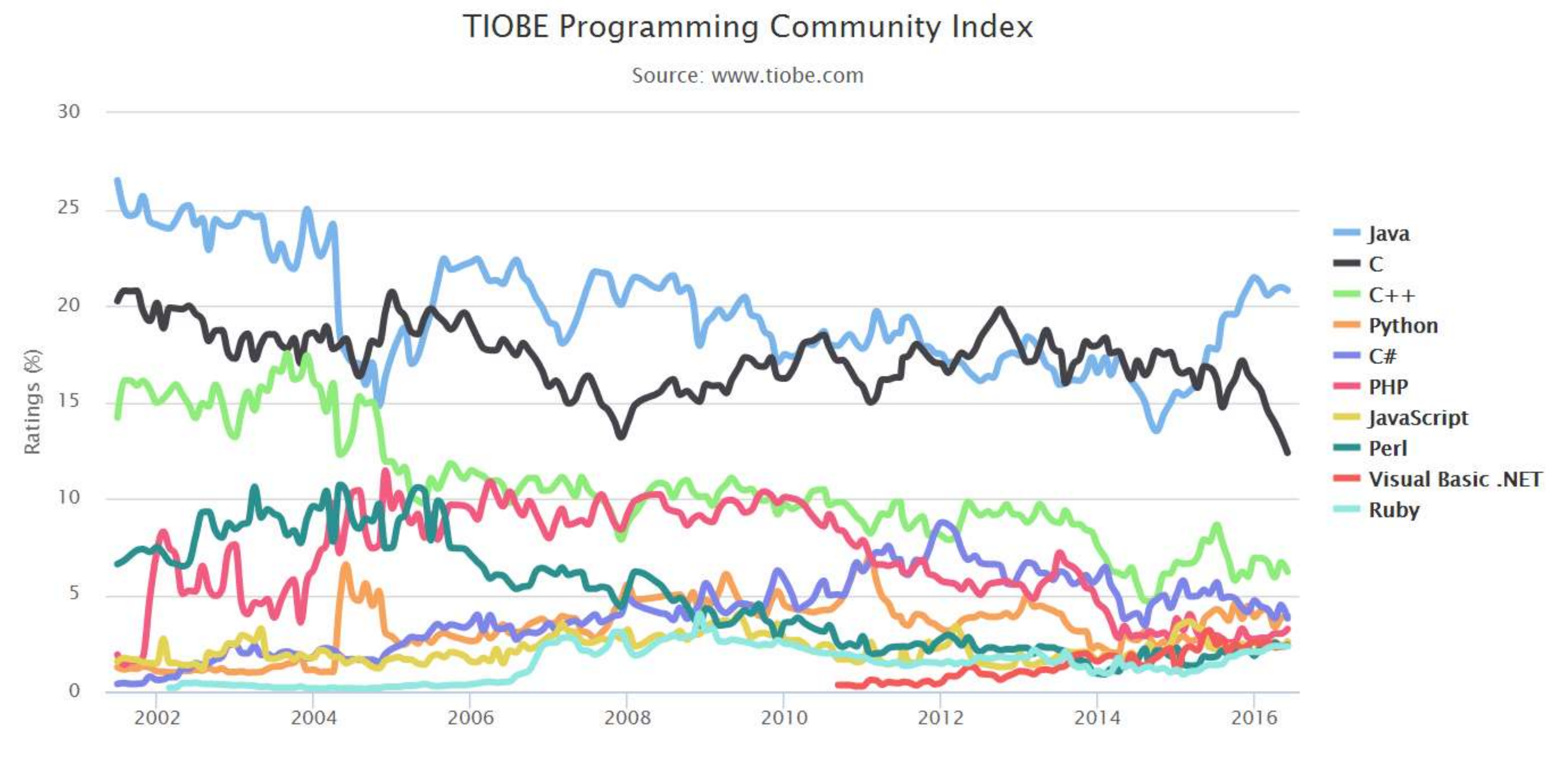
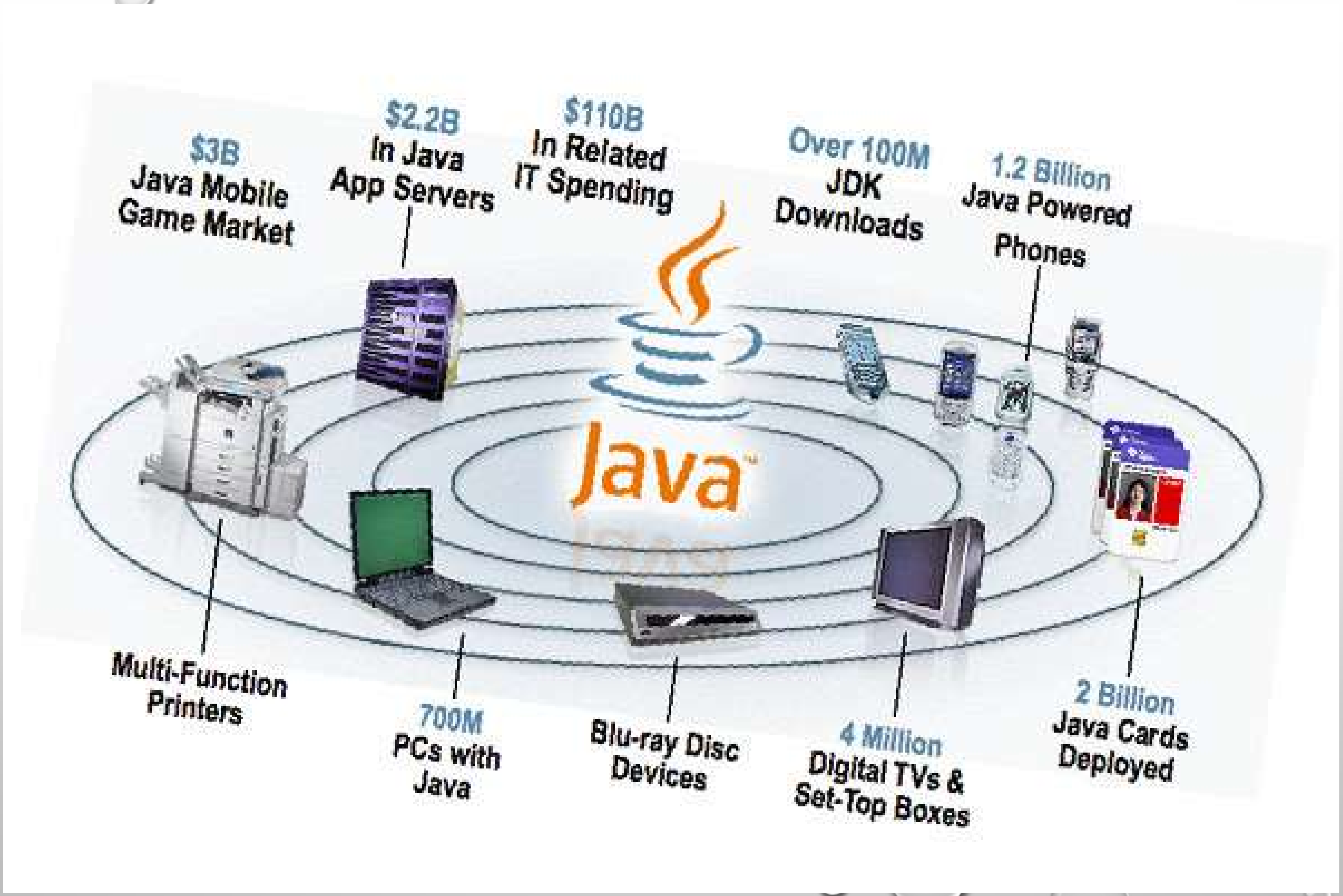
ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ JAVA

