Chapter 1 Introduction to Java

1. Introduction

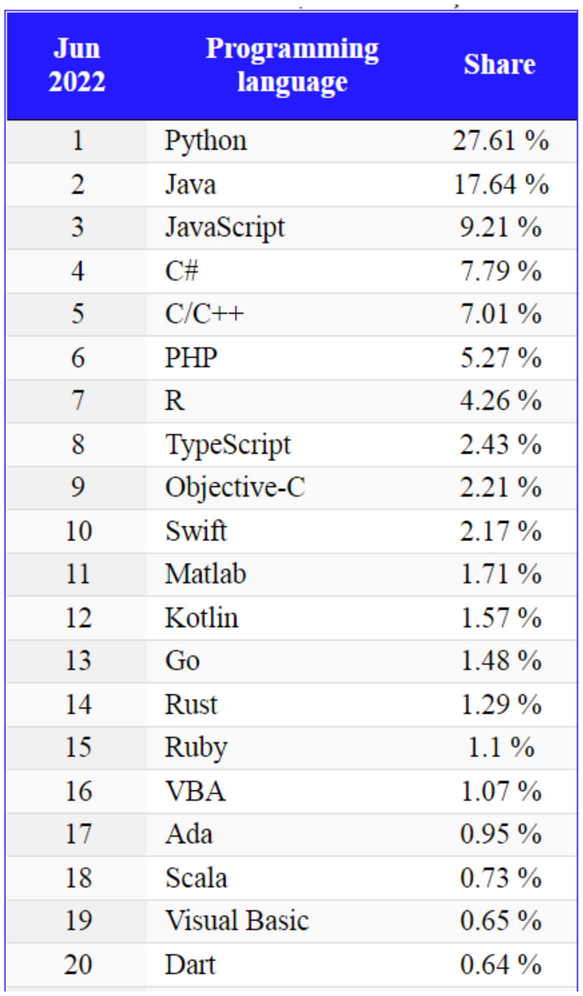
James Gosling, Mike Sheridan, and Patrick Naughton, a team of Sun engineers known as the *Green team* initiated the Java language in 1991. Java was introduced to the public in 1995.

Java is a popular programming language. Java is used to develop mobile applications, web applications, desktop applications, games and much more. It is owned by Oracle.

After the name OAK, the team decided to give a new name to it and the suggested words were Silk, Jolt, revolutionary, DNA, dynamic, Java, etc. Java is the name of an island in Indonesia where the first coffee (named java coffee) was produced. And this name was chosen by James Gosling while having coffee near his office.

Java is an object-oriented programming language. As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa.

Top Programming Languages (Jun 2022)



Jun 2022	Programming language	Share
1	Python	27.61 %
2	Java	17.64 %
3	JavaScript	9.21 %
4	C#	7.79 %
5	C/C++	7.01 %
6	PHP	5.27 %
7	R	4.26 %
8	TypeScript	2.43 %
9	Objective-C	2.21 %
10	Swift	2.17 %
11	Matlab	1.71 %
12	Kotlin	1.57 %
13	Go	1.48 %
14	Rust	1.29 %
15	Ruby	1.1 %
16	VBA	1.07 %
17	Ada	0.95 %
18	Scala	0.73 %
19	Visual Basic	0.65 %
20	Dart	0.64 %

https://statisticstimes.com/tech/top-computer-languages.php

2. Basics of Java

2.1 A Simple Java Program

Let's begin with a simple Java program that displays the message Welcome to Java! on the console.

Example 01: A simple Java program that displays a message.
Welcome.java

```
1 class Welcome {
2     public static void main(String[] args) {
3         // Display message Welcome to Java! on the console
4         System.out.println("Welcome to Java!");
5     }
6 }
```

Output:

```
Welcome to Java!
```

Line 1 defines a class. Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter. In this example, the class name is `Welcome`.

Line 2 defines the `main`() method. Every Java program is executed from the `main`() method. The `main`() method in this program contains the `System.out.println` statement. This statement displays the string `Welcome to Java!` on the console (line 4). A string must be enclosed in double quotation marks. Every statement in Java ends with a semicolon (;), known as the **statement terminator**.

Line 3 is a comment that documents what the program is and how it is constructed. Comments help programmers to communicate and understand the program. They are not programming statements, and thus are ignored by the compiler.

