested loops, first the *body of the loop* is executed then a *conditional expression* is evaluated and if that evaluation is *true* only then the *body of the loop* is executed again, otherwise, the loop terminated and control jumps to the next statement.





Java provides following types of loops:

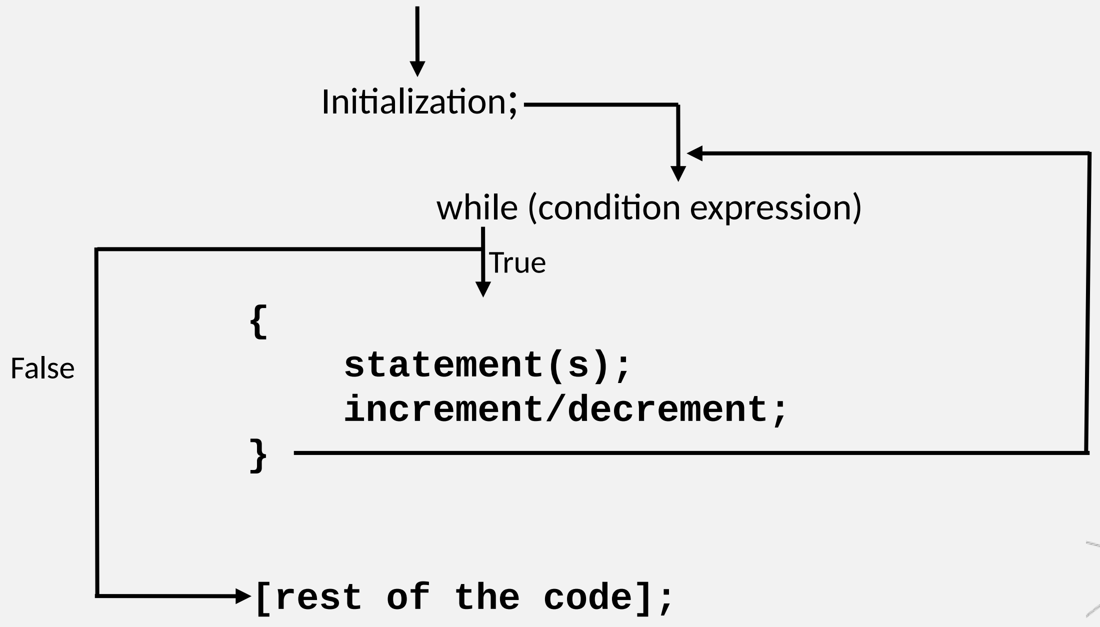
1. **while loop**
2. **do while loop**
3. **for loop**

while loop

A ***while loop*** continually executes a block of code while a particular condition is **true.**

Syntax: while loop

```
initialization;
while (conditional expression)
{
    statement(s);
    increment/decrement;
}
[rest of the code];
```



A while loop consists of a block, which is a collection of statements that form the body of the loop. A while loop checks the condition before the execution of the block *(block of the loop)*, so the control structure is known as **pre-test loop.**

Example: Suppose there is a situation to display number from 1 to 5.

```
int num=1;   //initialization
while(num<=5) //condition: it terminates loop when num>5
{
   System.out.println(num);
   num++; //increment value of num by 1, i.e., num=num+1;
}
System.out.println("End of loop");
```

The above code executes as follows:

at num=1	1<=5, which is true, execute body of loop	display num, i.e. 1	num=num+1
			num=1+1=2

at num=2	2<=5, which is true, execute body of loop	display num, i.e. 2	num=num+1
			num=2+1=3

at num=3	3<=5, which is true, execute body of loop	display num, i.e. 3	num=num+1
			num=3+1=4

at num=4	4<=5, which is true, execute body of loop	display num, i.e. 4	num=num+1
			num=4+1=5

at num=5	5<=5, which is true, execute body of loop	display num, i.e.5	num=num+1
			num=5+1=6

at num=6	6<=5, which is false, exit from the loop	display "end of loop"	

We can implement **an infinite loop** using **while** statement as follows:

```
    while (true)
    {
        Body of loop;
    }
```

PROGRAM 3-8

Description: *In this program we are going to display first ten natural numbers.*

```
class Display
{
    public static void main(String args[])
    {
        int value=1; //initialization
        while(value<=10)    //condition
        {
            System.out.println(value);
            value++;  //incrementing
        }
        System.out.println(“End of the loop!”);
    }
}
```

Output of Program 3-8

```
1
2
3
4
5
6
7
8
9
10
End of the Loop!
```

PROGRAM 3-9

Description: *In this program we are going to display first ten natural numbers.*

```
import java.io.*;
class Factorial
{
   public static void main(String args[])
   {
      int fact=1;  //to store factorial of n
```

```
        int n; //number, whose factorial is to be evaluated
        DataInputStream r=new DataInputReader(Sytem.in);
        try
        {
           System.out.println(“Enter n: “);
           n=Integer.parseInt(r.readLine());

           while (n>=1)    //terminates when n<1
           {
              fact=fact*n;
              n--;
           }
           System.out.println(“Factorial is : “+fact)l
        }
        catch(Exception e)
        {
           System.out.println(“You have entered a wrong value”);
        }
    }
}
```

Output of Program 3-9

```
Enter n: 5
Factorial is: 120
```

The above code executes as follows:

at fact=1 and n=5	5>=1, which is true, execute body of loop	fact=fact*n
		fact=1*5=5
			n - -
			n=5-1=4

at fact=5 and n=4	4>=1, which is true, execute body of loop	fact=fact*n
		fact=5*4=20
			n - -
			n=4-1=3

at fact=20 and n=3	3>=1, which is true, execute body of loop	fact=fact*n
		fact=20*3=60
			n - -
			n=3-1=2

at fact=60 and n=2	2>=1, which is true, execute body of loop	fact=fact*n
		fact=60*2=120
			n - -
			n=2-1=1

at fact=120 and n=1	1>=1, which is true, execute body of loop	fact=fact*n
		fact=120*1=120
			n - -
			n=1-1=0

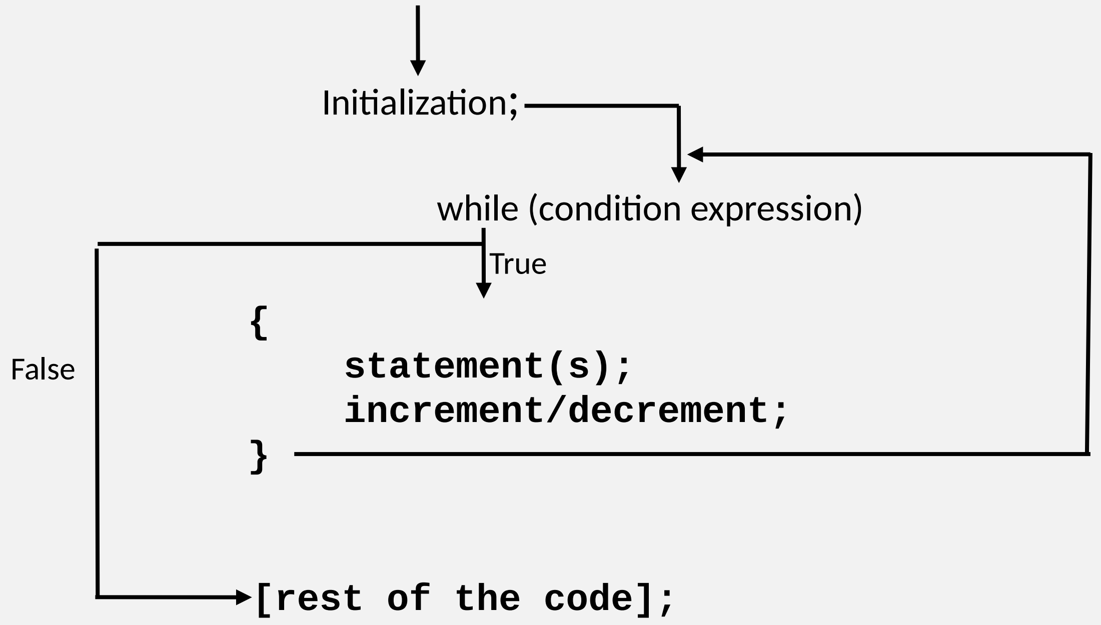
at fact=120 and n=0	0>=1, which is false, exit from the loop	display “factorial is: 120”	

do-while loop

In, **do-while loop,** the code within the block is executed and then the condition is evaluated. If the condition is true the code is executed again. This repeats until the condition becomes false.
A *d0-while loop* checks the condition after the block is executed, so the control statement is known as **post-test loop.**
The body of the loop is executed at least once whether the condition is true or not.

