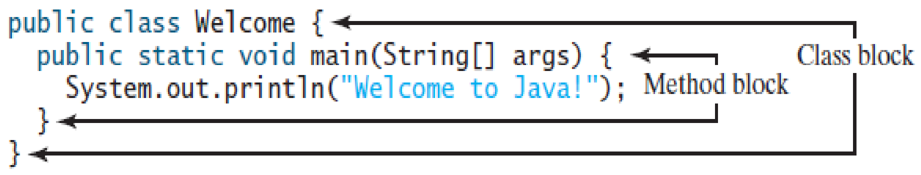
Simple Java Program

A Java program is executed from the main method in the class.

Let's begin with a simple Java program that displays the message Welcome to Java! on the console. (The word *console* is an old computer term that refers to the text entry and display device of a computer. *Console input* means to receive input from the keyboard, and *console output* means to display output on the monitor.) The program is shown in Listing 1.1.



```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

LISTING 1.1 Welcome.java

```java
public class Welcome {
    public static void main(String[ ] args) {
        // Display message Welcome to Java! on the console
    System.out.println("Welcome to Java!");
     }
}
```

```
Welcome to Java!
```

- **Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter.**
- **The program is executed from the main method. A class may contain several methods. The main method is the entry point where the program begins execution.**
- **Every statement in Java ends with a semicolon (;), known as the *statement terminator.***
- **Comments help programmers to communicate and understand the program. They are not programming statements and thus are ignored by the compiler. In Java, comments are preceded by two:**

- **slashes (//) on a line, called a *line comment,***
- **enclosed between /* and */ on one or several lines, called a *block comment* or *paragraph comment.***

```
// This application program displays Welcome to Java!

/* This application program
   displays Welcome to Java! */
```

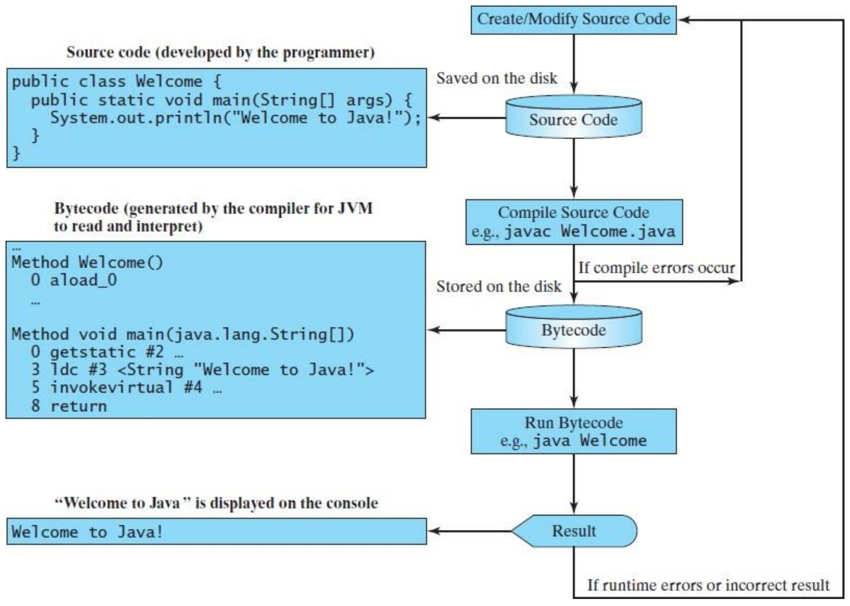
1. **A pair of curly braces ({ })in a program forms a *block* that groups the program's components. In Java, each block begins with an opening brace ({) and ends with a closing brace (}). Every class has a *class block* that groups the data and methods of the class. Similarly, every method has a *method block* that groups the statements in the method. Blocks can be *nested*, meaning that one block can be placed within another, as shown in the following code.**

Creating, Compiling, and Executing a Java Program

You save a Java program in a .java file and compile it into a .class file.

The Java Virtual Machine executes the class file.

As shown in Figure bellow.


Create/Modify Source Code
Source code (developed by the programmer)
Saved on the disk
Source Code
Bytecode (generated by the compiler for JVM to read and interpret)
Compile Source Code e.g., javac Welcome.java
If compile errors occur
Stored on the disk
Bytecode
Run Bytecode e.g., java Welcome
"Welcome to Java" is displayed on the console
Result
If runtime errors or incorrect result