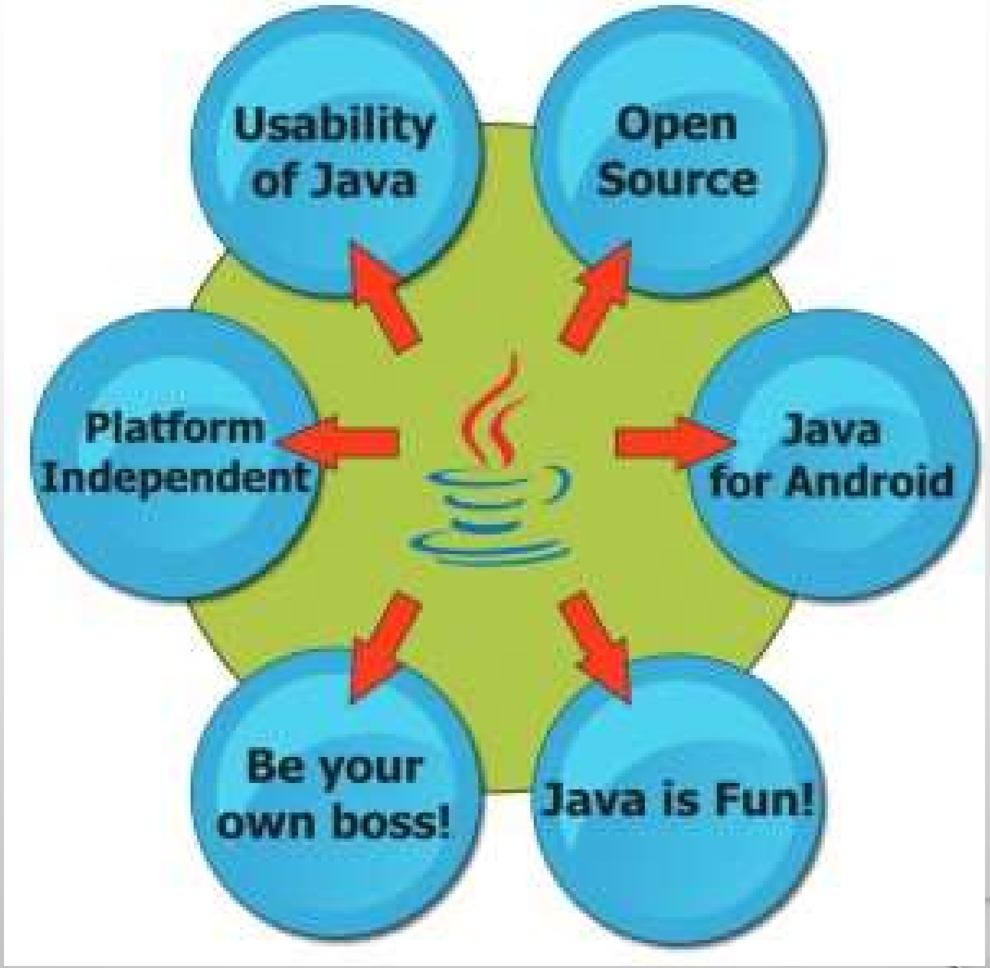
DR. EFTHIMIOS ALEPIS

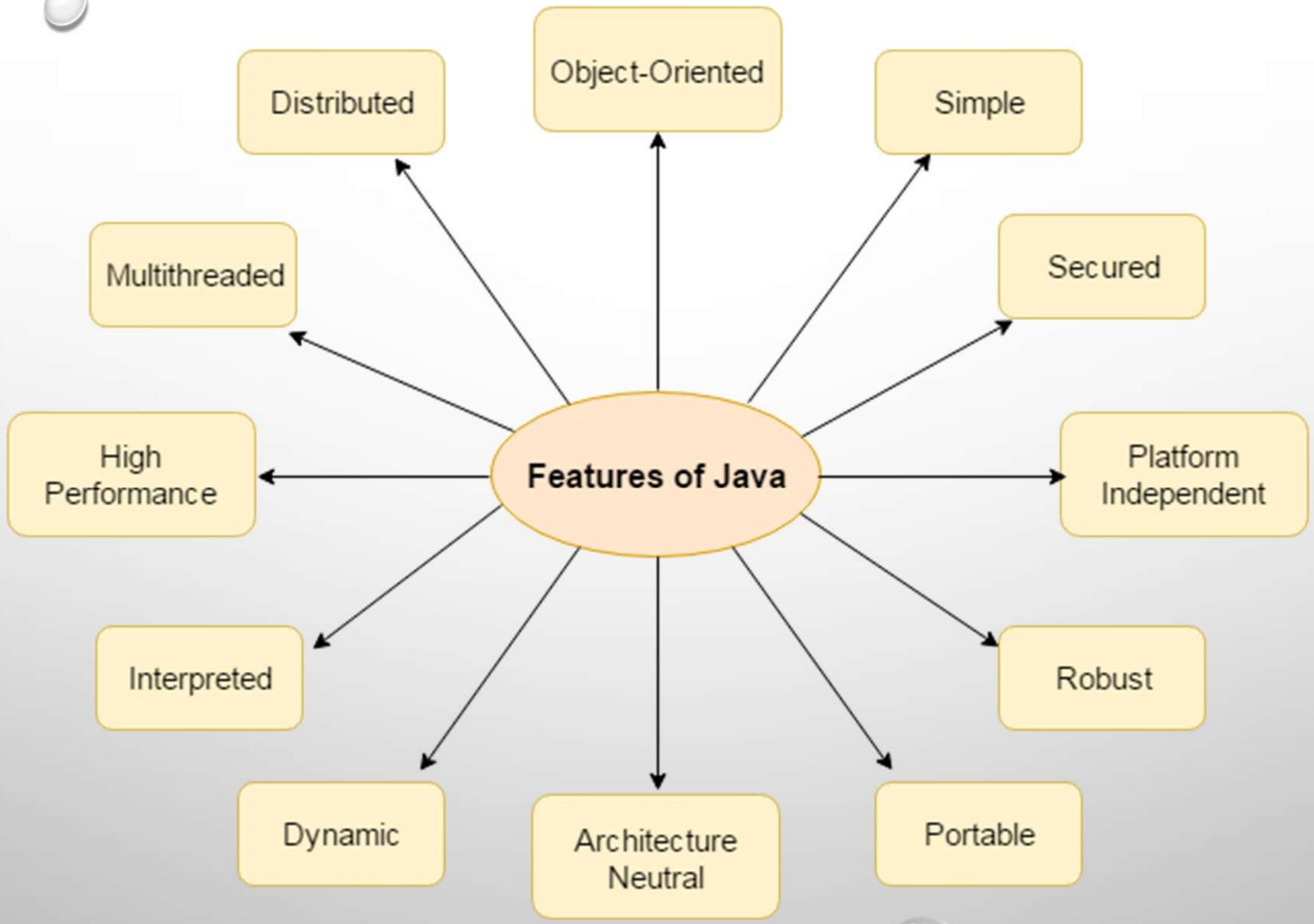
JAVA STATISTICS




$3B
Java Mobile
Game Market
$2.2B
In Java
App Servers
$110B
In Related
IT Spending
Over 100M
JDK
Downloads
1.2 Billion
Java Powered
Phones
Java™
Multi-Function
Printers
700M
PCs with
Java
Blu-ray Disc
Devices
4 Million
Digital TVs &
Set-Top Boxes
2 Billion
Java Cards
Deployed

INTRODUCTION (WHY JAVA?)