Syntax: do-while loop

```
initialization;
do
{
   statement(s);
   increment/decrement;
}while (conditional expression);
[rest of the code];
```



Example: Suppose there is a situation to display number from 1 to 5.

```
int num=1;   //initialization
do
{
   System.out.println(num);
   num++; //increment value of num by 1, i.e., num=num+1;
} while(num<=5); //condition: it terminates loop when num>5
System.out.println(“End of loop”);
```

The above code executes as follows:

at num=1	No condition is checked and we enter into the body of the loop.	display num, i.e. 1	num=num+1
			num=1+1=2

at num=2	2<=5, which is true, execute body of loop	display num, i.e. 2	num=num+1
			num=2+1=3

at num=3	3<=5, which is true, execute body of loop	display num, i.e. 3	num=num+1
			num=3+1=4

at num=4	4<=5, which is true, execute body of loop	display num, i.e. 4	num=num+1
			num=4+1=5

at num=5	5<=5, which is true, execute body of loop	display num, i.e.5	num=num+1
			num=5+1=6

at num=6	6<=5, which is false, exit from the loop	display “end of loop”	

PROGRAM 3-10

Description: *In this program we will find the sum of first 10 natural number*

```
class NaturalSum
{
   public static void main(String args[])
   {
      int sum=0;
      int N=1; //initialization
      do
      {
         sum=sum+N;
         N++;
      }while(N<=10);   //condition
      System.out.println(“Sum of first 10 Natural numbers is: “+sum);
      System.out.println(“   End of the program!”);
   }
}
```

Output of Program 3-10

```
Sum of first 10 Natural Numbers is: 55
End of the program!
```

The above code executes as follows:

At sum=0 and N=1	No condition is checked and we enter into the body of the loop	Evaluate sum=sum+n sum=0+1=1	Evaluate N++ N=1+1=2
At sum=1 and N=2	N<=10, i.e., 2<=10, true, execute the body of the loop	Evaluate sum=sum+n sum=1+2=3	Evaluate N++ N=2+1=3
At sum=3 and N=3	N<=10, i.e., 3<=10, true, execute the body of the loop	Evaluate sum=sum+n sum=3+3=6	Evaluate N++ N=3+1=4
At sum=6 and n=4	N<=10, i.e., 4<=10, true, execute the body of the loop	Evaluate sum=sum+n sum=6+4=10	Evaluate N++ N=4+1=5
At sum=10 and n=5	N<=10, i.e., 5<=10, true, execute the body of the loop	Evaluate sum=sum+n sum=10+5=15	Evaluate N++ N=5+1=6
At sum=15 and n=6	N<=10, i.e., 6<=10, true, execute the body of the loop	Evaluate sum=sum+n sum=15+6=21	Evaluate N++ N=6+1=7
At sum=21 and n=7	N<=10, i.e., 7<=10, true, execute the body of the loop	Evaluate sum=sum+n sum=21+7=28	Evaluate N++ N=7+1=8
At sum=28 and n=8	N<=10, i.e., 8<=10, true, execute the body of the loop	Evaluate sum=sum+n sum=28+8=36	Evaluate N++ N=8+1=9
At sum=36and n=9	N<=10, i.e., 9<=10, true, execute the body of the loop	Evaluate sum=sum+n sum=36+9=45	Evaluate N++ N=9+1=10
At sum=45 and n=10	N<=10, i.e., 10<=10, true, execute the body of the loop	Evaluate sum=sum+n sum=45+10=55	Evaluate N++ N=10+1=11
At sum=55 and n=11	N<=10, i.e., 11<=10, which is false, exit the loop	Display “ Sum of first 10 Natural Numbers is: 55” and “End of the Program!”	

PROGRAM 3-

Description: *In this program, a user has three choices i.e. 1-Area of Rectangle, 2-Area of Circle, 3-Exit. Depending on the user's choice required inputs will be taken and corresponding actions is performed. A user can make choice again and again.*

```
import java.io.*;
class MenuDriven
{
   public static void main(String args[])
   {
      int choice;  //to store user choice
      DataInputStream r=new DataInputStream(System.in);
      try
      {
         do
         {
         System.out.println("\n*****ENTER YOUR CHOICE*****);
         System.out.println("Enter 1 for Area of Rectangle");
         System.out.println("Enter 2 for Area of Circle");
         System.out.println("Enter 3 to Exit");

         System.out.println("Enter your Choice: ");
         choice=Integer.parseInt(r.readLine());
```

```
switch(choice)
{
   case 1:
      //Area of rectangle=Length*breadth
      double area, length, breadth;

      System.out.println("Enter Length of Rectangle:
");
      length=Double.parseDouble(r.readLine());
      System.out.println("Enter Breadth of Rectangle:
");
      breadth=Double.parseDouble(r.readLine());

      area=length*breadth;
      System.out.println("Area of Rectangle is
"+area);
      break;


   case 2:
      //Area of circe=pi*radius*radius
      final double pi=3.14
      double cirArea;
      double rd;
```

```
                    System.out.println(“Enter Radius of Circle: “);
                    rd=Double.parseDouble(r.readLine());

                    cirArea=pi*rd*rd;
                    System.out.println(“Area of Circle: “+cirArea);
                    break;

                 case 3:
                    System.exit(1);  //exit all
                 default:  //if no match found
                    System.out.println(“Invalid Choice”);
                 }
              }(while(choice!=3); //condition
              System.out.println(“End of Program
           }
           catch(Exception e)
           {
              System.out.println(“You have entered invalid value”+e);        }
        }
     }
```

Output of Program 3-11

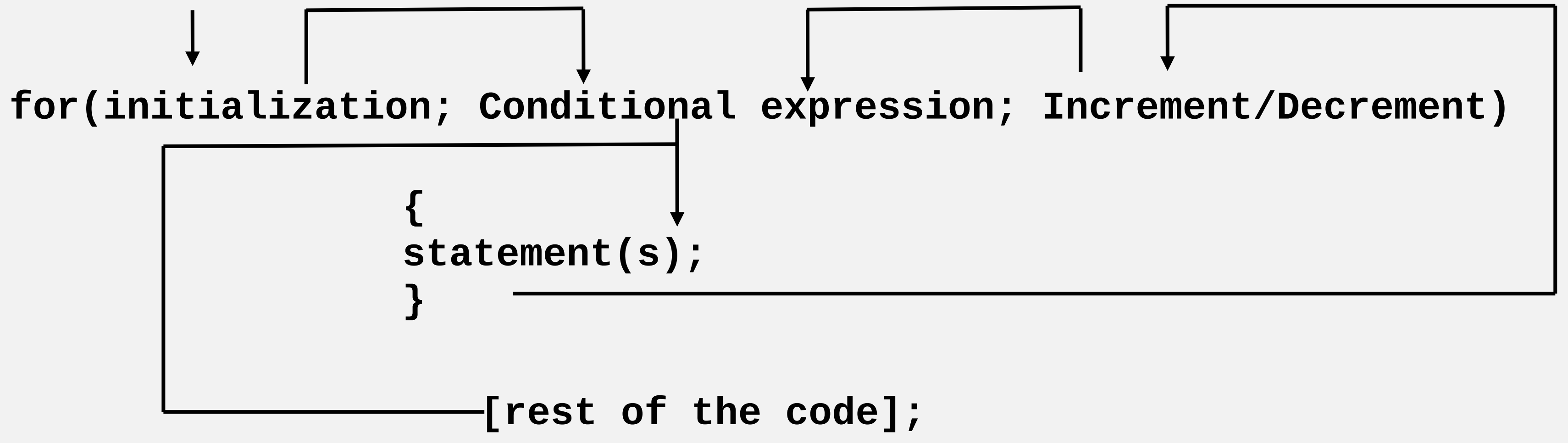
```
*****ENTER YOUR CHOICE*****
Enter 1 for Area of Rectangle
Enter 2 for Area of Circle
Enter 3 to Exit
Enter Your Choice: 1
Enter Length of Rectangle: 12.5
Enter Breadth of Rectangle: 10
Area of Rectangle is: 125.0

*****ENTER YOUR CHOICE*****
Enter 1 for Area of Rectangle
Enter 2 for Area of Circle
Enter 3 to Exit
Enter Your Choice: 2
Enter Radius of Circle: 2.5
Area of Circle is: 19.625

*****ENTER YOUR CHOIC*****
Enter 1 for Area of Rectangle
Enter 2 for Area of Circle
Enter 3 to Exit
Enter Your Choice: 3
```

for loop

This is and ***entry controlled loop.*** A ‘while’ and ‘do-while’ loop is useful when we don’t know in advance how many times a block of code is to be executed. *For loop* it is more **concise** loop structure than *while and do-while* loop structures. For loop has three actions, i.e. ***initialization***, ***conditional expression*** and ***increment*** or ***decrement*** are placed in ***for statement*** itself.