File name and class name should be same in Java. Let's be clear on one point; this requirement is not mandatory unless until there is a public class in the file.

2.2 Comments

You can place comments in your source code that are not executed as part of the program. Comments provide clarity to the source code allowing others to better understand what the code was intended to accomplish. Comments are especially important in large projects containing hundreds or thousands of lines of source code or in projects in which many contributors are working on the source code.

Adding source code comments to your source code is a highly recommended practice.

In Java, there are two types of comments:

- **Single-line comment** or **Comment line** starts with two forward slashes (//)
- **Multi-line comment** or **Paragraph comment** starts with a slash asterisk /* and ends with an asterisk slash */.

2.3 main() Function

Every Java program is executed from the main() method. The main() method in Java must be declared public, static and void, and must have a String[] parameter. if any of these are missing; java program will compile but at run time error will be thrown.

- The class that contains the main() method must have the same name as the source file. For example, the source file name is Welcome.java and the class name is Test, the compiler will throw an error as "Could not find or load main class Welcome".
- The main() method is public as that is how JVM knows which class to load and where is the entry point for the execution.
- The main() method has a return type as void as it does not return any value. The main method takes an array of strings as parameter.
- Having main method as static, the JVM can access the main() method without creating an instance of the class.

* Java source code is translated into bytecode by Java compiler. Java bytecode can be executed on any computer with a Java Virtual Machine.

* JVM (Java Virtual Machine) is an engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language.

2.4 Java Package

2.4.1 Package

A package is simply a container that groups related types (Java classes, interfaces and so on). For example, in core Java, the ResultSet interface belongs to the java.sql package. The package contains all the related types that are needed for the SQL query and database connection. If you want to use the ResultSet interface in your code, just import the java.sql package.

Based on whether the package is defined by the user or not, packages are divided into two categories:

- **Built-in Package** are existing java packages that come along with the JDK. For example, java.lang, java.util, java.io, etc.
- Java also allows you to create packages as per your need, which are called **User-defined packages**.

2.4.2 Package Naming Convention

The package name must be unique (like a domain name). Hence, there's a convention to create a package as a domain name, but in reverse order. For example, com.company or com.company.name.

2.4.3 Importing Package

Java has an import statement that allows you to import an entire package, or use only certain classes and interfaces defined in the package.

The general form of import statement is:

```
import package_name.ClassName;    // To import a certain class only
import package_name.*;            // To import the whole package
```

For example,

```
import java.util.Date; // imports only Date class
import java.util.*;      // imports everything inside java.util package
```

The import statement is optional in Java. You can use its fully qualified name, which includes its full package hierarchy. Here is an example to import a package using the `import` statement.

```java
import java.util.Date;

class MyClass implements Date {
    // body
}
```

The same task can be done using the fully qualified name as follows:

```java
class MyClass implements java.util.Date {
    //body
}
```

Let's take a look at `System.out.println` statement in the Example 1 above again.

- The `System` is a java class which belongs to `java.lang` package.
- The `out` is a `public static` field of type `PrintStream` in the `System` class. (We will discuss class, `public`, and `static` in later chapters.)
- The `println`() is a method of the `PrintStream` class.

By default `java.lang` package will be available to us. That is why in the Example 01, we did not need the `import` statement:

```java
import java.lang.*;
```

or

```java
import java.lang.System;
```

3. Programming Style and Documentation

Programming style deals with what programs look like. A program can compile and run properly even if written on only one line, but writing it all on one line would be bad programming style because it would be hard to read. Documentation is the body of explanatory remarks and comments pertaining to a program. Programming style and documentation are as important as coding.

3.1 Appropriate Comments and Comment Styles

Include a summary at the beginning of the program that explains what the program does, its key features, and any unique techniques it uses. In a long program, you should also include comments that introduce each major step and explain anything that is difficult to read. It is important to make comments concise so that they do not crowd the program or make it difficult to read.

3.2 Proper Indentation and Spacing

A consistent indentation style makes programs clear and easy to read, debug, and maintain. Java can read the program even if all of the statements are on the same long line, but humans find it easier to read and maintain code that is aligned properly.