Features of Java

Object-Oriented

Simple

Secured

Platform Independent

Robust

Portable

Architecture Neutral

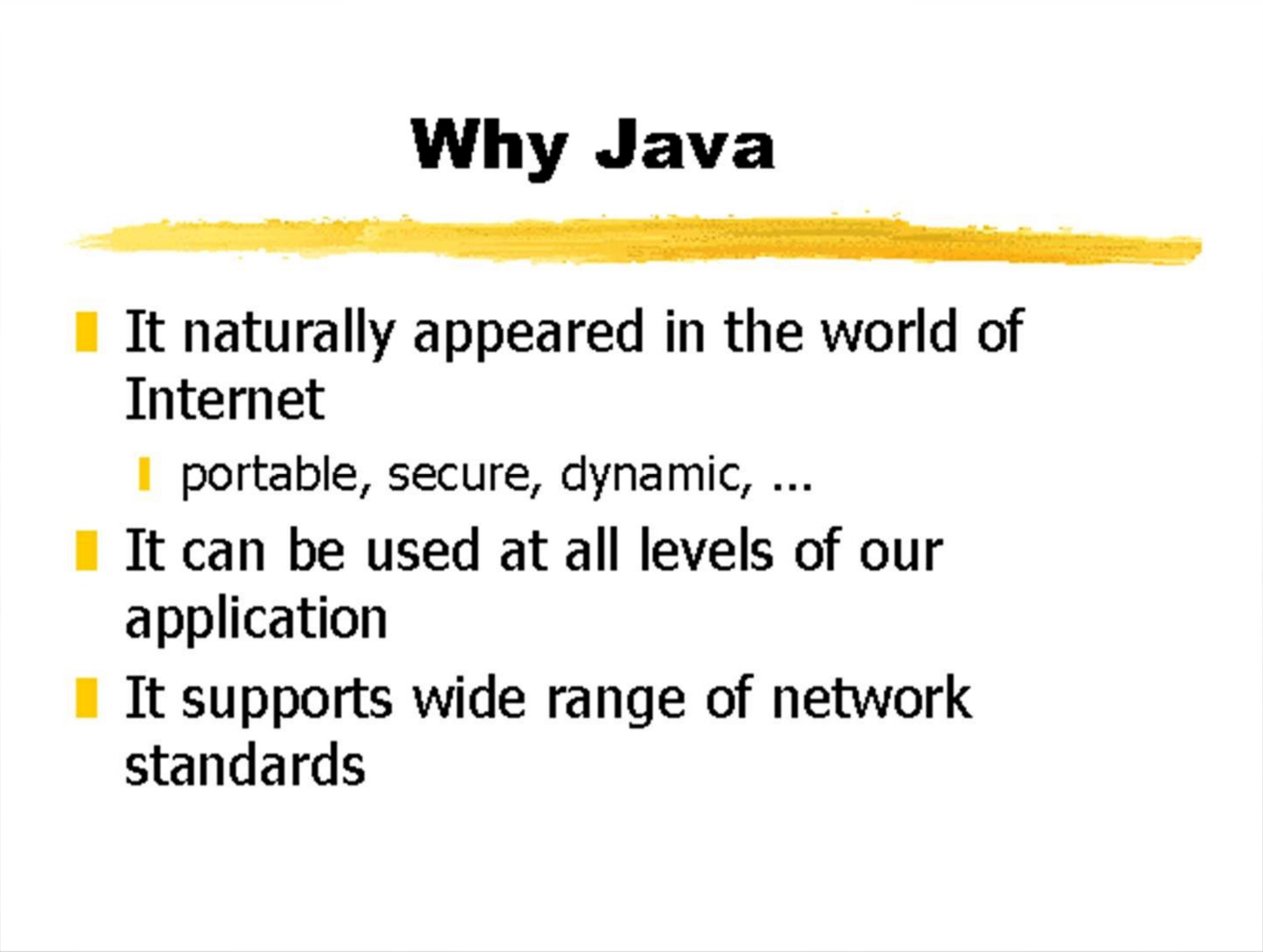
Dynamic

Interpreted

High Performance

Multithreaded

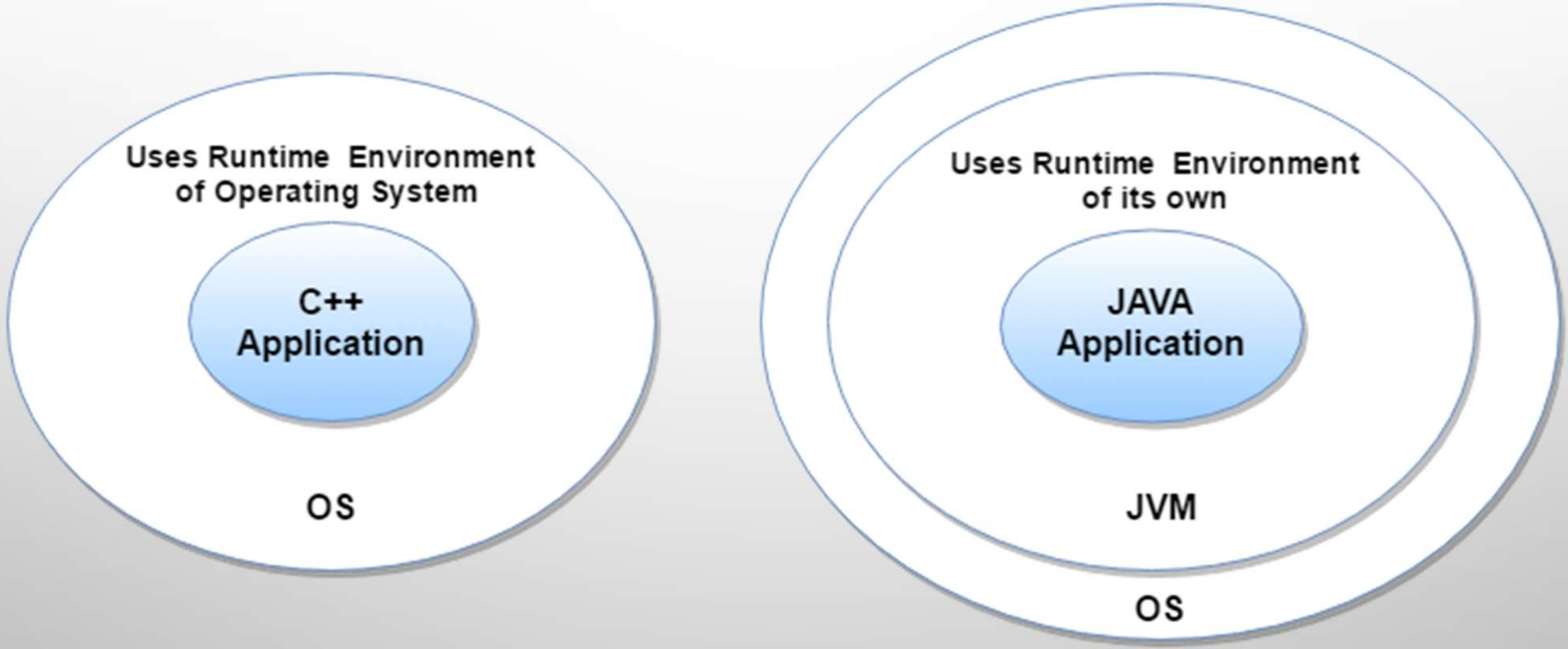
Distributed



Why Java

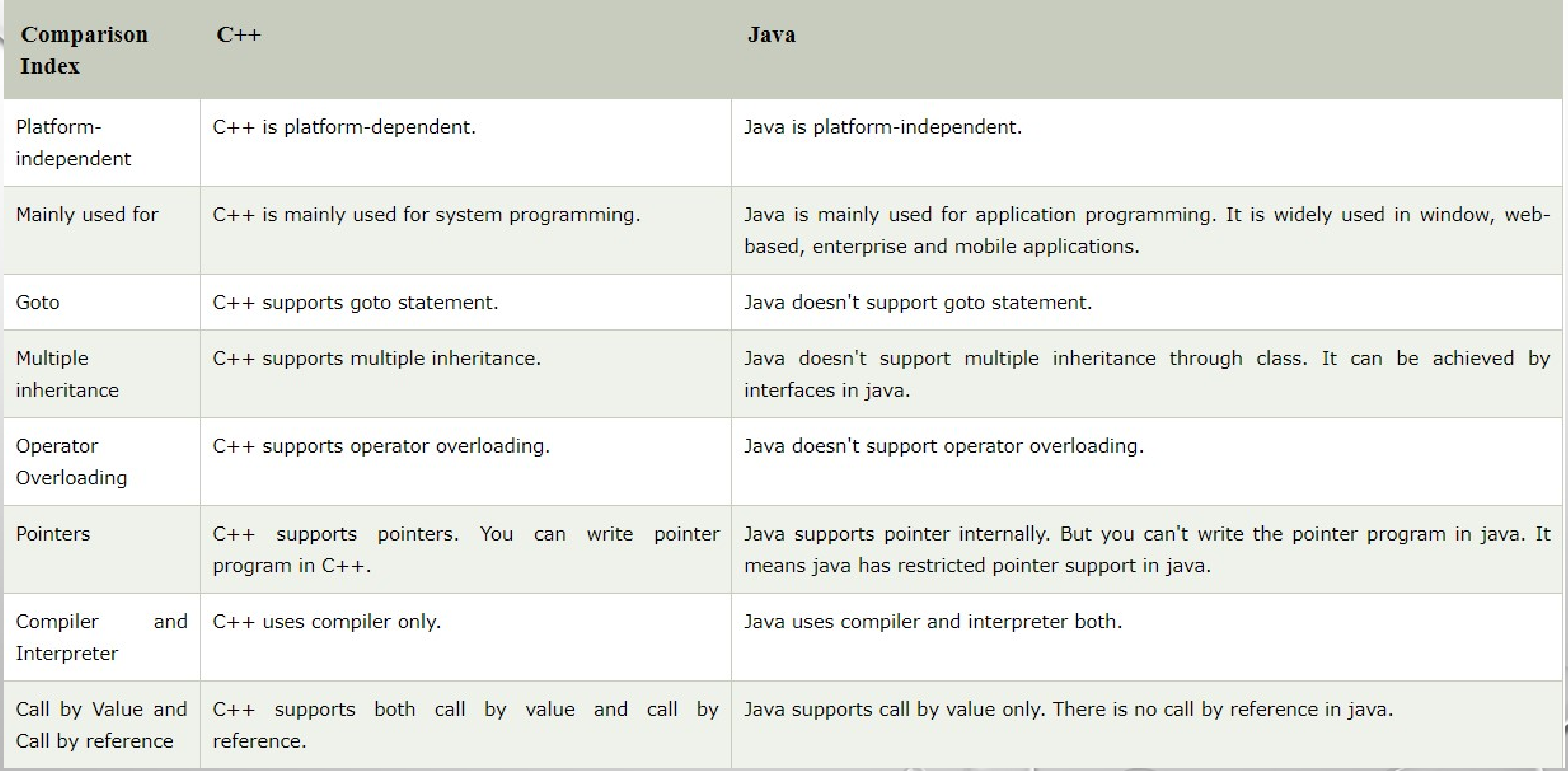
- It naturally appeared in the world of Internet
  - portable, secure, dynamic, ...
- It can be used at all levels of our application
- It supports wide range of network standards

IS QUITE SECURED

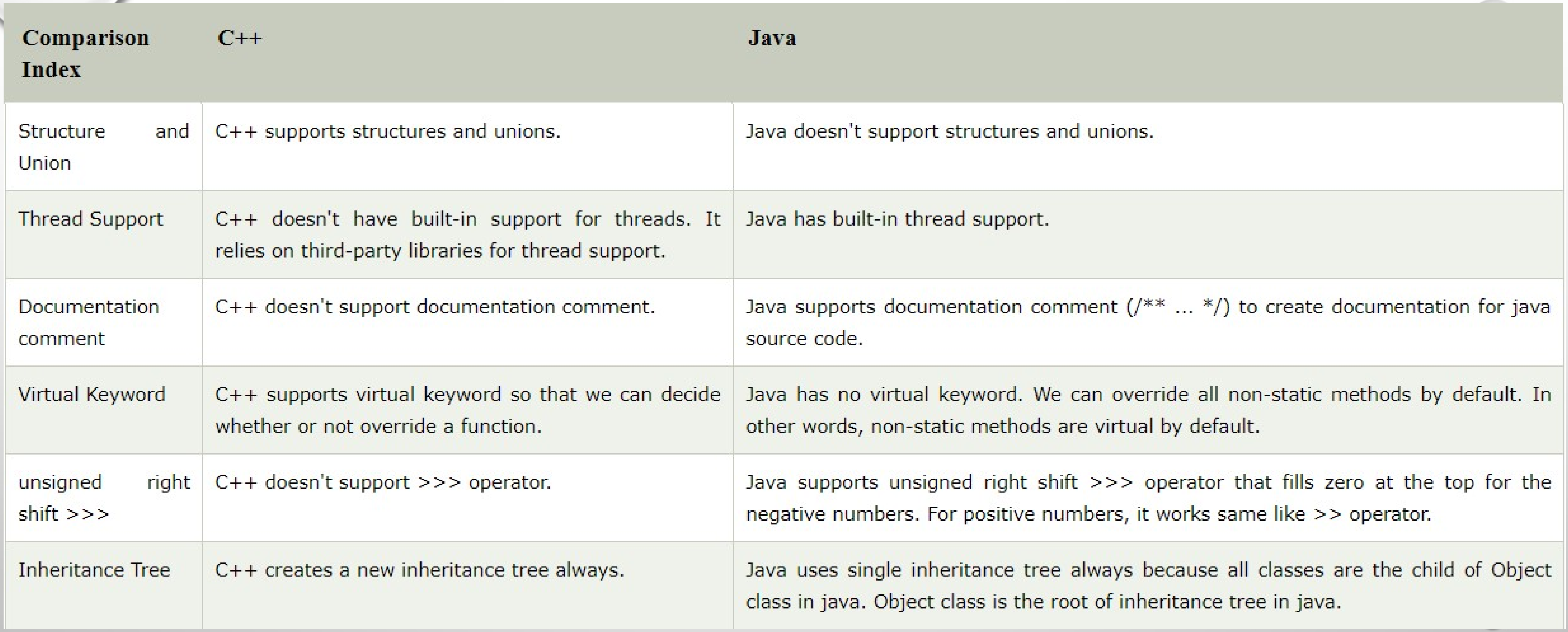

Uses Runtime Environment of Operating System
C++ Application
OS
Uses Runtime Environment of its own
JAVA Application
JVM
OS