Syntax: for loop

```
for(initialization; Conditional expression; Increment/Decrement)
{
   statement(s);
}
[rest of the code];
```



```
for(initialization; Conditional expression; Increment/Decrement)
            {
            statement(s);
            }

        [rest of the code];
```

Example: Suppose there is a situation to display number from 1 to 5.

```
for(int num=1; num<=5; num++)
{
   System.out.println(num);
}
System.out.println(“End of loop!”);
```

at num=1	No condition is checked and we enter into the body of the loop.	display num, i.e. 1	num=num+1
			num=1+1=2
			And check again

at num=2	2<=5, which is true, execute body of loop	display num, i.e. 2	num=num+1
			num=2+1=3
			And check again

at num=3	3<=5, which is true, execute body of loop	display num, i.e. 3	num=num+1
			num=3+1=4
			And check again

at num=4	4<=5, which is true, execute body of loop	display num, i.e. 4	num=num+1
			num=4+1=5
			And check again

at num=5	5<=5, which is true, execute body of loop	display num, i.e.5	num=num+1
			num=5+1=6
			And check again

at num=6	6<=5, which is false, exit from the loop	display “end of loop”	

All three expressions of the **for loop** are **optional.** We can create an **infinite loop** as follows:

```
    for(;;)
    {
        Body of loop;
    }
```

PROGRAM 3-12

Description: *In this program we are going to display table of 2*

```
class Table
{
    public static void main(String args[])
    {
        int num=2;
        int res=0;
        for(int i=1;i<=10;i++)
        {
            res=num*i;
            System.out.println(num+”*”+i+”=“+res);
        }
        System.out.pritnln(“End of Program!);
    }
}
```

Output of Program 3-12

```
2*1=2
2*2=4
2*3=6
2*4=8
2*5=10
2*6=12
2*7=14
2*8=16
2*9=18
2*10=20
End of Program!
```

At i=1	Check: i<=10 1<=10, true, execute the body of the loop	Evaluate res=num*i res=2*1=2 display 2*1=2	i++ i=1+1=2
At i=2	Check: i<=10 2<=10, true, execute the body of the loop	Evaluate res=num*i res=2*2=4 display 2*2=4	i++ i=2+1=3
At i=3	Check: i<=10 3<=10, true, execute the body of the loop	Evaluate res=num*i res=2*3=6 display 2*3=6	i++ i=3+1=4
At i=4	Check: i<=10 4<=10, true, execute the body of the loop	Evaluate res=num*i res=2*4=8 display 2*4=8	i++ i=4+1=5

At i=5	Check: i<=10 5<=10, true, execute the body of the loop	Evaluate res=num*i res=2*5=10 display 2*5=10	i++ i=5+1=6
At i=6	Check: i<=10 6<=10, true, execute the body of the loop	Evaluate res=num*i res=2*6=12 display 2*6=12	i++ i=6+1=7
At i=7	Check: i<=10 7<=10, true, execute the body of the loop	Evaluate res=num*i res=2*7=14 display 2*7=14	i++ i=7+1=8
At i=8	Check: i<=10 8<=10, true, execute the body of the loop	Evaluate res=num*i res=2*8=16 display 2*8=16	i++ i=8+1=9
At i=9	Check: i<=10 9<=10, true, execute the body of the loop	Evaluate res=num*i res=2*9=18 display 2*9=18	i++ i=9+1=10
At i=10	Check: i<=10 10<=10, true, execute the body of the loop	Evaluate res=num*i res=2*10=20 display 2*10=20	i++ i=10+1=11
At i=11	Check: i<=10 11<=10, which is false, exit the loop	Display “End of program!”	

A **for loop** has many features:

1. We can initialize more than **one variable** in a **for statement,** which are separated by ***comma operator(,)***

```
int i,j;
for(j=5,i=1;i<=5;i++)
{
    System.out.println(j);
}
```

2. We can also use more than one increment or decrement statement in a **for statement,** separated by ***comma operator***

```
int i,j;
    for(j=5,i=1;i<=j;i++,i--)
    {
        System.out.println(“Value of i=“+i);
        System.out.println(“Value of j=“+j);
    }
```

3. The **for statement** also has another form designed for *iteration* through ***collections*** and ***arrays.*** It is called as the ***enhanced for statement.***

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

```
String[] cities={“MGG”,”Ludhiana”,”Khanna”,”Amloh”,Sirhind”};
System.out.println(“*****LIST OF CITIES*****”);
for(String item:cities)
{
   System.out.println(item);
}
System.out.println(“End of program”);
}}
```

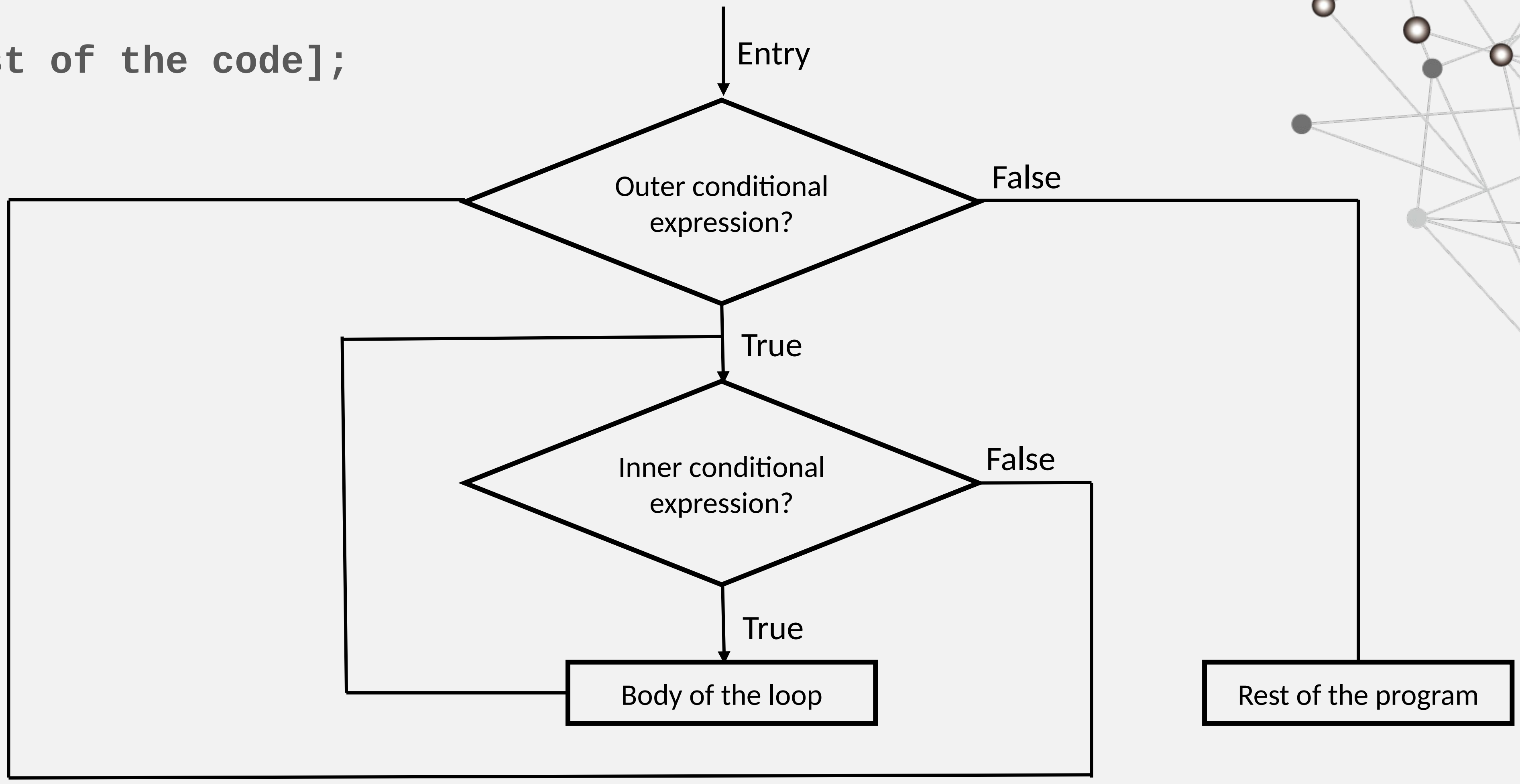
Nesting of loops

The placing of one loop **inside** the body of another loop is called **nesting of loop.** The **outer loop** takes control of the number of repetitions of the **inner loop.** *Nesting* can be done by any type of loop.

