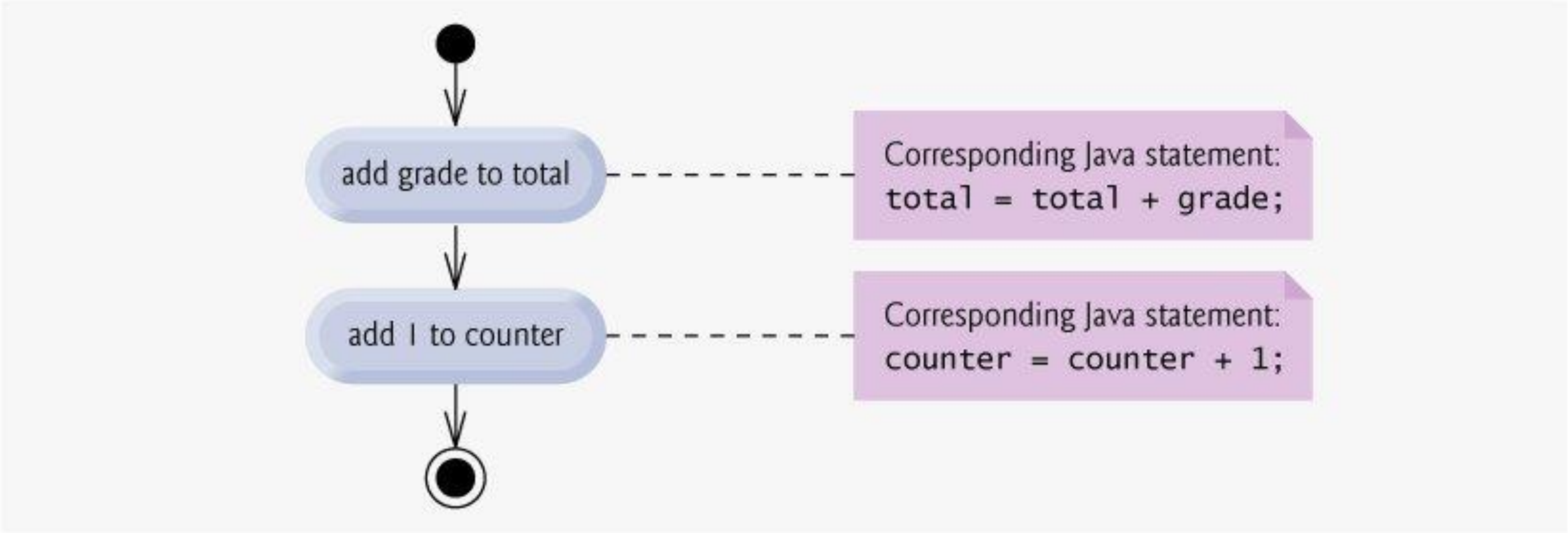
Tutorial 2

Control Flow, Methods, Arrays

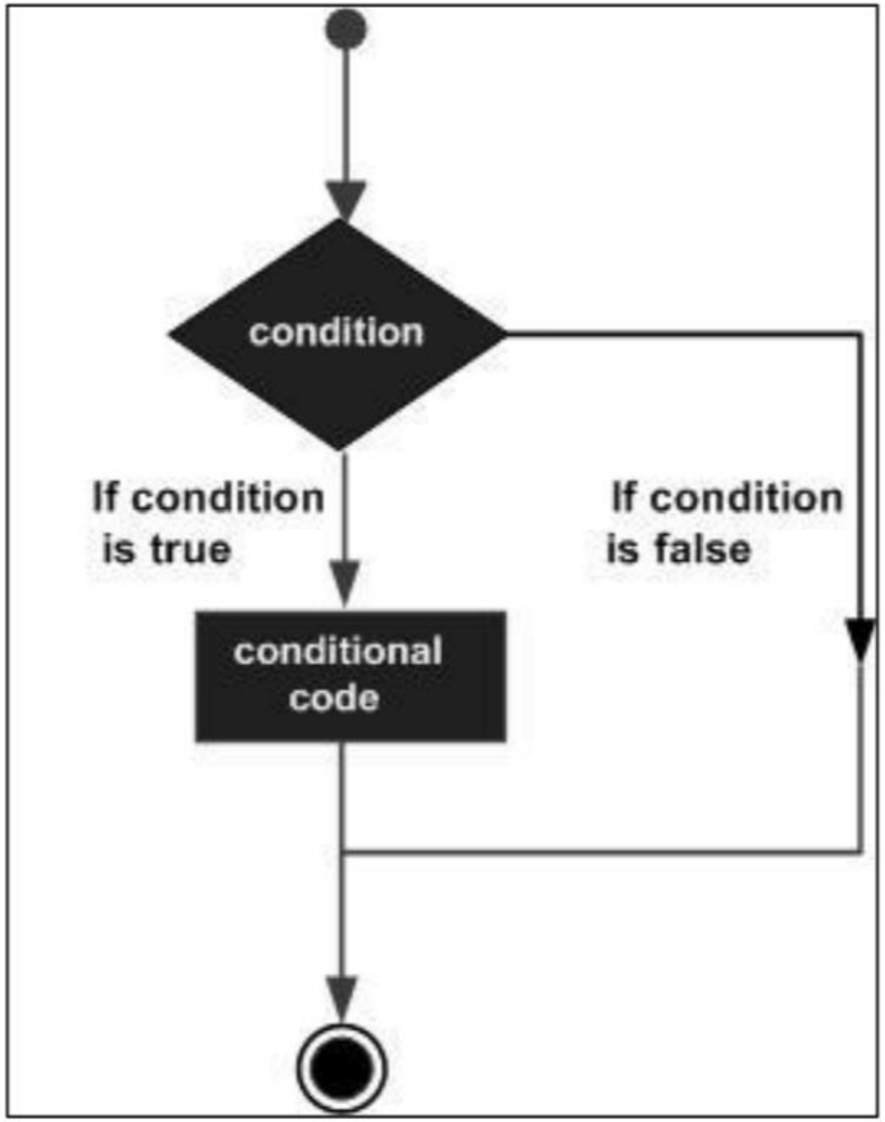
Control Flow

Sequence Structure

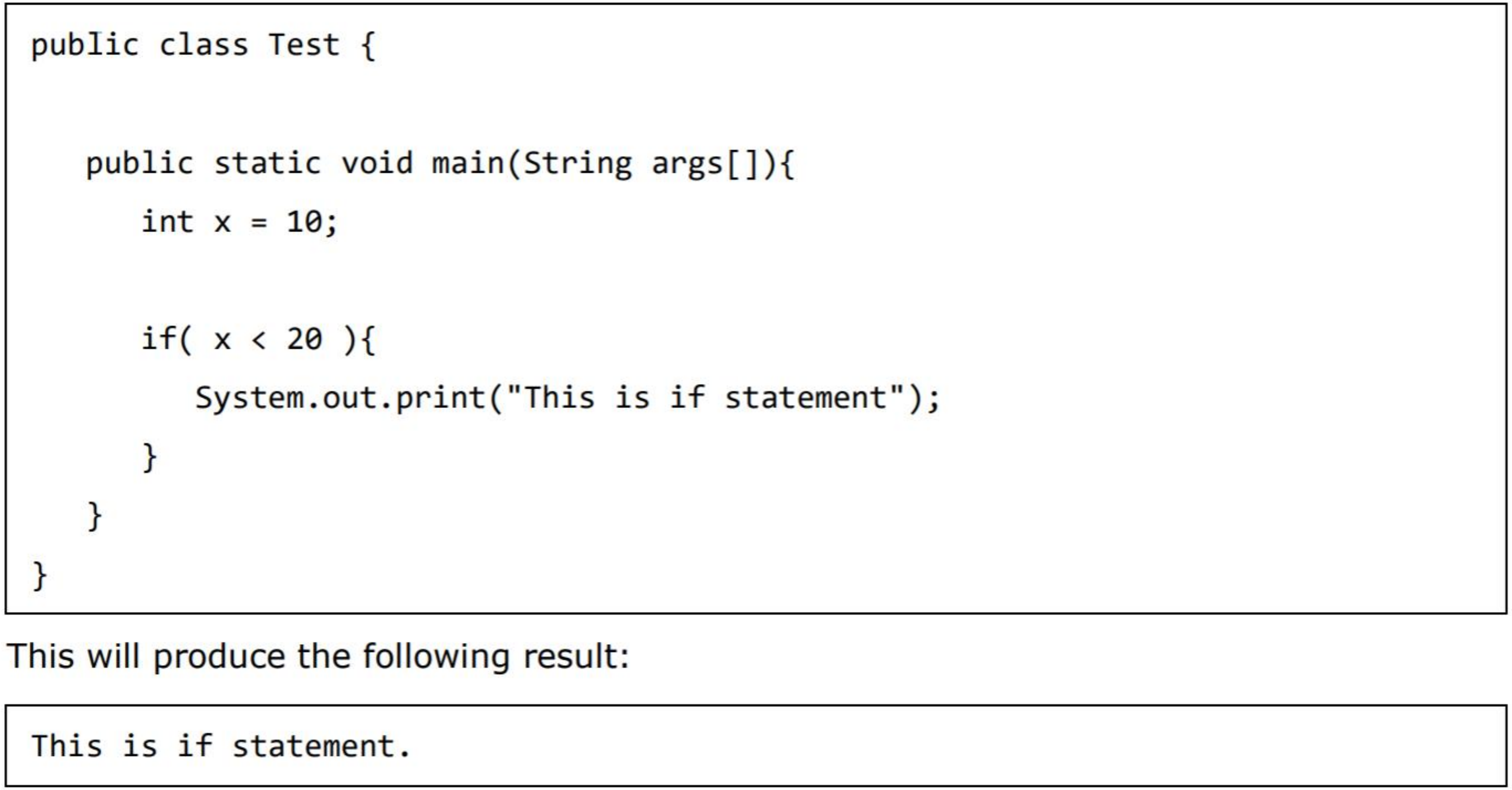


Selection Structure

- If Statement
- If-else statement
- If-else-if-else statement
- nested if statement
- switch statement
- Conditional operator



Example of if statement



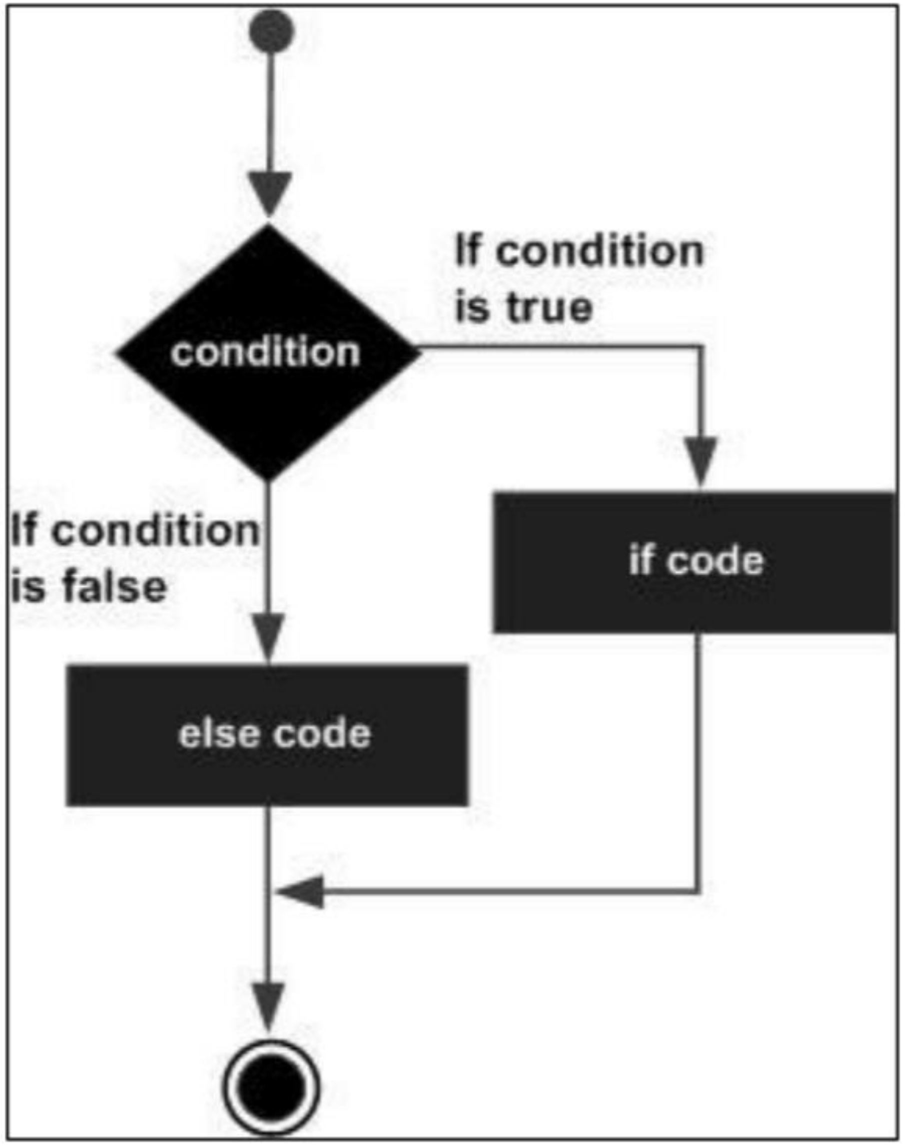
```
public class Test {


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


      if( x < 20 ){
         System.out.print("This is if statement");
      }
   }
}
```

This will produce the following result:

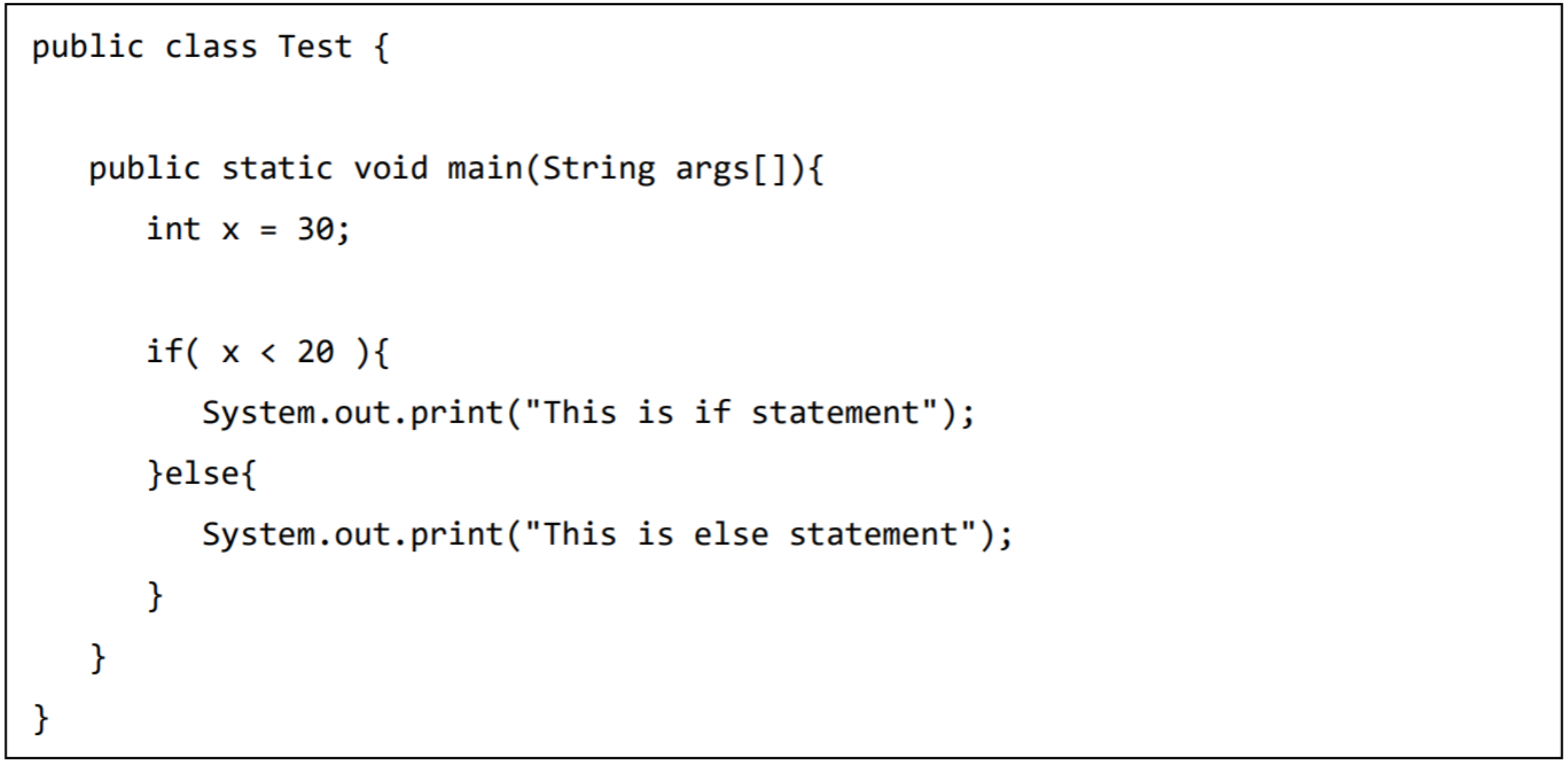
```
This is if statement.
```

Selection Structure

- If-else Statement

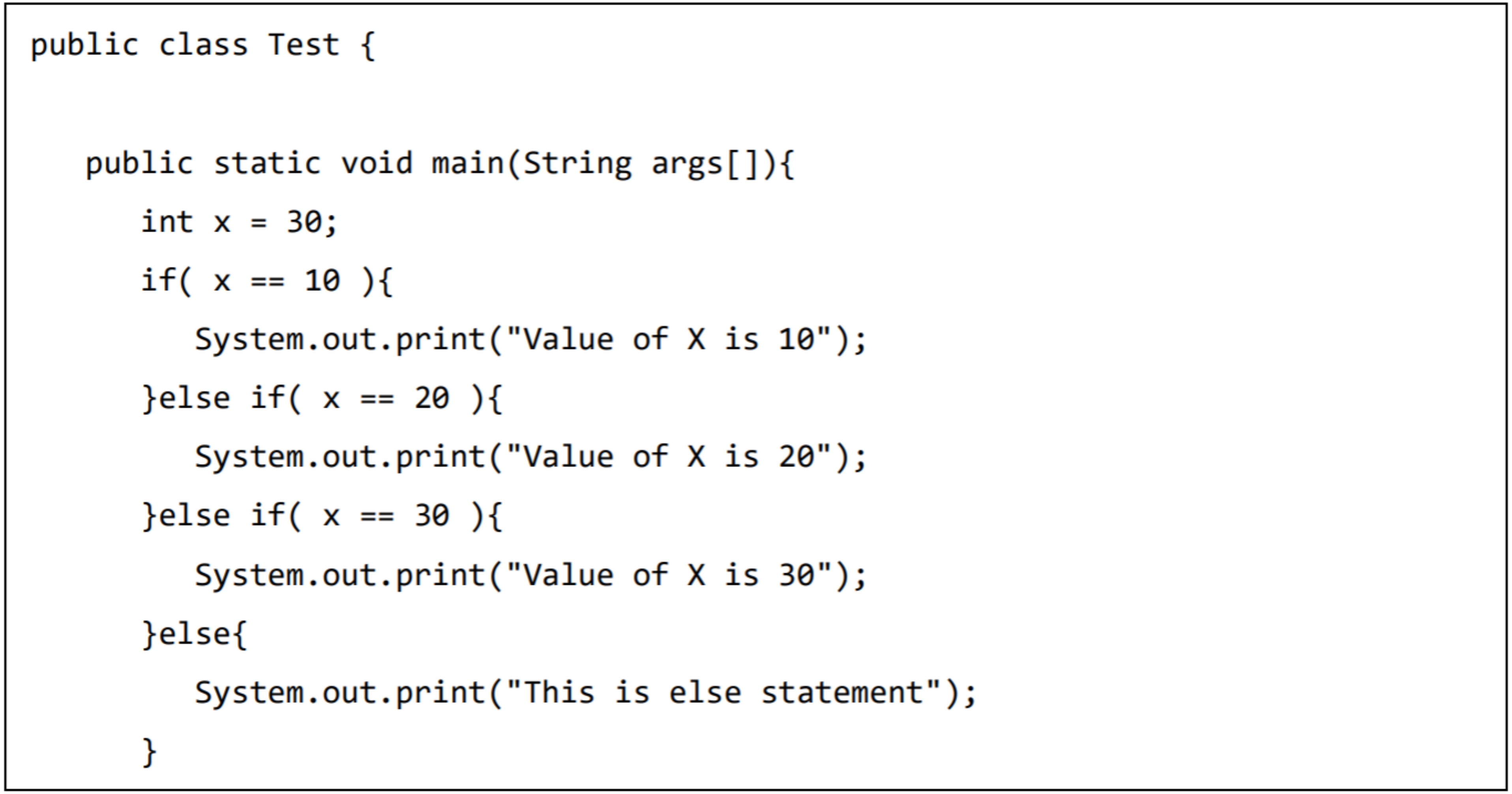


Example of if-else statement



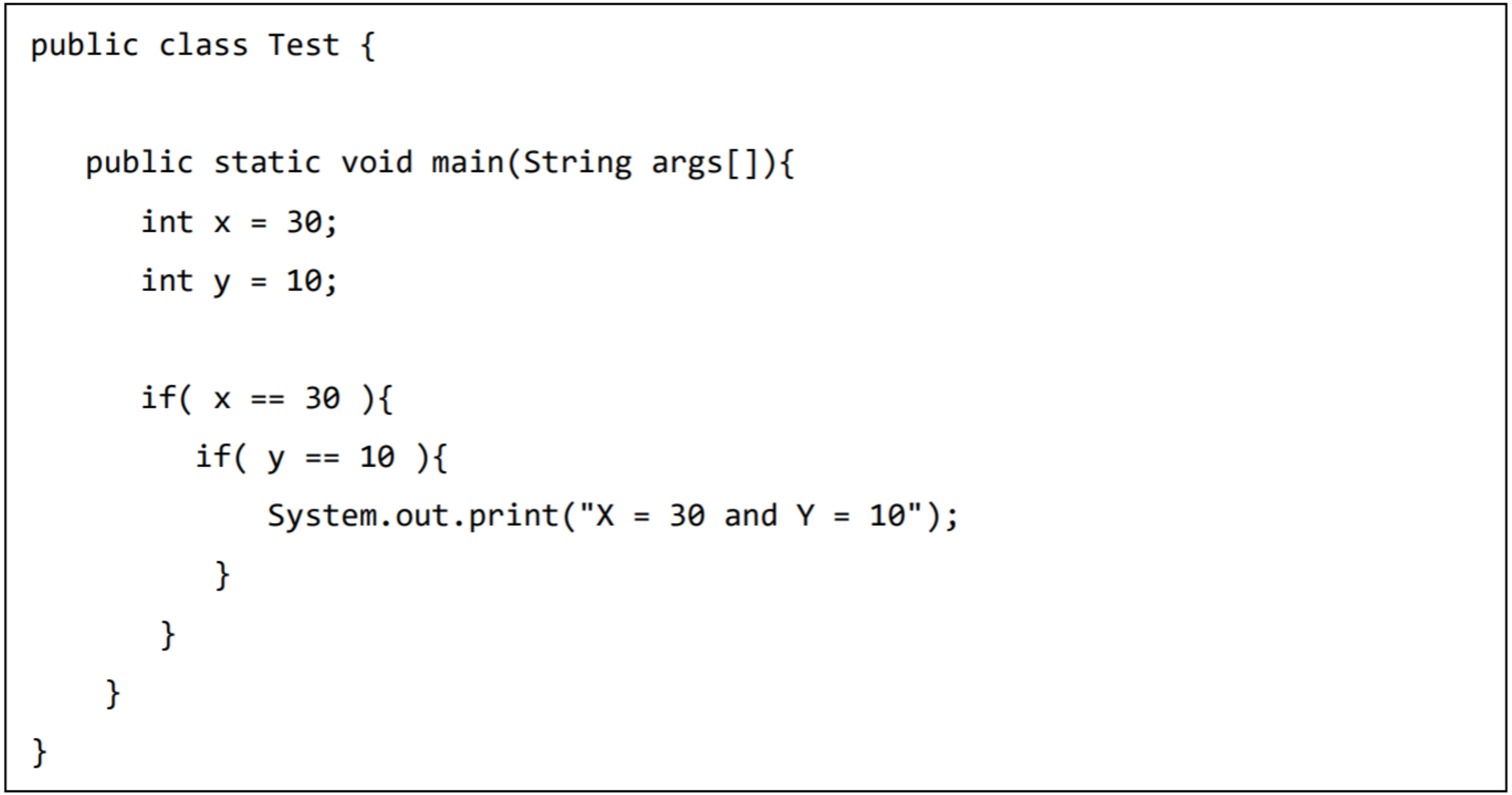
```
public class Test {

   public static void main(String args[]){
      int x = 30;

      if( x < 20 ){
         System.out.print("This is if statement");
      }else{
         System.out.print("This is else statement");
      }
   }
}
```

If-else-if-else statement



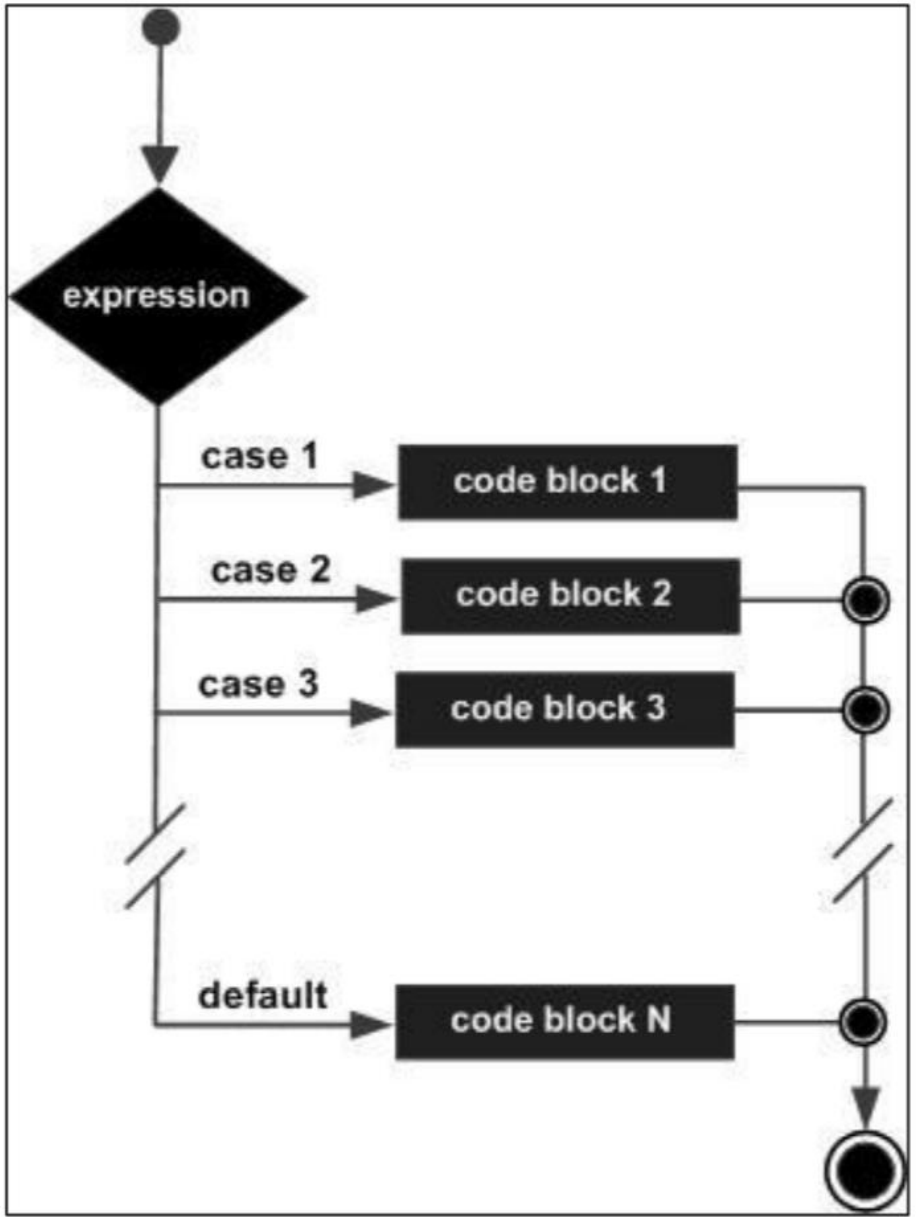
```
public class Test {


   public static void main(String args[]){
      int x = 30;
      if( x == 10 ){
         System.out.print("Value of X is 10");
      }else if( x == 20 ){
         System.out.print("Value of X is 20");
      }else if( x == 30 ){
         System.out.print("Value of X is 30");
      }else{
         System.out.print("This is else statement");
      }
```