```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

```
…
Method Welcome()
  0 aload_0
  …

Method void main(java.lang.String[])
  0 getstatic #2 …
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 …
  8 return
```

```
Welcome to Java!
```

FIGURE 1.6 The Java program-development process consists of repeatedly creating/modifying source code, compiling, and executing programs

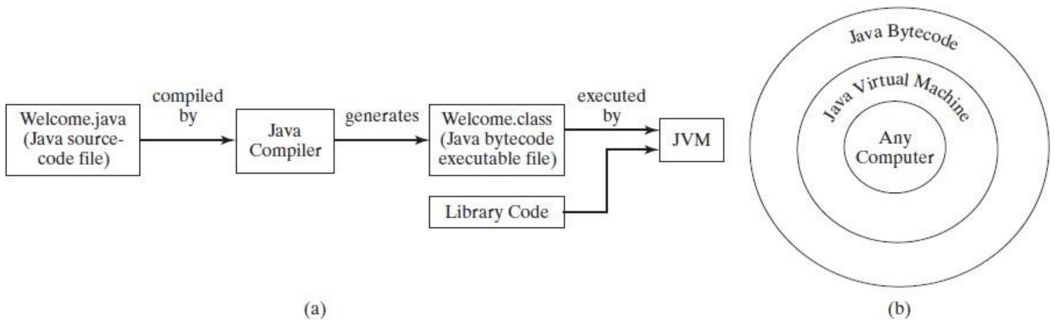
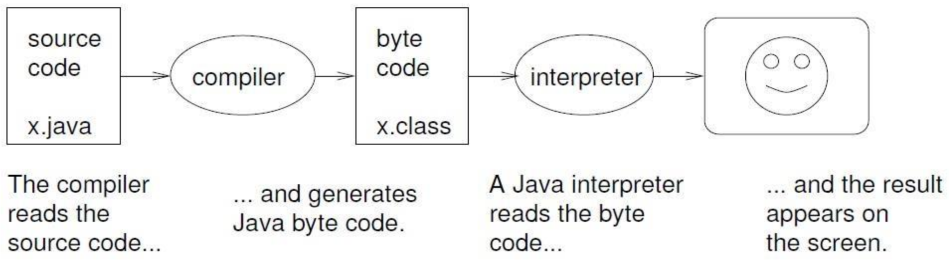


FIGURE 1.8 (a) Java source code is translated into bytecode. (b) Java bytecode can be executed on any computer with a Java Virtual Machine.

Java is both compiled and interpreted.

Instead of translating programs into machine language, the Java compiler generates byte code. Byte code is easy (and fast) to interpret, like machine language, but it is also portable, like a high-level language. Thus, it is possible to compile a program on one machine, transfer the byte code to another machine, and then interpret the byte code on the other machine. This ability is an advantage of Java over many other high-level languages.


source code
x.java
compiler
byte code
x.class
interpreter
The compiler reads the source code...
... and generates Java byte code.
A Java interpreter reads the byte code...
... and the result appears on the screen.

Programming Errors

Programming errors can be categorized into three types: syntax errors, runtime errors, and logic errors.

1. Syntax Errors

***Syntax Errors*: Errors that are detected by the compiler are called syntax errors or compile errors. Syntax errors result from errors in code construction, such as mistyping a keyword, or using an opening brace without a corresponding closing brace.**

For example,

LISTING 1.4 ShowSyntaxErrors.java

```
1 public class ShowSyntaxErrors {
2   public static main(String[ ] args) {
3    System.out.println("Welcome to Java);
4   }
5 }
```

Four errors are reported, but the program actually has two errors:

- **The keyword void is missing before main in line 2.**
- **The string Welcome to Java should be closed with a closing quotation mark in line 3.**

2. Runtime Errors

Runtime errors **are errors that cause a program to terminate abnormally. They occur while a program is running if the environment detects an operation that is impossible to carry out.**

For example

- if the program expects to read in a number, but instead the user enters a string, this causes data-type errors to occur in the program.
- Another example of runtime errors is division by zero.