Nesting of while loops

Syntax: Nested while loop

```
initialization;
while(conditional expression)
{
   initialization;
   while(conditional expression)
   {
        statement(s);
        increment/decrement;
    }
    statement(s);
    increment/decrement;
}
[rest of the code];
```



PROGRAM 3-13

Description: *In this program, we are going to display a pattern of stars as:*

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

```
class PrintStars
{
    public static void main(String args[])
    {
        int row;
        int column;
        row=1;
        while(row<=3)
        {
            column=1;
            while(column<=4)
            {
                System.out.print(“*”);
                column++;
            }
            System.out.println(“\n”);
            row++;
        }
        System.out.println(“End of program”);
    }
}
```

Output of Program 3-13

```
****
****
****
End of program
```

At row=1	Check: row<=3 1<=3, true, execute body of the loop	Set column=1	Check: column<=4 1<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=2, go to test inner loop condition again
		At column=2	Check: column<=4 2<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=3, go to test inner loop condition again
		At column=3	Check: column<=4 3<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=4, go to test inner loop condition again
		At column=4	Check: column<=4 4<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=5, go to test inner loop condition again
		At column=5	Check: column<=4 5<=4, false, exit from inner loop	Print: \n, i.e. set cursor to new line and evaluate row++, i.e., row=2, go to rest outer loop

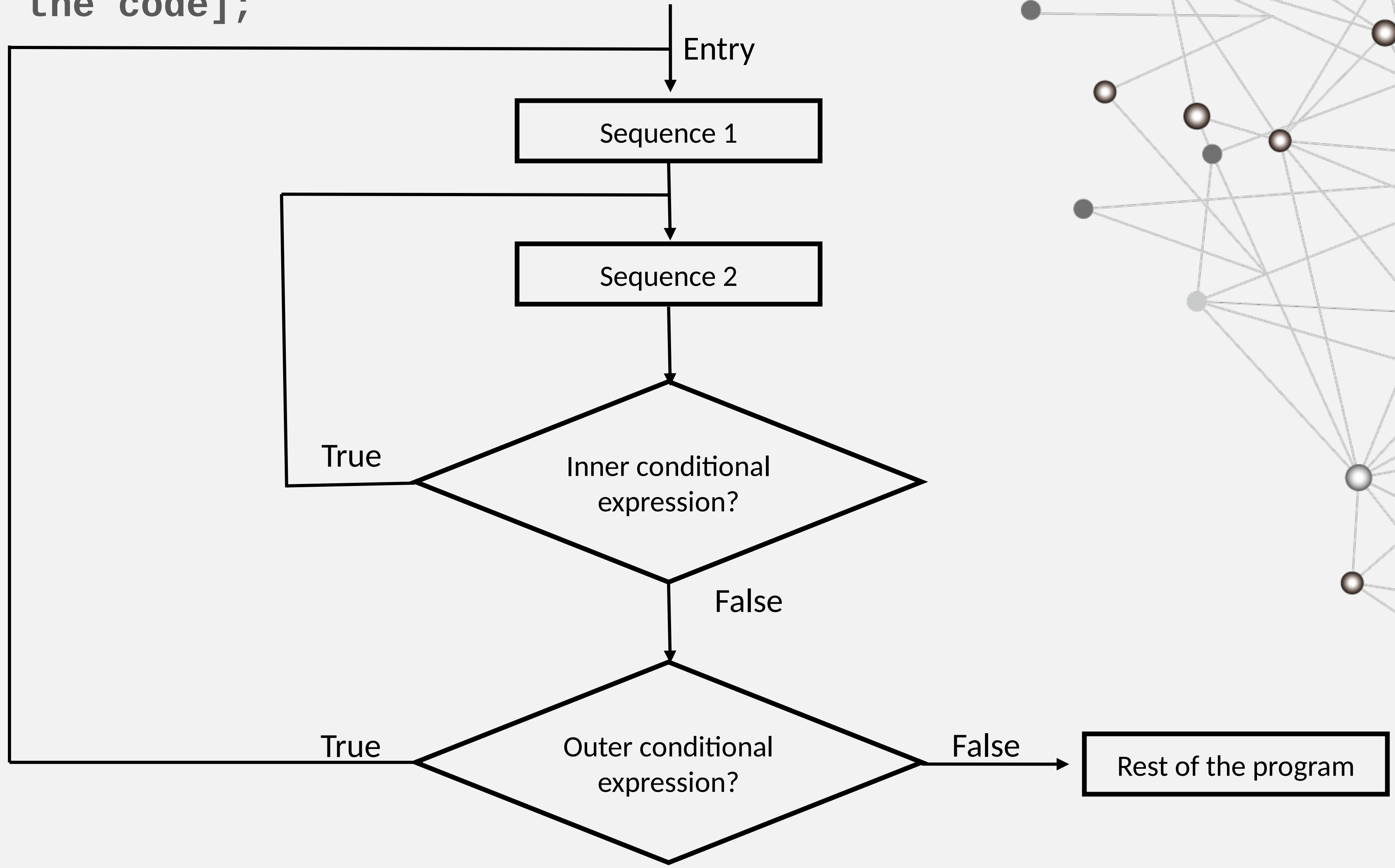
At row=2	Check: row<=3 2<=3, true, execute body of the loop	Again set column=1	Check: column<=4 1<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=2, go to test inner loop condition again
		At column=2	Check: column<=4 2<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=3, go to test inner loop condition again
		At column=3	Check: column<=4 3<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=4, go to test inner loop condition again
		At column=4	Check: column<=4 4<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=5, go to test inner loop condition again
		At column=5	Check: column<=4 5<=4, false, exit from inner loop	Print: \n, i.e. set cursor to new line and evaluate row++, i.e., row=3, go to rest outer loop
At row=3	Check: row<=3 3<=3, true, execute body of the loop	Again set column=1	Check: column<=4 1<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=2, go to test inner loop condition again
		At column=2	Check: column<=4 2<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=3, go to test inner loop condition again
		At column=3	Check: column<=4 3<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=4, go to test inner loop condition again

		At column=4	Check: column<=4 4<=4, true, execute the boy of the loop	Display * and evaluate column++. i.e. column=5, go to test inner loop condition again
		At column=5	Check: column<=4 5<=4, false, exit from inner loop	Print: \n, i.e. set cursor to new line and evaluate row++, i.e., row=4, go to rest outer loop
At row=4	Check: row<=3 4<=3, false, exit from the loop and execute rest of the code	Display: “End of program”		

Nesting of do-while loops

Syntax: Nested do-while loop

```
initialization;
do
{
    initialization;
    do
    {
        Statement(s);
        increment/decrement;
    }while(condition expression);
    statement(s);
    increment/decrement;
}while(condition expression);
[rest of the code];
```



PROGRAM 3-14

Description: *In this program, we are going to display a pattern of stars as:*

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

```
class PrintStars
{
    public static void main(String args[])
    {
        int row;
        int column;
        row=1;
        do
        {
            column=1;
            do
            {
                System.out.print(*);
                column++;
            }while(column<=row);
            System.out.println("\n");
            row++;
        }while(row<=3);
```

```
        System.out.println(“End of program”);
    }
}
```

Output of Program 3-14

```
*
**
***
End of program
```

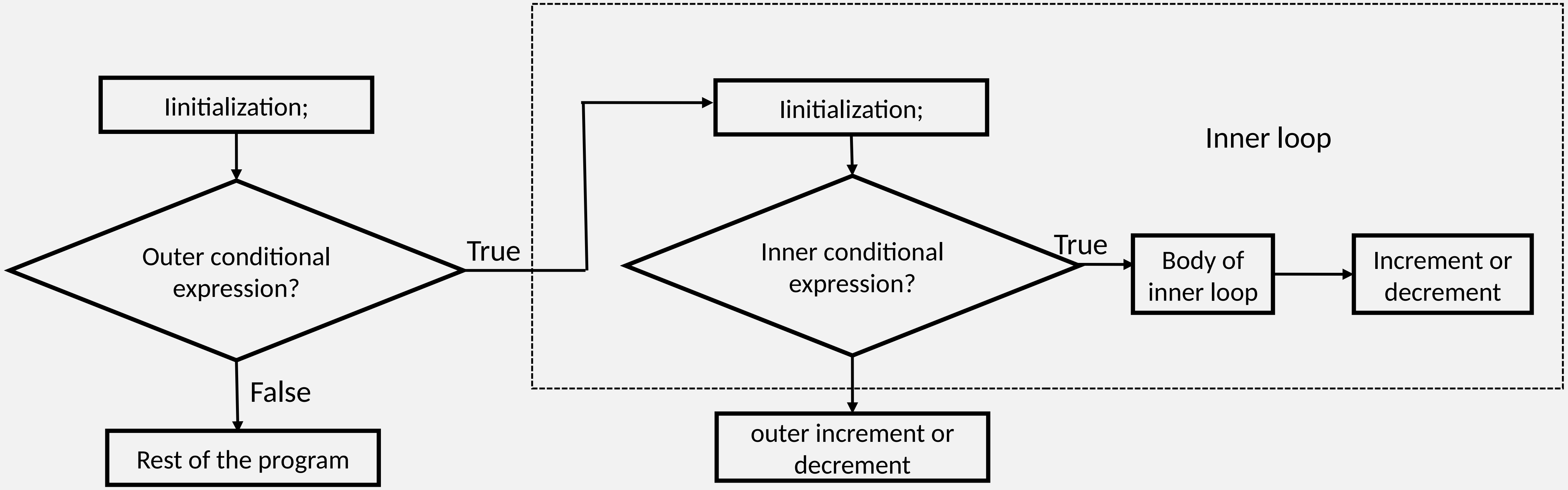
At row=1	Check: row<=3 1<=3, true, execute body of the loop	Set column=1	No condition is checked and we enter into the body of the inner loop	Display * and evaluate column++. i.e. column=2, go to test inner loop condition again
		At column=2	Check: column<=row 2<=1, false, exit from inner loop and execute outer loop statements	Print: \n , evaluate row++ Row=2 Go to test outer loop
At row=2	Check: row<=3 2<=3, true, execute body of the loop	Again set column=1	No condition is checked and we enter into the body of the inner loop	Display * and evaluate column++. i.e. column=2, go to test inner loop condition again
		At column=2	Check: column<=row 2<=2, true, execute the body of inner loop	Display * and evaluate column++. i.e. column=3, go to test inner loop condition again