IS QUITE SECURED

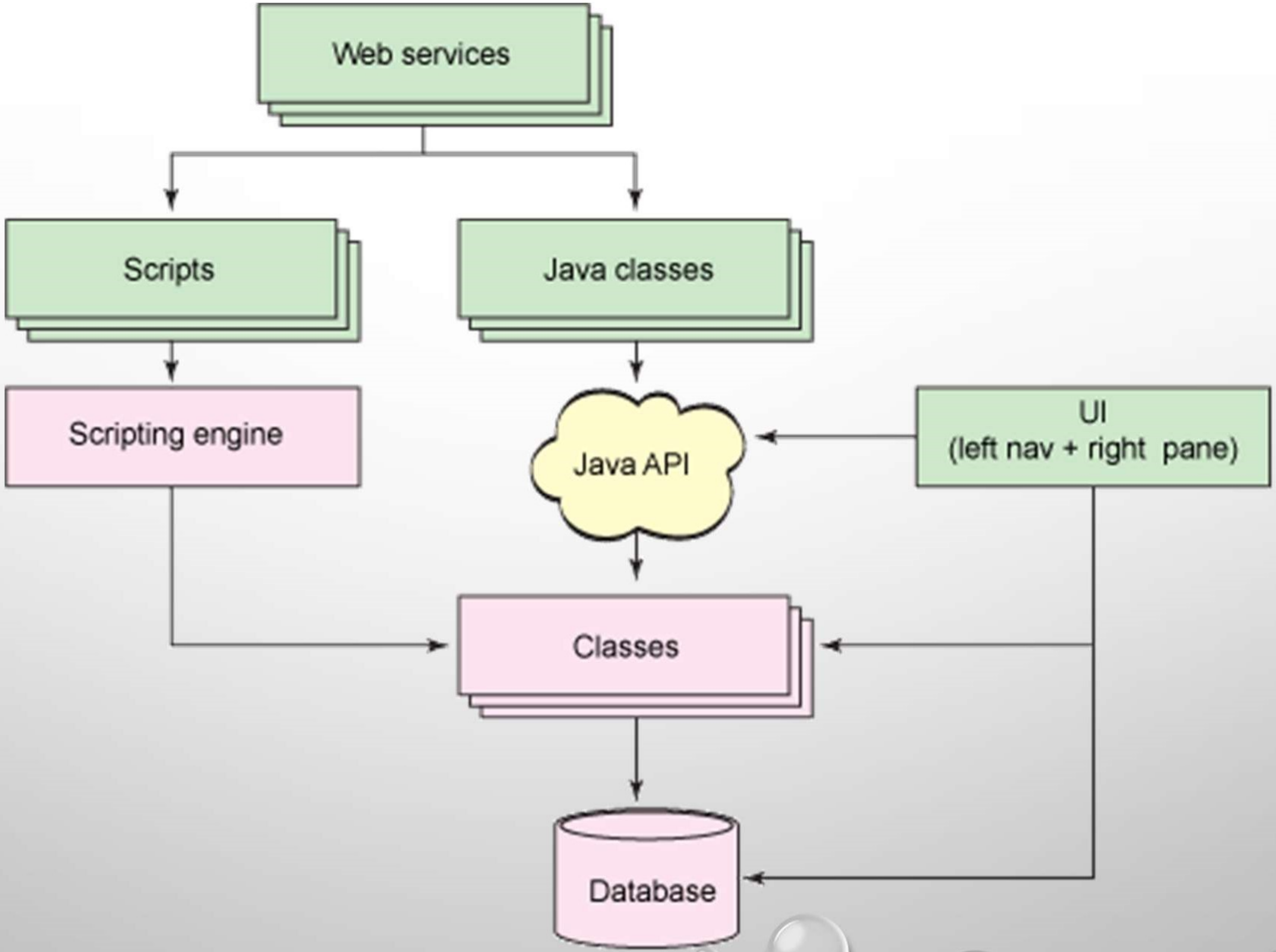
- NO EXPLICIT POINTER
- JAVA PROGRAMS RUN INSIDE VIRTUAL MACHINE SANDBOX
- CLASSLOADER: ADDS SECURITY BY SEPARATING THE PACKAGE FOR THE CLASSES OF THE LOCAL FILE SYSTEM FROM THOSE THAT ARE IMPORTED FROM NETWORK SOURCES.
- BYTECODE VERIFIER: CHECKS THE CODE FRAGMENTS FOR ILLEGAL CODE THAT CAN VIOLATE ACCESS RIGHT TO OBJECTS.
- SECURITY MANAGER: DETERMINES WHAT RESOURCES A CLASS CAN ACCESS SUCH AS READING AND WRITING TO THE LOCAL DISK.



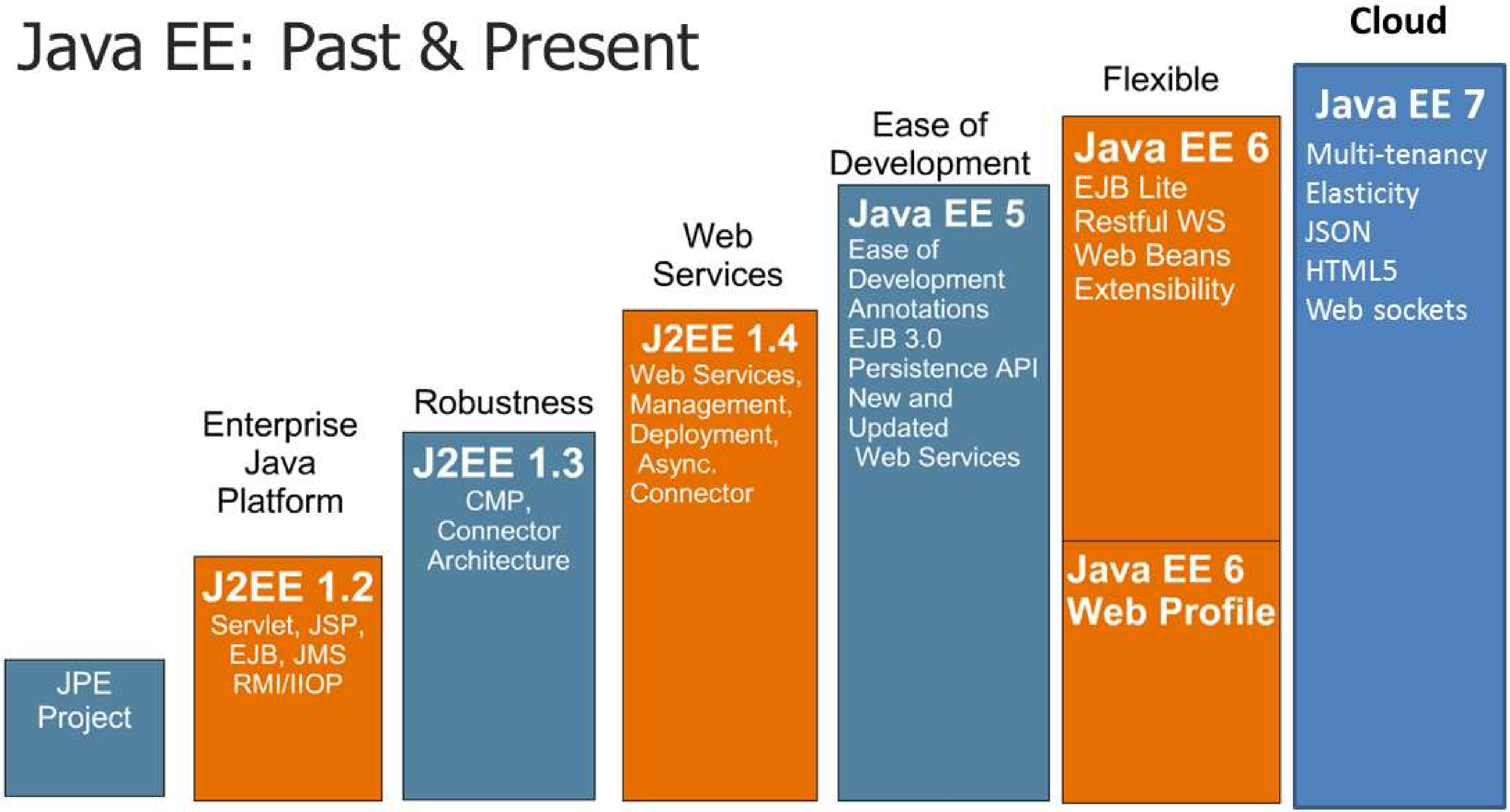
Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Goto	C++ supports goto statement.	Java doesn't support goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only.	Java uses compiler and interpreter both.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.



Comparison Index	C++	Java
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses single inheritance tree always because all classes are the child of Object class in java. Object class is the root of inheritance tree in java.


Web services
Scripts
Java classes
Scripting engine
Java API
UI
(left nav + right pane)
Classes
Database

Java EE: Past & Present



JPE Project

Enterprise Java Platform

J2EE 1.2
Servlet, JSP, EJB, JMS RMI/IIOP

Robustness

J2EE 1.3
CMP, Connector Architecture

Web Services

J2EE 1.4
Web Services, Management, Deployment, Async. Connector

Ease of Development

Java EE 5
Ease of Development
Annotations
EJB 3.0
Persistence API
New and Updated Web Services

