Evolution of Java

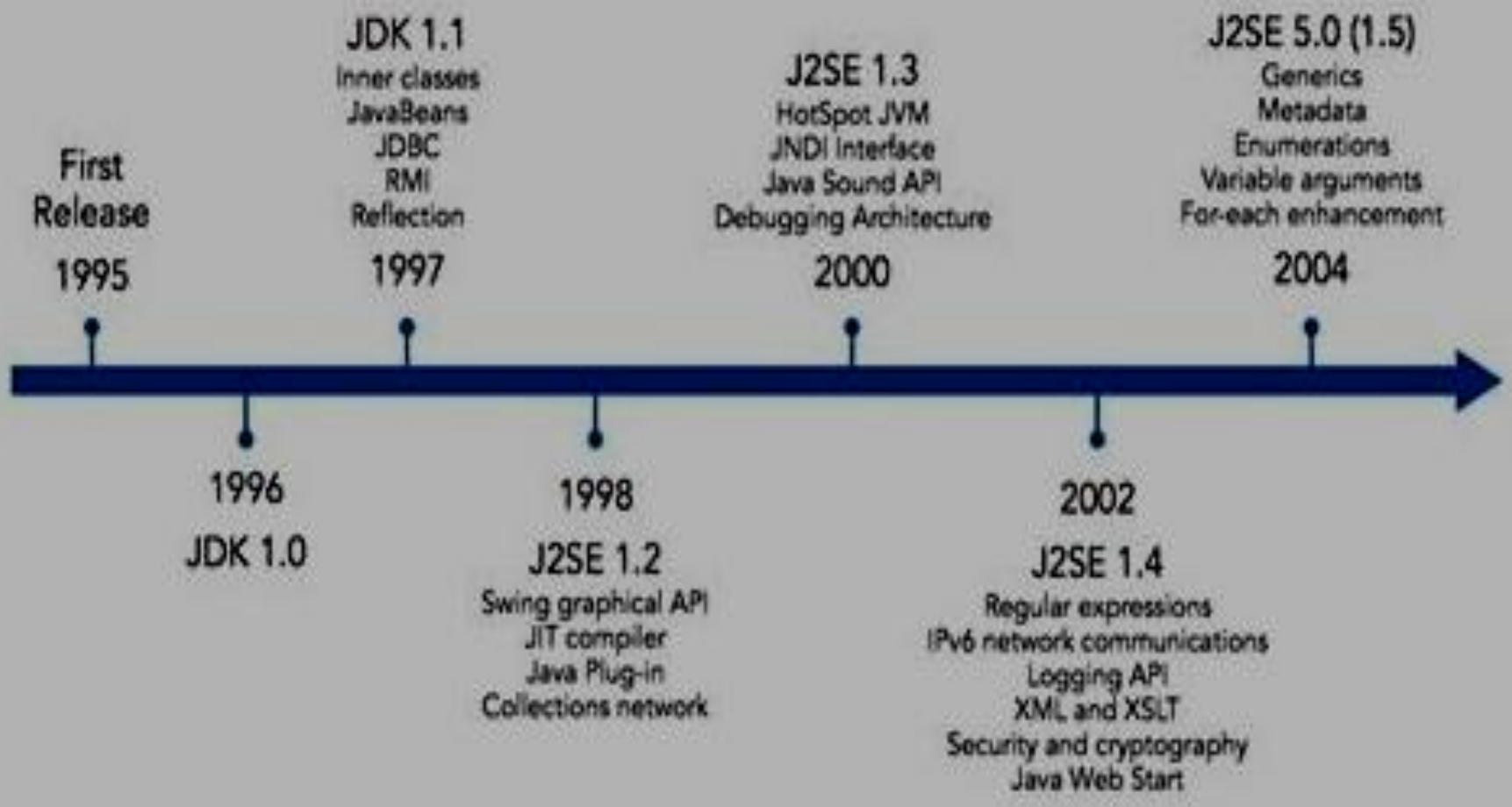
- Java is invented by *James Gosling*, *Patrick Naughton*, *Chris Warth*, *Ed Frank*, and *Mike Sheridan* at Sun Microsystems in 1991.

