Programming in Java

What is Java

Java is a programming language developed by Sun Microsystems (which is a part of Oracle now) in 1995. James Gosling is known as the father of Java. It is one of the most popular programming languages currently and is used on more than 3 billion devices.

There are many types of applications that are developed using Java for example:

- Mobile applications: Java is the primary technology behind Android application development. Some of the top companies using Java for their mobile applications include Spotify, Netflix, Google Earth, Uber, etc.
- Desktop Applications such as acrobat reader, media player, antivirus, etc.
- Web applications such as javatpoint.com, etc.
- Web servers and application servers
- Games like Mission Impossible III, Minecraft were created using Java
- Embedded Systems. For example, SIM cards that our phones utilize uses Java.
- Database connection
- And much, much more!

Significant Language Features

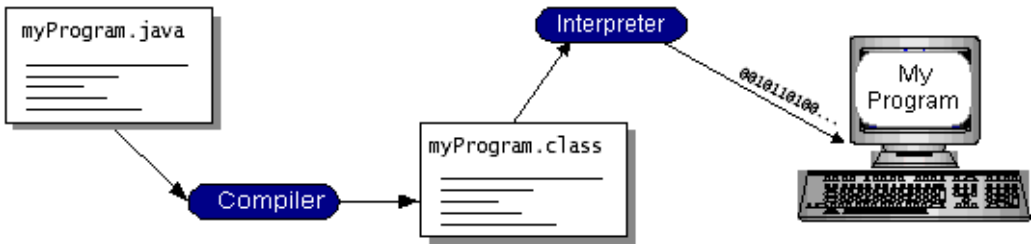
1. **Platform Independence** - Java compilers do not produce native object code for a particular platform but rather 'byte code' instructions for the Java Virtual Machine (JVM). Making Java code work on a particular platform is then simply a matter of writing a byte code interpreter to simulate a JVM. What this all means is that the same compiled byte code will run unmodified on any platform that supports Java.
2. **Object Orientation** - Java is a pure object-oriented language. This means that everything in a Java program is an object and everything is descended from a root object class.
3. **Rich Standard Library** - One of Java's most attractive features are its standard library. The Java environment includes hundreds of classes and methods in six major functional areas.

- *Language Support* classes for advanced language features such as strings, arrays, threads, and exception handling.
- *Utility* classes like a random number generator, date and time functions, and container classes.
- *Input/output* classes to read and write data of many types to and from a variety of sources.
- *Networking* classes to allow inter-computer communications over a local network or the Internet.
- *Abstract Window Toolkit* for creating platform-independent GUI applications.
- *Applet* is a class that lets you create Java programs that can be downloaded and run on a client browser.

4. **Applet Interface** - In addition to being able to create stand-alone applications, Java developers are capable of creating programs that can be downloaded from a web page and run on a client browser.
5. **Familiar C++-like Syntax** - One of the factors enabling the rapid adoption of Java is the similarity of the Java syntax to that of the popular C++ programming language.
6. **Garbage Collection** - Java does not require programmers to explicitly free dynamically allocated memory. This makes Java programs easier to write and less prone to memory errors.

Running Programs in Java

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first, you translate a program into an intermediate language called **Java bytecodes**-the **platform-independent codes** interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java bytecode instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed.



Java bytecodes help make "**write once, run anywhere**" possible. You can compile your program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. That means that if a computer has a Java VM, the same program written in the Java programming language can run on Windows 10, a Solaris workstation, or on an iMac.

