Unit-I

Fundamentals of Java Programming

Introduction Object-Oriented Programming (OOP)

Basic Concepts of OOP

- It is a programming paradigm based on the concept of "objects", which can contain data and code: data- in the form of fields (often known as attributes or properties), and code- in the form of procedures (often known as methods).
- A feature of objects is that an object's own procedures can access and often modify the data fields of itself (objects have a notion of this or self). In OOP, computer programs are designed by making them out of objects that interact with one another.

- the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.
- OOP refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

The History of Java

- Java is related to C++, which is a direct descendant of C.
- From C, Java derives its syntax.
- Many of Java’s object-oriented features were influenced by C++.
- By the end of the 1980s and the early 1990s, object-oriented programming using C++ took hold.

The History of Java

- with the object-oriented paradigm, it was a language that could be used to create a wide range of programs.
- Within a few years, the World Wide Web and the Internet would reach critical mass.
- This event would precipitate another revolution in programming.

The History of Java

- Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991.
- This language was initially called "Oak".
- Later the project went by the name *Green* and was finally renamed "***Java***" in **1995**, from Java coffee, the coffee from Indonesia.
- The original impetus for Java was not the Internet!
- Instead, the primary motivation was the need for a platform-independent language

The History of Java

- The language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls.
- C/C++ compilers are platform dependent, they convert source code into machine language understandable to the particular OS.
  - If your operating system is changed then you have to change your compiler also.

The History of Java

- Although it is possible to compile a C++ program for just about any type of CPU, to do so requires a full C++ compiler targeted for that CPU.
  - The problem is that compilers are expensive and time-consuming to create.
- Gosling and others began work on a portable, platform-independent language that could be used to produce code that would run on a variety of CPUs under differing environments. **This effort ultimately led to the creation of Java**.

The History of Java

- With the emergence of the World Wide Web, Java was propelled to the forefront of computer language design, because the Web, too, demanded portable programs.
- Sun Microsystems released the first public implementation as Java 1.0 in 1996. It promised **Write Once, Run Anywhere** (**WORA**) functionality.

Java Version History

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan 1996)
- JDK 1.1 (19th Feb 1997)
- J2SE 1.2 (8th Dec 1998)
- J2SE 1.3 (8th May 2000)
- J2SE 1.4 (6th Feb 2002)
- J2SE 5.0 (30th Sep 2004)
- Java SE 6 (11th Dec 2006)
- Java SE 7 (28th July 2011)
- Java SE 8 (18th Mar 2014)
- Java SE 9 (21st Sep 2017)
- Java SE 10 (20th Mar 2018)
- Java SE 11 (25th Sept 2018)
- Java SE 12 (19th Mar 2019)
- Java SE 13 (17th Sep 2019)
- Java SE 14 (17th Mar 2020)

Features of Java

Features of Java

- **Simple**
  - Java is easy to learn for the professional programmers.
  - Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble learning Java.
  - Complex features of C++ has omitted like pre-processor, operator overloading, multiple inheritance etc.
  - Error prone task such as pointers and memory management have been eliminated or are handled by the java environment automatically.

Features of Java

- **Object-Oriented**
  - Java follows “everything is an object” paradigm
  - Designed as a pure Object-Oriented Language
  - Includes all OOP concepts like class, object, encapsulation, inheritance and polymorphism