- Java is related to C++, which is inherited from the language C.

- The characters of Java is inherited from C and C++ language. It took approx. Eighteen months to develop the first working version.

- It was first named as “Oak” but was renamed as “Java” in 1995.

The History of Java



- The basic idea behind creating this language is to create a platform-independent language that is used to develop software for consumer electronic devices such as microwave ovens, remote controls, etc.

- Initially, it was not designed for Internet applications.

- It is possible to compile C++ code for any processor but to do so it requires a full C++ compiler targeted for that particular processor and platform.

- That makes it expensive and time-consuming.

- To overcome this, Gosling and others started working on a portable and platform-independent language, this leads to the creation of Java.

Java 2 Features	*Additional Features of J2SE 5.0*
• Compiled and Interpreted	• Ease of Development
• Platform-Independent and Portable	• Scalability and Performance
• Object-Oriented	• Monitoring and Manageability
• Robust and Secure	• Desktop Client
• Distributed	• Core XML Support
• Familiar, Simple and Small	• Supplementary character support
• Multithreaded and Interactive	• JDBC RowSet
• High Performance	
• Dynamic and Extensible	

Compiled and Interpreted

Usually a computer language is either compiled or interpreted. Java combines both these approaches thus making Java a two-stage system. First, Java compiler translates source code into what is known as *bytecode* instructions. Bytecodes are not machine instructions and therefore, in the second stage, Java interpreter generates machine code that can be directly executed by the machine that is running the Java program. We can thus say that Java is both a compiled and an interpreted language.

Platform-Independent and Portable

The most significant contribution of Java over other languages is its portability. Java programs can be easily moved from one computer system to another, *anywhere* and *anytime*. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs. This is the reason why Java has become a popular language for programming on Internet which interconnects different kinds of systems worldwide. We can download a Java applet from a remote computer onto our local system via Internet and execute it locally.

Object-Oriented

Java is a true object-oriented language. Almost everything in Java is an *object*. All program code and data reside within objects and classes. Java comes with an extensive set of *classes*, arranged in *packages*, that we can use in our programs by inheritance. The object model in Java is simple and easy to extend.

Robust and Secure

Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. It is designed as a garbage-collected language relieving the programmers virtually all memory management problems. Java also incorporates the concept of exception handling which captures series errors and eliminates any risk of crashing the system.

Security becomes an important issue for a language that is used for programming on Internet. Threat of viruses and abuse of resources are everywhere. Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet. The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

Distributed

Java is designed as a distributed language for creating applications on networks. It has the ability to share both data and programs. Java applications can open and access remote objects on Internet as easily as they can do in a local system. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

Simple, Small and Familiar

Java is a small and simple language. Many features of C and C++ that are either redundant or sources of unreliable code are not part of Java. For example, Java does not use pointers, preprocessor header files, **goto** statement and many others. It also eliminates operator overloading and multiple inheritance.

Multithreaded and Interactive

Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs. This means that we need not wait for the application to finish one task before beginning another. For example, we can listen to an audio clip while scrolling a page and at the same time download an applet from a distant computer. This feature greatly improves the interactive performance of graphical applications.

Difference between Java and C

- Java does not include the C unique statement keywords **sizeof**, and **typedef**.
- Java does not contain the data types **struct** and **union**.
- Java does not define the type modifiers keywords **auto, extern, register, signed,** and **unsigned.**
- Java does not support an explicit pointer type.
- Java does not have a preprocessor and therefore we cannot use **# define, # include**, and **# ifdef** statements.
- Java requires that the functions with no arguments must be declared with empty parenthesis and not with the **void** keyword as done in C.
- Java adds new operators such as **instanceof** and >>>.
- Java adds labelled **break** and **continue** statements.
- Java adds many features required for object-oriented programming.

Difference between Java and C++

- Java does not support operator overloading.
- Java does not have template classes as in C++.
- Java does not support multiple inheritance of classes. This is accomplished using a new feature called "interface".
- Java does not support global variables. Every variable and method is declared within a class and forms part of that class.
- Java does not use pointers.
- Java has replaced the destructor function with a finalize( ) function.
- There are no header files in Java.