		At column=3	Check: column<=row 3<=2, false, exit from inner loop and execute outer loop statements	Print: \n , evaluate row++ Row=3 Go to test outer loop
At row=3	Check: row<=3 3<=3, true, execute body of the loop	Again set column=1	No condition is checked and we enter into the body of the inner loop	Display * and evaluate column++. i.e. column=2, go to test inner loop condition again
		At column=2	Check: column<=row 2<=3, true, execute the body of inner loop	Display * and evaluate column++. i.e. column=3, go to test inner loop condition again
		At column=3	Check: column<=row 3<=3, true, execute the body of inner loop	Display * and evaluate column++. i.e. column=4 go to test inner loop condition again
		At column=4	Check: column<=row 4<=3, false, exit from inner loop and execute outer loop statements	Print: \n , evaluate row++ Row=4 Go to test outer loop
At row=4	Check: row<=3 4<=3, true, execute from outer loop and execute rest of the program	Display: “End of program”		

Nesting of for loops

Syntax: Nested for loops:

```
for(initialization; conditional expression; increment/decrement)
{
    for(initialization; conditional expression; increment/decrement)
    {
        statement(s);
    }
    Statement(s);
}
```



PROGRAM 3-15

Description: *In this program, we are going to display a pattern as:*

```
1
22
333
```

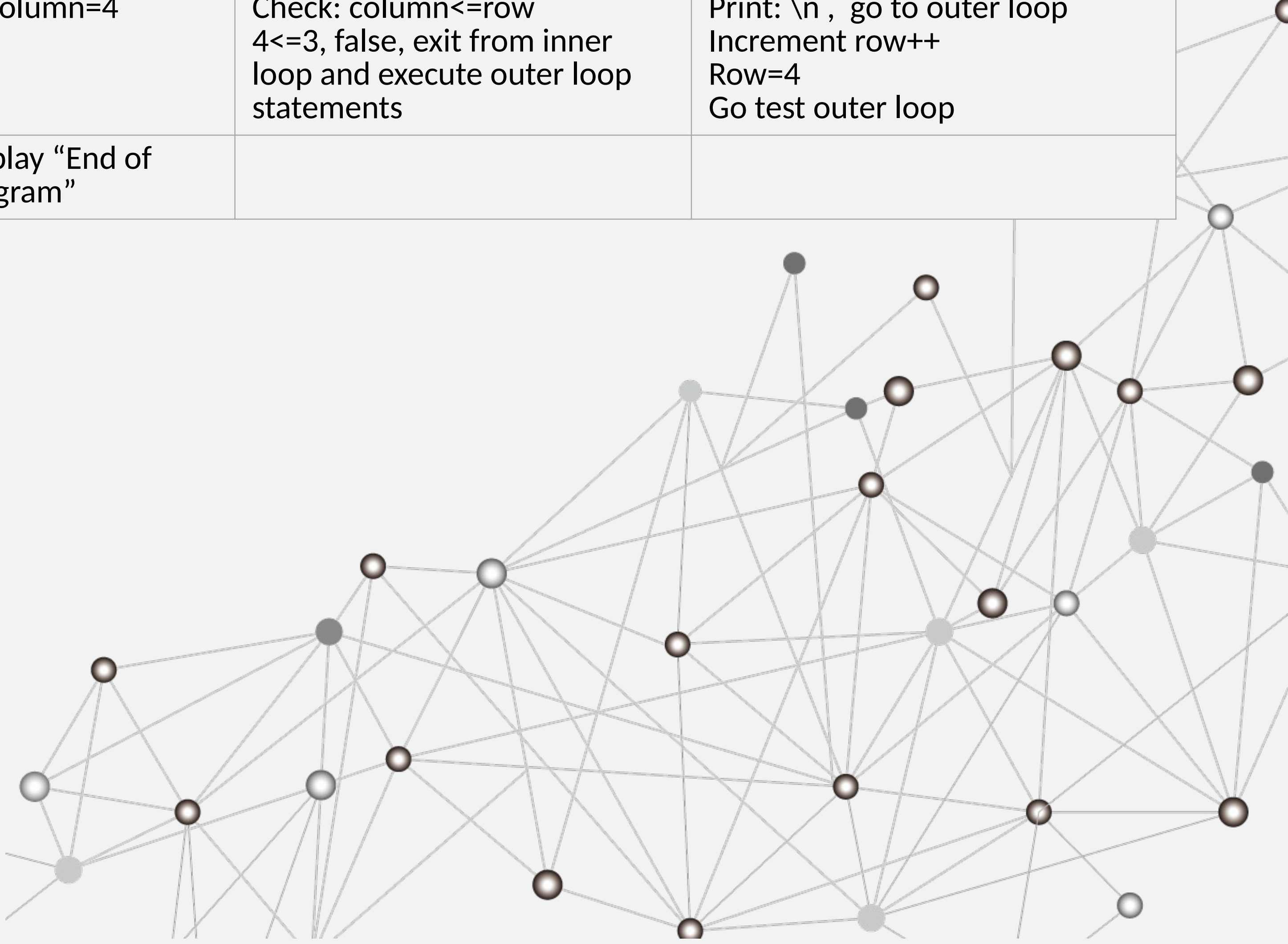
```
class DemoNestedFor
{
    public static void main(String args[])
    {
        int row;
        int column;
        for(row=1;row<=3;row++)
        {
            for(column=1;column<=row;column++)
            {
                System.out.print(row);
            }
            System.out.println(“\n”);
        }
        System.out.println(“End of program”);
    }
}
```

Output of Program 3-15

```
1
22
333
End of program
```

At row=1	Check: row<=3 1<=3, true, execute body of the loop	Set column=1	Check: column<=row 1<=1, true, execute inner loop	Display row i.e. 1 and go to inner loop increment column++ Column=2, go test inner loop
		At column=2	Check: column<=row 2<=1, false, exit from inner loop and execute outer loop statements	Print: \n , go to outer loop Increment row++ Row=2 Go test outer loop
At row=2	Check: row<=3 2<=3, true, execute body of the loop	Again set column=1	Check: column<=row 1<=2, true, execute inner loop	Display row i.e. 2 and go to inner loop increment column++ Column=2, go test inner loop
		At column=2	Check: column<=row 2<=2, true, execute the body of inner loop	Display row i.e. 2 and go to inner loop increment column++ Column=3, go test inner loop
		At column=3	Check: column<=row 3<=2, false, exit from inner loop and execute outer loop statements	Print: \n , go to outer loop Increment row++ Row=3 Go test outer loop
At row=3	Check: row<=3 1<=3, true, execute body of the loop	Again set column=1	Check: column<=row 1<=3, true, execute inner loop	Display row i.e. 3 and go to inner loop increment column++ Column=2, go test inner loop

		At column=2	Check: column<=row 2<=3, true, execute the body of inner loop	Display row i.e. 3 and go to inner loop increment column++ Column=3, go test inner loop
		At column=3	Check: column<=row 3<=3, true, execute the body of inner loop	Display row i.e. 3 and go to inner loop increment column++ Column=4, go test inner loop
		At column=4	Check: column<=row 4<=3, false, exit from inner loop and execute outer loop statements	Print: \n , go to outer loop Increment row++ Row=4 Go test outer loop
At row=4	Check: row<=4 4<=3, false, exit the outer loop	Display “End of program”		