Features of Java

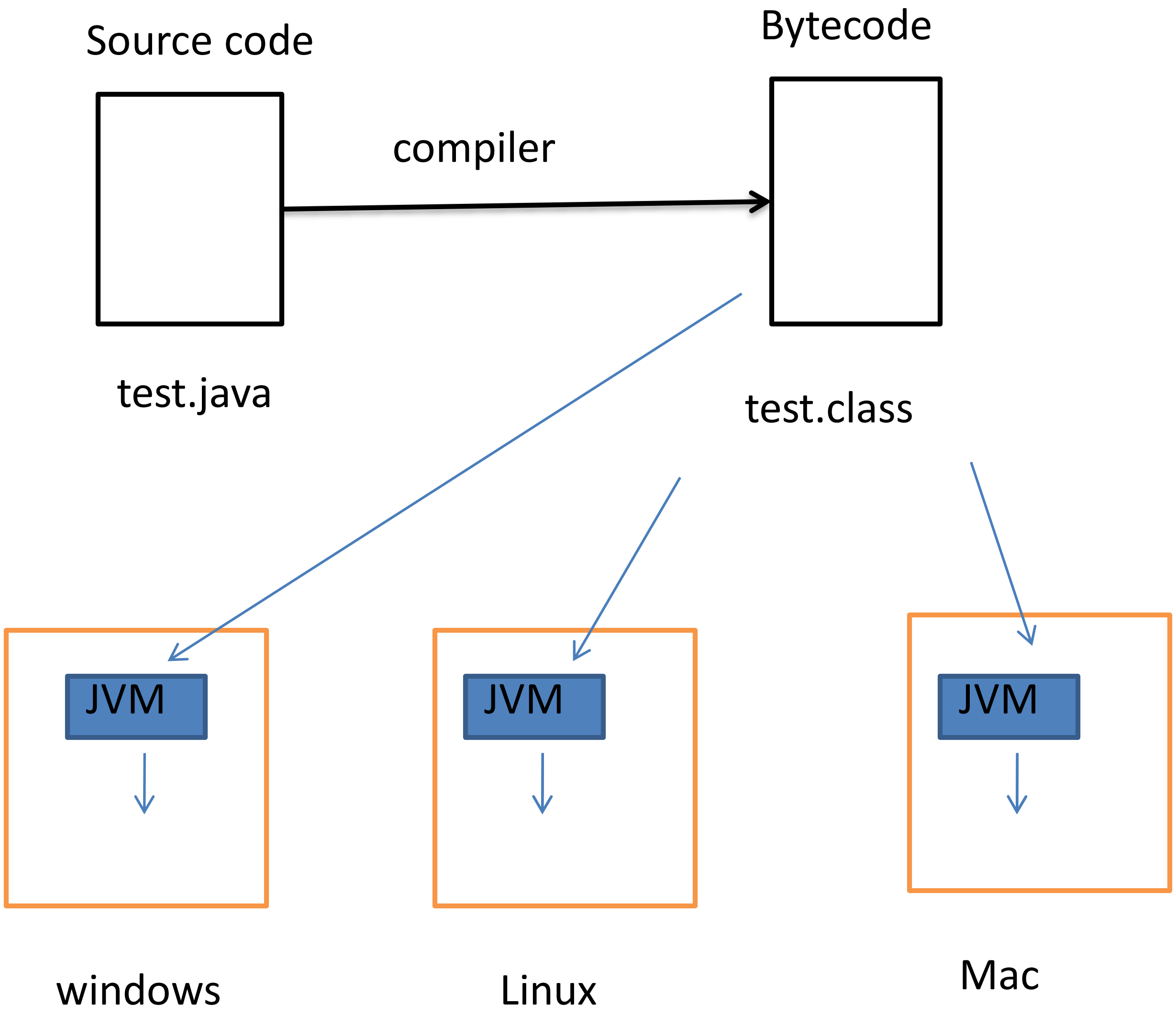
- **Robust**
  - Java is a strictly typed language
  - It checks your code at compile time, as well as at run time.
  - Java uses automatic garbage collection, which manages memory by preventing memory leaks.
  - In a well-written Java program, all run-time errors can—and should—be managed by your program.

Features of Java

- **Multithreaded**
  - Multithreaded programming allows you to write programs that execute more than one task (thread) at the same time.
  - It is possible to create multithreaded applications in Java.
  - Java uses threads to utilize the CPU idle time to perform the necessary garbage clean-up and general system maintenance.

Features of Java

- **Architecture-Neutral**
  - Java designers goal was “write once; run anywhere, any time, forever.”
  - The Java compiler compiles the source code and generates **bytecode**.
  - Bytecode is intermediate between source and native machine code.
  - This bytecode is neutral and has nothing to do with a particular computer architecture.
  - **Java Virtual Machine** (JVM) converts the bytecode into native code for a particular processor.


Source code
Bytecode
compiler
test.java
test.class
JVM
JVM
JVM
windows
Linux
Mac

Features of Java

- **Interpreted and High Performance**
  - The Java compiler compiles the Java source code into Bytecode.
  - Bytecode is an executable for JVM.
  - The bytecode is then interpreted by a Java interpreter which converts the bytecode into platform specific code and executes it.
  - Java improves its performance because of **Just-in-time** compiler.

Just-in-time Compiler

- JIT compiler is part of the JVM
- JIT compiles selected portions of bytecode into executable code in real time, on a piece-by-piece, demand basis.
- It is not practical to compile an entire Java program into executable code all at once, because Java performs various run-time checks that can be done only at run time.
- Instead, a JIT compiler compiles code as it is needed, during execution.
- Not all sequences of bytecode are compiled-only those that will benefit from compilation.
- The remaining code is simply interpreted.
- This improves the performance of the program.

Features of Java

- **Distributed**
  - Java is designed for the distributed environment of the Internet because it handles TCP/IP.
  - Java also supports ***Remote Method Invocation (RMI)****. This feature enables a program to* invoke methods across a network.

Features of Java

- **Dynamic**
  - Java programs carry substantial amounts of run-time type information
  - That is used to verify and resolve accesses to objects at run time.
  - This makes it possible to dynamically link code in a safe and expedient manner.
  - In Java small fragments of bytecode may be dynamically updated on a running system.

Features of Java

- **Secure**
  - A Java program is executed by the JVM also helps to make it secure.
  - Because the JVM is in control, it can contain the program and prevent it from generating side effects outside of the system.