Java Environment

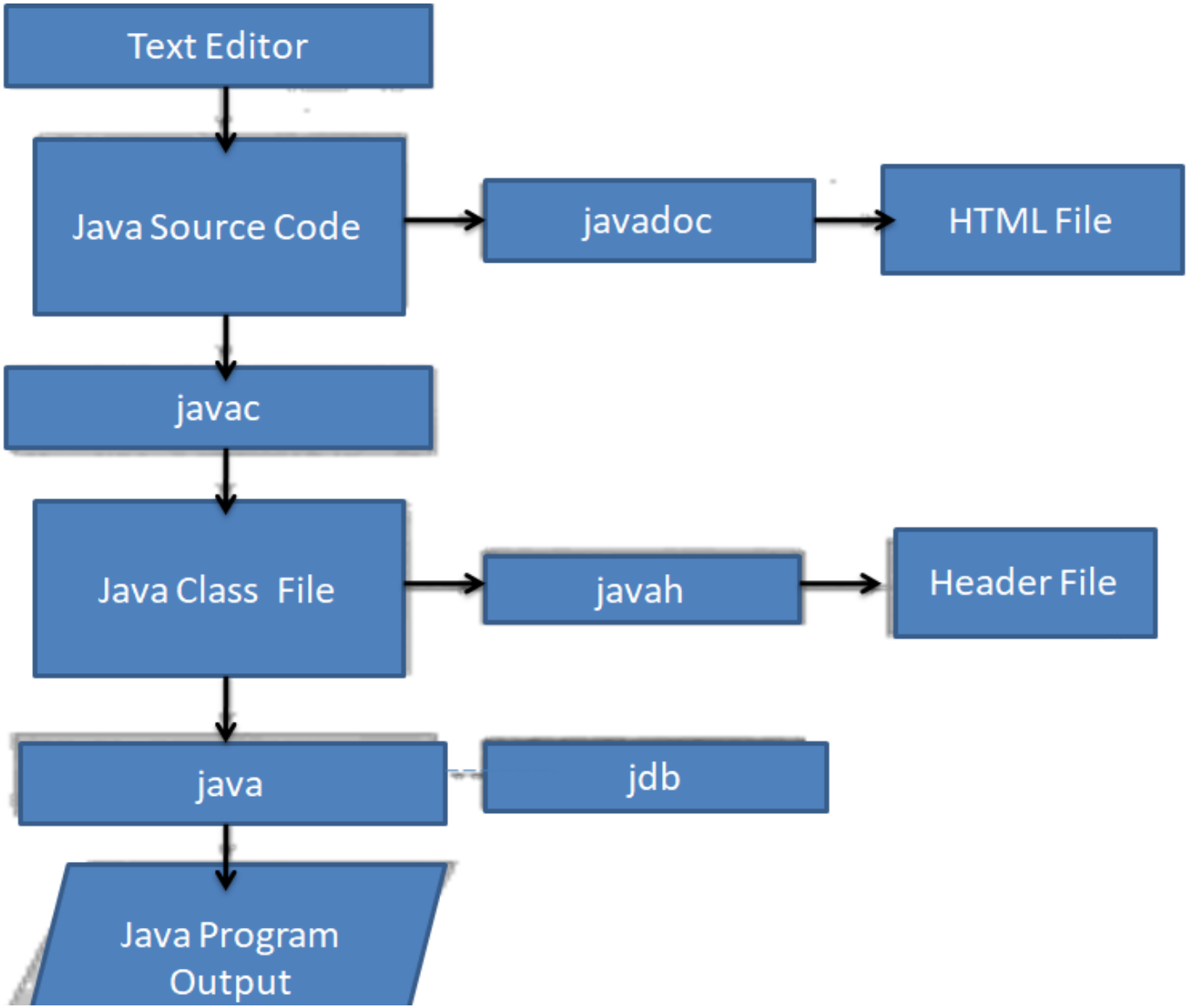
Java environment includes a large number of development tools and hundreds of classes and methods. The development tools are part of the system known as *Java Development Kit* (JDK) and the classes and methods are part of the *Java Standard Library* (JSL), also known as the *Application Programming Interface* (API).