LISTING 1.5 ShowRuntimeErrors.java

```java
public class ShowRuntimeErrors {
  public static void main(String[ ] args) {
  System.out.println(1 / 0);
  }
}
```

FIGURE 1.11 The runtime error causes the program to terminate abnormally.

3. Logic Errors

Logic errors **occur when a program does not perform the way it was intended to. Errors of this kind occur for many different reasons. For example, suppose you wrote the program in Listing 1.6 to convert Celsius 35 degrees to a Fahrenheit degree:**

LISTING 1.6 ShowLogicErrors.java

```java
public class ShowLogicErrors {
 public static void main(String[] args) {
 System.out.println("Celsius 35 is Fahrenheit degree ");
 System.out.println((9 / 5) * 35 + 32);
 }
}
```

You will get Fahrenheit 67 degrees, which is wrong. It should be 95.0

To get the correct result, you need to use 9.0 / 5

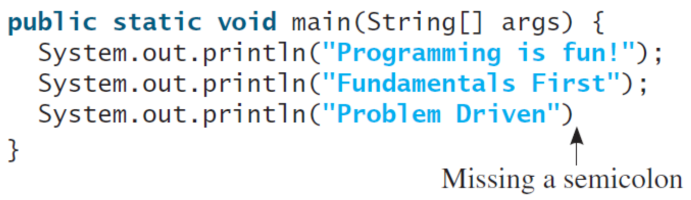
Common Errors

- **Common Error 1: Missing Braces**

```java
public class Welcome {

}
```

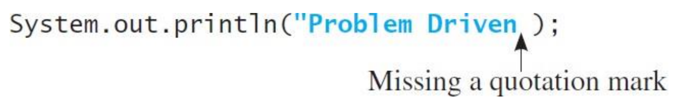
Type this closing brace right away to match the opening brace

- **Common Error 2: Missing Semicolons**



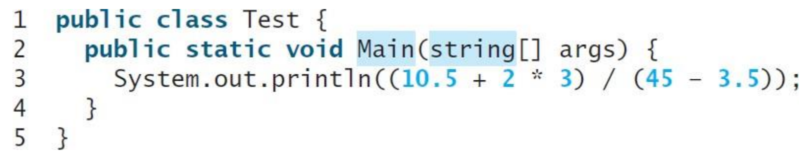
```
public static void main(String[] args) {
  System.out.println("Programming is fun!");
  System.out.println("Fundamentals First");
  System.out.println("Problem Driven")
}
```

- **Common Error 3: Missing Quotation Marks**



```
System.out.println("Problem Driven );
```

- **Common Error 4: Misspelling Names**



```
public class Test {
  public static void Main(string[] args) {
    System.out.println((10.5 + 2 * 3) / (45 - 3.5));
  }
}
```

Display output in the Console

Java uses System.out to refer to the standard output device. By default, the output device is the display monitor.

To perform console output, you simply use the print ( ) or println( ) methods to display a primitive value or a string to the console. For example:

System.out.print("Enter a number for radius: ");

System.out.print(**"Enter your Name: "**);

System.out.println(4+ **1**);

int sum, num1, num2;
sum = num1 + num2;
System.out.println(**"the sum of num1 and num2 is: " + Sum**);

Reading Input from the Console

Reading input from the console enables the program to accept input from the user.