Nested if-else statements

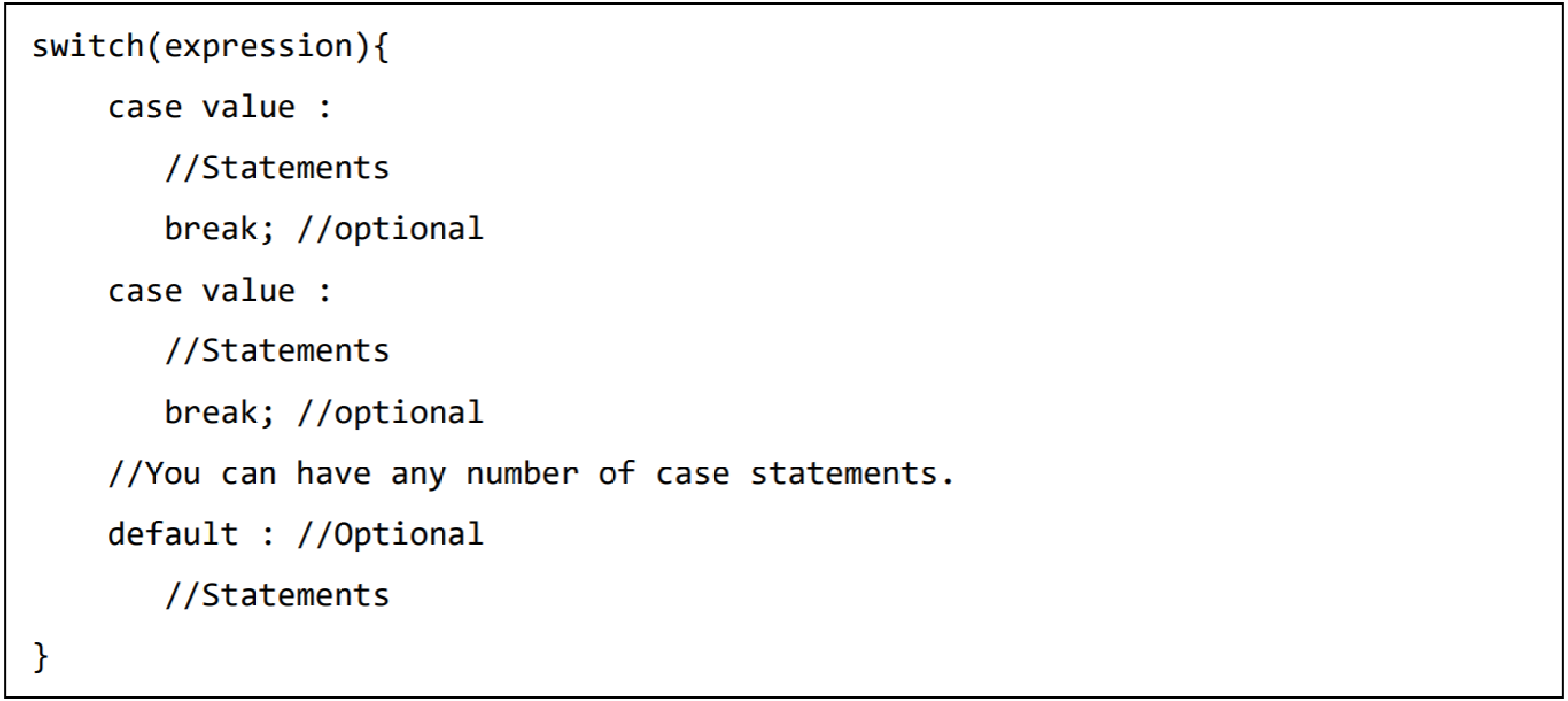


```
public class Test {

   public static void main(String args[]){
      int x = 30;
      int y = 10;

      if( x == 30 ){
         if( y == 10 ){
             System.out.print("X = 30 and Y = 10");
          }
       }
    }
}
```

Switch statement



Switch statement

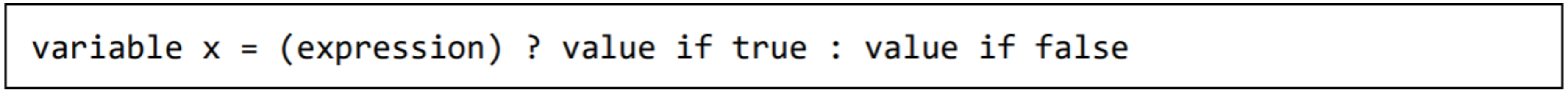


```
switch(expression){
    case value :
       //Statements
       break; //optional
    case value :
       //Statements
       break; //optional
    //You can have any number of case statements.
    default : //Optional
       //Statements
}
```

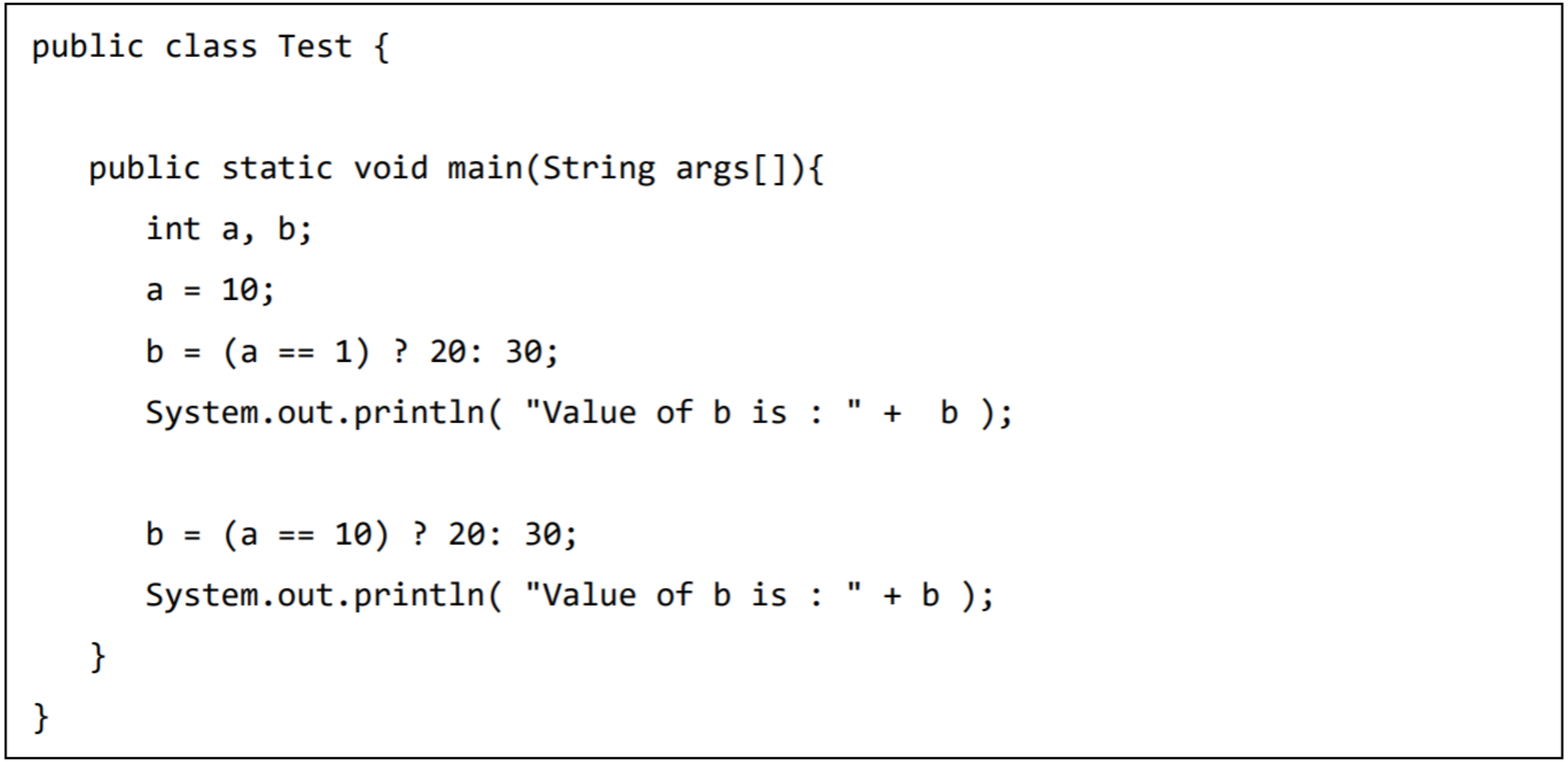
Switch Statement

- Test for all possible values
- Always provide a default case
- Place the default case last so that the break for that case is not required

Conditional Operator (The ?: Operator)



```
variable x = (expression) ? value if true : value if false
```