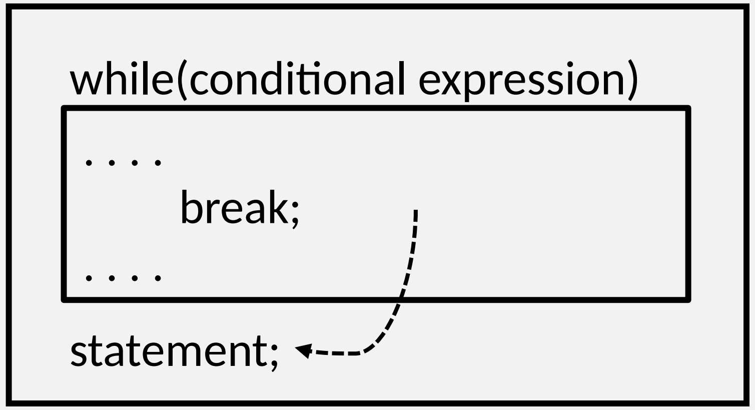


BRANCHING STATEMENTS

Sometimes, while executing a loop, it is desired to skip a part of the loop body and continue with the next iteration of the loop or it may also exit from the loop before the specified condition for the loop become false. This can be achieved by ***branching statements*** also called as ***jumping and skipping constructs.*** These statements are as follows:

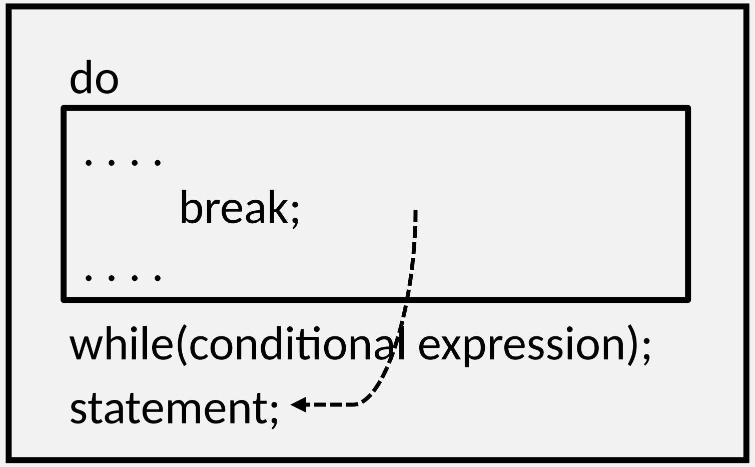
1. break statement

When we need to exit from a loop as soon as certain condition is occurred, then we use **break** statement. When a break statement encounters inside a loop, the loop immediately **exit** the program continuing with the statement immediately following the body of the loop. The break statement can be used in **while loop, do while loop, for loop** and also **switch statement.**



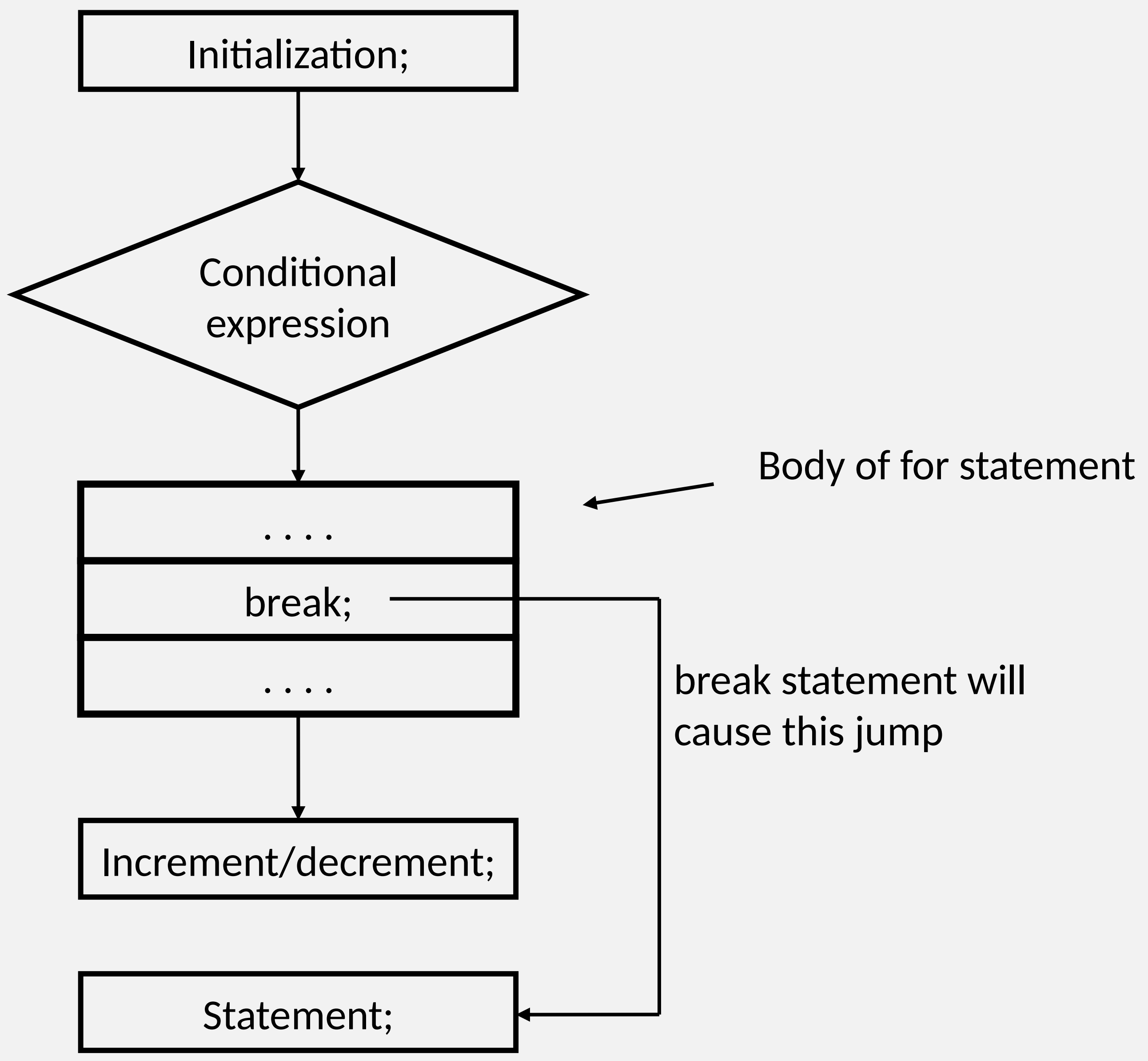
```
while(conditional expression)
    . . . .
            break;
    . . . .
statement;
```

break in while loop



```
do
    . . . .
            break;
    . . . .
while(conditional expression);
statement;
```

break in do-while loop



break in for loop

PROGRAM 3-

Description:** In this program, we are going to demonstrate the concept of **break statement.

```
class DemoBreak
{
    public static void main(String args[])
    {
        int num;
        for(num=1;num<=10;num++)
        {
            if(num==5)
            {
                System.out.println(“Break encountered! So, jump out of
            loop!”);
                break;
            }
            System.out.println(num);
        }
        System.out.println(“End of program”);
    }
}
```

Output of Program 3-16

```
1
2
3
4
Break encountered! So, jump out of loop!
End of program
```

at num=1	1<=10, which is true, execute body of loop	Check num==5
		1==5, false, so
		display num, i.e. 1
			num=num+1
			num=1+1=2

at num=2	2<=10, which is true, execute body of loop	Check num==5
		2==5, false, so
		display num, i.e. 2
			num=num+1
			num=2+1=3

at num=3	3<=10, which is true, execute body of loop	Check num==5
		3==5, false, so
		display num, i.e. 3
			num=num+1
			num=3+1=4

at num=4	4<=10, which is true, execute body of loop	Check num==5
		4==5, false, so
		display num, i.e. 4
			num=num+1
			num=4+1=5

at num=5	5<=5, which is true, execute body of loop	Check num==5 1==5, true, so display “Break encountered! So, jump out of loop! And execute break.	Display: “End of program”

The break statement can be ***labelled*** or ***unlabelled.*** We use unlabelled break statement in ***switch statement.*** We can also use an unlabelled break to terminate a while, do-while or for loop. An unlabelled break statement terminates the **innermost** switch, for, while or do-while statement where as a *labelled break* can terminate an outer statement. The labelled break statement terminates the labelled statement. It does not transfer the flow of control to the label. Control flow is transferred to the statement immediately following the labelled *(terminated)* statement.