A single space should be added on both sides of a binary operator, as shown in below:

```
System.out.println(3 + 4 * 4);
```

(a) Good style

```
System.out.println(3+4*4);
```

(b) Bad style

3.3 Block Styles

A block is a group of statements surrounded by braces. There are two popular styles, next-line style and end-of-line style, as shown below.

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

Next-line style

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

End-of-line style

The next-line style aligns braces vertically and makes programs easy to read, whereas the end-of-line style saves space and may help avoid some subtle programming errors. Both are acceptable block styles. The choice depends on personal or organizational preference. You should use a block style consistently—mixing styles is not recommended.

4. `boolean` Datatype

Java uses `boolean` instead of bool. When a `boolean` is expected, it must in fact be a boolean expression (`true` or `false`). No conversions from any other types, including int.

Example 02: Using boolean variables.
Example.java

```
public class Example {
    public static void main(String[] args) {
        boolean b1 = true;
        boolean b2 = false;

        System.out.println(b1);
        System.out.println(b2);
    }
}
```

Output:

```
true
false
```

5. No Global Variables or Functions

Java does NOT support global variables. Every variable in Java is either local or a member of a class. So, you cannot declare variables outside of class.

Java does not have functions; it has only methods. In Java, everything (objects) belongs to some class. So, we cannot declare functions (methods) outside a class.

6. Constant

A constant variable is a variable that is declared to have a value in it that will be constant till the program ends. They are declared using a keyword known as "`final`".

The variables which are declared as "`final`" will only be initialized once and then their value will be constant and will not change during the execution of the program. For example:

```
class English {
    final int VOWELS = 5;
}
```

7. Naming Convention

There is a standard naming convention, which all the standard Java libraries follow, and which you MUST follow in our APL class.

- Names of classes are in MixedCase starting with a capital letter. If the most natural name for the class is a phrase, start each word with a capital letter, as in `StringBuffer`.
- Names of constants are ALL_UPPER_CASE. Separate words of phrases with underscores as in MIN_VALUE.
- Other names (methods, variables, etc.) are in lower case or mixedCase, starting with a lower-case letter. Example, insertBook().

8. Important Points to Keep in Mind

Here are some important points to keep in mind while working with Java Source file:

8.1 Number of Classes in A Java Source File

A Java file can contain any number of classes, and at most, one of them can be declared as a public class. In simple words, a java file can contain either zero public class, or if there is a public class present, it cannot be more than one.

8.2 Name of the Java Source File

The name of the Java source file can be anything provided that no class is declared as public.

Explanation: Java allows us to name the Java source file with anything if there is not a single class that is declared as public. But if there is a class that is declared as public, the name of the Java source file must be the same as the public class name. The Java source file extension must be .java.
Conventionally, each class definition should go in a separate file, and the name of the source file should be exactly the same (including case) as the name of the class, with ".java" appended. For example, the definition of Pair must go in file Pair.java.

8.3 Number of .class Files

The number of .class files generated is equal to the number of classes declared in the Java program.

Explanation: While compiling a program, the javac compiler generates as many .class files as there are classes declared in the Java source file.

The .class file contains Java byte code and is later executed by the JVM. The Java interpreter (JVM) knows where the class files are located and can load them as needed.

9. Working with `float` and `double`

Literals are data used for representing fixed values. They can be used directly in the code. For example,

```
int a = 1;
char b = 'F';
float c = 2.5f;
double d = 2.5;
boolean flag1 = false;
```

Here, `1`, `'F'`, `2.5f`, `2.5`, and `false` are literals.

Floating-point literals are used to initialize `float` and `double` type variables. A floating-point literal is of type `float` if it is suffixed with an ASCII letter `F` or `f`; otherwise, its type is `double` and it can optionally be suffixed with an ASCII letter `D` or `d`.

```
double d1 = 2.5;
double d2 = 2.5d;
double d3 = 2.5D;
```

In Java, we cannot assign a decimal value to a float variable. For example:

```
float n1 = 5.89;    // Error
float n2 = 10.56f;
```

In Java, we can assign a float value to a double variable, but not via versa. For example:

```
float n1 = 5.89d;    // Error
double n2 = 5.89f;
```

We can provide an integer value using a float keyword and the compiler will treat that as a floating number.

```
float n3 = 10f;     ->  10.0f
```