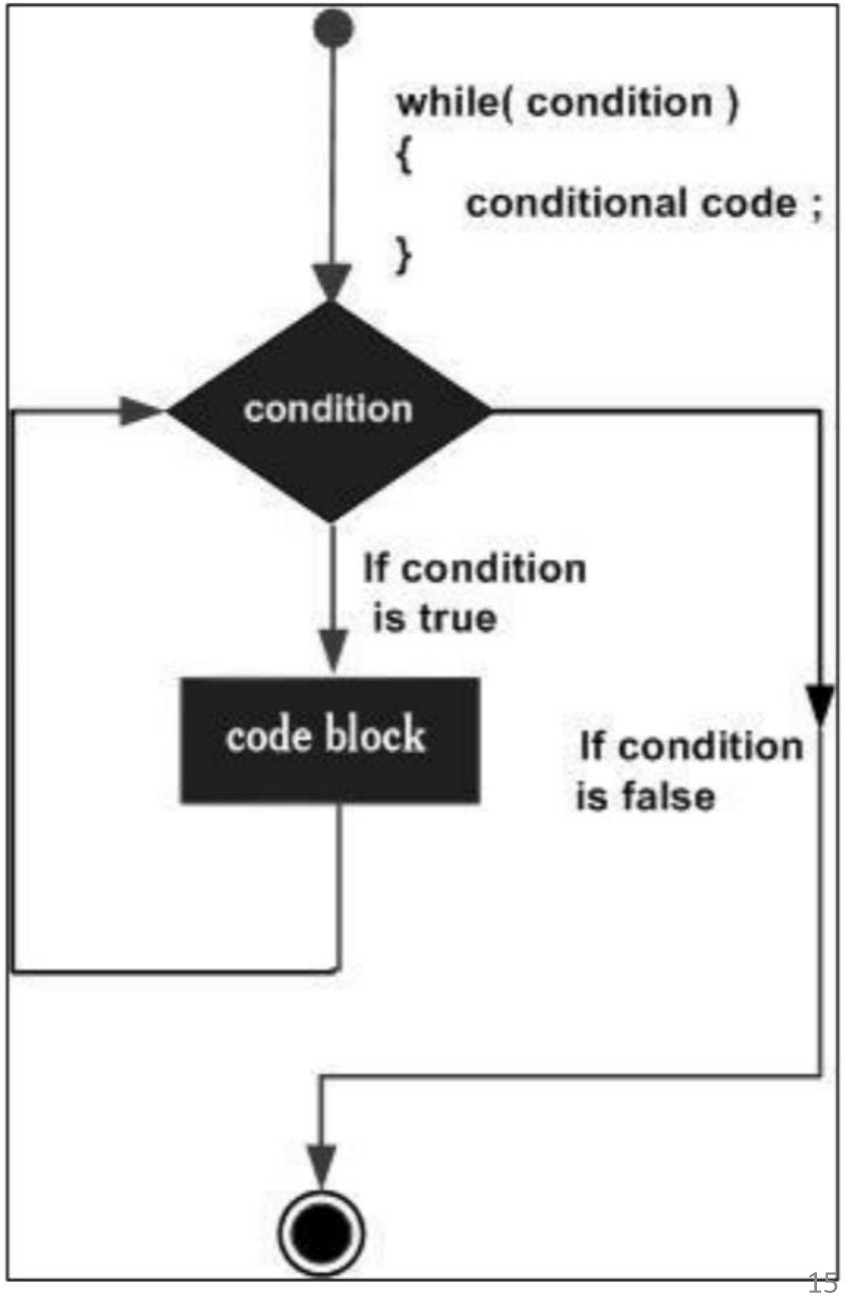


```java
public class Test {

   public static void main(String args[]){
      int a, b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

Repetition Structure

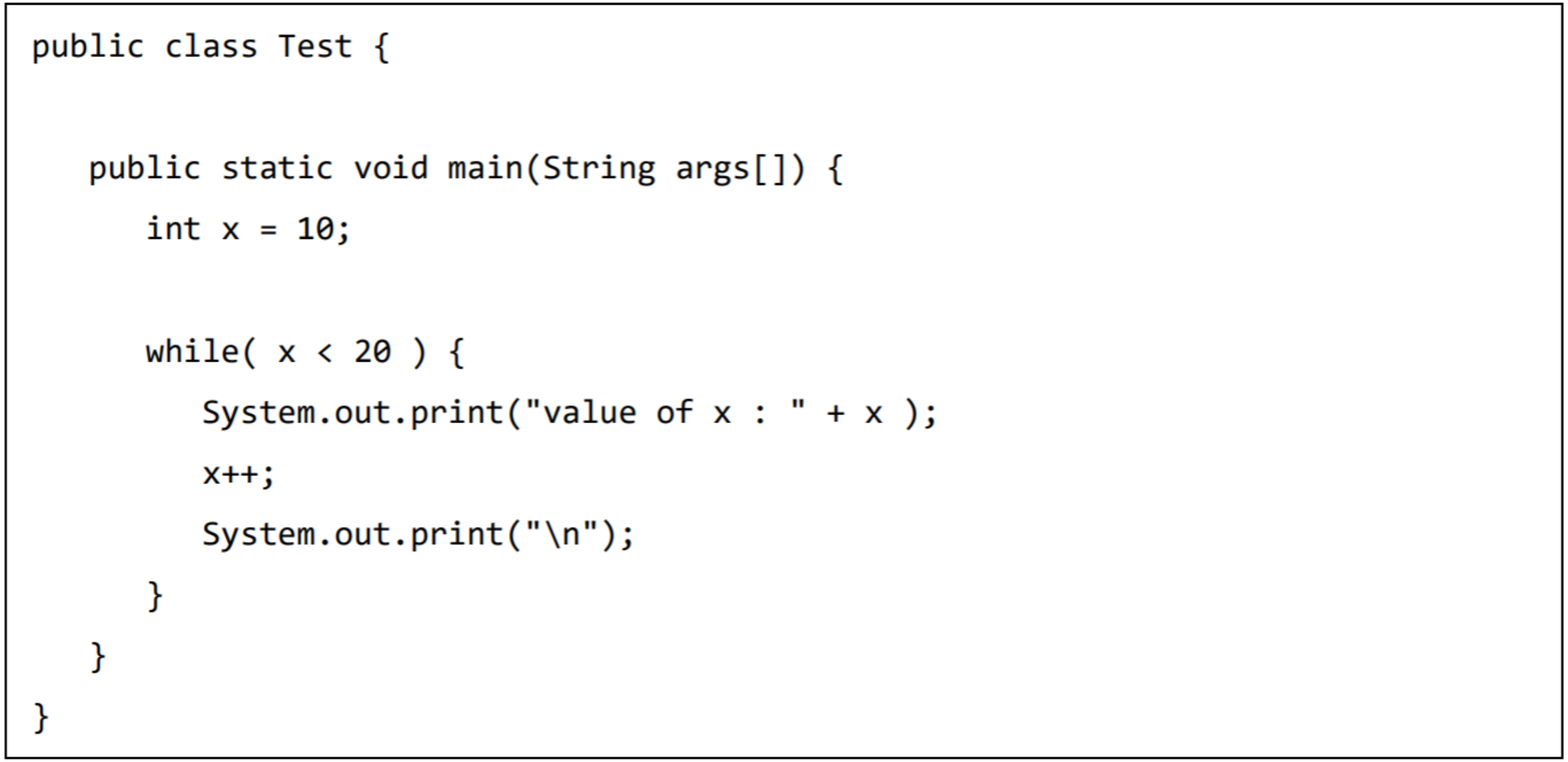
- While loop
- For loop
- Do…while loop

While loop



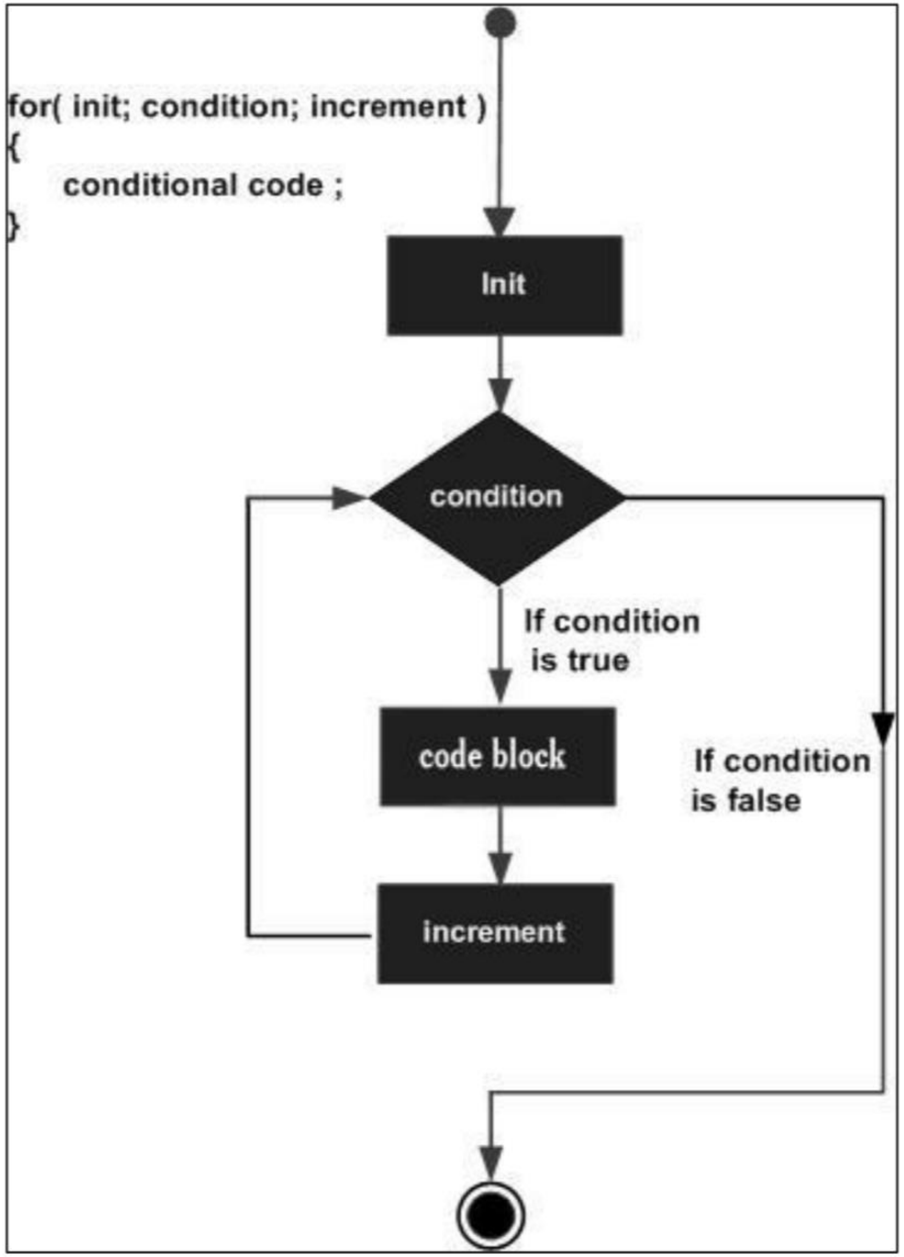
while(condition)
{
conditional code ;
}
condition
If condition is true
code block
If condition is false

While loop



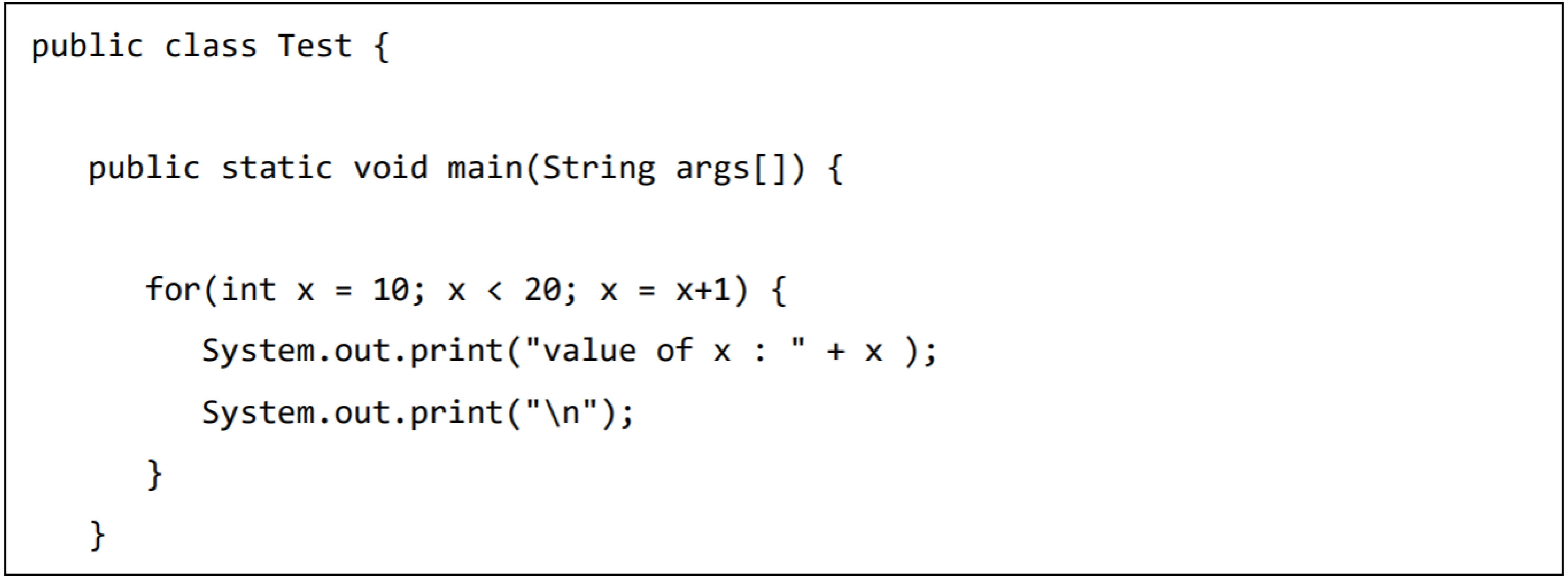
```
public class Test {