PROGRAM 3-17

Description: *In this program, we are going to demonstrate the concept of unlabelled break statement.*

```
class UnlabelledBreak
{
   public static void main(String args[])
   {
      int row, column;
      for(row=1;row<=3;row++)
      {
         for(column=1;column<=row;column++)
         {
            if(column==2)
            {
               System.out.println("\t Break Encountered, jump
         out of inner loop!");
               break; //unlabelled break
```

```
                }
                System.out.print(column);
            }
            System.out.printl(“\t Go for next iteration of outer loop!”);     }
        System.out.println(“End of program”);
    }
}
```

Output of Program 3-17

```
1   Go for next iteration of outer loop
1   Break Encountered, jump out of inner loop
    Go for next iteration of outer loop
1   Break encountered, jump out of inner loop
    Go for next iteration of outer loop
End of program
```

PROGRAM 3-18

Description: *In this program, we are going to demonstrate the concept of* ***labelled break statement.***

```
class LabelledBreak
{
    public static void main(String args[])
    {
        int row, column;
        outer:    //label
        for(row=1;row<=3;row++)
        {
            for(column=1;column<=row;column++)
            {
                if(column==2)
                {
                    System.out.println(“\t Break Encountered, jump
            out of inner loop!”);
                    break outer; //labelled break
                }
                System.out.println(column);
            }
        System.out.println(“\t Go for next iteration of outer loop”);
        }
```

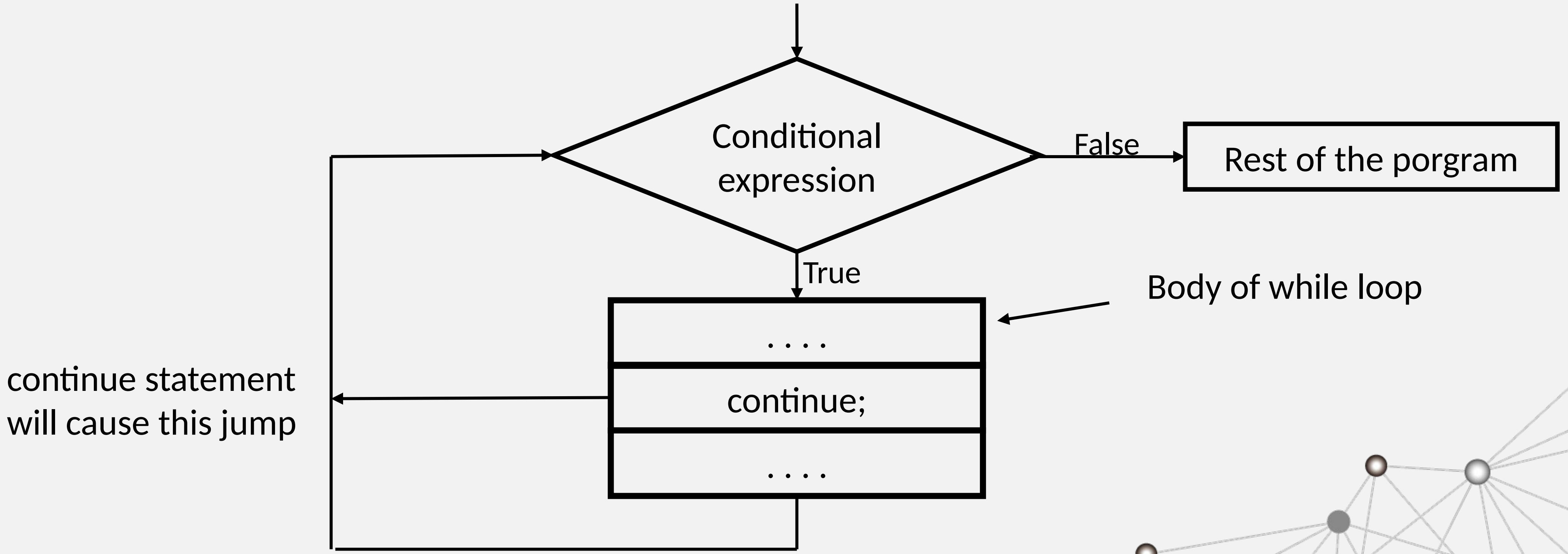
```
      System.out.println(“End of program”);
   }
}
```

Output of Program 3-18

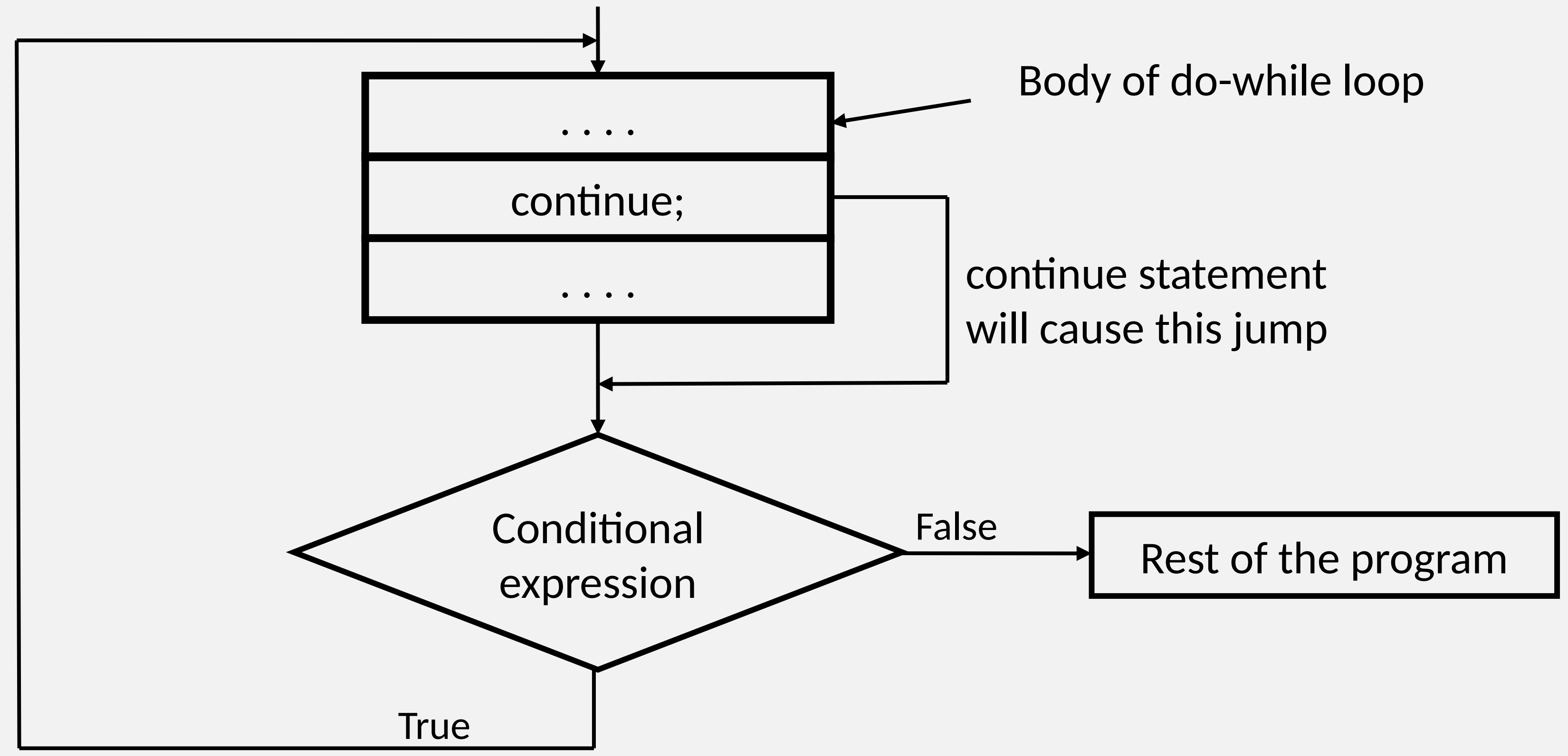
```
1   Go for next iteration of outer loop
1   Break Encountered, jump out of inner loop
End of program
```

