Loops in java

In this Lecture we Learn

- Loops in Java

introduction

- Java's Iteration statements are , these statements are commonly called as loops. A loop repeatedly executes the same set of instructions until a termination condition is met.

- **Problems Without Loops**
- For example, if we want to print Hello for 10 times, we need to call the print statement 10 times. This could be achieved by explicitly calling print statement 10 times as shown in next slide.

introduction

```
System.out.println("Hello");
System.out.println("Hello");
System.out.println("Hello");
System.out.println("Hello");
System.out.println("Hello");
System.out.println("Hello");
System.out.println("Hello");
System.out.println("Hello");
System.out.println("Hello");
System.out.println("Hello");
```

We print same output 10 time . It is very time consuming

introduction

- Although this approach works, it has the following disadvantages:
  - It is very time consuming, inefficient, error prone and has lots of duplicate code.
  - If we want print Hello World instead of Hello, we have to change all the ten lines.
  - If we want print 100 lines more or 1000 lines more, we have to add so many more lines. It is very time consuming to do it.
  - What if the number of lines to be printed is given by the user and we do not know the value until the program runs. If user wants 7 lines, we should print 7, if he wants 12 we should print 12.

introduction

- The alternative approach is to use loops which will take care of all the disadvantages mentioned above.
- As discussed earlier for any loop, there is loop body consisting of set of instructions and there is termination condition. As long as the termination condition is true, the loop body is executed. When the condition is false, loop stops.
- Since we want to call print 10 times, we want a termination condition which is true for ten times, but false after that. Such termination condition can be defined using a variable counter which is initialized to 1 and the condition is counter <= 10. We will increment the counter by 1 after every execution of loop body. So as long as the counter is less than or equal to 10, i.e. for values 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10, the loop body executes. When the counter becomes 11, the loop is terminated.

introduction

- Consider the following declarations:

```
int a, b, c, d, e, f, g, h, i, j;
```

Issues:

1. Variables of the same **data**_type, i.e. ***int***.
2. Individual operation (e.g., input) is needed for each variable

Better Solution:

- Use Loops. (One variable will be enough for the 10 input values).

Loops in Java

- In computer programming, loop cause a **certain piece** of **program** to be **executed** a **certain number** of **times**. Consider these scenarios:
  - You want to execute some code/s certain number of time.
  - You want to execute some code/s certain number of times depending upon input from user.
- These types of task can be solved in programming using loops.

Repetition (Loops)

- Loops sometime also called **Repetition.**
- **Repetition** structures, or loops, are used when a program needs to **repeatedly process one** or **more** instructions until some condition is met, at which time the loop ends.
- Repetition **allows** the programmer to **efficiently** use **variables**.

Repetition (Loops)

- Every loop can be defined by **three** way:
  - Starting Point
  - Ending Point
  - Sequence of Moving

While Loop in Java

While Loop in Java

- Executes from **zero** to many times, **depending** on expression.

- The **while loop** checks whether the **test** expression is **true or not**. If it is true, code/s inside the body of while loop is executed, that is, code/s inside the braces { } are executed. Then again the test expression is checked whether test expression is true or not. This process continues until the test expression becomes false.

While Loop Declaration

```
while(test condition)
{
    block of code;
}
```

- The **test condition** must be enclosed in parentheses. The **block of code** is called the body of the loop and is enclosed in braces and indented for readability. (The braces are not required if the body is composed of only ONE statement.) Semi-colons follow the statements within the block only

While Loop Declaration

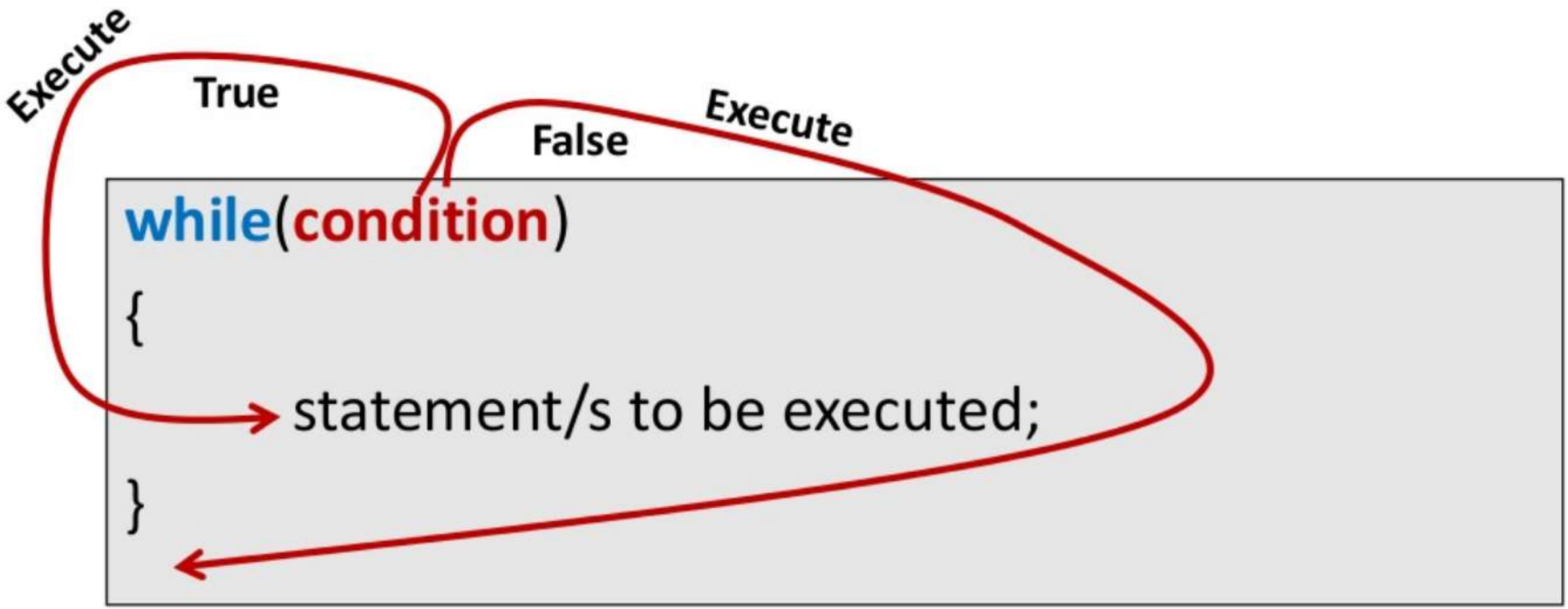
```
while(test condition)
{
    block of code;
}
```

When a program encounters a **while** loop, the test condition is evaluated first. If the condition is **TRUE,** the program executes the body of the loop. The program then returns to the test condition and reevaluates. If the condition is still **TRUE**, the body executes again. This cycle of testing and execution continues until the test condition evaluates to **0** or **FALSE.** If you want the loop to eventually terminate, something within the body of the loop must affect the test condition. Otherwise, a disastrous **INFINITE LOOP** is the result!