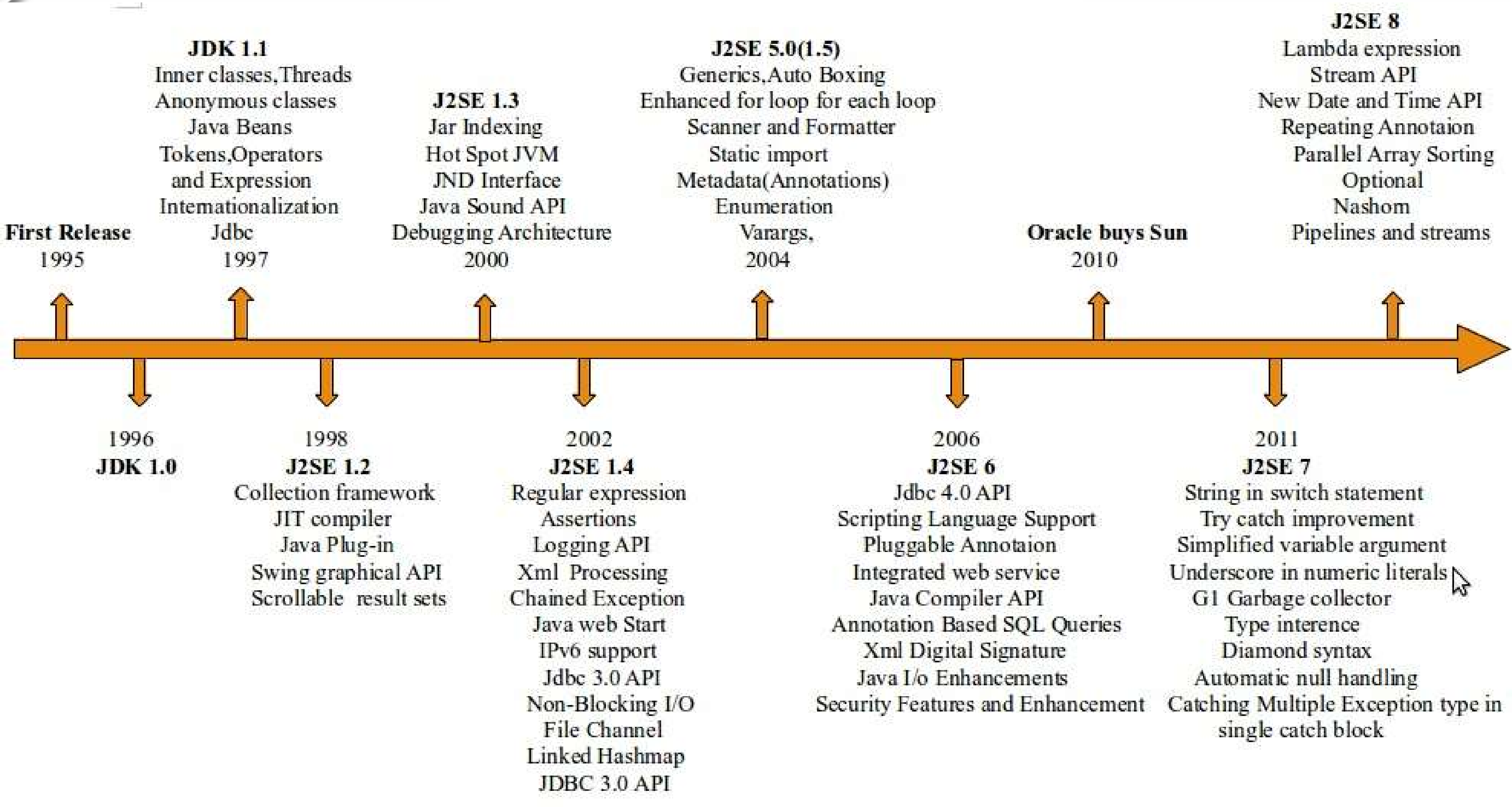
Flexible

Java EE 6
EJB Lite
Restful WS
Web Beans
Extensibility

Java EE 6 Web Profile

Cloud

Java EE 7
Multi-tenancy
Elasticity
JSON
HTML5
Web sockets



First Release
1995

1996
JDK 1.0

JDK 1.1
Inner classes,Threads
Anonymous classes
Java Beans
Tokens,Operators
and Expression
Internationalization
Jdbc
1997

1998
J2SE 1.2
Collection framework
JIT compiler
Java Plug-in
Swing graphical API
Scrollable result sets

J2SE 1.3
Jar Indexing
Hot Spot JVM
JND Interface
Java Sound API
Debugging Architecture
2000

2002
J2SE 1.4
Regular expression
Assertions
Logging API
Xml Processing
Chained Exception
Java web Start
IPv6 support
Jdbc 3.0 API
Non-Blocking I/O
File Channel
Linked Hashmap
JDBC 3.0 API

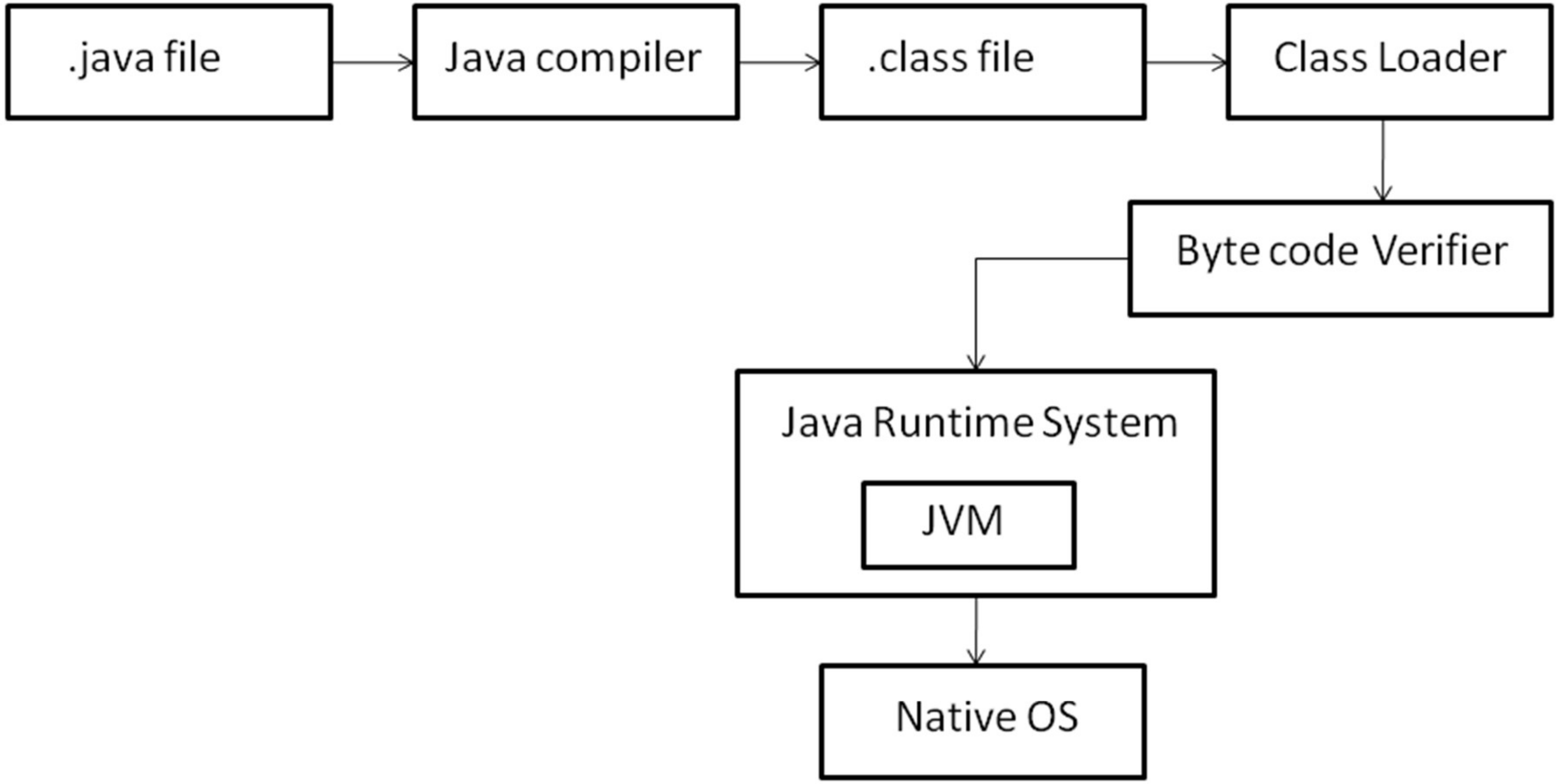
J2SE 5.0(1.5)
Generics,Auto Boxing
Enhanced for loop for each loop
Scanner and Formatter
Static import
Metadata(Annotations)
Enumeration
Varargs,
2004

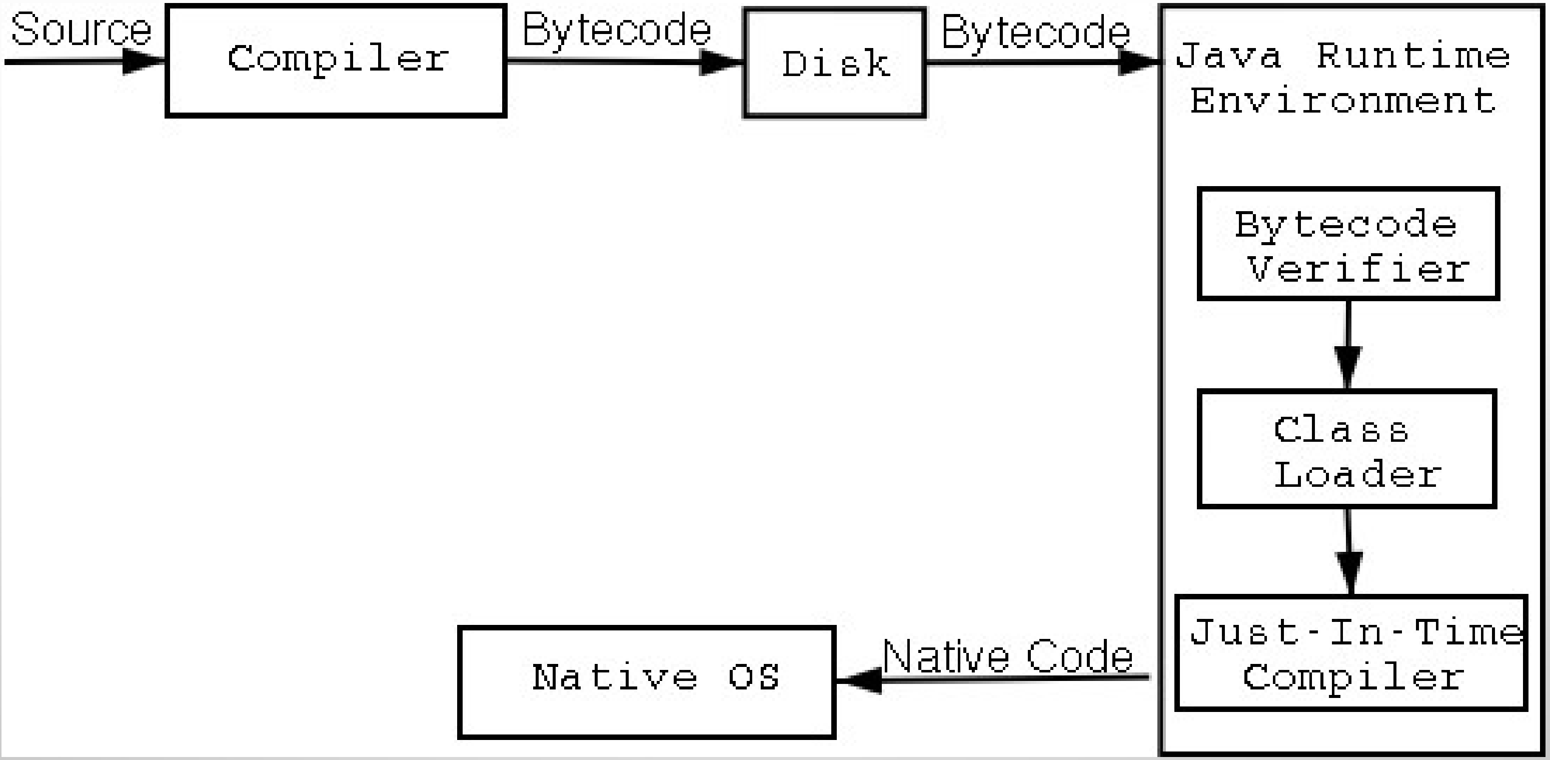
2006
J2SE 6
Jdbc 4.0 API
Scripting Language Support
Pluggable Annotaion
Integrated web service
Java Compiler API
Annotation Based SQL Queries
Xml Digital Signature
Java I/o Enhancements
Security Features and Enhancement