2. continue statement

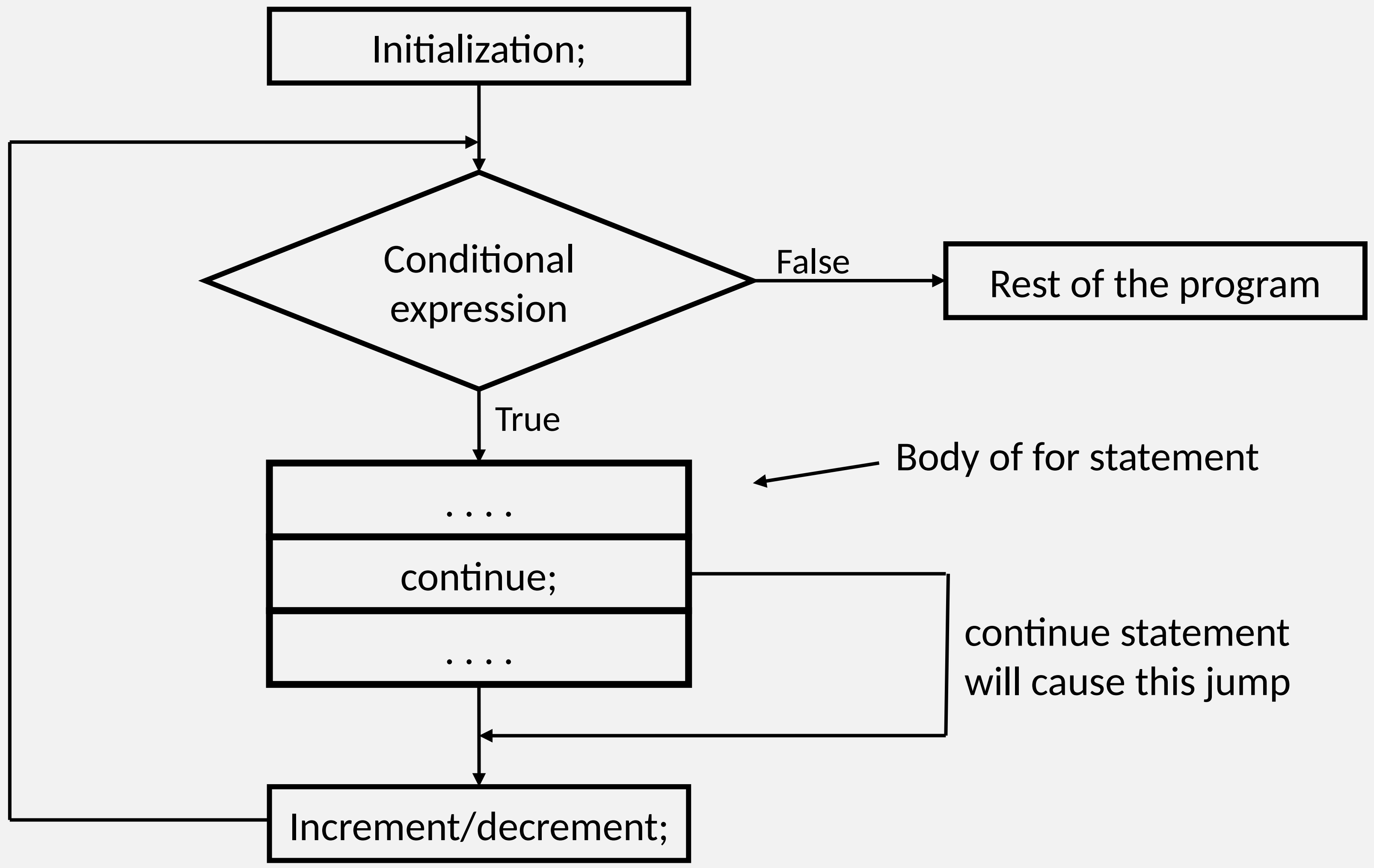
When we need to **skip a part** of the body of a loop when a certain condition is occurred, then we use **continue statement.** The continue statement skips the current iteration of an innermost **for, while or do-while loop** and starts the next iteration immediately.



continue in while loop



continue in do-while loop



continue in for loop

PROGRAM 3-19

Description: *In this program, we are going to demonstrate the concept of* ***continue statement.***

```
class DemoContinue
{
    public static void main(String args[])
    {
        int num;
        for(num=1;num<=20;num++)
        {
            if(num%5==0)
            {
                System.out.println(“Continue encountered, jump to next
            iteration of loop”);
                continue; //unlabelled continue
            }
            System.out.println(num);
        }
        System.out.println(“End of program!”);
    }
}
```

Output of Program 3-19

```
1
2
3
4
Continue encountered, jump to next iteration of loop
6
7
8
9
Continue encountered, jump to next iteration of loop
11
12
13
14
Continue encountered, jump to next iteration of loop
15
16
17
18
19
Continue encountered, jump to next iteration of loop
```

PROGRAM 3-20

Description: *In this program, we are going to demonstrate the concept of* ***labelled continue statement.***

```
class LabelledContinue
{
    public static void main(String args[])
    {
        int row,col;
        outer:
        for(row=1;row<=4;row++)
        {
            for(col=1;col<=row;col++)
            {
                if(col==3)
                {
                    System.out.println(“\t Continue encountered,
    jump next iteration of outer loop”);
                    continue outer; //labelled continue
                }
                System.out.print(col);
            }
            System.out.println();
        }
        System.out.println(“End of program”);
    }
}
```

Output of Program 3-20

```
1
12
12 Continue encountered, jump next iteration of outer loop
12 Continue encountered, jump next iteration of outer loop
End of program
```

3. return statement

The return statement is used within a method. When this statement is executed all the statements after **return statement** will be **skipped** because the return statement exits from the current method, and control flow returns to where the method was invoked (called).

For example: to return a value,

```
        return Value;
```

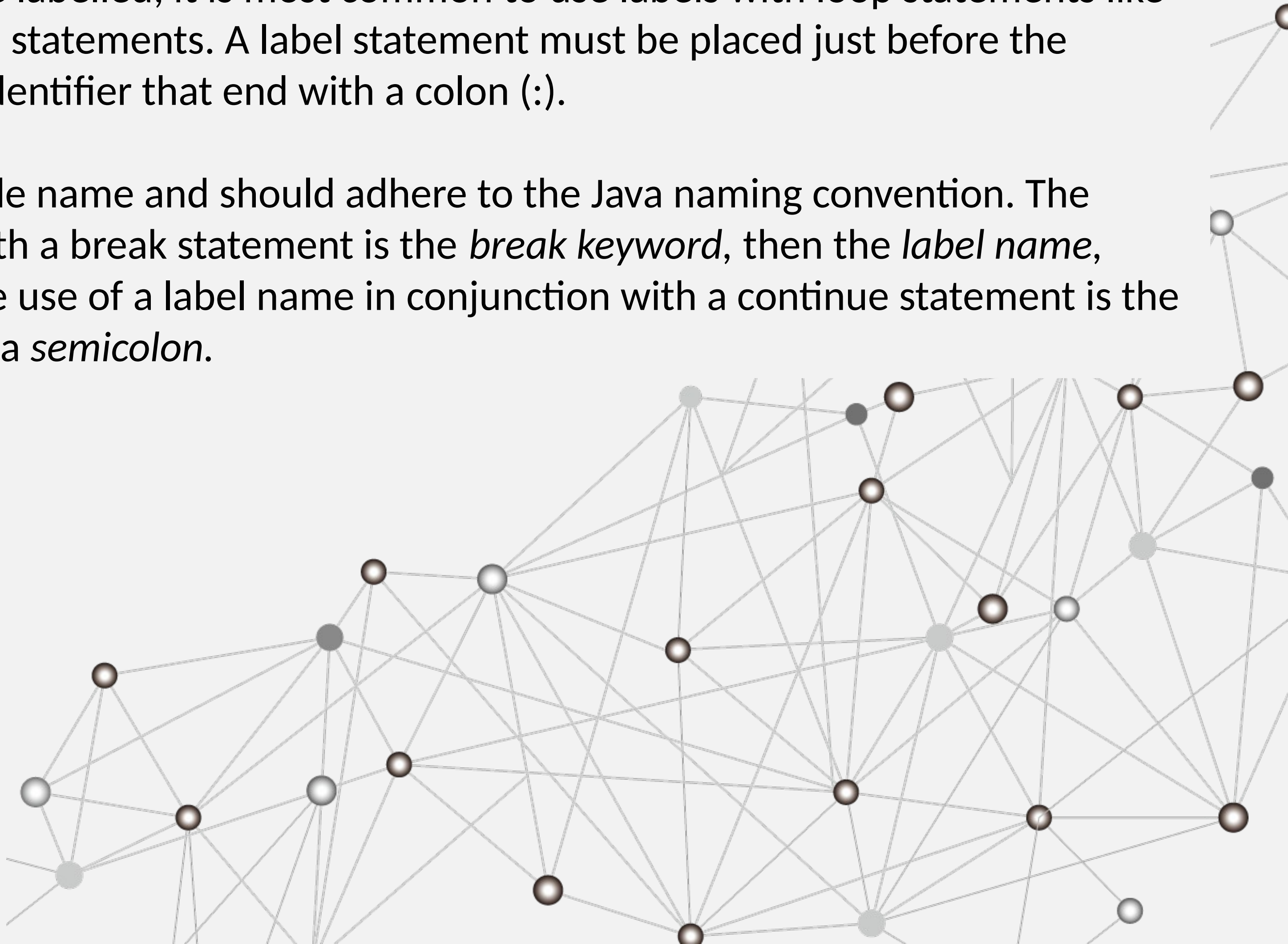
The **data type** of returned **Value** must match the **return type** of the method. When a method is declared **void,** use the form of **return** that does not return any value:

```
        return;
```

LABELLED LOOPS

Although many statements in a Java program can be labelled, it is most common to use labels with loop statements like for or while, in conjunction with break and continue statements. A label statement must be placed just before the statement being labelled, and it consists of a valid identifier that end with a colon (:).

The label must adhere to the rules for a valid variable name and should adhere to the Java naming convention. The syntax for the use of a label name in conjunction with a break statement is the *break keyword*, then the *label name*, followed by a *semicolon*. Similarly, the syntax for the use of a label name in conjunction with a continue statement is the *continue keyword*, then the *label name* followed by a *semicolon*.



THANK YOU