Syntax of while Loop

```
while(condition)
{
        statement/s to be executed;
}
```

Syntax of while Loop



While Loop Example in Java

```
12  public class Test {
13      public static void main(String[] args) {
14          int x = 10;
15
16        while( x < 15 ) {
17          System.out.print("value of x : " + x );
18          x++;
19          System.out.print("\n");
20        }
21      }
22
23  }
```

test.Test

Output - Test (run)

```
run:
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
```

While Loop Example in Java

```
public class Test {
   public static void main(String[] args) {
      int x = 10;

     while( x < 15 ) {
        System.out.print("value of x : " + x );
        x++;
        System.out.print("\n");
     }
   }
}
```

Output is:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
```

Start point of while loop

Ending point of while loop

Print Value of x

Sequence of while loop

\n use of new line

```
int x = 10;
   while( x < 15 ) {
      System.out.print("value of x : " + x );
      x++;
      System.out.print("\n");
```

Test Expression and ending point

x++ equal to x=x+1

Initially, x = 10, test expression x <15 is true, Print value of x is 10

Variable x is updated to 11, test expression x<15 is true, Print value of x is 11

Variable x is updated to 12, test expression x<15 is true, Print value of x is 12

Variable x is updated to 13, test expression x<15 is true, Print value of x is 13

Variable x is updated to 14, test expression x<15 is true, Print value of x is 14

Variable x is updated to 15, test expression x<15 is false, and while loop is terminated.

Escape Sequences

- A character preceded by a backslash (\) is an escape sequence and has a special meaning to the compiler.

Escape Sequence	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point.
\f	Inserts a form feed in the text at this point.
\'	Inserts a single quote character in the text at this point.
\"	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.

Another While Loop Example

```
12  public class Test {
13      public static void main(String[] args) {
14          int i=5;
15          while(i>=1){
16              System.out.println(i);
17              i--;
18          }
19      }
20
21  }
```

test.Test

Output - Test (run)

```
run:
5
4
3
2
1
```

Explanation of While Loop Example

```
public class Test {
    public static void main(String[] args) {
        int i=5;
        while(i>=1){
            System.out.println(i);
            i--;
        }
    }
}
```

Output is:

5
4
3
2
1

Starting Point of while loop

Test Expression and Ending point of while loop

Print the value of i

Sequence of while loop, Decrement by 1

Problem Statement

Java program to find factorial of a positive integer entered by user. (Factorial of n = 1*2*3...*n)

Solution of Previous Problem

```
12  import java.util.Scanner;
13  public class Test {
14      public static void main(String[] args) {
15          int number, i = 1, factorial = 1;
16          Scanner user_input = new Scanner(System.in);
17          System.out.println("Please Enter a Number: ");
18          number=user_input.nextInt();
19          while ( i <= number) {
20          factorial *= i;       //factorial = factorial * i;
21          ++i;
22          }
23          System.out.println("Factorial of "+number+" = "+factorial);
24      }
25  }
```

Output - Test (run)

```
run:
Please Enter a Number:
5
Factorial of 5 = 120
```

Problem Statement

Write a java program which displays **odd** numbers **between 1 and 10**.

Solution of Previous Program

```
13  public class Test {
14      public static void main(String[] args) {
15          int number = 0;
16           while (number < 10)
17           {
18              if (number%2 != 0)
19              System.out.println(number);
20              number=number+1;
21           }
22      }
23  }
24
```

test.Test > main > while (number < 10)

Output - Test (run)

```
run:
1
3
5
7
9
```

Infinite While Loop

```
public class Test {
    public static void main(String args[]){
        int i=10
        while(i>1)
        {
            System.out.println(i);
             i++;
        }
    }
}
```

Write a Java program to calculate the sum of first 10 natural number using while loop.

Increment and Decrement Operators

- The increment/decrement unary operator can be placed before the operand (prefix operator), or after the operands (postfix operator).
- They takes on different meaning in operations.

Increment and Decrement Operators

Operator	Name	Example expression	Meaning
++	Postfix increment	x++	add 1 to x and return the old value
++	Prefix increment	++x	add 1 to x and return the new value
--	Postfix decrement	x--	take 1 from x and return the old value
--	Prefix decrement	--x	take 1 from x and return the new value

++a increments and then uses the variable.
a++ uses and then increments the variable.

Increment and Decrement Operators

```
13    public class Test {
14        public static void main(String[] args) {
15            int number = 5;
16                System.out.println("Behaviour of Postincrement");
17                System.out.println(number++);
18                System.out.println(number);
19                number=5;
20                System.out.println("Behaviour of Preincrement");
21                System.out.println(++number);
22                System.out.println(number);
23        }
24    }
```

Output - Test (run)

```
run:
Behaviour of Postincrement
5
6
Behaviour of Preincrement
6
6
```

Some Examples

Expression	Initial value of x	Value of y	Final Value of x
Y = ++x;	10	11	11
Y = x++;	10	10	11
Y = --x;	10	9	9
Y = x--;	10	10	9

do while Loop in Java

do-while Loop

- The **do-while** loop is similar to the **while** loop, except that the **test condition** occurs at the **end** of the loop. Having the test condition at the end, guarantees that the body of the loop always executes at least one time.

- The **test condition** must be enclosed in parentheses and **FOLLOWED BY A SEMI-COLON.**

do-while Loop Declaration

```
do
{
    block of code;
}
while (test condition);
```

The **test condition** must be enclosed in parentheses and **FOLLOWED BY A SEMI-COLON.** Semi-colons also follow each of the statements within the block. The body of the loop (the block of code) is enclosed in braces and indented for readability. (The braces are not required if only **ONE** statement is used in the body of the loop.)

do-while Loop Declaration