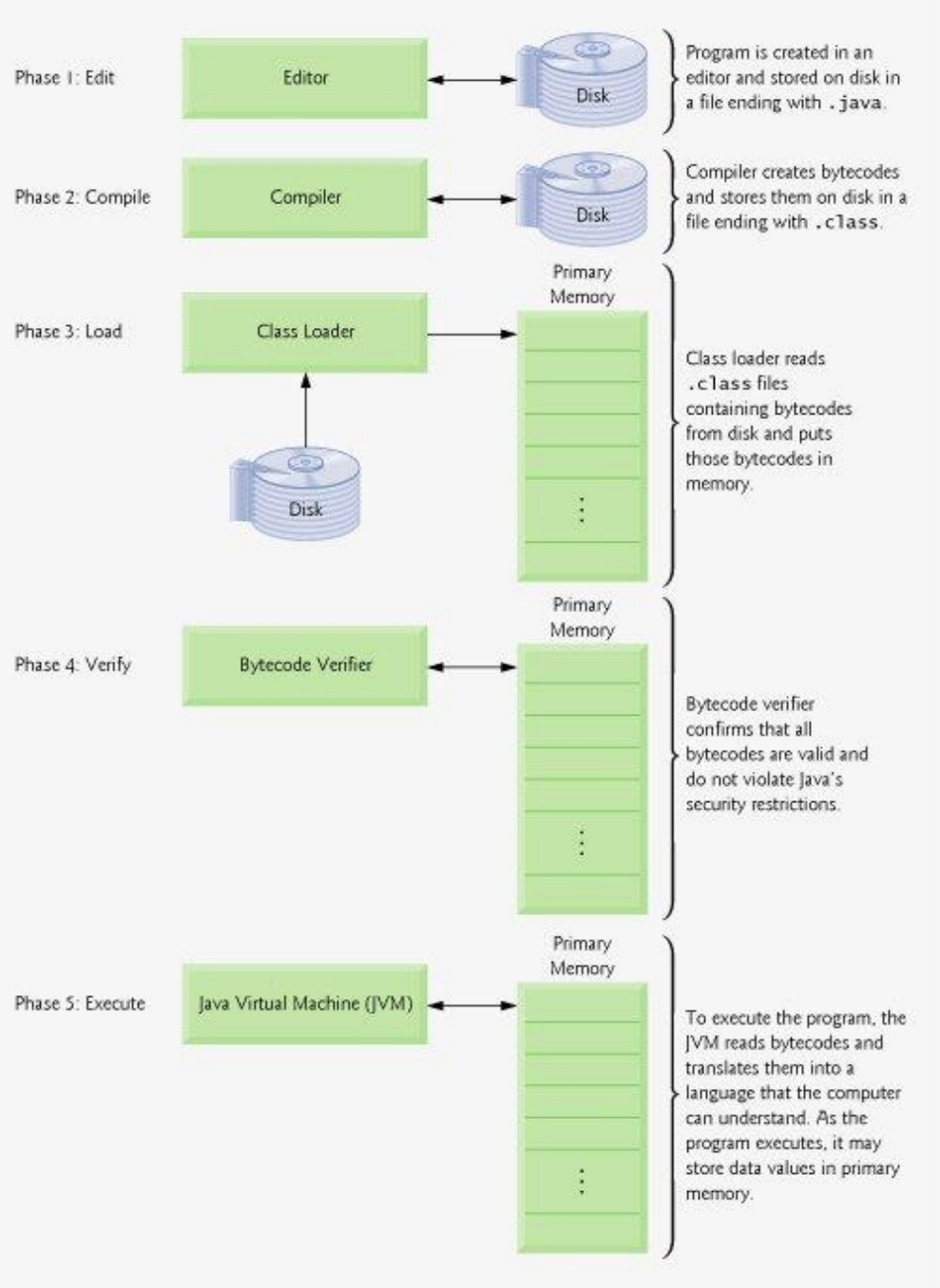


Figure 1The life cycle of a Java program

Types of Java Programs

There are two types of programs in Java:

- Applications
- Applets

An **application** is a program (such as a word-processor program, a spreadsheet program, a drawing program, or an e-mail program) that normally is stored and executed from the user's local computer.

To create an application in Java you must do the following steps:

1. Create a source file (writing the code).
2. Compile the source file into a bytecode file.
3. Run the program (code) contained in the bytecode file.

Example:

```
/* java application used to display the message "Welcome to Java
programming!" on the standard output. */
// Welcome1.java
// Text-printing program.
public class Welcome1 {
      // main method begins execution of Java application
      public static void main (String [] args )
      {
            System.out.println( "Welcome to Java Programming!");
      } // end method main
} // end class Welcome1
```

Output:

Welcome to Java Programming!

An **applet** is a small program that normally is stored on a remote computer that users connect to via a World Wide Web browser. The remote computer is known as a Web server. Applets are loaded from a remote computer into the browser, executed in the browser and discarded when

execution completes. To execute an applet again, the user must point a browser at the appropriate location on the World Wide Web and reload the program into the browser.

Java - Basic Syntax

When we consider a Java program it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods and instance variables mean.