Java Development Kit

The Java Development Kit comes with a collection of tools that are used for developing and running Java programs. They include:

- appletviewer (for viewing Java applets)
- javac (Java compiler)
- java (Java interpreter)
- javap (Java disassembler)
- javah (for C header files)
- javadoc (for creating HTML documents)
- jdb (Java debugger)

Process of developing and executing Java Application program


Text Editor
Java Source Code
javadoc
HTML File
javac
Java Class File
javah
Header File
java
jdb
Java Program Output

Application Programming Interface

The Java Standard Library (or API) includes hundreds of classes and methods grouped into several functional packages (see Appendix G). Most commonly used packages are:

- **Language Support Package:** A collection of classes and methods required for implementing basic features of Java.
- **Utilities Package:** A collection of classes to provide utility functions such as date and time functions.
- **Input/Output Package:** A collection of classes required for input/output manipulation.
- **Networking Package:** A collection of classes for communicating with other computers via Internet.
- **AWT Package:** The Abstract Window Tool Kit package contains classes that implements platform-independent graphical user interface.
- **Applet Package:** This includes a set of classes that allows us to create Java applets.

Overview of Java Language

Java is a general-purpose, object-oriented programming language. We can develop two types of Java programs:

- Stand-alone applications
- Web applets

Stand-alone Application

Stand-alone applications are programs written in Java to carry out certain tasks on a stand-alone local computer. In fact, Java can be used to develop programs for all kinds of applications Executing a stand-alone Java program involves two steps:

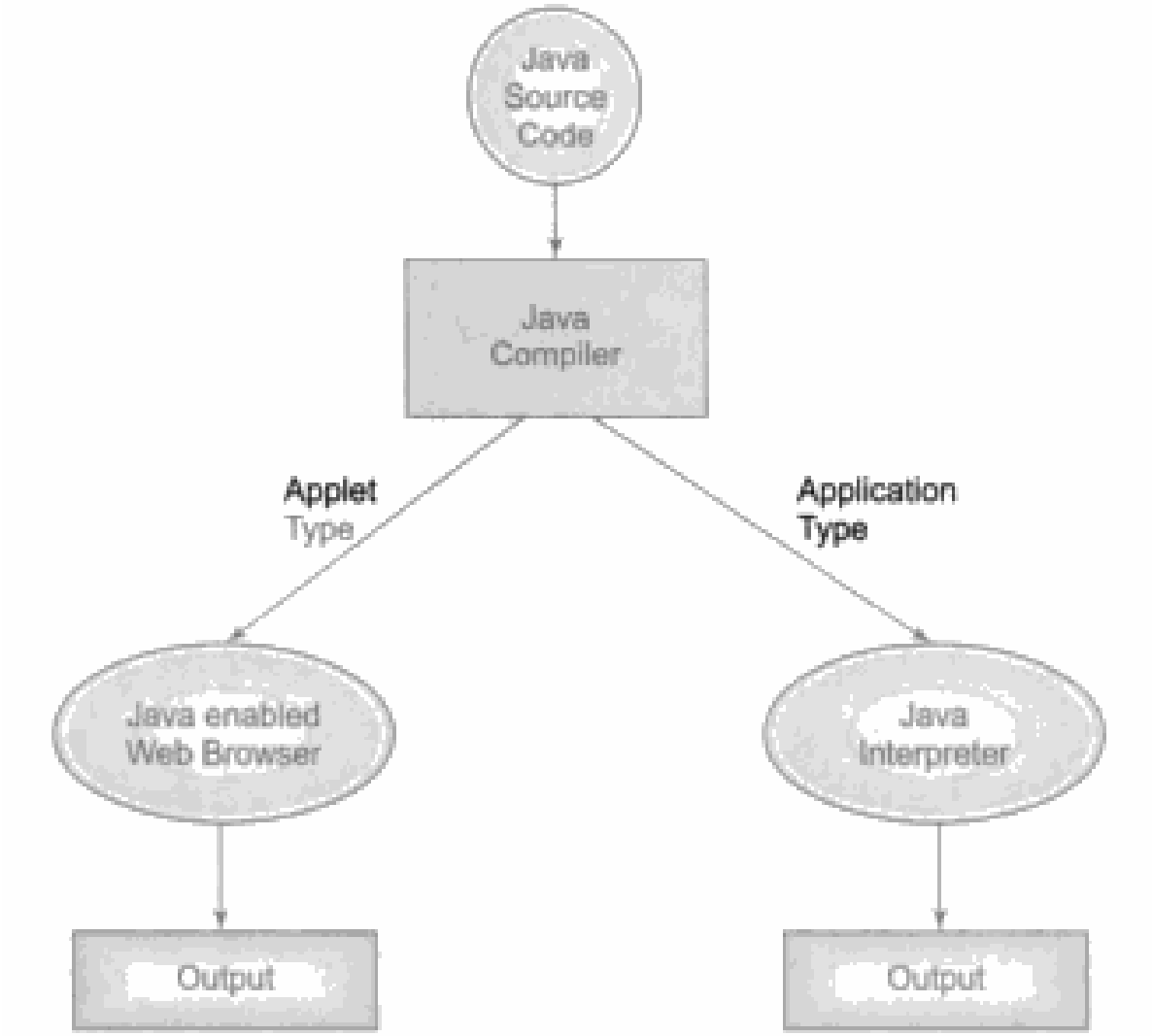
1. Compiling source code into bytecode using **javac** compiler
2. Executing the bytecode program using **java** interpreter.