```
do
{
    block of code;
}
while (test condition);
```

The **do-while** loop is an **exit-condition loop**. This means that the body of the loop is always executed first. Then, the test **condition** is evaluated. If the test condition is **TRUE,** the program executes the body of the loop again. If the test condition is **FALSE**, the loop terminates and program execution continues with the statement following the **while.**

do-while Loop Declaration

```
do
{
    block of code;
}
while (test condition);
```

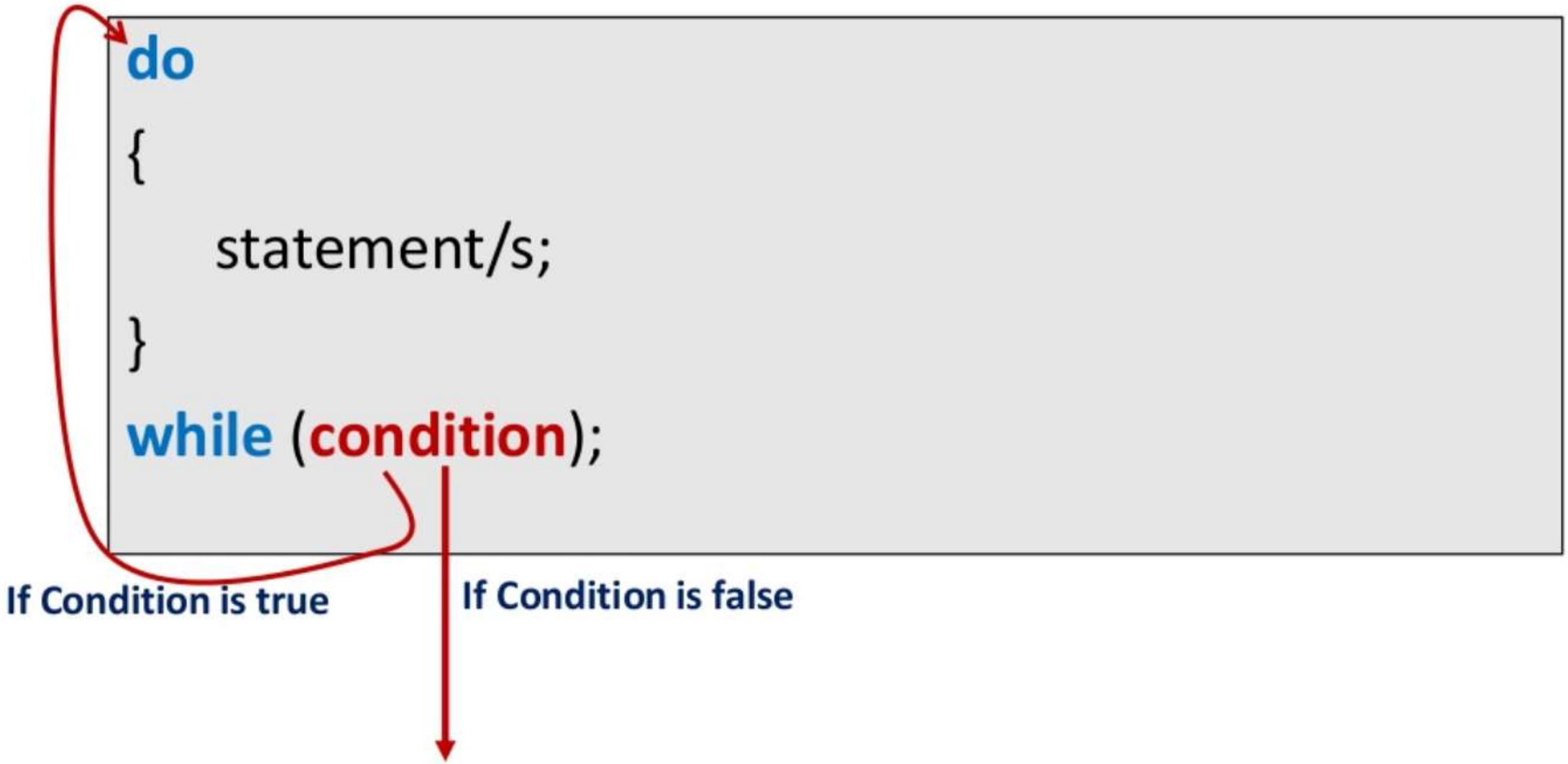
Note:-

- If body of loop is only one statement than braces ({}) are not needed.
- Body of the loop may have one or more statement.
- Semicolon is present after while.
- At least once body of loop is executed even condition is false.

Syntax of Do While Loop

```
do
{
    statement/s;
}
while (condition);
```

Syntax of Do While Loop



do while Loop Example in Java

```
13    public class Test {
14        public static void main(String[] args) {
15            int x = 10;
16          do {
17             System.out.print("value of x : " + x );
18             x++;
19             System.out.print("\n");
20              }while( x < 15 );
21        }
22    }
```

test.Test

Output - Test (run)

```
run:
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
```

Explanation of Do While Loop Example

```
public class Test {
   public static void main(String[] args) {
      int x = 10;
     do {
        System.out.print("value of x : " + x );
        x++;
        System.out.print("\n");
         } while( x < 15 );
   }
}
```

Starting point of do while loop

This statement one time must execute before checking condition

Sequence of Loop/ Increment by 1

```
int x = 10;
    do {
       System.out.print("value of x : " + x );
       x++;
       System.out.print("\n");
        } while( x < 15 );
```

Start point

Test Condition

- Initially, x= 10, Print value of x is 10 and Variable x is updated to 11
- Now Checking x=11 with Test Expression x<15 which is **true**, Print value of x is 11 and Variable x is Updated to 12
- Now Checking x=12 with Test Expression x<15 which is **true**, Print value of x is 12 and Variable x is Updated to 13
- Now Checking x=13 with Test Expression x<15 which is **true**, Print value of x is 13 and Variable x is Updated to 14
- Now Checking x=14 with Test Expression x<15 which is **true**, Print value of x is 14 and Variable x is Updated to 15.
- Now Checking x=15 with Test Expression x<15 which is **false**, **loop is terminated.**

Problem Statement

Write a Java program to add numbers entered by user until user enters 0.

Solution of Previous Problem

```
12  import java.util.Scanner;
13  public class Test {
14      public static void main(String[] args) {
15          long number,sum=0;
16          do{
17          Scanner user_input = new Scanner(System.in);
18          System.out.println("Please Enter a Number: ");
19          number=user_input.nextInt();
20          sum=sum+number;}
            while(number!=0);
22          System.out.println("Total Sum is: "+sum);
23
24      }
25  }
```

test.Test > main > do ... while (number != 0)

Output - Test (run)