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

      while( x < 20 ) {
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
   }
}
```

For loop



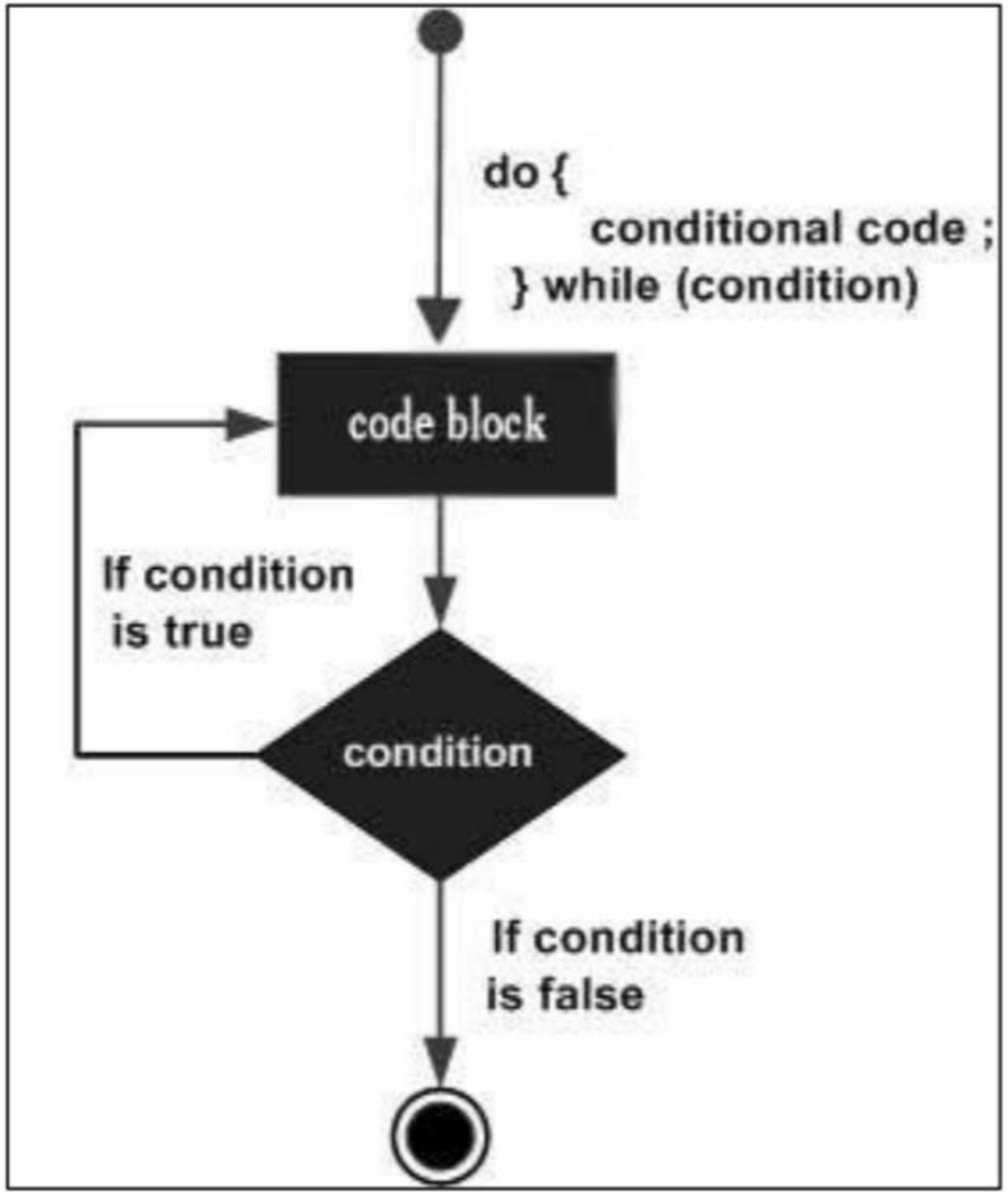
```
for( init; condition; increment )
{
    conditional code ;
}
```

For loop

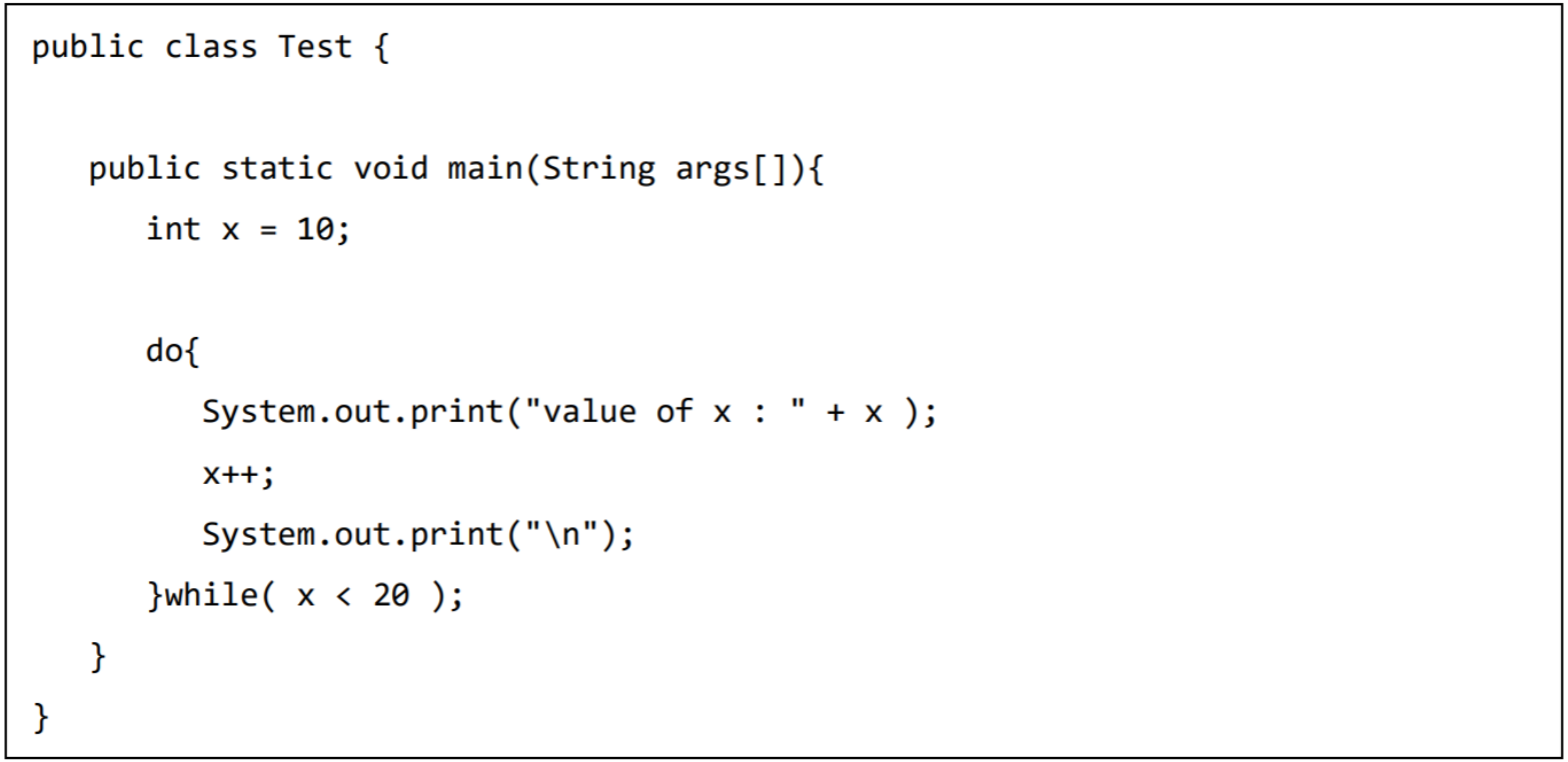


```
public class Test {

   public static void main(String args[]) {

      for(int x = 10; x < 20; x = x+1) {
         System.out.print("value of x : " + x );
         System.out.print("\n");
      }
   }
```

Do While loop



Do While loop

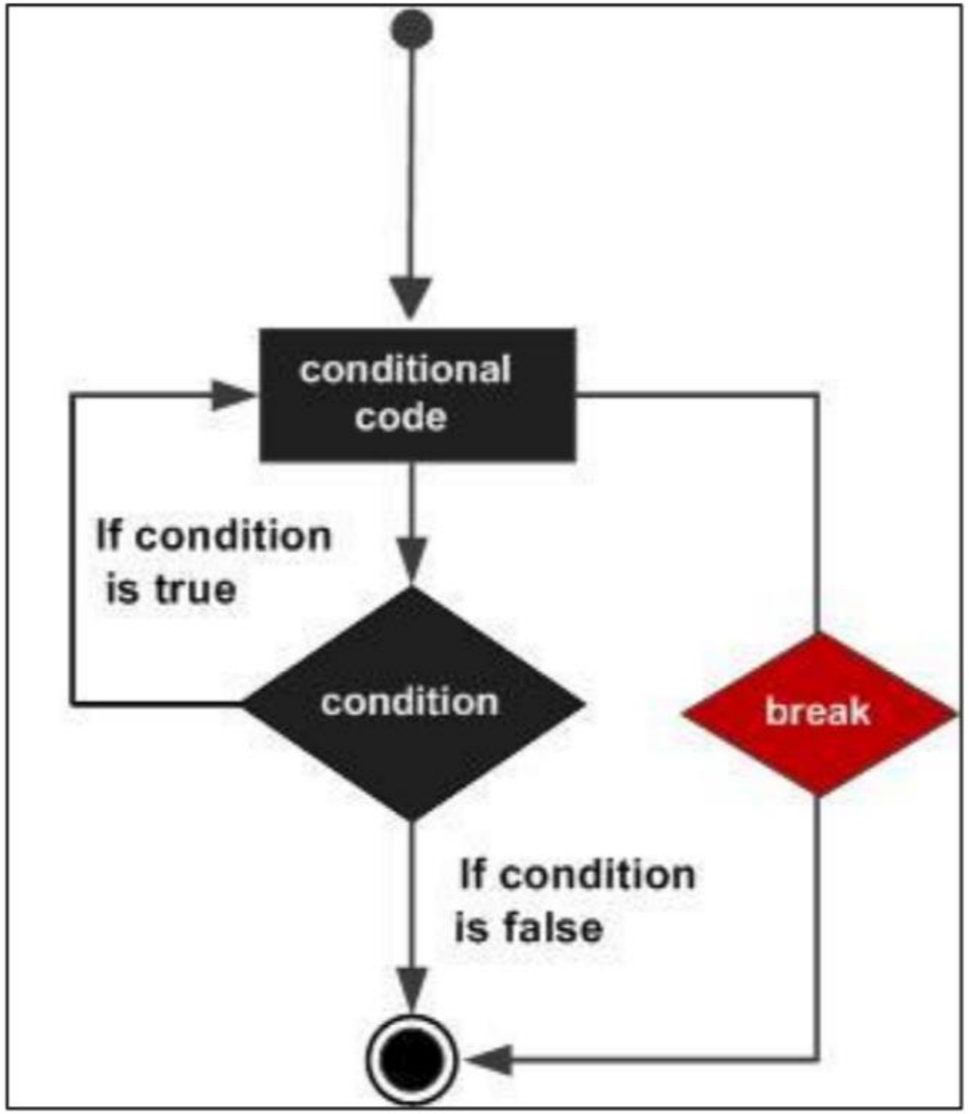


```
public class Test {

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

      do{
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }while( x < 20 );
   }
}
```

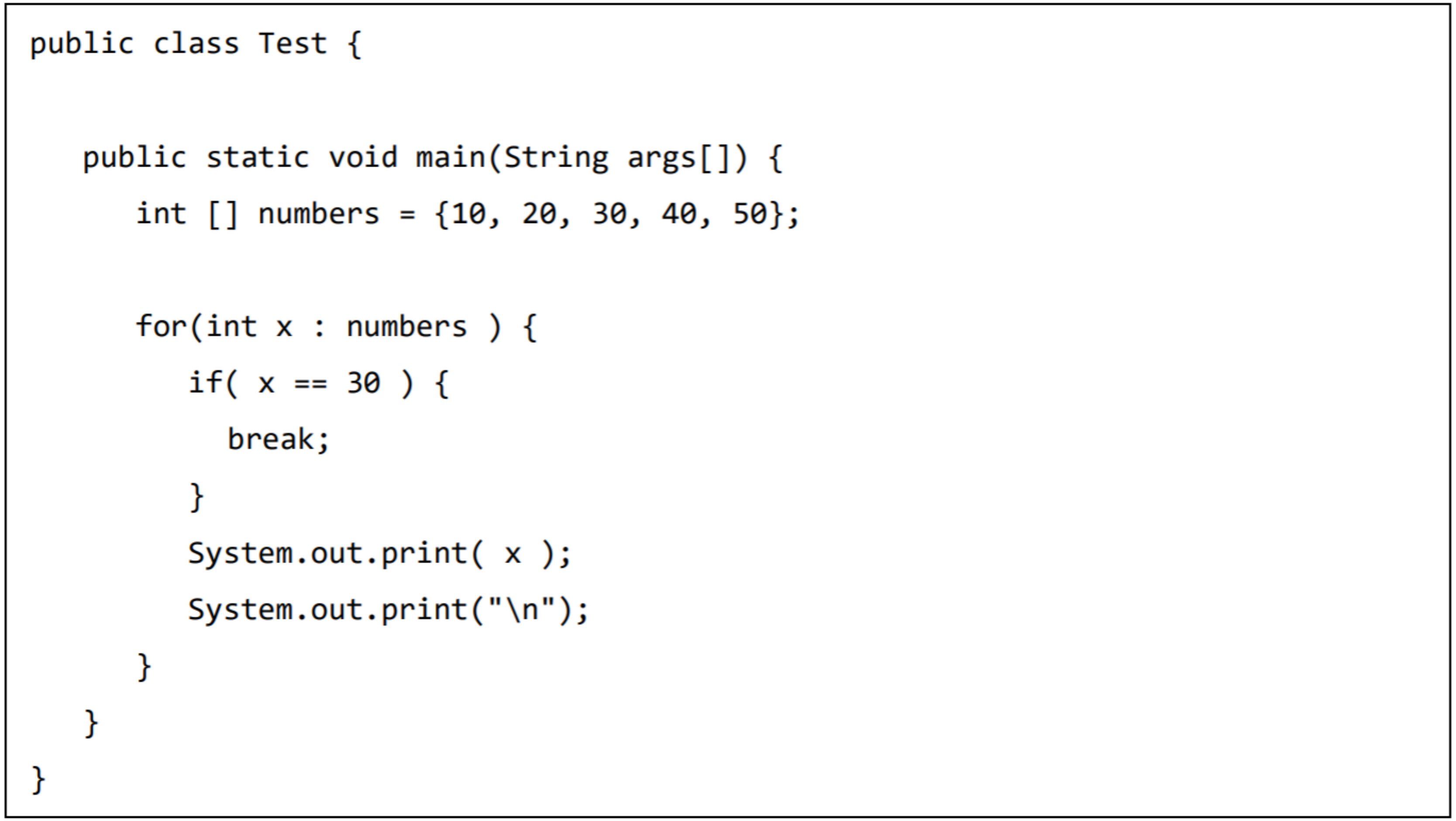
Loop Control Statements

- Break statement:
  - Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch
- Continue Statement:
  - Causes the loop to skip the reminder of its body and immediately retest its condition prior to reiterating

Break statement

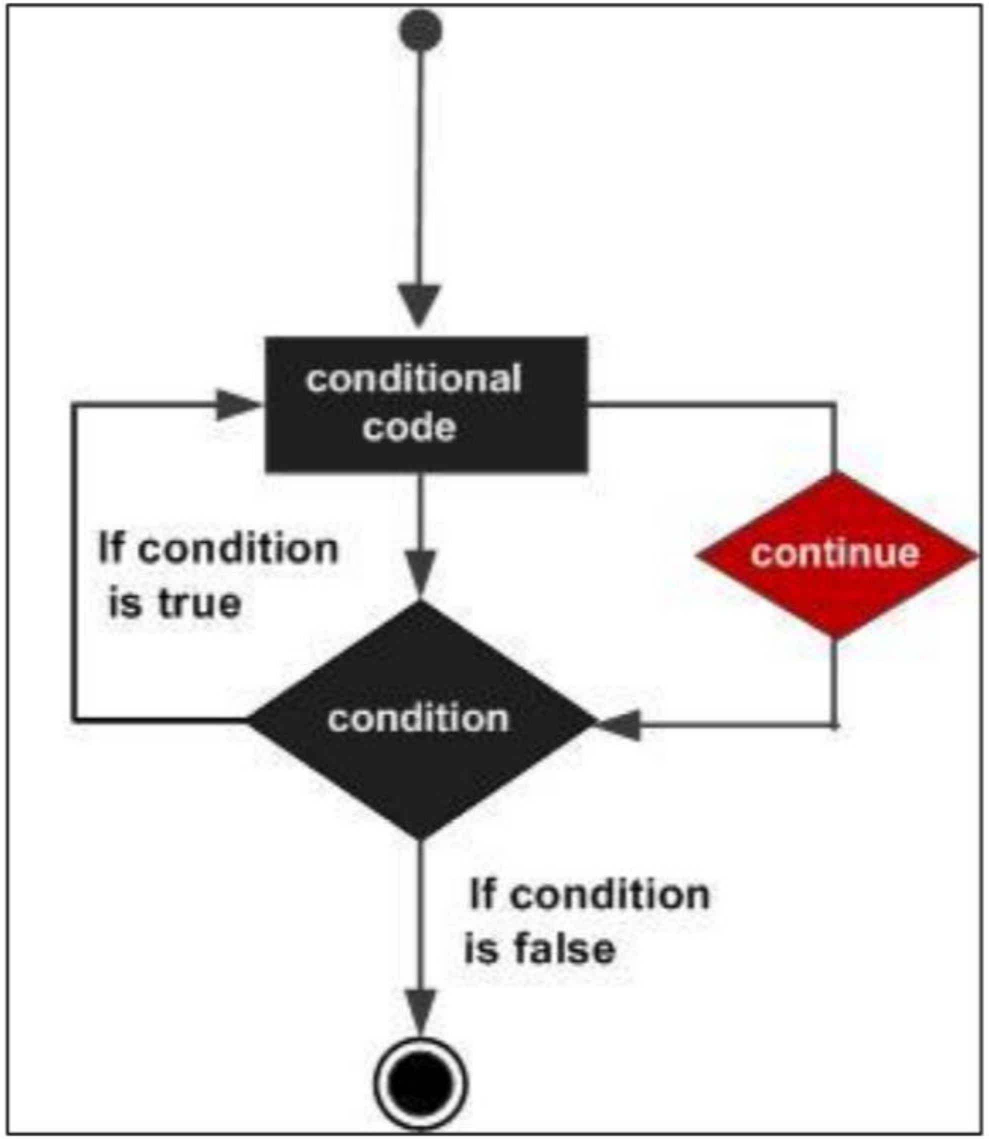


Break statement

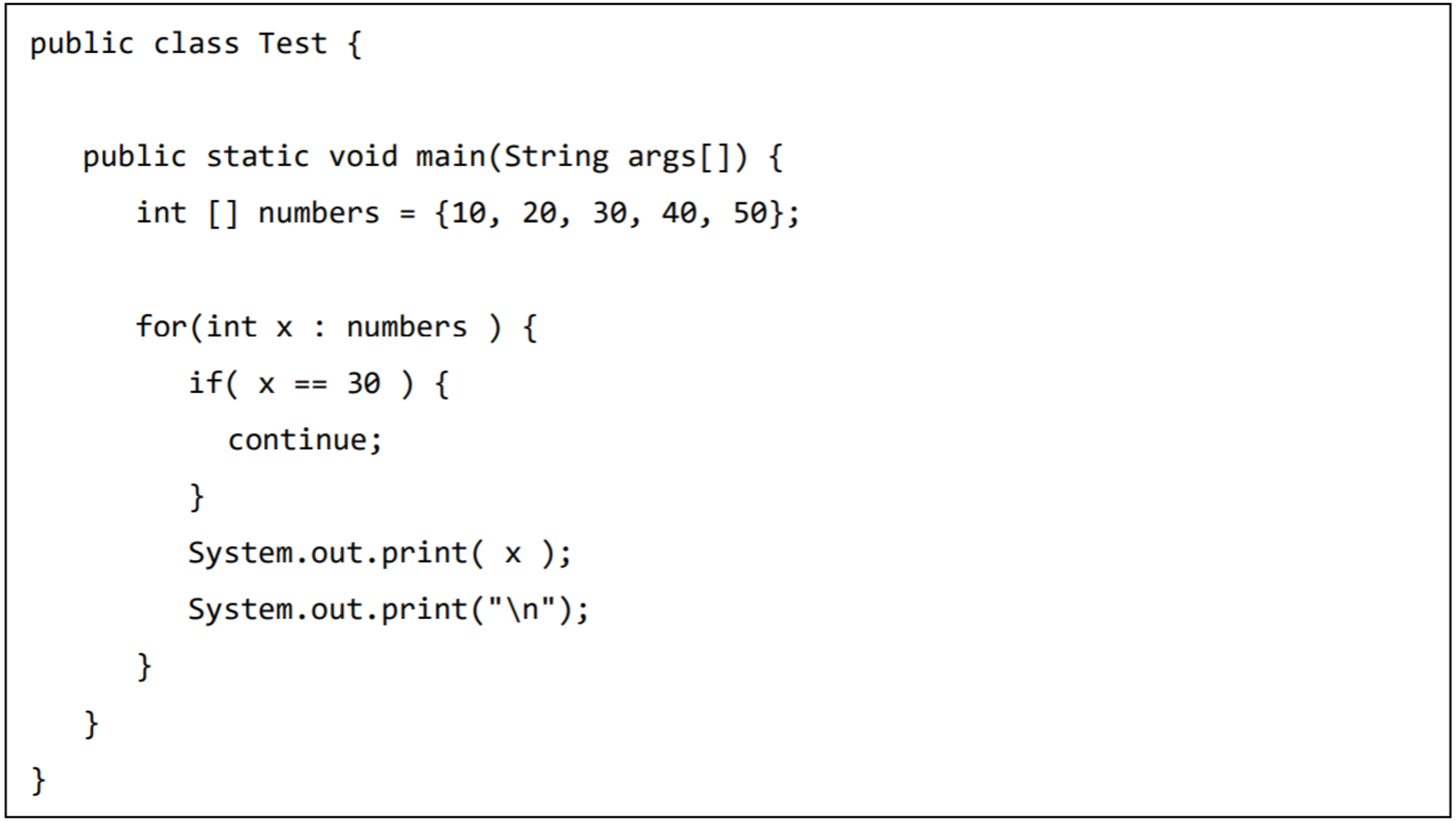