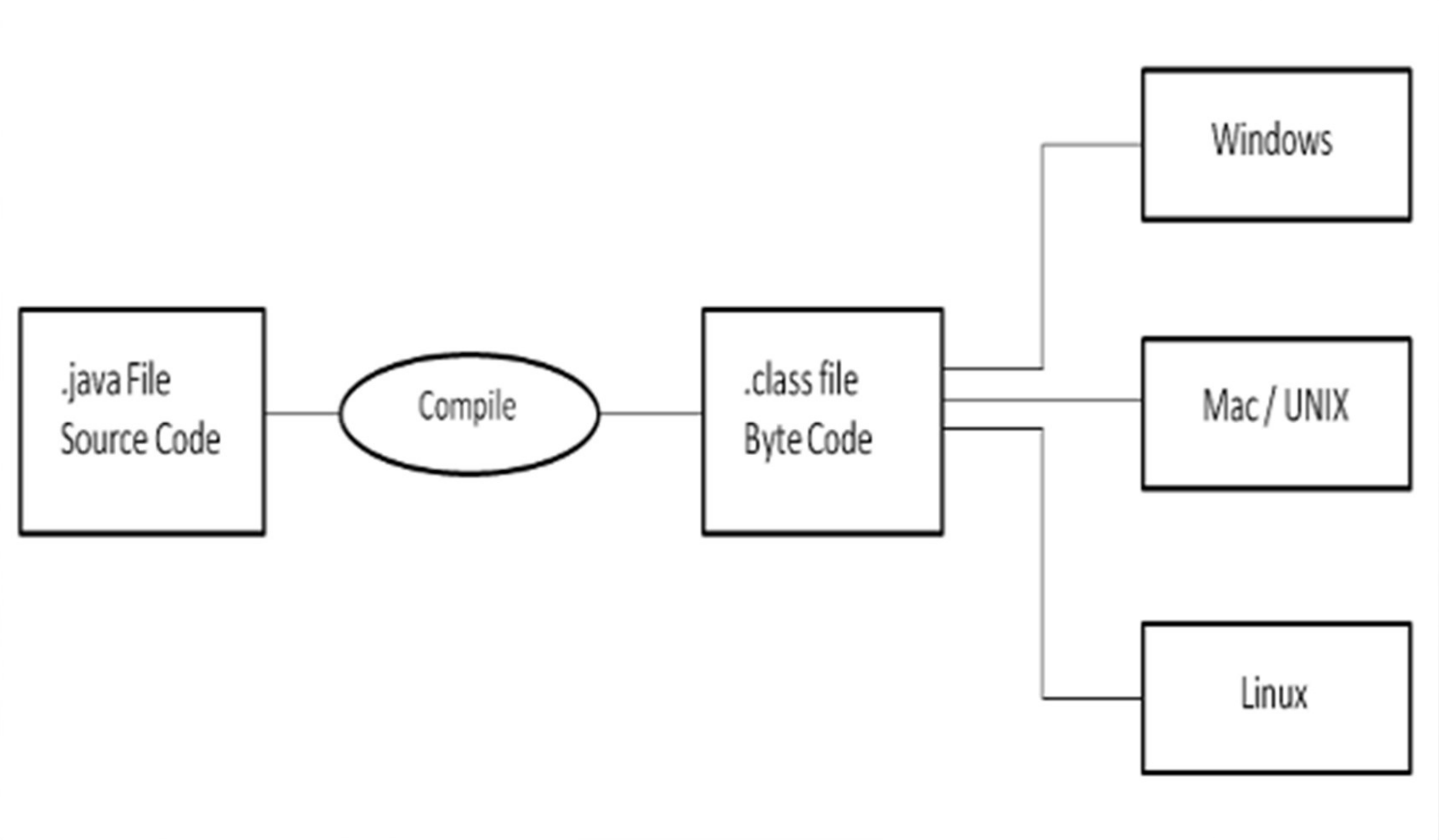
Oracle buys Sun
2010

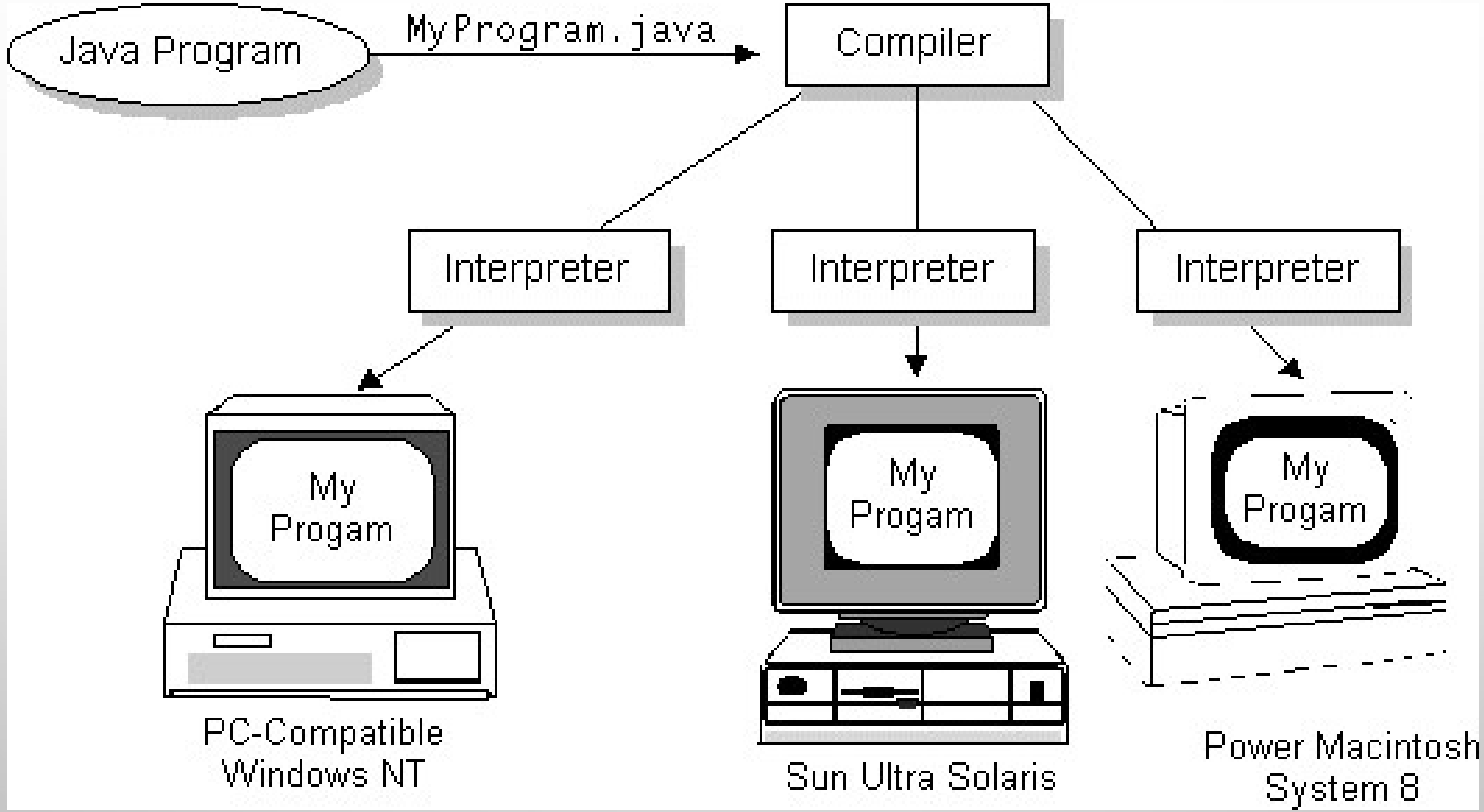
2011
J2SE 7
String in switch statement
Try catch improvement
Simplified variable argument
Underscore in numeric literals
G1 Garbage collector
Type inference
Diamond syntax
Automatic null handling
Catching Multiple Exception type in single catch block

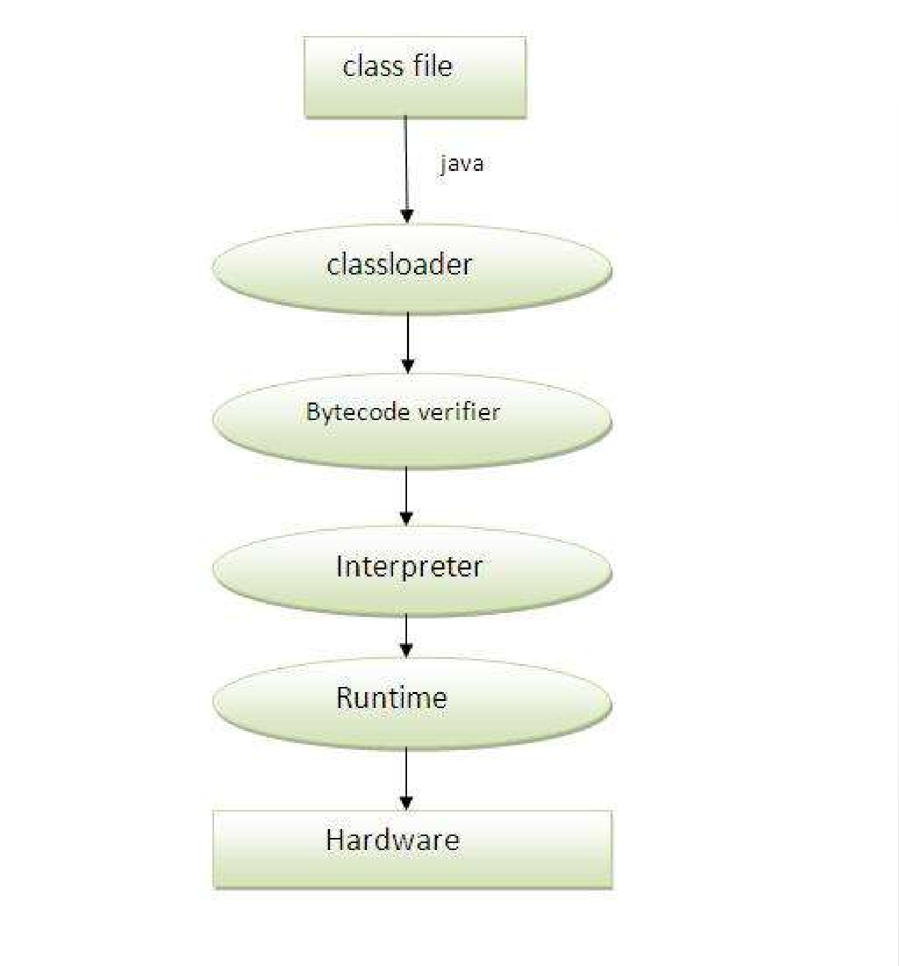
J2SE 8
Lambda expression
Stream API
New Date and Time API
Repeating Annotaion
Parallel Array Sorting
Optional
Nashorn
Pipelines and streams