```
run:
Please Enter a Number:
4
Please Enter a Number:
6
Please Enter a Number:
0
Total Sum is: 10
```

Java Infinitive do-while Loop

- If you pass **true** in the do-while loop, it will be infinitive do-while loop.
- **Syntax:**

```
do{

    //code to be executed

}while(true);
```

Write a Java program to enter the numbers till the user wants and at the end it should display the maximum and minimum number entered.

for Loop in Java

For Loop in Java

- The initialization statement is executed only once at the beginning of the for loop. Then the test expression is checked by the program. If the test expression is false, for loop is terminated.
- But if test expression is true then the code/s inside body of for loop is executed and then update expression is updated.
- This process repeats until test expression is false.

For Loop Declaration in Java

```
for(startExpression; testExpression; countExpression)
{
   block of code;
}
```

The **startExpression** is evaluated before the loop begins. It is acceptable to declare **and** assign in the **startExpression** (such as **int x = 1;**). This **startExpression** is evaluated only once at the beginning of the loop.

The **testExpression** will evaluate to **TRUE** (nonzero) or **FALSE** (zero). While TRUE, the body of the loop repeat. When **testExpression** becomes **FALSE**, the looping stops and the program continues with the statement immediately following the **for** loop body in the program code.

The **countExpression** executes after each trip through the loop. The count may increase/decrease by an increment of 1 or of some other value.

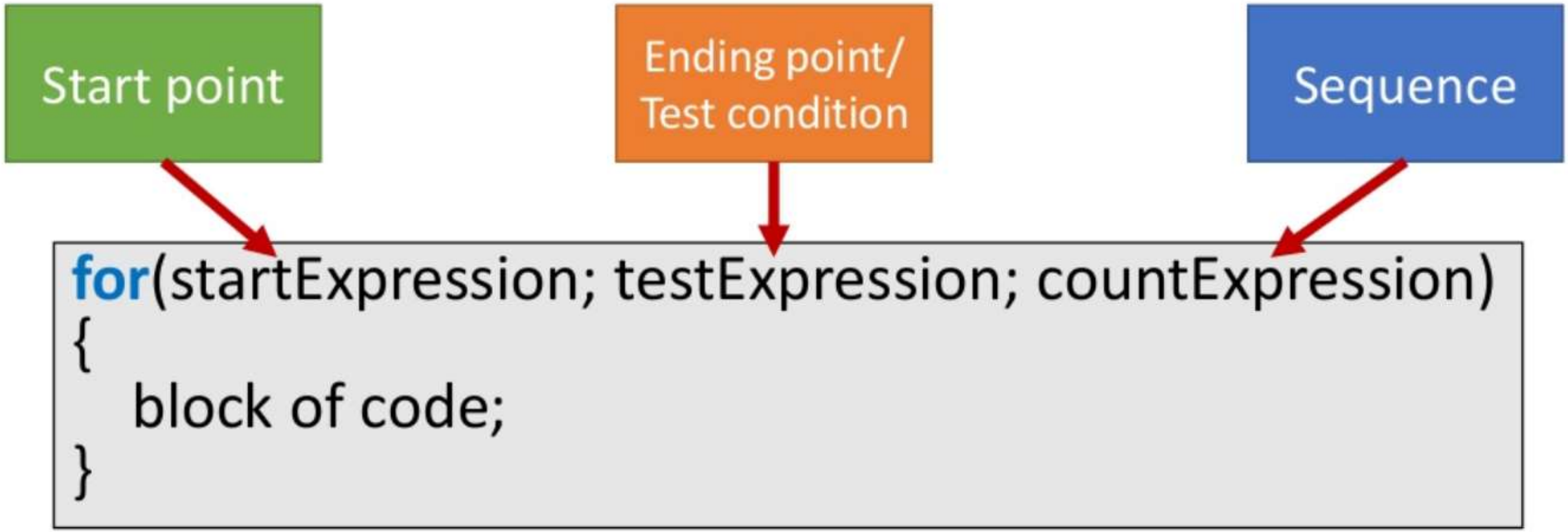
For Loop in Java

CAREFUL: When a **for** loop terminates, the value stored in the computer's memory under the looping variable will be "beyond" the testExpression in the loop. It must be sufficiently large (or small) to cause a false condition. Consider:

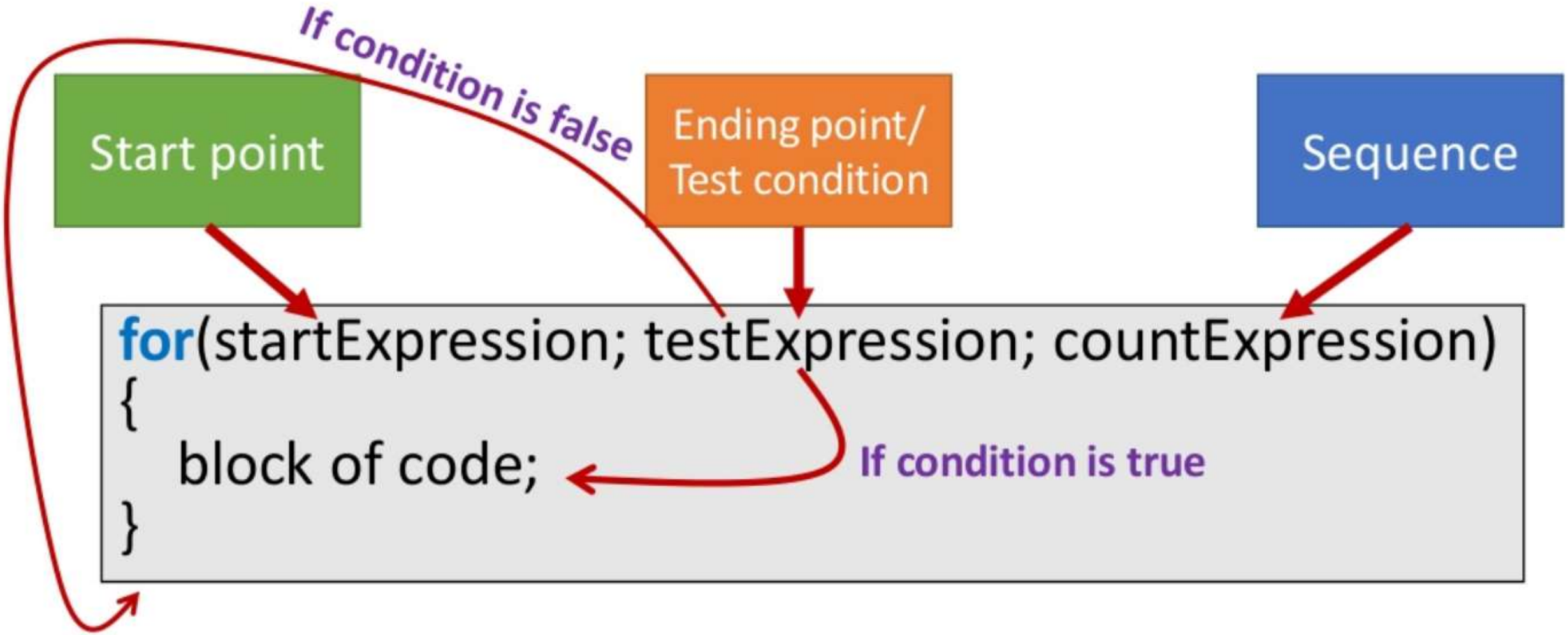
```
for (x = 0; x <= 13; x++){
    cout<<"Melody";
}
```

When this loop is finished, the value 14 is stored in x.

Syntax of For Loop



Syntax of For Loop



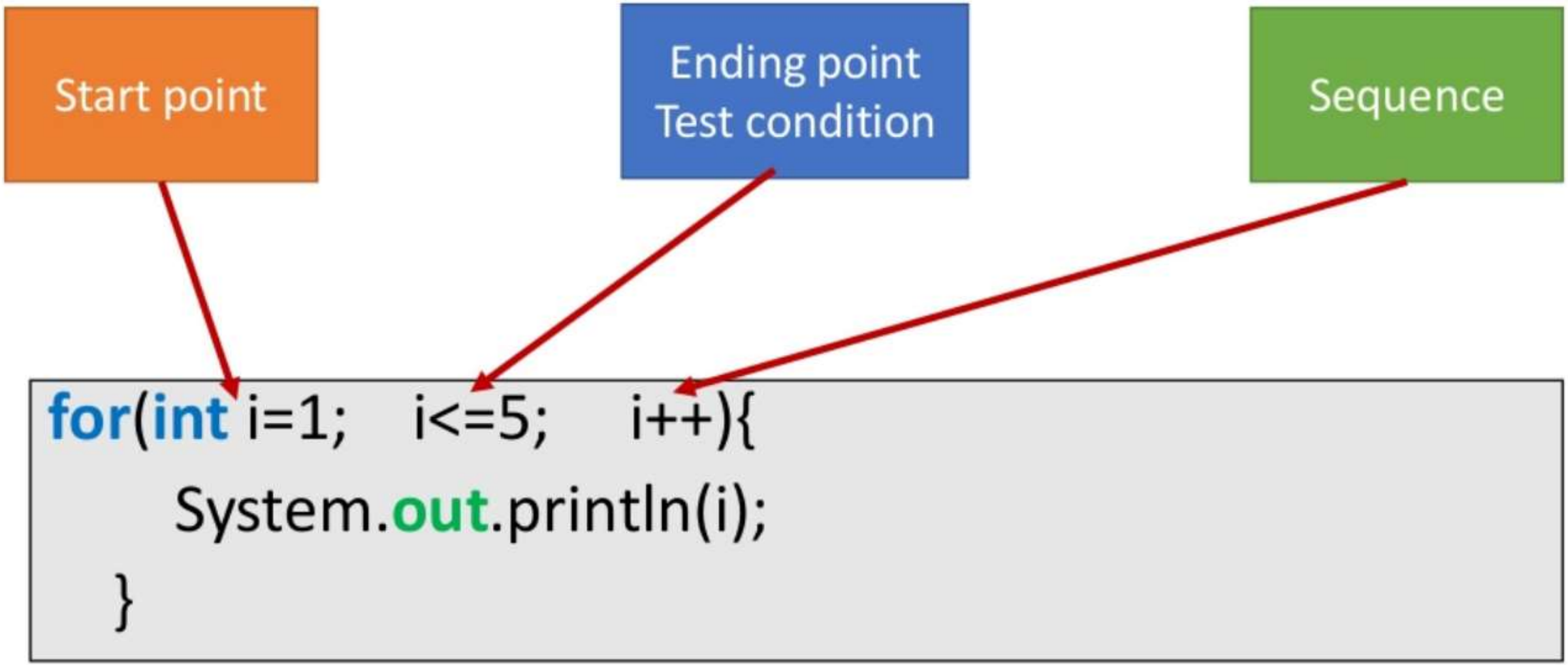
For Loop Example in java