Java Environment

JDK - Java Development Kit

- JDK is a software development environment which is used to develop and run java applications.
- The JDK contains a private Java Virtual Machine (JVM)
- It contain other resources such as an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) etc.

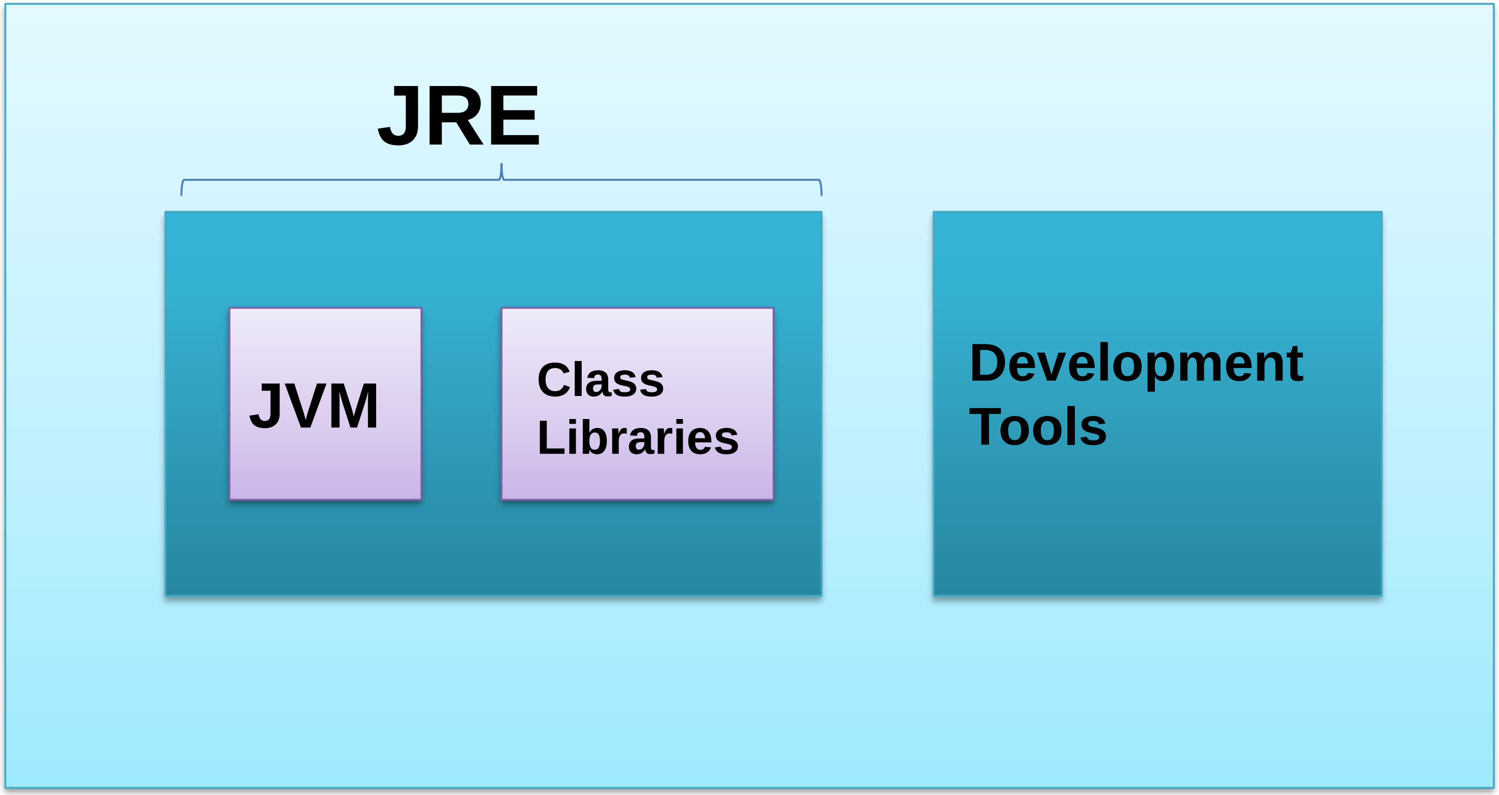
Components of JDK (Development Tools)

Components	Meaning
javac	It specifies the Java compiler, which converts source code into Java bytecode.
java	The loader for Java applications. This tool is an interpreter and can interpret the class files (Bytecode).
javadoc	The documentation generator, which automatically generates documentation from source code comments.
jdb	The Java debugger.
javap	Disassembling classes. List methods and class members.
javah	Header file generator. Used when combining Java with C/C++ programs.
jar	The specifies the archiver, which packages related class libraries into a single JAR file.
appletviewer	run and debug Java applets without a web browser.