```
public class Test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
           break;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

Continue Statement



Continue statement



```
public class Test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
           continue;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

Creating a Method

```
public static int methodName(int a, int b) {
  // body
}
```

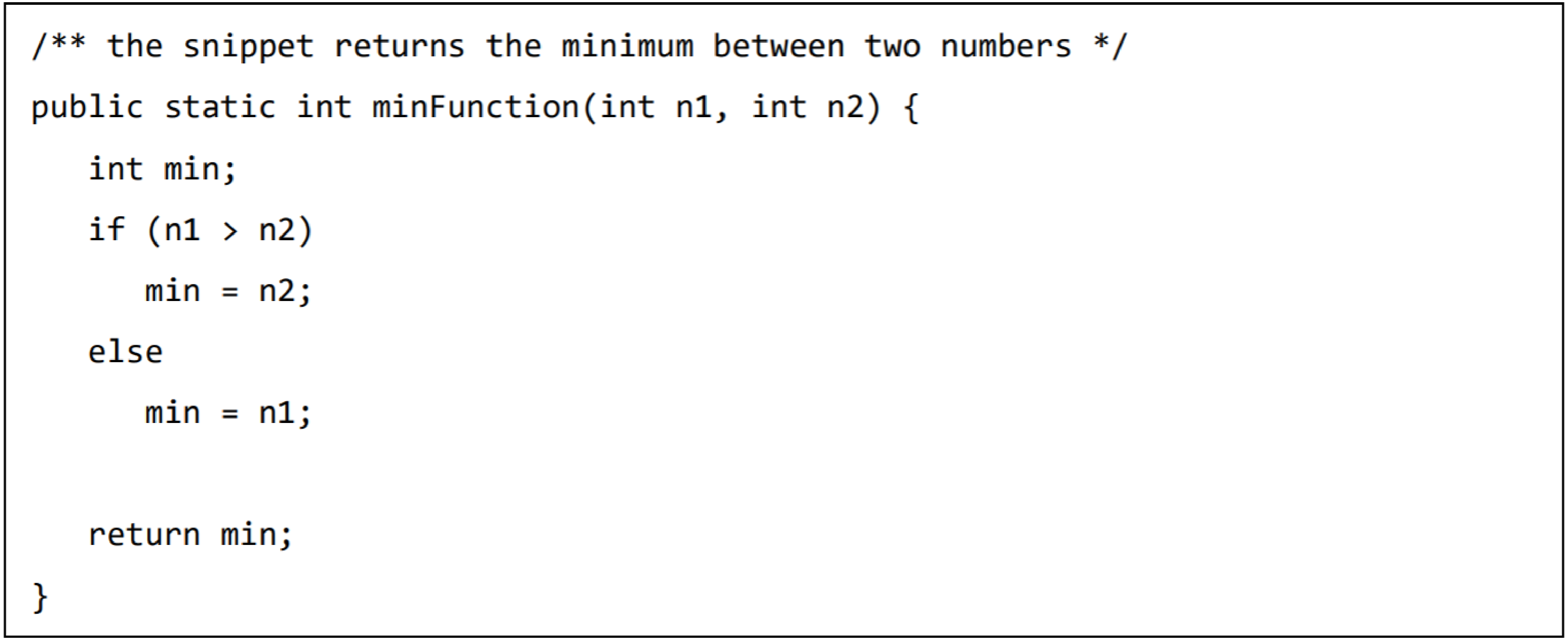
- **public static**: modifier
- **int**: return type
- **methodName**: name of the method
- **a, b**: formal parameters
- **int a, int b**: list of parameters

Methods

Creating a Method

- **Modifier**: it defines the access type of the method and it is optional to use.
- **Return type**: method may return a value.
- **nameOfMethod**: This is the method name. the method signature of the method name and parameter list.
- **Parameter list**: the list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- **Method body**: the method body defines what the method does with the statement.

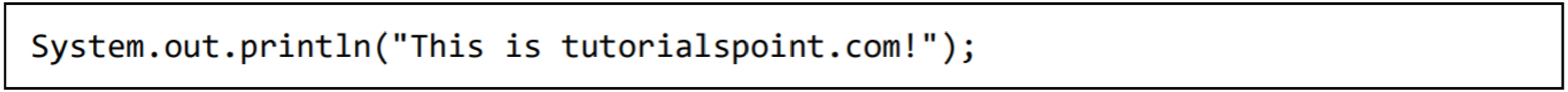
Creating a Method



```
/** the snippet returns the minimum between two numbers */
public static int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

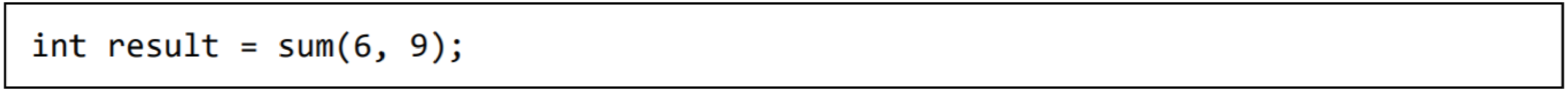
Method calling

- When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when:
    - the return statement is executed.
    - it reaches the method ending closing brace.
- The methods returning void is considered as call to a statement:



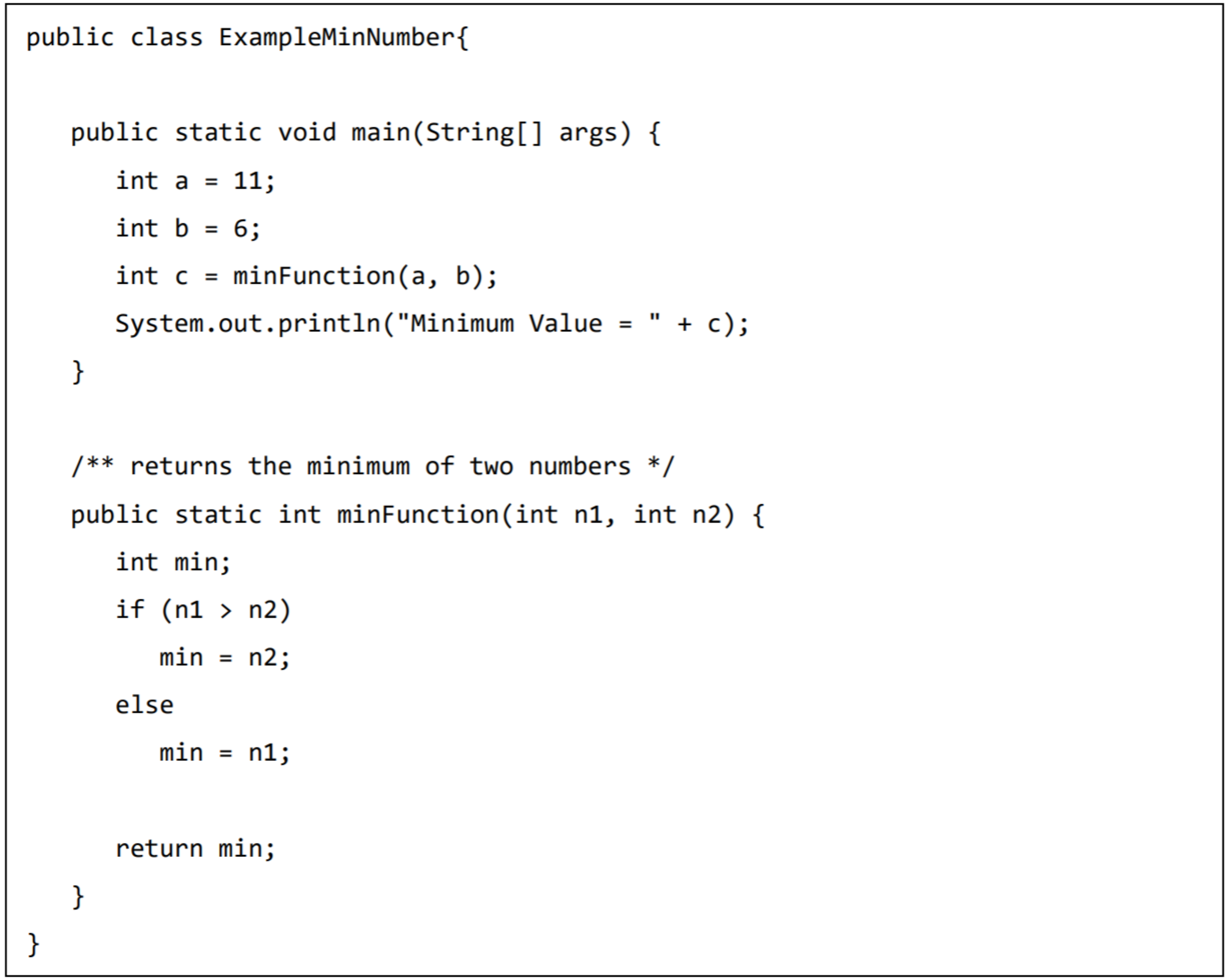
```
System.out.println("This is tutorialspoint.com!");
```