```
13  public class Test {
14      public static void main(String[] args) {
15          for(int i=1;i<=5;i++){
16          System.out.println(i);
17      }
18
19      }
20  }
```

test.Test

Output - Test (run)

```
run:
1
2
3
4
5
```


Start point
Ending point
Test condition
Sequence
for(int i=1; i<=5; i++){
System.out.println(i);
}

Problem Statement

- Write the table of 5 using for loop.

Solution of Previous Problem

```
13    public class Test {
14        public static void main(String[] args) {
15            for(int i=1;i<=10;i++){
              System.out.println("5X"+i+" = "+(5*i));
17        }
18
19        }
20    }
```

test.Test > main > for (int i = 1; i <= 10; i++)

Output - Test (run)

```
run:
5X1 = 5
5X2 = 10
5X3 = 15
5X4 = 20
5X5 = 25
5X6 = 30
5X7 = 35
5X8 = 40
5X9 = 45
5X10 = 50
```

Problem Statement

- Write a Java program to calculate the factorial of a number.
- **Hint**
- The factorial of a number 'n' is the product of all number from 1 up to the number 'n‘ it is denoted by n!. For example n=5 then factorial of 5 will be 1*2*3*4*5= 120. 5!= 120.

Another Example of For Loop

```
import java.util.Scanner;
public class Test {
   public static void main(String[] args) {
      int i, n, factorial=1;
      Scanner user_input= new Scanner(System.in);
      System.out.println("Please Enter a Number: ");
      n=user_input.nextInt();
      for (i = 1; i <= n; ++i) {
      factorial *= i;     // factorial = factorial * i
      }
      System.out.println("Factorial of "+n+" is "+factorial);
   }
}
```

Please Note:

- In the Previous program you can see that, variable i is not used outside for loop. In such case, it is better to define variable at the time of initialization statement.