JRE - Java Run-time Environment

- JRE is an environment which is used to run java applications. (e.g. Client machine)
- Hence, JRE is the part of JDK.
- It has JVM.
- It has Class Libraries such as Lang and util.
- It has other libraries like Java Database Connectivity (JDBC) and Java Naming and Directory Interface (JNDI), Java Management Extensions (JMX), Java Native Interface (JNI) and Java for XML Processing (JAX-WS).

JDK

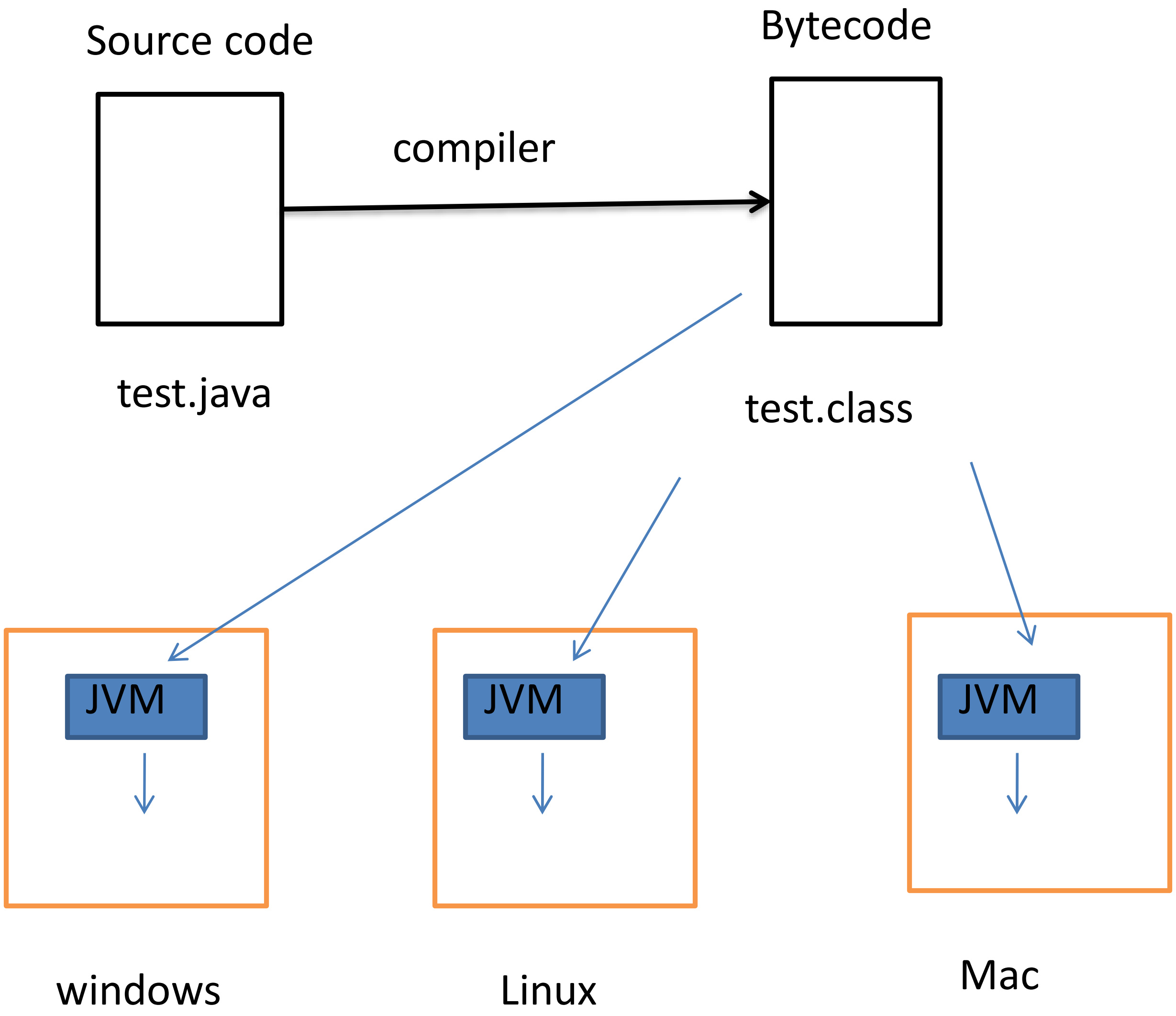


JRE

JVM

Class Libraries

Development Tools



Source code

compiler

Bytecode

test.java

test.class

JVM

JVM

JVM

windows

Linux

Mac

JVM – Java Virtual Machine

- JVM is an abstract machine. Which provides runtime environment in which java bytecode can be executed.
- In JRE, JVM is responsible to run Java application line by line.
- JVM is an interpreter.
- JVM is platform dependent and makes Java platform independent.
- JVM contains
  - Class loader
  - Memory area
  - Execution engine

Java **API (application programming interface)**

- Large collection of ready-made software components.
- Provide many useful capabilities, like GUI, event handling, I/O, file handling, networking etc.
- API is grouped into libraries of related classes and interfaces (packages)
- https://docs.oracle.com/javase/7/docs/api/

Summary

- JDK = JRE + Development tools (javac, java, javadoc etc.)
- JRE = JVM + Library Classes (packages)
- JVM – Class loader + Memory area + Execution engine

Application of Java

- **Mobile Applications**
  - Android application Package(APK) = JVM + DVK (Dalvik Virtual Machine)
  - Security
- **Desktop GUI Applications**
  - AWT, Swing

- **Embedded Systems**
  - SIM cards, blue-ray disk players, utility meters and televisions, use embedded Java technologies.