Applet

Applets are small Java programs developed for Internet applications. An applet located on a distant computer (Server) can be downloaded via Internet and executed on a local computer (Client) using a Java-capable browser.

We can develop applets for doing everything from simple animated graphics to complex games and utilities. Since applets are embedded in an HTML (Hypertext Markup Language) document and run inside a Web page, creating and running applets are more complex than creating an application.

Stand-alone programs can read and write files and perform certain operations that applets cannot do. An applet can only run within a Web browser.


Java
Source
Code
Java
Compiler
Applet
Type
Application
Type
Java enabled
Web Browser
Java
Interpreter
Output
Output

Simple Java Program

```
class SampleOne
{
  public static void main (String args[ ])
  {
      System.out.println("Java is better than C++.");
  }
}
```

This line contains a number of keywords, **public, static** and **void**.

Public:	The keyword **public** is an access specifier that declares the **main** method as unprotected and therefore making it accessible to all other classes. This is similar to the C++ **public** modifier.
Static:	Next appears the keyword **static**, which declares this method as one that belongs to the entire class and not a part of any objects of the class. The **main** must always be declared as **static** since the interpreter uses this method before any objects are created. More about static methods and variables will be discussed later in Chapter 8.
Void:	The type modifier **void** states that the **main** method does not return any value (but simply prints some text to the screen.)

All parameters to a method are declared inside a pair of parentheses. Here, **String args[ ]** declares a parameter named **args**, which contains an array of objects of the class type **String**.

The println method is a member of out object which is static data member of System Class

System.out.println(“Hello Java”);

Program to find Squareroot

```
import java.lang.Math;
class SquareRoot
{
      public static void main(String args[ ])
      {
          double x = 5 ;  // Declaration and initialization
          double y;       // Simple declaration
          y = Math.sqrt(x) ;
          System.out.println("y = " + y);
      }
}
```