```
for (int i = 0; i <= n; ++i){
  ... .. ...
}
```

In the above Java code, i is local to for loop, that is, you cannot use it outside for loop but makes program more readable.

- Two numbers are entered through the keyboard. Write a java program to find the value of one number raised to the power of another.

Java Infinitive For Loop

- If you use two semicolons ;; in the for loop, it will be infinitive for loop.
- **Syntax:**

```
for(;;){
    //code to be executed
}
```

Java Infinitive For Loop

```
public class Test {
public static void main(String[] args) {
    for(;;){
        System.out.println("infinitive loop");
    }
}
}
```

Output is:

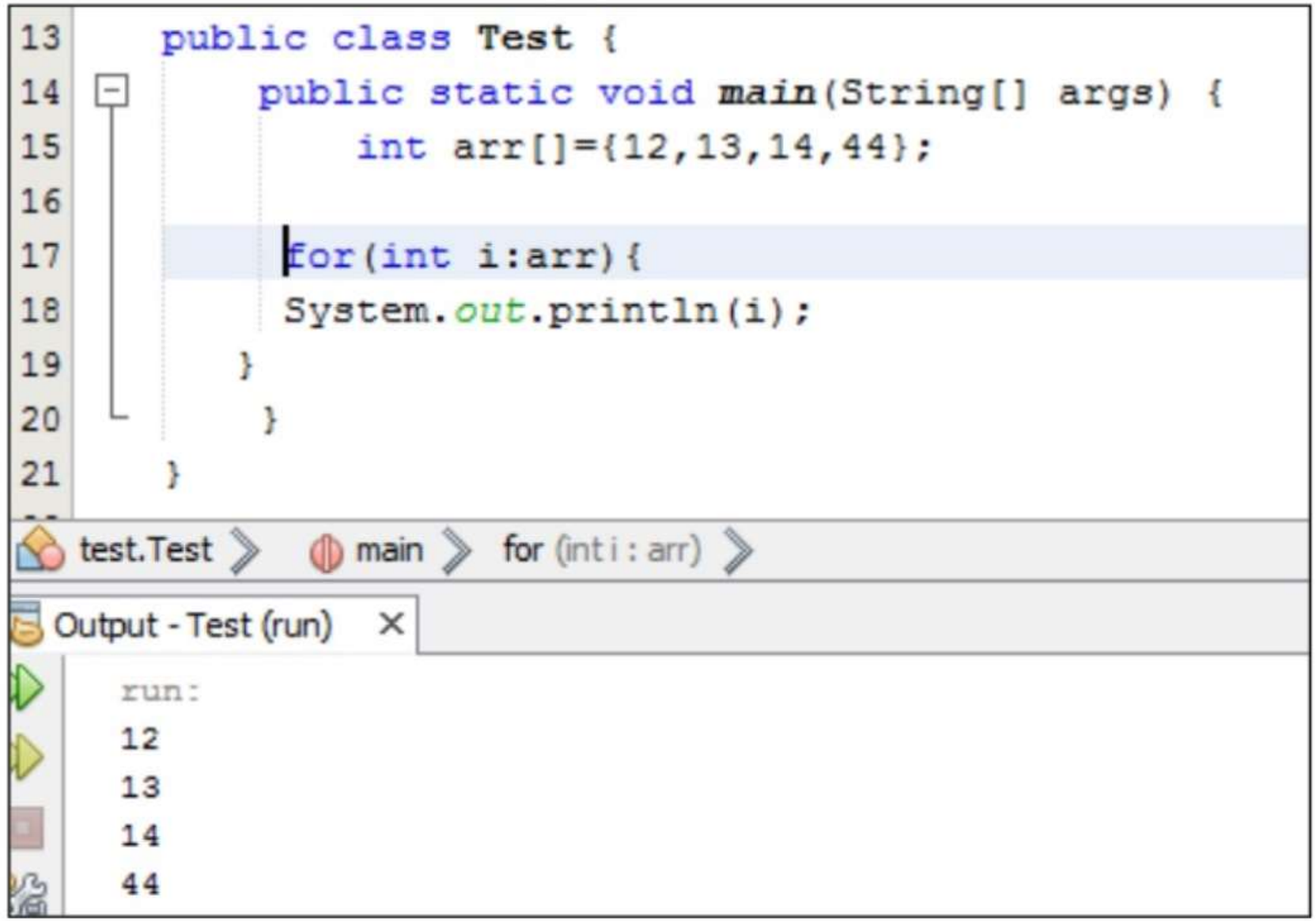
```
infinitive loop
infinitive loop
infinitive loop
infinitive loop
infinitive loop
ctrl+c
```

For-Each Loop

- The for-each loop introduced in Java5. It is mainly used to traverse array or collection elements. The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.
- **Advantage of for-each loop:**
  - It makes the code more readable.
  - It eliminates the possibility of programming errors.
- **Syntax of for-each loop:**

```
for(data_type variable : array | collection){}
```

Example of for-each loop for traversing the array elements:



Example of for-each loop for traversing the array elements:

```
public class Test {
    public static void main(String[] args) {
        int arr[]={12,13,14,44};

     for(int i:arr){
     System.out.println(i);
   }
    }
}
```

Another Example of For Each Loop

```
13  public class Test {
14      public static void main(String[] args) {
15          String Strex;
16          Strex ="Name is: Adil Aslam and age is: 21 and salary is: 400$";
17                for (String Strsplit: Strex.split(" and ")){
18
19                    System.out.println(Strsplit);
20
21                        }
22
23      }
24  }
```

test.Test

Output - Test (run)

```
run:
Name is: Adil Aslam
age is: 21
salary is: 400$
```

Nested Loops in Java

- The placing of one loop inside the body of another loop is called **nesting**. When you "**nest**" two loops, the **outer loop** takes control of the number of complete repetitions of the inner loop.
- We can use loops inside loop body
- First inner loop complete its iteration and then control shift to outer one, this process continues till end.

Nested Loops in Java

- **Nested While Loop Syntax**

```
while(condition)
{
      while(condition)
   {
        statement(s);
    }
  statement(s);
}
```

Nested Loops in Java

- **Nested While Loop Syntax**