- Object: Objects have states and behaviors
  Example: A dog has states: color, name, and breed as well as behaviors: wagging, barking, eating. An object is an instance of a class.
- Class: A class can be defined as a template/blueprint that describes the behaviors/states that an object of its type supports.
- Methods: A method is a behavior. A class can contain many methods. It is in methods where the logic is written, data is manipulated, and all the actions are executed.
- Instance Variables: Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Important notes about Java syntax

About Java programs, it is very important to keep in mind the following points:

- Case Sensitivity: Java is case sensitive, which means identifier Hello and hello would have different meanings in Java.
- Class Names: For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

  Example

  ```
  class MyFirstJavaClass
  ```

- Method Names: All method names should start with a Lower-Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

Example

```
public void myMethodName()
```

- Program File Name – The name of the program file should exactly match the class name.

Java Identifiers:

All Java components require names. Names used for classes, variables, and methods are called identifiers.

In Java, there are several points to remember about identifiers. They are as follows:

- All identifiers should begin with a letter (A to Z or a to z), a currency character ($) or an underscore (_).
- After the first character, identifiers can have any combination of characters.
- No spaces or special characters are allowed in the variable names of Java.
- A keyword cannot be used as an identifier.
- Most importantly, identifiers are case-sensitive.

  Examples of legal identifiers: age, $salary, _value, __1_value

  Examples of illegal identifiers: 123abc, -salary

Java - Basic Data types

There are two data types available in Java: Primitive Data Types and Reference/Object Data Types.

- Primitive Data Types:

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword.

1. byte: **byte data type is an 8-bit signed two's complement integer.**

   Example: byte a = 100 , byte b = -50
2. short: **short data type is a 16-bit signed two's complement integer.**

   Example: short s = 10000, short r = -20000
3. int: **int data type is a 32-bit signed two's complement integer.**

   Example: int a = 100000, int b = -200000

4. long: **long data type is a 64-bit signed two's complement integer.**

   Example: long a = 100000L, long b = -200000L

5. float: **float data type is a single-precision 32-bit IEEE 754 floating point.**

   Example: float f1 = 234.5f

6. double: **double data type is a double-precision 64-bit IEEE 754 floating point.**

   Example: double d1 = 123.4

7. boolean: **boolean data type represents one bit of information.**

   Example: boolean one = true

8. char: **char data type is a single 16-bit Unicode character.**

   Example: char letterA ='A'

- Reference/Object Data Types

Non-primitive data types are called reference types because they refer to objects. The main difference between primitive and non-primitive data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type always has a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types start with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc.

Final Variables

The value of a final variable cannot be changed after it has been initialized. Such variables are similar to constants in other programming languages.

To declare a final variable, use the final keyword in the variable declaration before the type:

```
final int aFinalVar = 0;
```

The previous statement declares **aFinal** variable and initializes it, all at once. Subsequent attempts to assign a value to **aFinalVar** result in a compiler error. You may, if necessary, defer initialization of a final local variable. Simply declare the local variable and initialize it later, like this:

```
final int blankfinal;
```

[…]

```
blankfinal = 0;
```

A final local variable that has been declared but not yet initialized is called a **blankfinal**. Again, once a final local variable has been initialized, it cannot be set, and any later attempts to assign a value to **blankfinal** result in a compile-time error.

Java - Basic Operators

The Arithmetic Operators:

Operator	Result
+	Addition (also unary plus)
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
––	Decrement

Example:

Assume integer variable A holds 10 and variable B holds 20, then the following simple example program demonstrates the arithmetic operators.

```
public class Test {
public static void main(String args[]) {
int a = 10;
int b = 20;
int c = 25;
int d = 25;
System.out.println("a + b = " + (a + b) );
System.out.println("a - b = " + (a - b) );
System.out.println("a * b = " + (a * b) );
System.out.println("b / a = " + (b / a) );
System.out.println("b % a = " + (b % a) );
System.out.println("c % a = " + (c % a) );
System.out.println("a++ = " + (a++) );
System.out.println("b-- = " + (a--) );
```

```
// Check the difference in d++ and ++d
System.out.println("d++ = " + (d++) );
System.out.println("++d = " + (++d) );
}
}// end class
```

This would produce the following result:

a + b = 30

a - b = -10

a * b = 200

b / a = 2

b % a = 0

c % a = 5

a++ = 10

b-- = 11

d++ = 25

++d = 27