- The method returning value can be understood by the following example:



```
int result = sum(6, 9);
```

Defining and calling a method



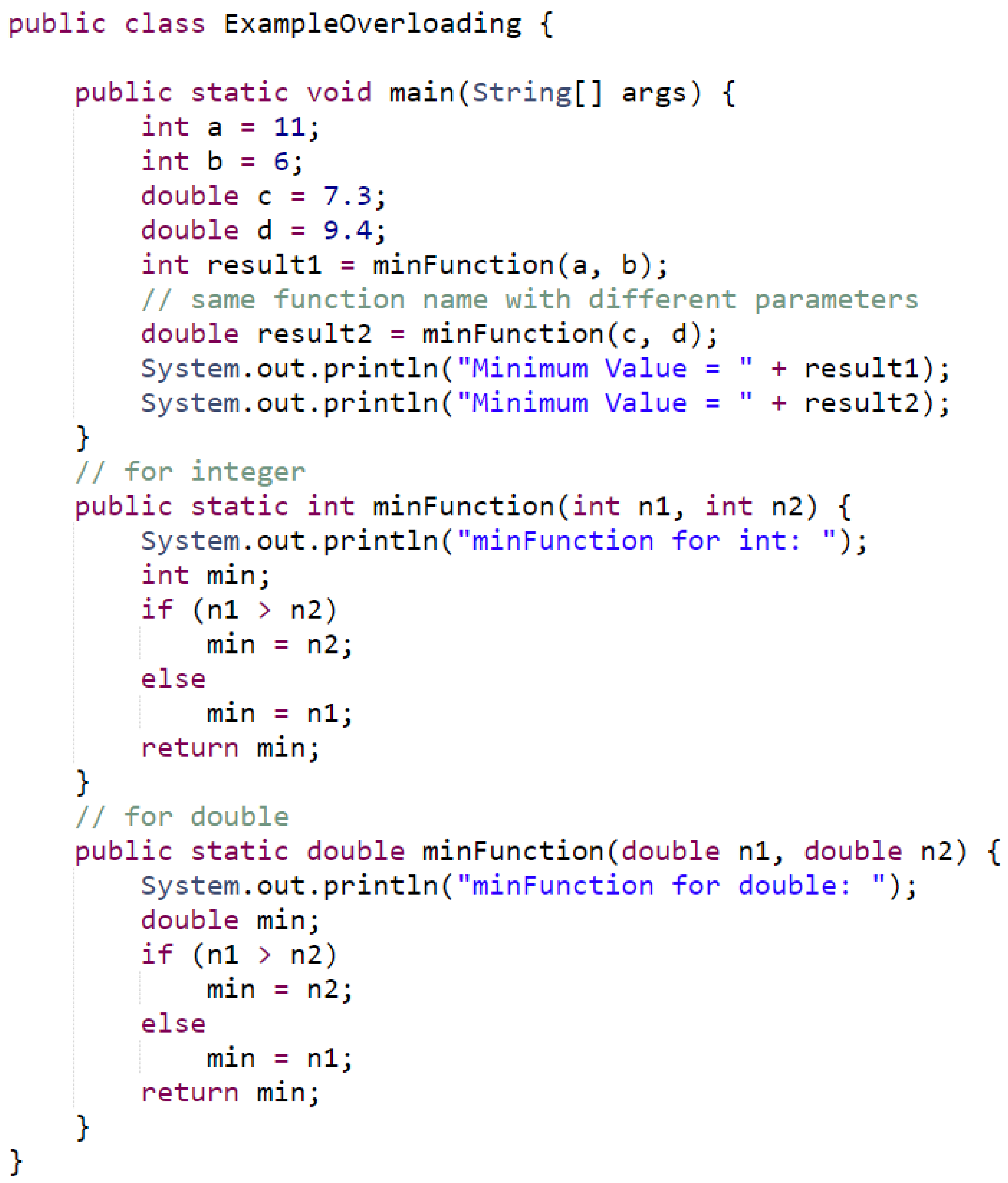
```
public class ExampleMinNumber{

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      int c = minFunction(a, b);
      System.out.println("Minimum Value = " + c);
   }

   /** returns the minimum of two numbers */
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;

      return min;
   }
}
```

Static Methods

- The static keyword is used to create methods that will exist independently of any instances created for the class.
- Static methods do not use any instance variables of any object of the class they are defined in. Static methods take all the data from parameters and compute something from those parameters, with no reference to variables.
- Class variables and methods can be accessed using the class name followed by a dot and the name of the variable or method.

Method Overloading

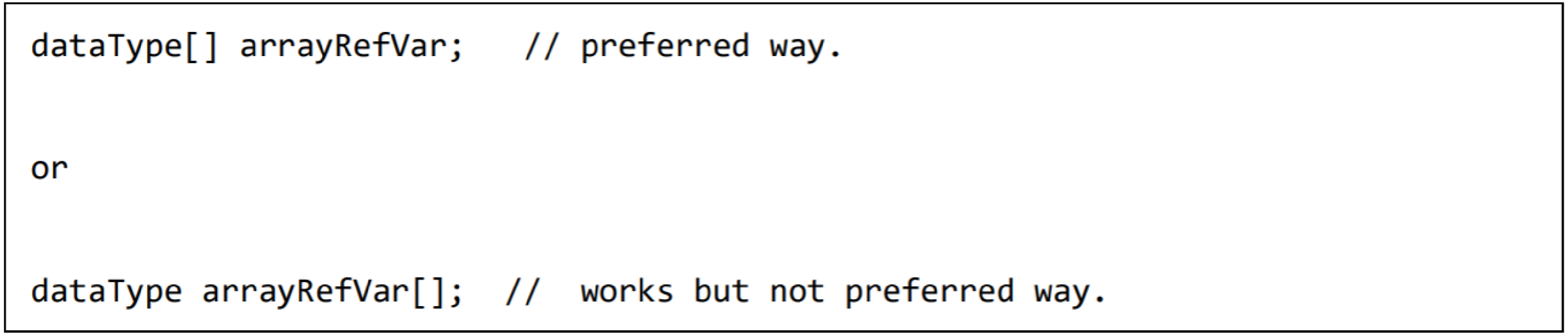
- When a class has two or more methods by the same name but different parameters, it is known as method overloading. It is different from overriding. In overriding, a method has the same method name, type, number of parameters, etc.
- Overloading methods makes program readable. Here, two methods are given by the same name but with different parameters. The minimum number from integer and double types is the result.



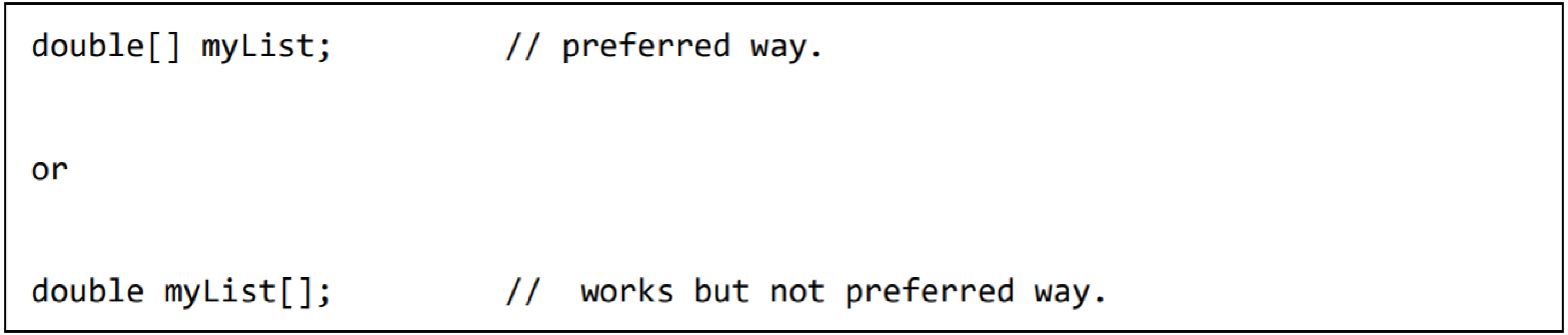
```
public class ExampleOverloading {

    public static void main(String[] args) {
        int a = 11;
        int b = 6;
        double c = 7.3;
        double d = 9.4;
        int result1 = minFunction(a, b);
        // same function name with different parameters
        double result2 = minFunction(c, d);
        System.out.println("Minimum Value = " + result1);
        System.out.println("Minimum Value = " + result2);
    }
    // for integer
    public static int minFunction(int n1, int n2) {
        System.out.println("minFunction for int: ");
        int min;
        if (n1 > n2)
            min = n2;
        else
            min = n1;
        return min;
    }
    // for double
    public static double minFunction(double n1, double n2) {
        System.out.println("minFunction for double: ");
        double min;
        if (n1 > n2)
            min = n2;
        else
            min = n1;
        return min;
    }
}
```

Arrays

Declaring Array Variables



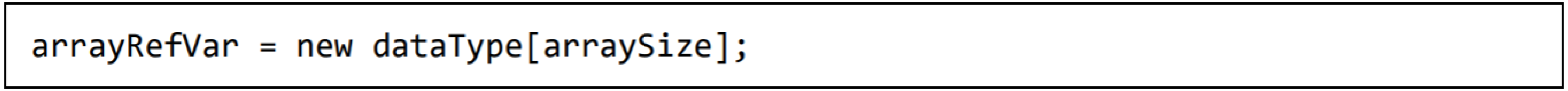
```
dataType[] arrayRefVar;   // preferred way.

or

dataType arrayRefVar[];  //  works but not preferred way.
```



```
double[] myList;         // preferred way.

or

double myList[];         //  works but not preferred way.
```

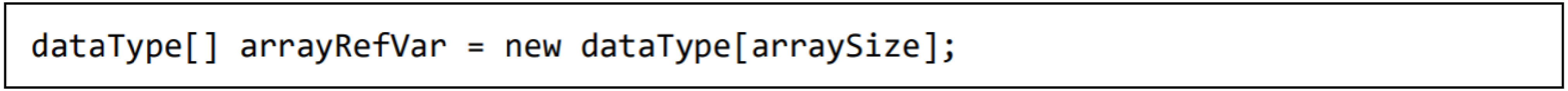
Creating arrays

- You can create an array by using the new operator with the following syntax:



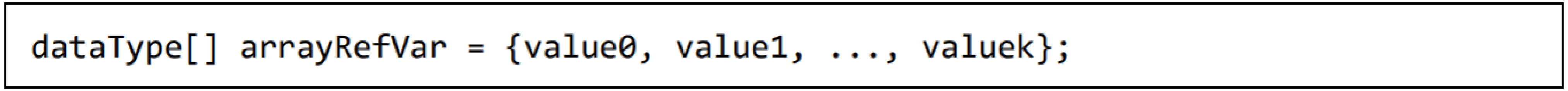
```
arrayRefVar = new dataType[arraySize];
```

- Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:



```
dataType[] arrayRefVar = new dataType[arraySize];
```