```
while(condition)
{
      while(condition)
   {
       statement(s);
    }
  statement(s);
}
```

Example of Nested While Loop

```
13    public class Test {
14        public static void main(String[] args) {
15            int outer = 1;
16             while(outer < 3)
17             {
18                 int inner = 5;
19                 while(inner < 8)
20                 {
21                     System.out.println(outer + "   " + inner);
22                     inner++;
23                 }
24                 outer++;
25             }
26        }
27    }
```

test.Test > main > while (outer < 3) > while (inner < 8)

Output - Test (run)

```
run:
1   5
1   6
1   7
2   5
2   6
2   7
```

```
int outer = 1;
      while(outer < 3)  {
         int inner = 5;
         while(inner < 8){
            System.out.println(outer + "  " + inner);
            inner++; }
         outer++; }
```

Output is:

```
1   5
1   6
1   7
2   5
2   6
2   7
```

- Here outer loop is executed for values going from 1 till less than 3 i.e. 1 and 2
- Similarly inner loop is executed from 5 till less than 8 i.e. 5, 6 and 7...
- When outer is 1, inner will become 5, 6, and 7
- And when outer is 2, then also inner will become 5, 6, and 7.

Nested Do While Loop Example

```
13  public class Test {
14      public static void main(String[] args) {
15          int i = 0;
16          do
17          {
18          int j = 0;
19              do
20                {
21              System.out.println("i= "+i+" j= "+j);
22              j++;
23               }while(j<3);
24              i++;
25          }while(i<2 );
26      }
27  }
```

test.Test

Output - Test (run)

```
run:
i= 0 j= 0
i= 0 j= 1
i= 0 j= 2
i= 1 j= 0
i= 1 j= 1
i= 1 j= 2
```

Nested Loops in Java

- **Nested For Loop Syntax**

```
for(initialize; condition; increment)
{
      for (initialize; condition; increment)
   {
        statement(s);
   }
 statement(s);
}
```

Nested For Loop Example

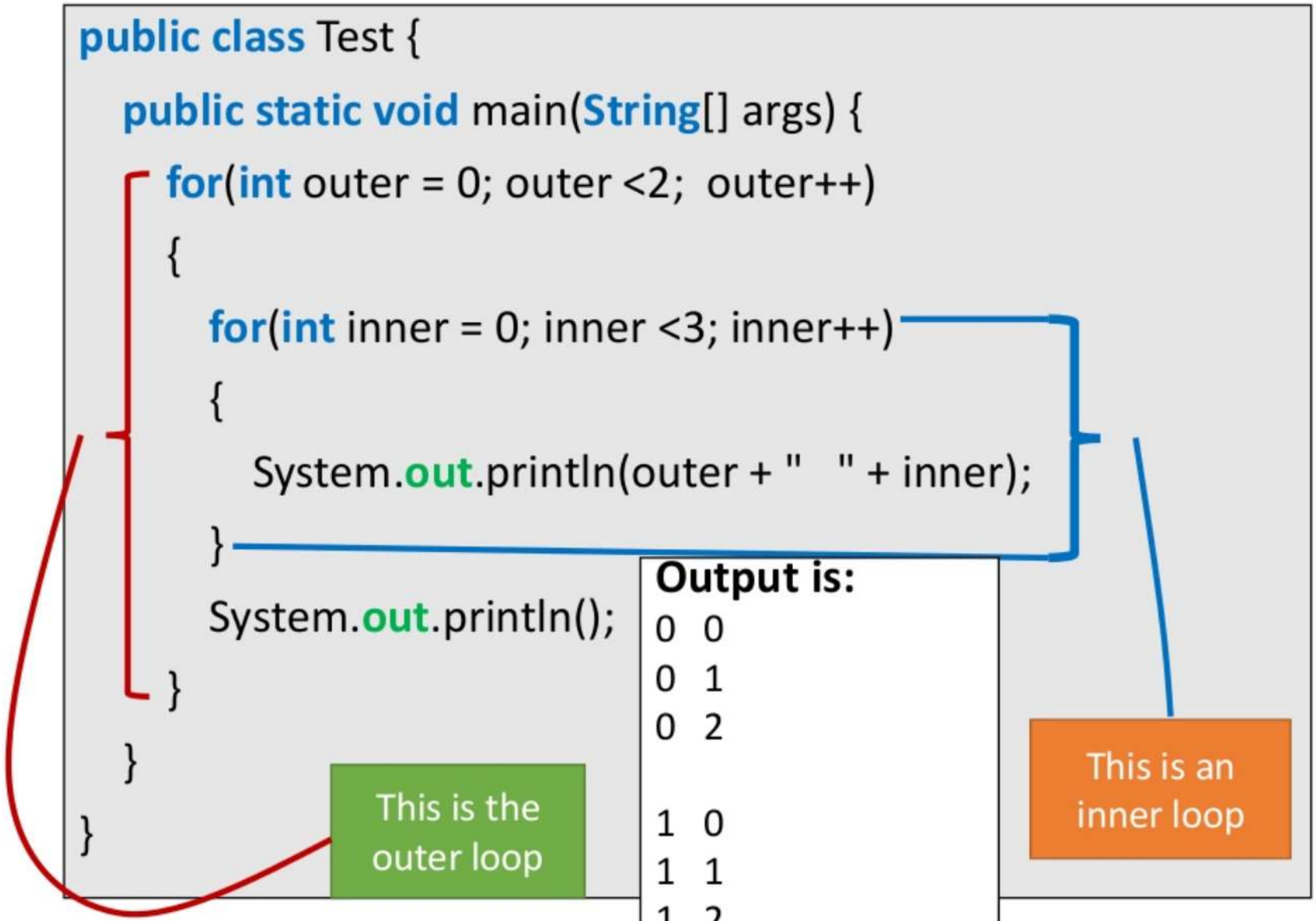
```
13    public class Test {
14        public static void main(String[] args) {
15            for(int outer = 0; outer <2;  outer++)
16            {
17                for(int inner = 0; inner <3; inner++)
18                {
19                    System.out.println(outer + "    " + inner);
20                }
21                System.out.println();
22            }
23        }
24    }
```

test.Test > main >

Output - Test (run)