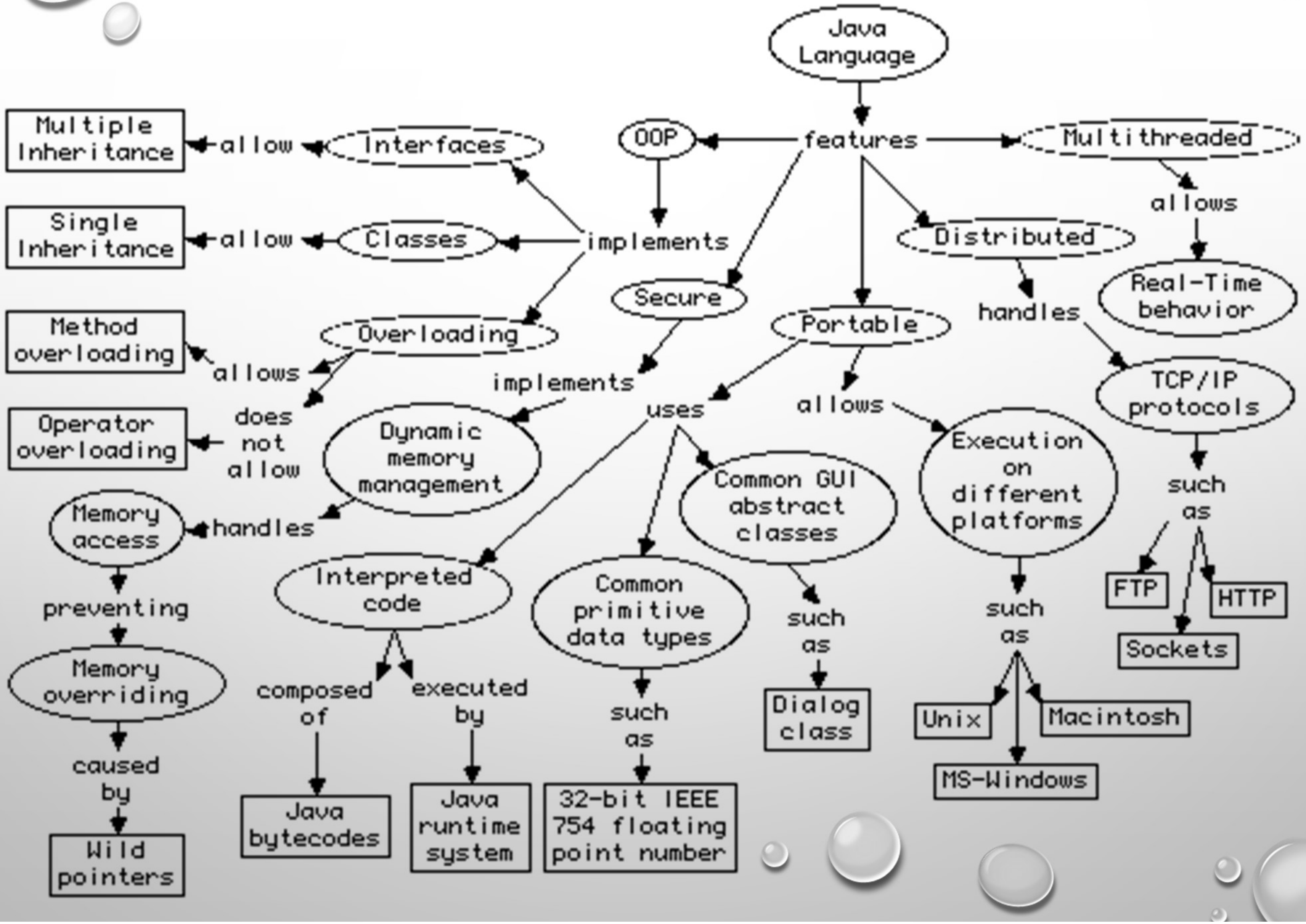

.java file
Java compiler
.class file
Class Loader
Byte code Verifier
Java Runtime System
JVM
Native OS


Source
Compiler
Bytecode
Disk
Bytecode
Java Runtime Environment
Bytecode Verifier
Class Loader
Just-In-Time Compiler
Native Code
Native OS


.java File
Source Code
Compile
.class file
Byte Code
Windows
Mac / UNIX
Linux


Java Program
MyProgram.java
Compiler
Interpreter
Interpreter
Interpreter
My Progam
My Progam
My Progam
PC-Compatible Windows NT
Sun Ultra Solaris
Power Macintosh System 8