The compiler knows how to convert integer literals into float literals or double literals.

```
float n4 = 10;      ->  10.0f
double n5 = 10;      ->  10.0d
```

Example 03: Initialize `float` and `double` variables.

```
class Example {
    public static void main(String[] args) {
        float myFloat1 = 1f;
        float myFloat2 = 2.5F;
        float myFloat3 = 3;      // can be converted into 2.0f
        double myDouble1 = 3.5d;
        double myDouble2 = 4D;
        double myDouble3 = 5.5f;
        double myDouble4 = 6.5; // by default double
        double myDouble5 = 7;   // can be converted into 7.0d

        System.out.println(myFloat1);
        System.out.println(myFloat2);
        System.out.println(myFloat3);
        System.out.println(myDouble1);
        System.out.println(myDouble2);
        System.out.println(myDouble3);
        System.out.println(myDouble4);
        System.out.println(myDouble5);
    }
}
```

Output:

```
1.0
2.5
3.0
3.5
4.0
5.5
6.5
7.0
```

Example 04: Perform some calculations on `float` values.

```
class Example {
    public static void main(String[] args) {
        float n1 = 1.5f;
        float n2 = 2f;
        float n3 = n1 * n2 * 3;      // this works
        float n4  = n1 * n2 * 3.5f;
      //float n4  = n1 * n2 * 3.5;  // error, cannot convert double to float

        System.out.println("n3 = " + n3);
        System.out.println("n4 = " + n4);
    }
}
```

Output:

```
n3 = 9.0
n4 = 10.5
```

10. Input and Output

10.1 Output

In Java, to send output to standard output (screen), you can simply use:

```
System.out.print();
System.out.println();
System.out.printf();
```

The print() method displays a string on the console and retains the cursor in the same line. It works only with an argument.

The println() method also displays a string on the console but moves the cursor to the next line. It can also work without arguments.

The printf() method provides string formatting (similar to printf in C/C++ programming).

Example 05: Print concatenated strings.

```
class Example {
    public static void main(String[] args) {
        double number = -10.6;

        System.out.println("Java is " + "awesome.");
        System.out.print("number = " + number);
    }
}
```

Output:

```
Java is awesome.
number = -10.6
```

Example 06: Using the printf() method.

```
class Example {
    public static void main(String[] args) {
        char c = 'a';
        int n = 10;
        float f = 2.0f;
        double d = 5.5;
        String s = "Hello";

        System.out.printf("c = %c\n", c);
        System.out.printf("n = %d\n", n);
        System.out.printf("f = %.2f\n", f);
        System.out.printf("d = %.2f\n", d);
        System.out.printf("s = %s\n", s);
    }
}
```

Output:

```
c = a
n = 10
f = 2.00
d = 5.50
s = Hello
```

10.2 Input

We can get an input from the user using object from the `Scanner` class. In order to use the object of Scanner, we need to import `java.util.Scanner` package.

Example 07: Get integer input from the user.

```java
import java.util.Scanner;

class Example {
    public static void main(String[] args) {
        // Create an object of Scanner
        Scanner input = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        // Take integer input from the user
        int number = input.nextInt();

        System.out.println("You entered " + number);

        // Close the Scanner object
        input.close();
    }
}
```

Output:

```
Enter an integer: 10
You entered 10
```

In the above example, we have created an object named input of the `Scanner` class. We then call the `nextInt` () method of the Scanner class to get an integer input from the user.

Similarly, we can use `nextLong`(), `nextFloat`(), `nextDouble`(), `next`() and `nextLine`() methods to get long, float, double, and string input respectively from the user.