Java uses System.in to the standard input device. By default, the input device is the keyboard. . Console input can use the Scanner class to create an object to read input from System.in, as follows:

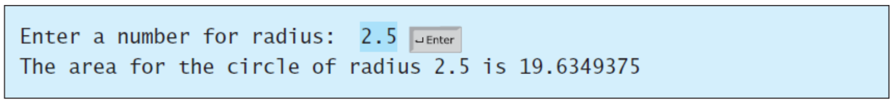
`Scanner input = new Scanner ( System.in );` creates a **Scanner** object and assigns its reference to the variable **input**

`double radius = input.nextDouble();` reads a number from the keyboard and assigns the number to **radius**

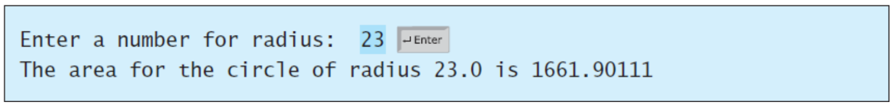
LISTING 2.2 ComputeAreaWithConsoleInput.java

```java
import  java.util.Scanner;
//import class
public class ComputeAreaWithConsoleInput {
 public static void main(String[ ] args) {
   // Create a Scanner object
        Scanner input = new Scanner(System.in);
      //create a Scanner
   // Prompt the user to enter a radius
      System.out.print("Enter a number for radius: ");
      double radius = input.nextDouble();
    //read a double value from user
   // Compute area
      double area = radius * radius * 3.14159;
// Display results
   System.out.println("The area for the circle of radius " + radius + " is " + area);
   }
 }
```

The area for the circle of radius 100 is 5



```
Enter a number for radius:  2.5 ↵Enter
The area for the circle of radius 2.5 is 19.6349375
```



```
Enter a number for radius:  23 ↵Enter
The area for the circle of radius 23.0 is 1661.90111
```

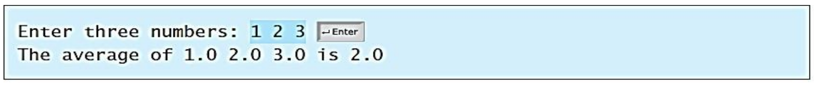
Note: The Scanner class is in the java.util package.

The following statement imports from the package.

import java.util.Scanner;
import java.util.*;

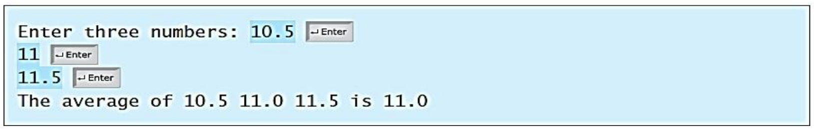
LISTING 2.3 ComputeAverage.java

```
import java.util.Scanner;
// Scanner is in the java.util package
public class ComputeAverage {
 public static void main(String[] args) {
  // Create a Scanner object
    Scanner input = new Scanner(System.in);
// Prompt the user to enter three numbers
    System.out.print("Enter three numbers: ");
    double number1 = input.nextDouble();
    double number2 = input.nextDouble();
double number3 = input.nextDouble();
//Compute average
    double average = (number1 + number2 + number3) / 3;
// Display results
System.out.println("The average of " + number1 + " " + number2
                                       + " " + number3 + " is " + average);
  }
}
```



```
Enter three numbers: 1 2 3 ↵Enter
The average of 1.0 2.0 3.0 is 2.0
```

enter input in one line



```
Enter three numbers: 10.5 ↵Enter
11 ↵Enter
11.5 ↵Enter
The average of 10.5 11.0 11.5 is 11.0
```

enter input in multiple lines

Variables

Variables are used to store values to be used later in a program.

They are called variables because their values can be changed. (The value of a variable may change during the execution of a program)

The *variable declaration* tells the compiler to allocate appropriate memory space for the variable based on its data type.