- **Enterprise Applications**
  - Network applications, web-services
  - Banking applications

- **Web-based Applications**
  - Servlet, JSP

- **Scientific Applications**
  - Scientific calculations and mathematical operations
  - Highly secure and fast
  - Portability
  - MATLAB

- **Gaming Applications**
  - Support most powerful 3D-Engine
  - 3D Games

- **Big Data technologies**
  - Big Data technologies like Hadoop

- **Business Applications**
  - security and reliability
  - reduces the complexity of enterprise application

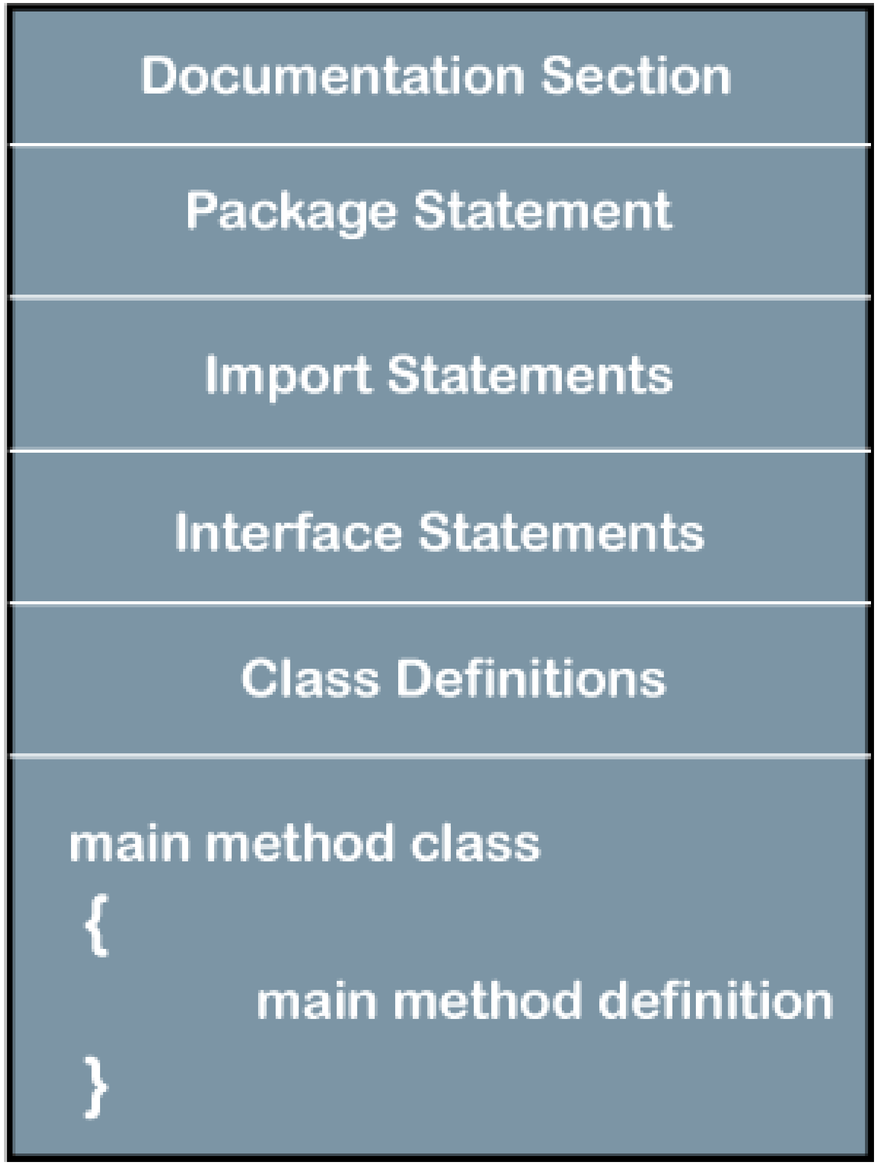
- **Distributed Applications**
  - Jini (Java Intelligent Networking Infrastructure)
  - provide, register, and find distributed services

- **Cloud-Based Applications**
  - companies can build their applications remotely
  - companies can share data

Simple Java program

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

- Save the above file as First.java
- **To compile:** javac First.java
- **To execute:** java First