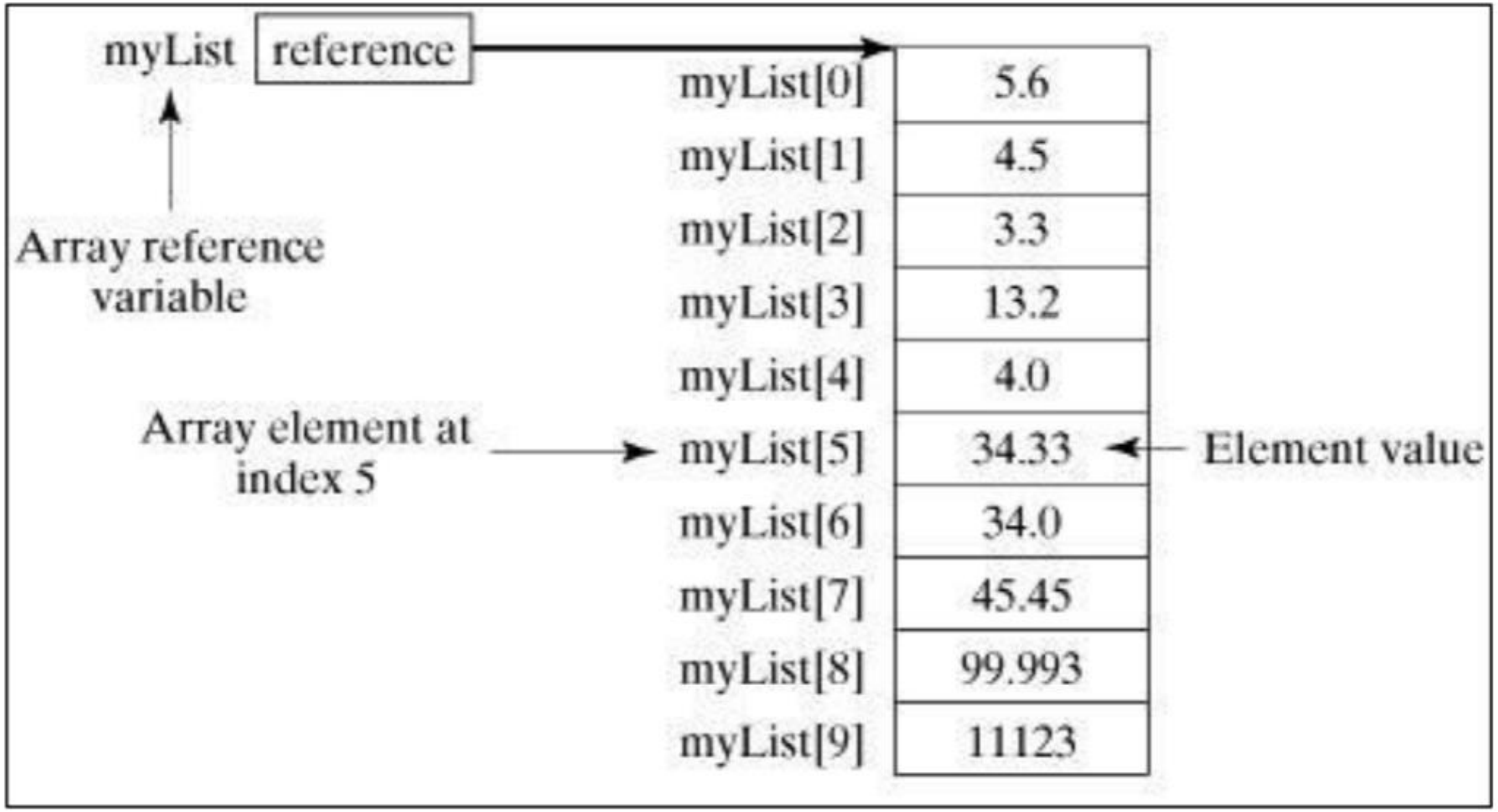
- Alternatively you can create arrays as follows:



```
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

- Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:

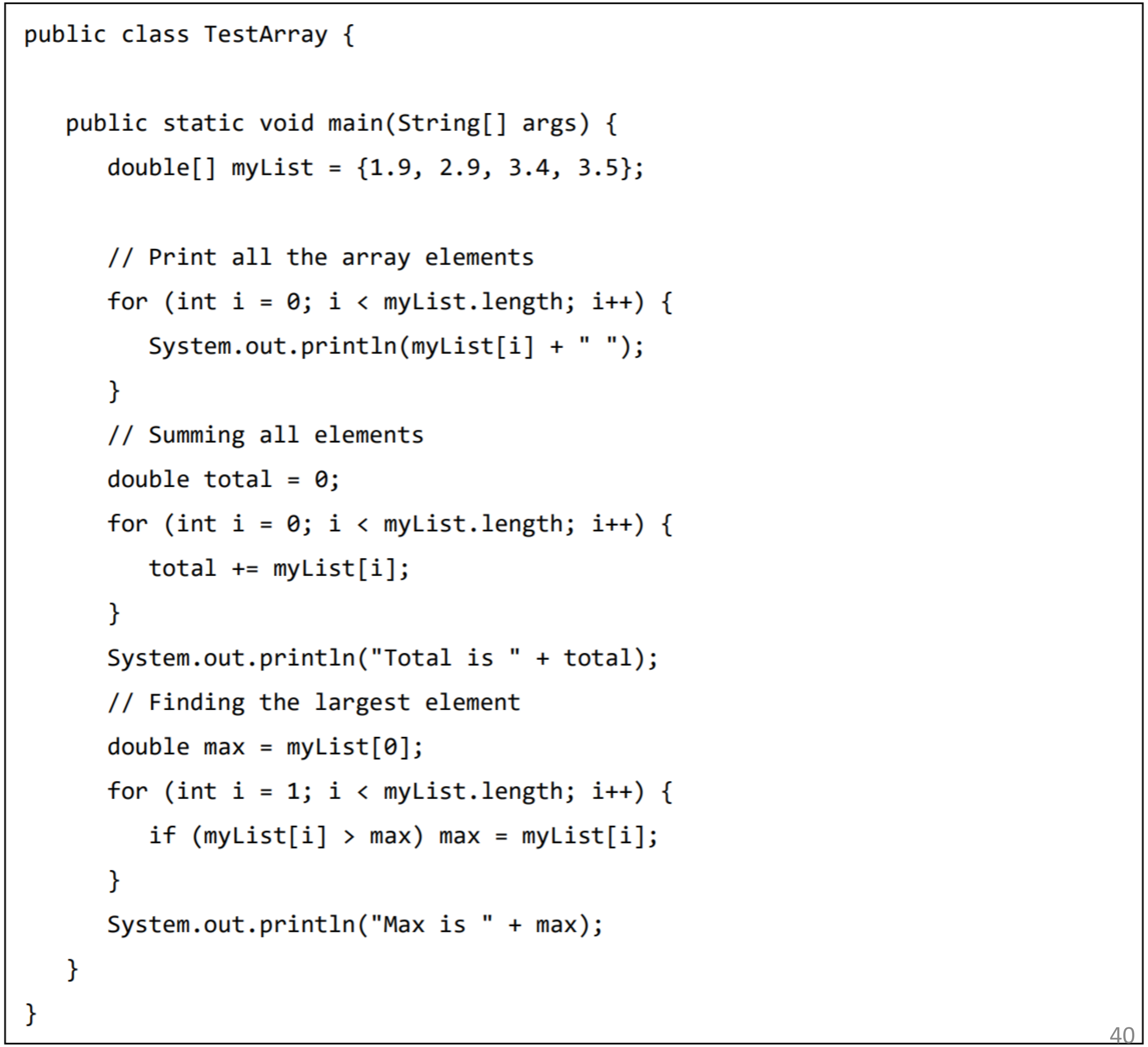
```
double[] myList = new double[10];
```



Processing Arrays

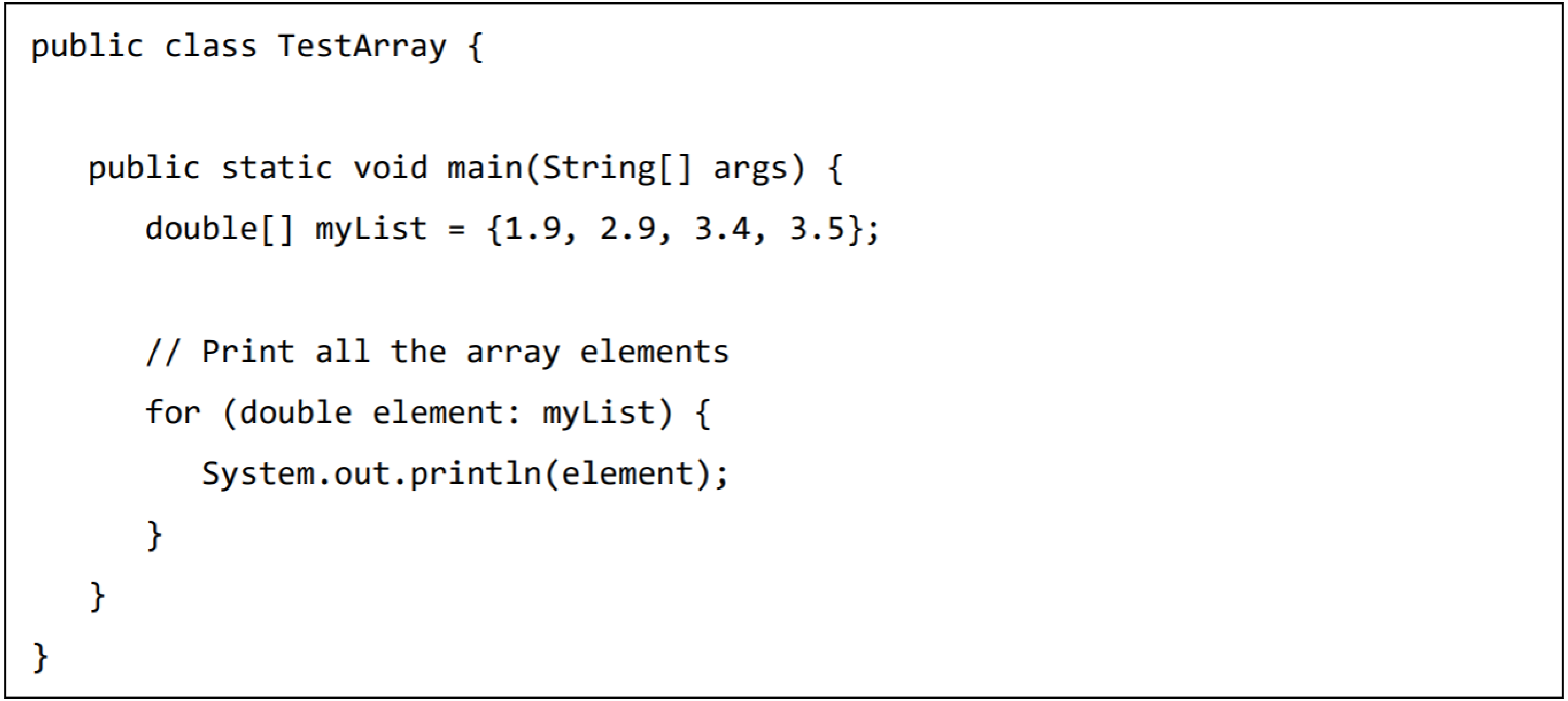
- When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

For loop



```
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (int i = 0; i < myList.length; i++) {
         System.out.println(myList[i] + " ");
      }
      // Summing all elements
      double total = 0;
      for (int i = 0; i < myList.length; i++) {
         total += myList[i];
      }
      System.out.println("Total is " + total);
      // Finding the largest element
      double max = myList[0];
      for (int i = 1; i < myList.length; i++) {
         if (myList[i] > max) max = myList[i];
      }
      System.out.println("Max is " + max);
   }
}
```

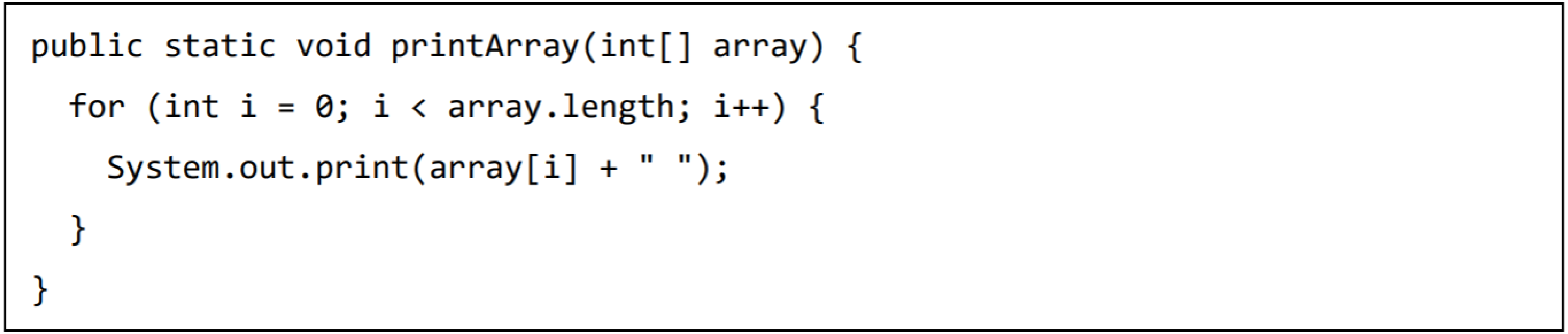
Foreach loop



```
public class TestArray {

   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};

      // Print all the array elements
      for (double element: myList) {
         System.out.println(element);
      }
   }
}
```

Passing Arrays to Methods

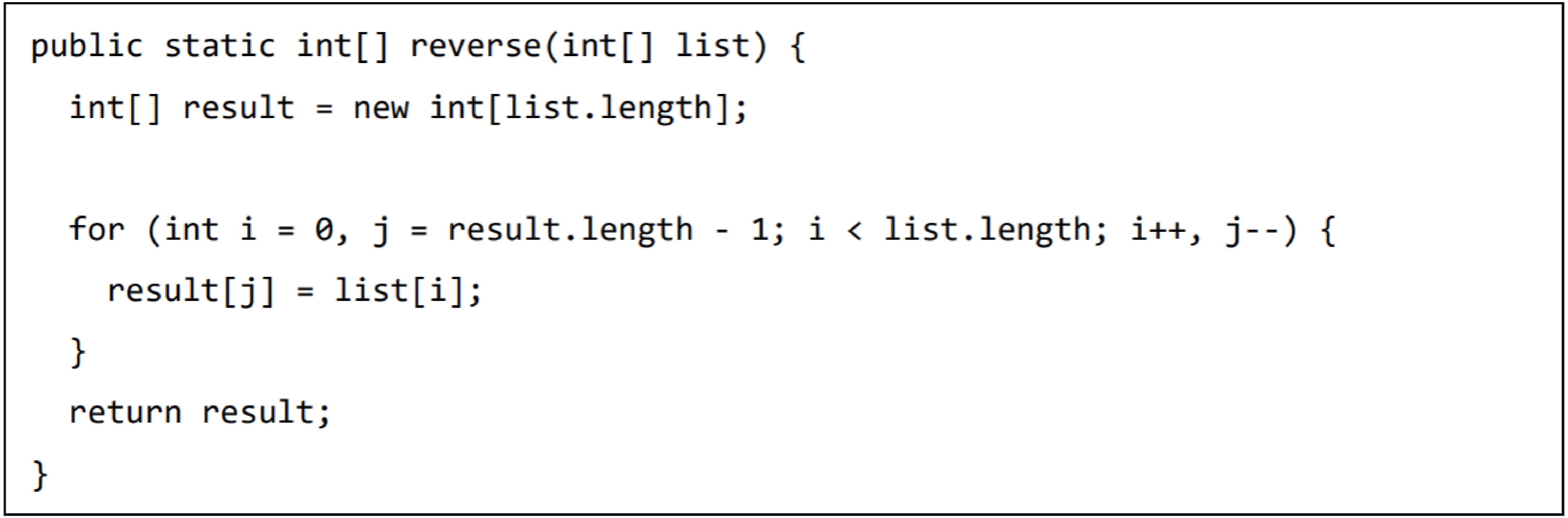
- Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an int array:



```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

Returning an array from a Method

- A method may also return an array. For example, the following method returns an array that is the reversal of another array:



```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
    result[j] = list[i];
  }
  return result;
}
```