Example 08: Get float, double and string input from the user.

```
import java.util.Scanner;

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

        // Create an object of Scanner
        Scanner input = new Scanner(System.in);

         // Getting float input
         System.out.print("Enter float: ");
         float myFloat = input.nextFloat();

         // Getting double input
         System.out.print("Enter double: ");
         double myDouble = input.nextDouble();

         input.nextLine();      // notice this line

         // Getting String input
         System.out.print("Enter string: ");
         String myString = input.nextLine();

         // Getting char input
         System.out.print("Enter char: ");
         char myChar = input.next().charAt(0);

         System.out.println("\nFloat entered = " + myFloat);
         System.out.println("Double entered = " + myDouble);
         System.out.println("String entered = " + myString);
         System.out.println("Char entered = " + myChar);

         // Close the Scanner object
         input.close();
    }
}
```

Output:

```
Enter float: 2.5
Enter double: 3.6
Enter string: Hello Java
Enter char: b

Float entered = 2.5
Double entered = 3.6
String entered = Hello Java
Char entered = b
```

If you remove the `input.nextLine();` the string input after the double (same for int, float, char) input would be skipped. This is a common problem, and it happens because the nextDouble method does not read the newline character of your input, so when you issue the command `nextLine`, the Scanner finds the newline character and gives you that as a line.

11. Mathematical Methods

Java `Math` class provides several methods to work on math calculations like `min`(), `max`(), `pow`(), `sqrt`(), `avg`(), `sin`(), `cos`(), `tan`(), `round`(), `ceil`(), `floor`(), `abs`() etc. More Mathematical Methods here.

You do not need to import the `Math` class since it is in the `java.lang` package, which is imported by default.

Example 09: Using mathematical methods.

```
class Example {
    public static void main(String[] args) {
        double x = 2;
        double y = 4.5;

        System.out.println(Math.max(x, y));
        System.out.println(Math.sqrt(y));
        System.out.println(Math.pow(x, 2));
        System.out.println(Math.round(y));
        System.out.println(Math.log10(x));
    }
}
```

Output:

```
4.5
2.1213203435596424
4.0
5
0.3010299956639812
```

12. Control Statements

12.1 Selections

`if`, `if-else`, Ternary Conditional Operator (?:), `if-else-if`, and `switch`.

12.2 Iterations

`for` loop, `while` loop, and `do`...`while` loop

12.3 Special Control Statements

`break` and `continue`.

Exercises

Write the following programs:

1. Set the total to 0 to start with. While the total is 50 or less, ask the user to input a number. Add that number to the total and print a message "The total is [total]". Stop the loop when the total is over 50.
2. Ask the user to enter a number. Keep asking until they enter a value over 5 and then display the message "The last number you entered was a [number]" and stop the program.
3. Ask the user to enter a number and then enter another number. Add these two numbers together and then ask if they want to add another number. If they enter "y", ask them to enter another number and keep adding numbers until they do not answer "y". Once the loop has stopped, display the total.
4. Create a variable called computer_num and set the value to 50. Ask the user to enter a number. While their guess is not the same as the computer_num value, tell them if their guess is too low or too high and ask them to have another guess. If they enter the same value as computer_num, display the message "Well done, you took [count] attempts".
5. Find the largest even integer **n** such that $\mathbf{n}^3$ is less than 11000.
6. Write a program that reads an unspecified number of integers, determines how many positive and negative values have been read, and computes the total and average of the input values (not counting zeros). Your program ends with the input **0**. Display the total number of positive integers, the total number of negative integers, the total, and the average as a floating-point number.
7. (*Find numbers divisible by 5 or 6, but not both*) Write a program that displays ten numbers per line, all the numbers from 100 to 200 that are divisible by 5 or 6, but not both. The numbers are separated by exactly one space.
8. (*Sum a series*) Write a program to sum the following series:
$$\frac{1}{3} + \frac{3}{5} + \frac{5}{7} + \frac{7}{9} + \frac{9}{11} + \frac{11}{13} + \dots + \frac{95}{97} + \frac{97}{99}$$
9. (*Perfect number*) A positive integer is called a *perfect number* if it is equal to the sum of all of its positive divisors, excluding itself. For example, 6 is the first perfect number, because 6 = 3 + 2 + 1. The next is 28 = 14 + 7 + 4 + 2 + 1. There are four perfect numbers less than 10,000. Write a program to find these four numbers.
10. Write a program that calculates the following summation:
$$\frac{1}{1+\sqrt{2}} + \frac{1}{\sqrt{2}+\sqrt{3}} + \frac{1}{\sqrt{3}+\sqrt{4}} + \dots + \frac{1}{\sqrt{624}+\sqrt{625}}$$

Reference

[1] Y. Daniel Liang. 'Introduction to Java Programming', 11e – 2019

[2] https://www.programiz.com/java-programming