Documentation Section

Package Statement

Import Statements

Interface Statements

Class Definitions

main method class
{
main method definition
}

Structure of Java Program

```
class CommandLine{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
```

- Save the above file as CommandLine.java
- compile by > javac CommandLine.java
- run by > java CommandLineExample

```
class  CommandLine {
public static void main(String args[])
{
  for(int i=0;i<args.length;i++)
System.out.println(args[i]);
 }
}
```

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

    int number=13;
    //Check if the number is divisible by 2 or not
    if(number%2==0){
        System.out.println("even number");
    }
else{
        System.out.println("odd number");
    }
}
}
```

```
public class SwitchExample {
public static void main(String[] args) {
      int number=20;
        switch(number){
       case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
       default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

```
public class WhileExample {
public static void main(String[] args) {
    int i=1;
    while(i<=10){
        System.out.println(i);
    i++;
    }
}
}
```

```
public class DoWhileExample {
public static void main(String[] args) {
    int i=1;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
}
```

```
class Fibonacci{
public static void main(String args[])
{
 int n1=0,n2=1,n3,i,count=10;
 System.out.print(n1+" "+n2);//printing 0 and 1

 for(i=2;i<count;++i)
 {
  n3=n1+n2;
  System.out.print(" "+n3);
  n1=n2;
  n2=n3;
 }
  }
}
```

Java Array

- array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java.
- Advantages
- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Syntax to declare array in java

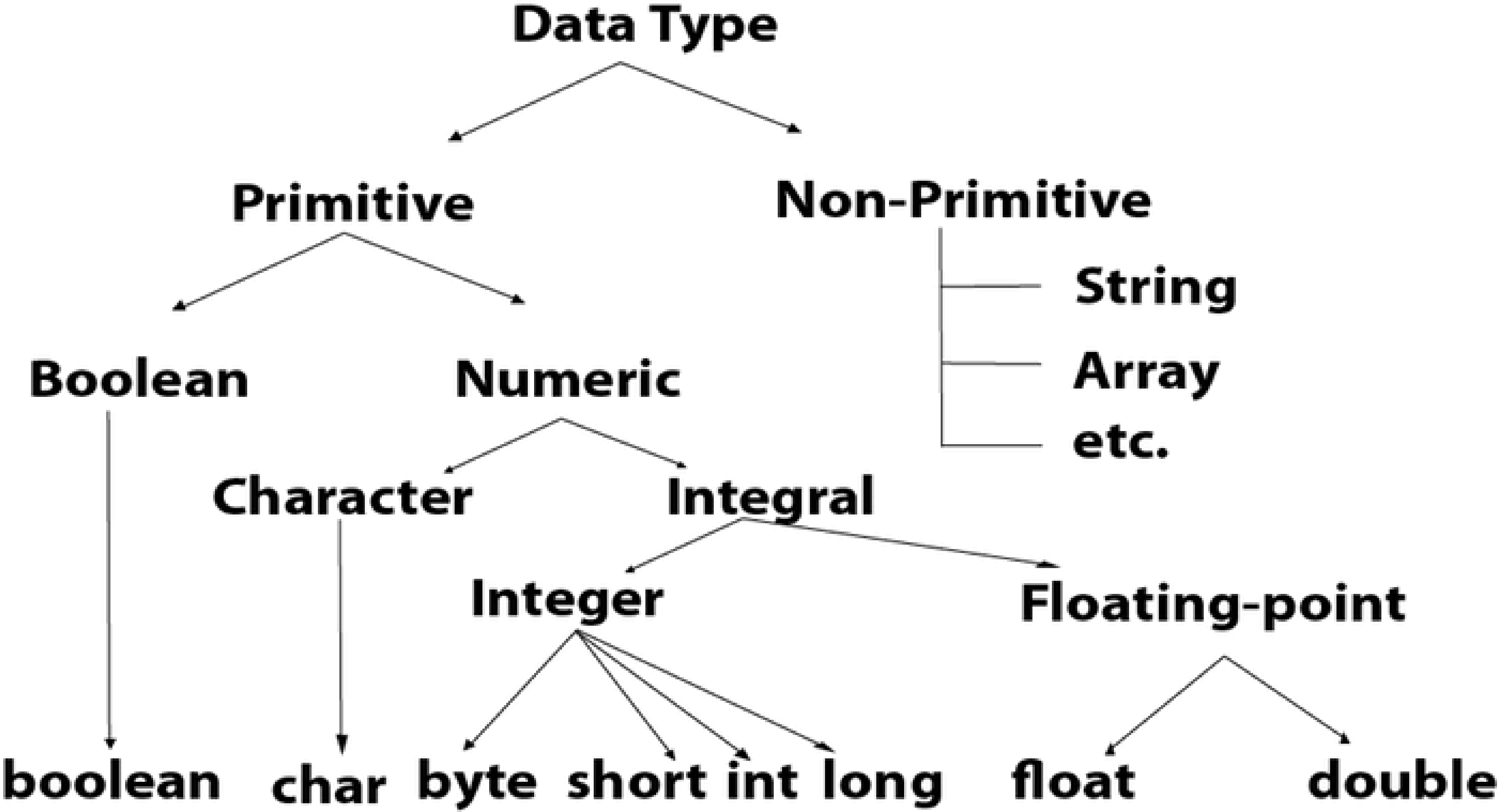
- dataType[] arr; (or)
- dataType []arr; (or)
- dataType arr[];

```
class TestArray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=1;//initialization
a[1]=2;
a[2]=3;
a[3]=4;
a[4]=5;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array

System.out.println(a[i]);
}}
```