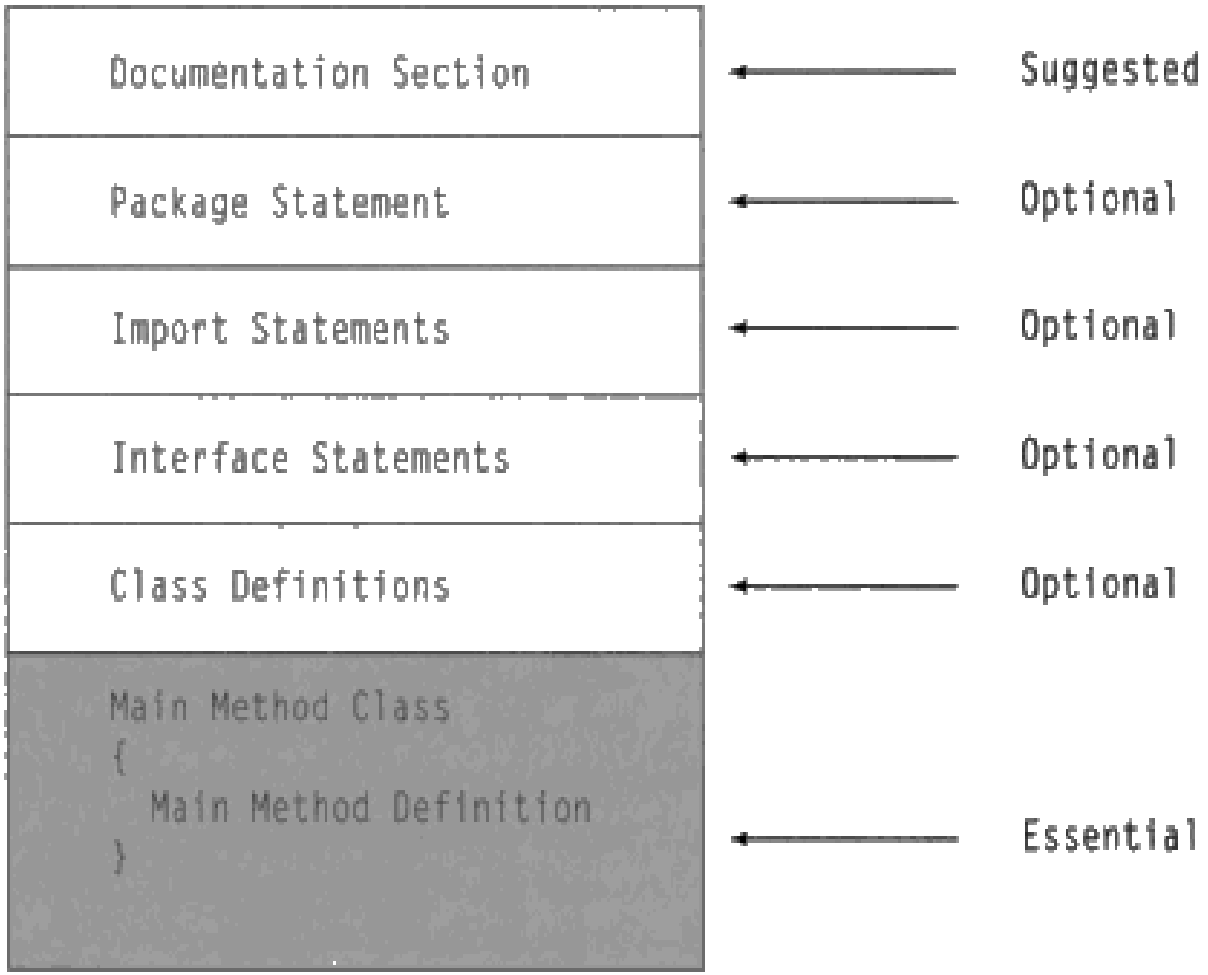
An Application of two class

Both the examples discussed above use only one class that contains the main method. A real-life application will generally require multiple classes. Program 3.3 illustrates a Java application with two classes.

```
class Room
{
      float length;
      float breadth;

      void getdata(float a, float b)
      {
          length  = a;
          breadth = b;
      }
}
class RoomArea
{
    public static void main (String args[ ])
    {
        float area;
        Room room1 = new Room( ); // Creates an object room1
        room1.getdata(14, 10);    // Assigns values to length and breadth
        area = room1.length * room1. breadth;
        System.out.println ("Area =" + area);
    }
}
```

Java Program Structure



Java Tokens

- Smallest unit of a program is called as token.

- There are five tokens namely:

1. Reserved keywords
2. Identifiers
3. Literals
4. Operators
5. Separators

Reserved Keywords

- Reserved keywords are essential part of language definition.
- Java language has 50 reserved keywords.
- They cannot be used as identifiers.