The syntax for declaring a variable is

    datatype variableName;

Here are some examples of variable declarations:

```
int count ;                    // Declare count to be an integer variable
float radius;                  // Declare radius to be a double variable
double interestRate; // Declare interestRate to be a double variable
```

Variables often have initial values. You can declare a variable and initialize it in one step. Consider, for instance, the following code:

```
int count = 1;
```

This is equivalent to the next two statements:

```
int count;
count = 1;
```

You can also use a shorthand form to declare and initialize variables of the same type together. For example,

```
int i = 1, j = 2, x= 5;
```

Assignment operator

The syntax for assignment statements is as follows:

```
Variable =  expression;
```

For example, consider the following code:

```
int y = 1;                     // Assign 1 to variable y
double radius = 1.0; // Assign 1.0 to variable radius
int x = 5 * (3 / 2);      // Assign the value of the expression to x
x = y + 1;                     // Assign the addition of y and 1 to x
double area = radius * radius * 3.14159;         // Compute area
```

for multiple variables assigned, you can use this syntax:

```
i = j = k = 1;
```

which is equivalent to

```
k = 1;
j = k;
i = j;
```

Named Constants

A named constant is an identifier that represents a permanent value.

Constant is a variable data that never changes during the execution of a program

The syntax for declaring a constant:

```
final datatype CONSTANTNAME = value;
```

The word final is a Java keyword for declaring a constant.

For example,

```
final double PI = 3.14159;          // Declare a constant
```

error: PI = PI +5;

Numeric Data Types and Operations

Java has six numeric types for integers and floating-point numbers with operators

{ + , - , * , / , and %}.

TABLE 2.1 Numeric Data Types

Name	*Range*	*Storage Size*	
`byte`	-2^7 to $2^7 - 1$ (−128 to 127)	8-bit signed	`byte` type
`short`	-2^{15} to $2^{15} - 1$ (−32768 to 32767)	16-bit signed	`short` type
`int`	-2^{31} to $2^{31} - 1$ (−2147483648 to 2147483647)	32-bit signed	`int` type
`long`	-2^{63} to $2^{63} - 1$ (i.e., −9223372036854775808 to 9223372036854775807)	64-bit signed	`long` type
`float`	Negative range: −3.4028235E + 38 to −1.4E − 45 Positive range: 1.4E − 45 to 3.4028235E + 38	32-bit IEEE 754	`float` type
`double`	Negative range: −1.7976931348623157E + 308 to −4.9E − 324 Positive range: 4.9E − 324 to 1.7976931348623157E + 308	64-bit IEEE 754	`double` type

Reading Numbers from the Keyboard

You know how to use the nextDouble() method in the Scanner class to read a double value from the keyboard.

TABLE 2.2 Methods for `Scanner` Objects

Method	*Description*
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

Here are examples for reading values of various types from the keyboard:

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a byte value: ");
byte byteValue = input.nextByte();

System.out.print("Enter a short value: ");
short shortValue = input.nextShort();

System.out.print("Enter an int value: ");
int intValue = input.nextInt();

System.out.print("Enter a long value: ");
long longValue = input.nextLong();

System.out.print("Enter a float value: ");
float floatValue = input.nextFloat();
```

Numeric Operators

The operators for numeric data types include the standard arithmetic operators: addition (+), subtraction (–), multiplication (*), division (/), and remainder (%), as shown in Table 2.3. The *operands* are the values operated by an operator.

TABLE 2.3 Numeric Operators

Name	*Meaning*	*Example*	*Result*
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Exponent Operations

The Math.pow(a, b) method can be used to compute a^b. The pow method is defined in the Math class in the Java API.

For example,

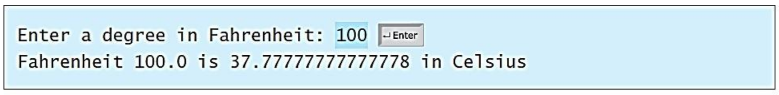
```
System.out.println( Math.pow(2, 3));       // Displays 8.0
System.out.println( Math.pow(4, 0.5));    // Displays 2.0