Data Types in Java

- Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:
- **Primitive data types**: The primitive data types include boolean, char, byte, short, int, long, float and double.
- **Non-primitive data types**: The non-primitive data types include Classes, Interfaces, and Arrays.



Data Type

Primitive

Non-Primitive

String

Array

etc.

Boolean

Numeric

Character

Integral

Integer

Floating-point

boolean

char

byte

short

int

long

float

double

Operators in Java

- Unary Operator i++,i--,++a
- Arithmetic Operator * / % + -
- Shift Operator << >> >>>
- Relational Operator < > <= >= == !=
- Bitwise Operator & ^ |
- Logical Operator && , ||
- Ternary Operator ? :
- Assignment Operator = += -= *= /= %= &= ^= |= <<= >>= >>>=

Lexical Issues

Programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.

- **Whitespace –** A space, tab, or newline
- **Identifiers –** Class names, method names, and variable names e.g.
  - Valid : `a4    $test this_is_okAvgTemp _count`
  - Invalid : `2count    high-temp    Not/ok`

- **Literals –** A constant value e.g.
  - `100 5 05  0x5     98.6 1000.0 1.0E+03`
    `'X'      "This is a test"`
- **Comments**
  - Single-line   // -----------------
  - Multiline   /* --

    ------------------------------------- */
  - Documentation comments
    - produce an HTML file that documents your program

      /** ----------------------------------- */

- **Keywords**
  - 50 keywords currently defined

abstract
assert
boolean
break
byte
case
catch
char
class
const

continue
default
do
double
else
enum
extends
final
finally
float

for
goto
if
implements
import
instanceof
int
interface
long
native

new
package
private
protected
public
return
short
static
strictfp
super

switch
synchronized
this
throw
throws
transient
try
void
volatile
while

Data Types

- **The Primitive Types**
  - Integers : **byte, short, int, and long**
  - Floating-point numbers : **float and double**
  - Characters : **char**
  - Boolean : **boolean**

Operators in java

- **Arithmetic Operators**

  + – * / %

  – cannot be useed on **boolean** types, but can be used them on **char** types

  – can be applied to floating-point types as well as integer types

  – a % b is the same as:

  ```
  a - ((int)(a / b) * b)
  ```

  – This means that a%b is the floating point equivalent of the remainder after division.

Operators in java

- **Increment and Decrement**
  - increases its operand by one

    ++ (*postfix* and *prefix*)
  - Decreases its operand by one

    -- (*postfix* and *prefix*)

Operators in java

- **The Bitwise Operators**
  - act upon the individual bits of their operands

~ & | ^ >> >>> << &= |=
^= >>= >>>= <<=

  - Operator new in Java

>>> (Shift right zero fill)

>>>= (Shift right zero fill assignment)

Operators in java

- **Relational Operators**
  - determine the relationship between two operands
  - The outcome of these operations is a **boolean** value

== != > < >= <=

Operators in java

- **Boolean Logical Operators**
  - combine two **boolean** values to form a resultant **boolean** value

<table>
<tr><th></th><th></th><th></th></tr>
<tr><td>&</td><td>Logical AND</td><td>(evaluate the right-hand operand)</td></tr>
<tr><td>|</td><td>Logical OR</td><td>(evaluate the right-hand operand)</td></tr>
<tr><td>^</td><td>Logical XOR (exclusive OR)</td><td></td></tr>
<tr><td>||</td><td>Short-circuit OR</td><td>(Java will not bother to evaluate the right-hand operand)</td></tr>
<tr><td>&&</td><td>Short-circuit AND</td><td>(Java will not bother to evaluate the right-hand operand)</td></tr>
<tr><td>!</td><td>Logical unary NOT</td><td></td></tr>
<tr><td>&=</td><td>AND assignment</td><td></td></tr>
<tr><td>|=</td><td>OR assignment</td><td></td></tr>
<tr><td>^=</td><td>XOR assignment</td><td></td></tr>
<tr><td>==</td><td>Equal to</td><td></td></tr>
<tr><td>!=</td><td>Not equal to</td><td></td></tr>
</table>

Operators in java

- **The Assignment Operator**

  var = expression;