Identifiers

Identifiers are programmer-designed tokens. They are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program. Java identifiers follow the following rules:

1. They can have alphabets, digits, and the underscore and dollar sign characters.
2. They must not begin with a digit.
3. Uppercase and lowercase letters are distinct.
4. They can be of any length.

Identifier must be meaningful, short enough to be quickly and easily typed and long enough to be descriptive and easily read. Java developers have followed some naming conventions.

- Names of all public methods and instance variables start with a leading lowercase letter. Examples:

```
average
sum
```

- When more than one words are used in a name, the second and subsequent words are marked with a leading uppercase letters. Examples:

```
dayTemperature
firstDayOfMonth
totalMarks
```

- All private and local variables use only lowercase letters combined with underscores. Examples:

```
length
batch_strength
```

- All classes and interfaces start with a leading uppercase letter (and each subsequent word with a leading uppercase letter). Examples:

```
Student
HelloJava
Vehicle
MotorCycle
```

- Variables that represent constant values use all uppercase letters and underscores between words. Examples:

```
TOTAL
F_MAX
PRINCIPAL_AMOUNT
```

LITERALS

Literals in Java are a sequence of characters (digits, letters, and other characters) that represent constant values to be stored in variables. Java language specifies five major types of literals. They are:

- Integer literals
- Floating_point literals
- Character literals
- String literals
- Boolean literals

Operators

An operator is a symbol that takes one or more arguments and *operates* on them to produce a result.

Separators

Separators are symbols used to indicate where groups of code are divided and arranged.

parentheses ( )	Used to enclose parameters in method definition and invocation, also used for defining precedence in expressions, containing expressions for flow control, and surrounding cast types.
braces { }	Used to contain the values of automatically initialized arrays and to define a block of code for classes, methods and local scopes
brackets [ ]	Used to declare array types and for dereferencing array values
semicolon ;	Used to separate statements
comma ,	Used to separate consecutive identifiers in a variable declaration, also used to chain statements together inside a 'for' statement
period .	Used to separate package names from sub-packages and classes; also used to separate a variable or method from a reference variable.

Java Virtual Machine

Java compiler produces an intermedia code known as *bytecode* for a machine that does not exist. This machine is called the *Java Virtual Machine* and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer. Figure illustrates the process of compiling a Java program into bytecode which is also referred to as *virtual machine code*.

Process of Compilation

The virtual machine code is not machine specific. The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine

Process of converting bytecode into machine code

Command Line Argument

There may be occasions when we may like our program to act in a particular way depending on the input provided at the time of execution. This is achieved in Java programs by using what are known as command line arguments. Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution.

Recall the signature of the **main( )** method used in our earlier example programs:

```
public static void main (String args[ ])
```

As pointed out earlier, **args** is declared as an array of strings (known as string objects). Any arguments provided in the command line (at the time of execution) are passed to the array **args** as its elements.

Consider the following statement where program name is Test and others are argument

Java Test Basic Fortran C++ Java

This command line contains four arguments. These are assigned to the array **args** as follows:

Let us see a program that will print all the command line arguments and its count

```
Class ComLineTest
{
    public static void main (String args[ ])
    {
        int count, i=0;
        String string;
        count = args.length;
        System.out.println("Number of arguments = " + count);
        while (i < count)
        {
            string = args[i];
            i = i + 1;
            System.out.println(i+ " : " + "Java is" + string+ "!");
        }
    }
}
```

Program illustrates the use of command line arguments. Compile and run the program with the command line as follows:

```
java ComLineTest Simple Object_Oriented Distributed Robust
Secure Portable Multithreaded Dynamic
```

The output of the program would be as follows:

```
Number of arguments = 8
    1  :  Java is Simple!
    2  :  Java is Object_Oriented!
    3  :  Java is Distributed!
    4  :  Java is Robust!
    5  :  Java is Secure!
    6  :  Java is Portable!
    7  :  Java is Multithreaded!
    8  :  Java is Dynamic!
```