System.out.println( Math.pow(2.5, 2));    // Displays 6.25
System.out.println( Math.pow(2.5, -2));  // Displays 0.16
```

Listing 2.6 gives a program that converts a Fahrenheit degree to Celsius using the formula

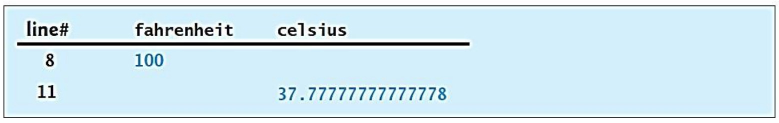
celsius = (59) (fahrenheit - 32).

LISTING 2.6 FahrenheitToCelsius.java

```
1 import java.util.Scanner;
2
3 public class FahrenheitToCelsius {
4   public static void main(String[] args) {
5     Scanner input = new Scanner(System.in);
6
7      System.out.print("Enter a degree in Fahrenheit: ");
8      double fahrenheit = input.nextDouble();
9
10     // Convert Fahrenheit to Celsius
11     double celsius = (5.0 / 9) * (fahrenheit - 32);
12     System.out.println("Fahrenheit " + fahrenheit + " is " +celsius + " in Celsius");
13     }
14  }
```



```
Enter a degree in Fahrenheit: 100 ↵Enter
Fahrenheit 100.0 is 37.77777777777778 in Celsius
```



line#	fahrenheit	celsius
8	100	
11		37.77777777777778

Augmented Assignment Operators

The operators +, -, *, /, and % can be combined with the assignment operator to form augmented operators.

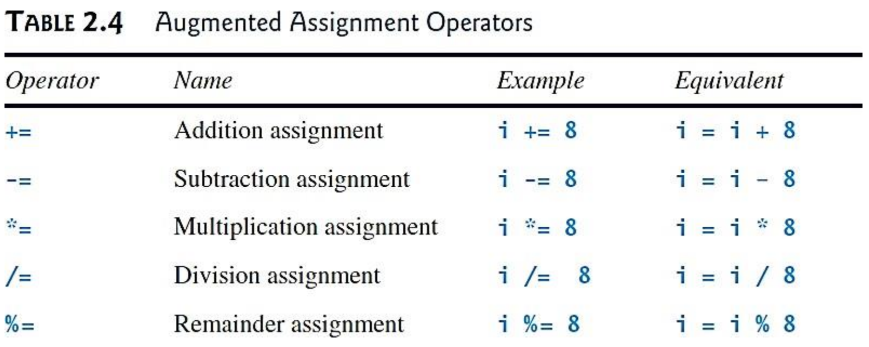


TABLE 2.4 Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

Increment and Decrement Operators

The increment operator (++) and decrement operator (– –) are for incrementing and decrementing a variable by 1.

- **postincrement**

```
int i = 3, j = 3;
i++; // i becomes 4
j--; // j becomes 2
```

- **preincrement**

```
int i = 3, j = 3;
++i; // i becomes 4
--j; // j becomes 2
```

TABLE 2.5 Increment and Decrement Operators

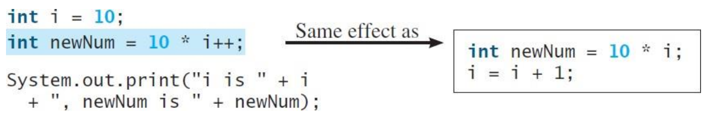
Operator	*Name*	*Description*	*Example (assume i = 1)*

`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;`
			`// j is 2, i is 2`

`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;`
			`// j is 1, i is 2`

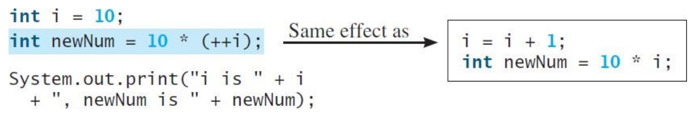
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;`
			`// j is 0, i is 0`

`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;`
			`// j is 1, i is 0`



```
int i = 10;
int newNum = 10 * i++;

System.out.print("i is " + i
  + ", newNum is " + newNum);
```

```
int newNum = 10 * i;
i = i + 1;
```



```
int i = 10;
int newNum = 10 * (++i);

System.out.print("i is " + i
  + ", newNum is " + newNum);
```

```
i = i + 1;
int newNum = 10 * i;
```