- **The ? Operator**

  expression1 ? expression2 : expression3;

Arrays

- An *array* is a group of same type of variables that are referred to by a common name.
- Create an array

```
            type var-name[ ];
```

- Allocate memory for that

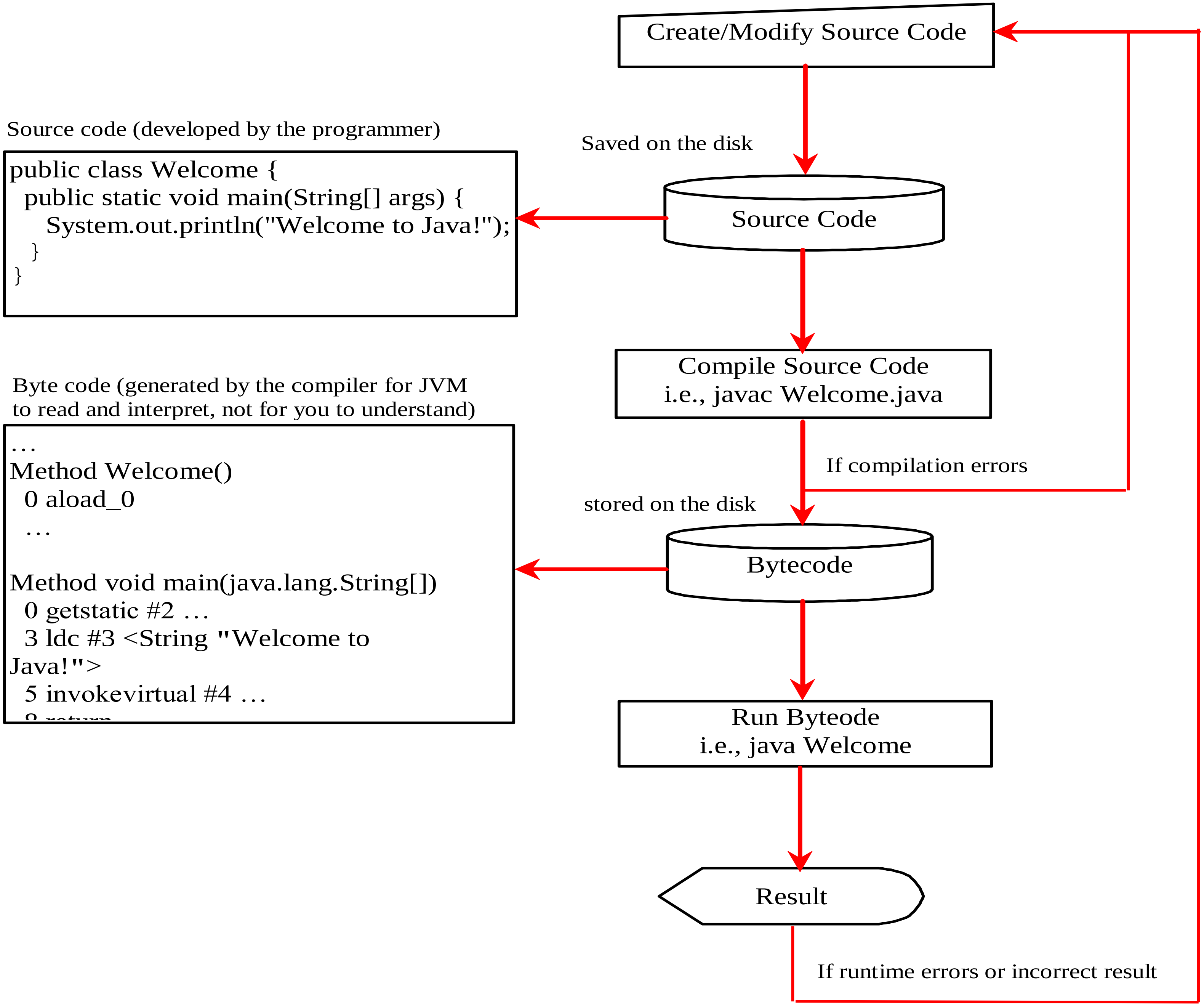
```
            array-var = new type [size];
```

OR

```
type var-name[ ] =  new type [size];
```

- Using the Enhanced for Loop
  - process arrays and collections
  - Syntax:

```
for (type identifier : array)
{
      statements...
}
```



Create/Modify Source Code

Saved on the disk

Source Code

Compile Source Code
i.e., javac Welcome.java

If compilation errors

stored on the disk

Bytecode

Run Byteode
i.e., java Welcome

Result

If runtime errors or incorrect result

Source code (developed by the programmer)

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

Byte code (generated by the compiler for JVM
to read and interpret, not for you to understand)

```
…
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
```

References:

Herbert Schildt, Complete Reference Java (8th edition), Tata McGraw Hill Edition