```
run:
0    0
0    1
0    2

1    0
1    1
1    2
```


public class Test {
public static void main(String[] args) {
for(int outer = 0; outer <2; outer++)
{
for(int inner = 0; inner <3; inner++)
{
System.out.println(outer + " " + inner);
}
System.out.println();
}
}
}
Output is:
0 0
0 1
0 2
1 0
1 1
1 2
This is the outer loop
This is an inner loop

Another Nested for Loop Example

```
13  public class Test {
14      public static void main(String[] args) {
15          int n=5;
16          for (int i = 1; i <= n; i++) {
17              for (int j = 1; j <= i; j++) {
18              System.out.print("#");
19              }
20          System.out.println();
21
22          }
23      }
24  }
```

test.Test > main

Output - Test (run)

```
run:
#
##
###
####
#####
```

Controlling Loops/Jumping out of Loop

- Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain condition becomes true, that is jump out of loop.

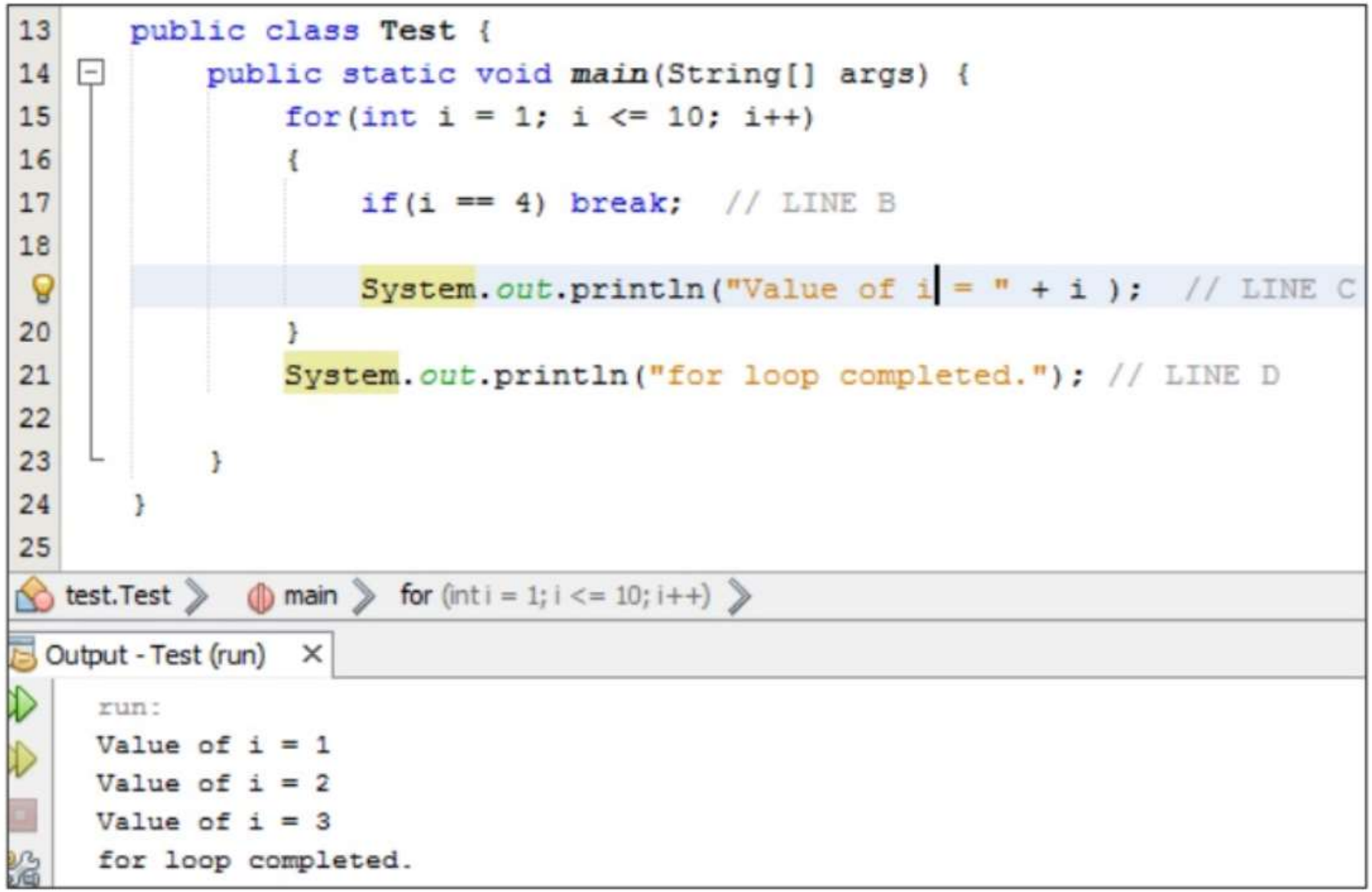
- We can use **break** and **continue** statement to control the interactions of a loop.

Controlling Loops/Jumping out of Loop

- **Break Statement**
  - **break** statement **stops the loops** and start executing program **from the line after loop**.

- **continue statement**
  - **continue** statement breaks only the **current iteration**.

Java Break Statement With Loop



Java Break Statement With Inner Loop

```
13  public class Test {
14      public static void main(String[] args) {
15          for(int i=1;i<3;i++){
                for(int j=1;j<3;j++){
17                  if(i==2&&j==2){
18                      break;
19                  }
20                  System.out.println(i+" "+j);
21              }
22          }
23
24      }
25  }
```

test.Test > main > for (int i = 1; i < 3; i++) > for (int j = 1; j < 3; j++) >

Output - Test (run)

```
run:
1 1
1 2
2 1
```

Java Continue Statement

- The Java *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.
- **Syntax:**

```
jump-statement;
continue;
```

Example of Continue Statement

```
public class Test {
    public static void main(String[] args) {
        for(int a = 1; a < 10; a++){
          if(a == 5)
          {continue;}
          System.out.print(a + " ");
           }
    }
}
```

Output is:

1 2 3 4 6 7 8 9

- The **println**("...") method prints the string "..." and moves the cursor to a new line.
- The **print**("...") method instead prints just the string "...", but does not move the cursor to a new line.

- So simply we can say:
- **Print()** = print on line
  println() = print on line, then move to next line
- Both methods belongs to PrintStream class.

Please Note:

- **print() Example**

```
for(int i = 0; i < 5; i++)
System.out.print(" " + i);
```

Output is:
0 1 2 3 4

- **println() Example**

```
for(int i = 0; i < 5; i++)
System.out.println(" " + i);
```

Output is:
0
1
2
3