class file
java
classloader
Bytecode verifier
Interpreter
Runtime
Hardware



Java Language

features

OOP

Multithreaded

Multiple Inheritance

allow

Interfaces

implements

allows

Single Inheritance

allow

Classes

Distributed

Real-Time behavior

Secure

Portable

handles

Method overloading

allows

Overloading

implements

TCP/IP protocols

does not allow

Operator overloading

uses

allows

Dynamic memory management

Common GUI abstract classes

Execution on different platforms

such as

Memory access

handles

FTP

HTTP

Interpreted code

Common primitive data types

such as

such as

Sockets

preventing

Memory overriding

composed of

executed by

such as

Dialog class

Unix

Macintosh

caused by

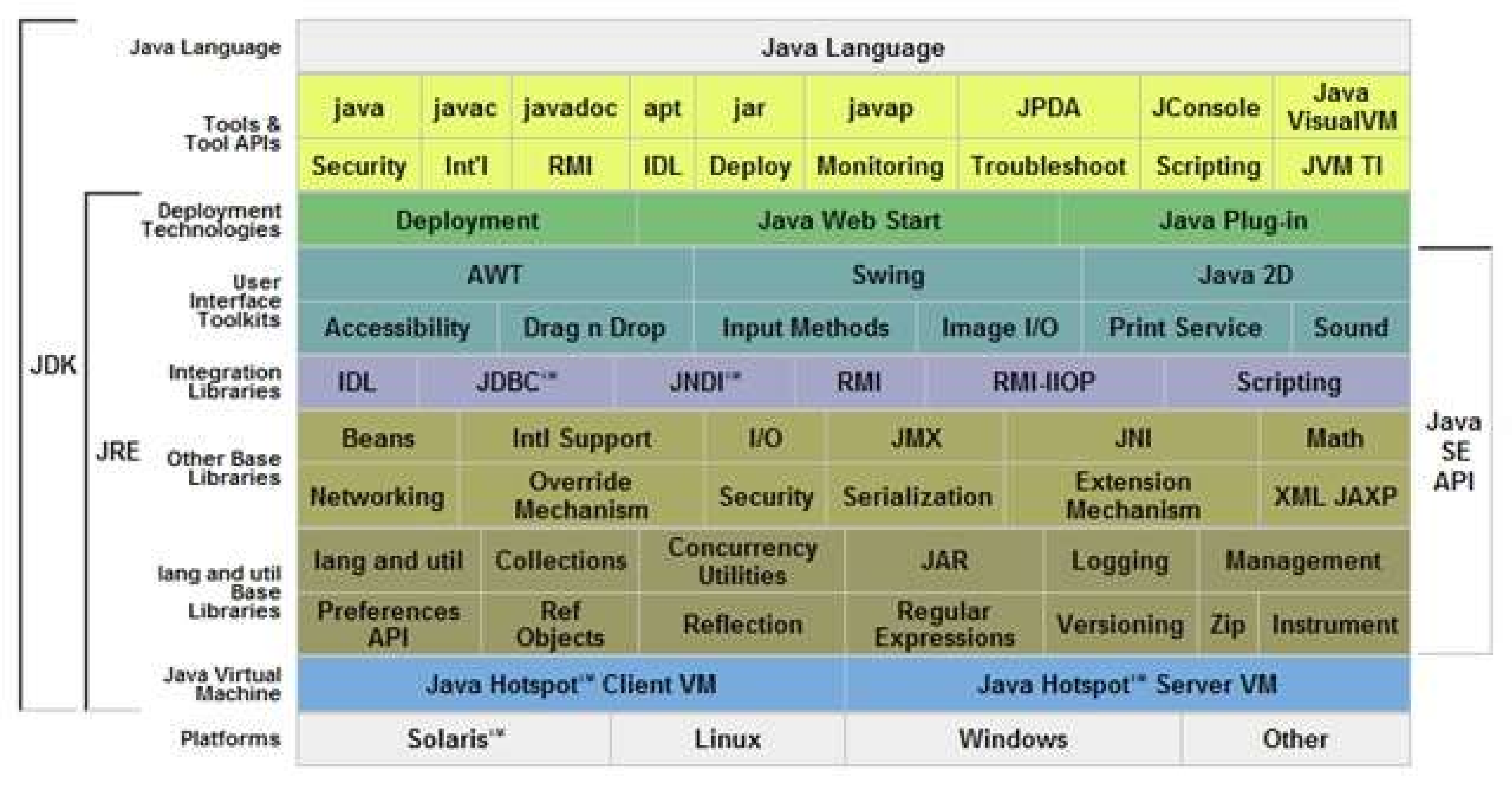
MS-Windows

Java bytecodes

Java runtime system

32-bit IEEE 754 floating point number

Wild pointers



JDK

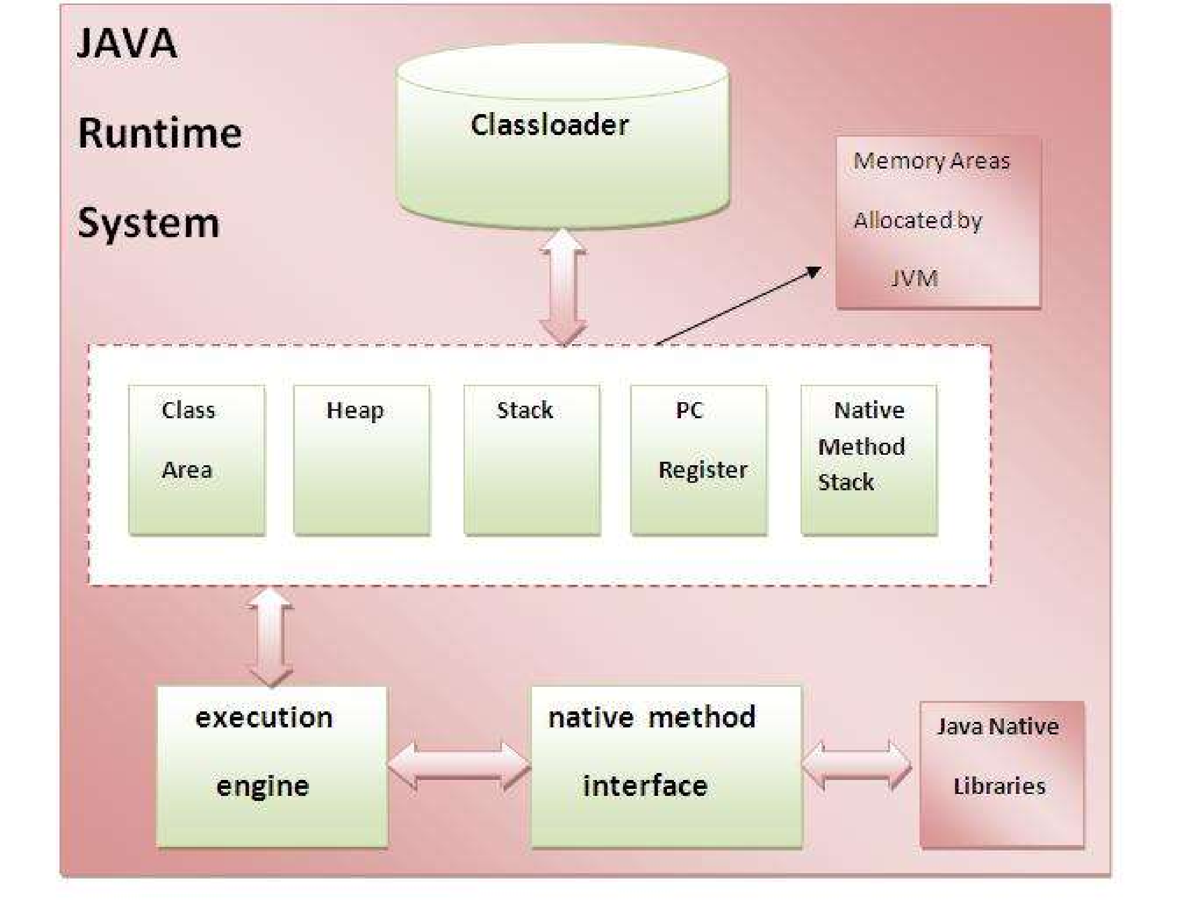
JRE

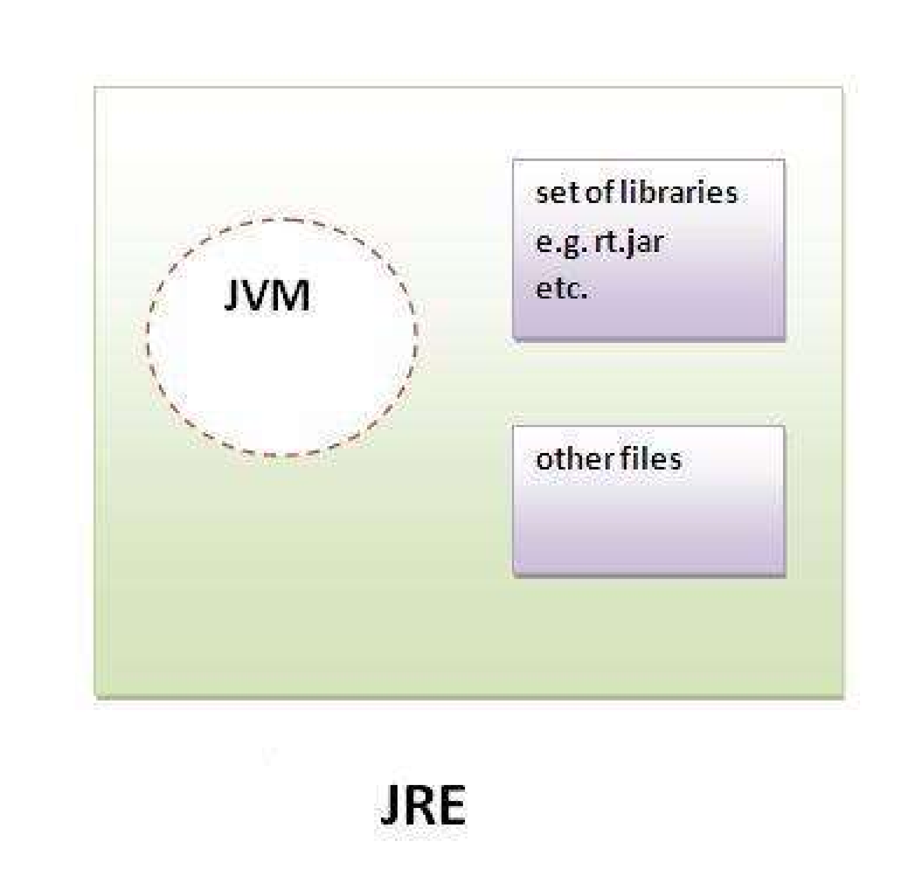
Java SE API

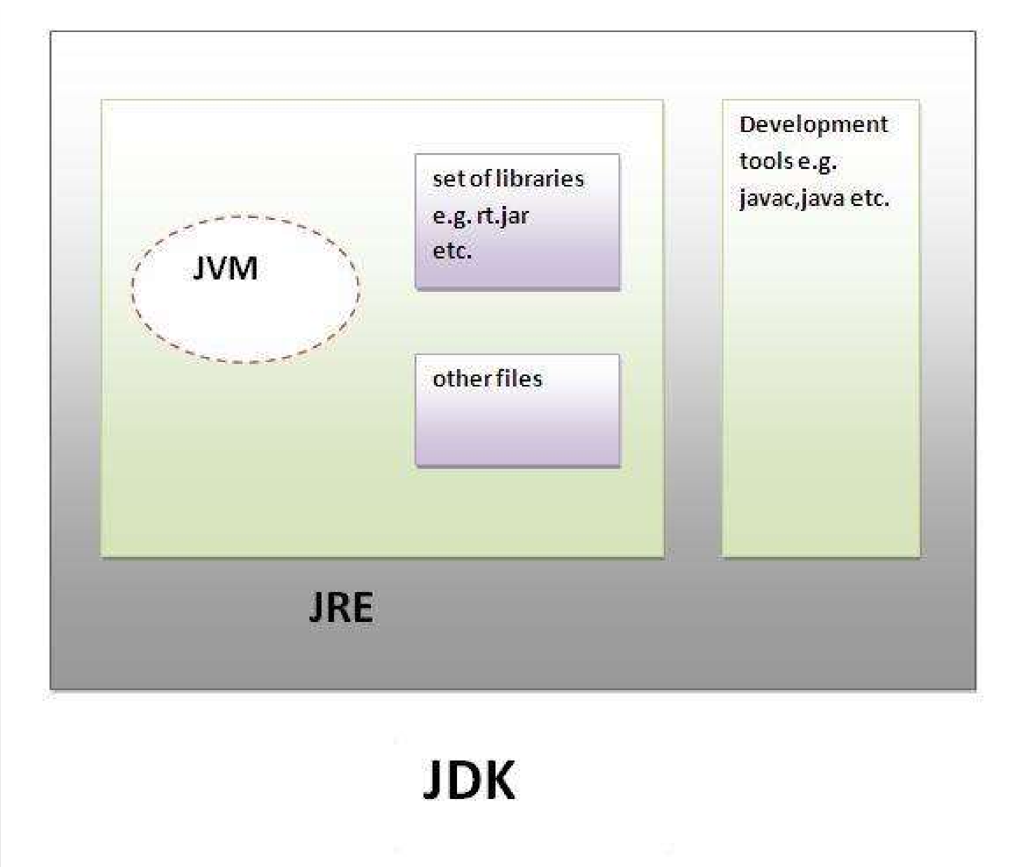
Java Language	Java Language								
Tools & Tool APIs	java	javac	javadoc	apt	jar	javap	JPDA	JConsole	Java VisualVM
	Security	Int'l	RMI	IDL	Deploy	Monitoring	Troubleshoot	Scripting	JVM TI
Deployment Technologies	Deployment	Java Web Start	Java Plug-in						
User Interface Toolkits	AWT	Swing	Java 2D						
	Accessibility	Drag n Drop	Input Methods	Image I/O	Print Service	Sound			
Integration Libraries	IDL	JDBC™	JNDI™	RMI	RMI-IIOP	Scripting			
Other Base Libraries	Beans	Intl Support	I/O	JMX	JNI	Math			
	Networking	Override Mechanism	Security	Serialization	Extension Mechanism	XML JAXP			
lang and util Base Libraries	lang and util	Collections	Concurrency Utilities	JAR	Logging	Management			
	Preferences API	Ref Objects	Reflection	Regular Expressions	Versioning	Zip	Instrument		
Java Virtual Machine	Java Hotspot™ Client VM	Java Hotspot™ Server VM							
Platforms	Solaris™	Linux	Windows	Other					

JDK, JRE AND **JVM**

- A SPECIFICATION THAT PROVIDES RUNTIME ENVIRONMENT IN WHICH JAVA BYTECODE CAN BE EXECUTED
- JVMS ARE AVAILABLE FOR MANY HARDWARE AND SOFTWARE PLATFORMS
- JVMS ARE PLATFORM DEPENDENT BECAUSE CONFIGURATION OF EACH OS DIFFERS
- JVM MAIN TASKS:
  - LOAD CODE
  - VERIFY CODE
  - EXECUTE CODE
  - PROVIDE RUNTIME ENVIRONMENT


JAVA
Runtime
System
Classloader
Memory Areas
Allocated by
JVM
Class
Area
Heap
Stack
PC
Register
Native
Method
Stack
execution
engine
native method
interface
Java Native
Libraries


JVM
set of libraries
e.g. rt.jar
etc.
other files
JRE

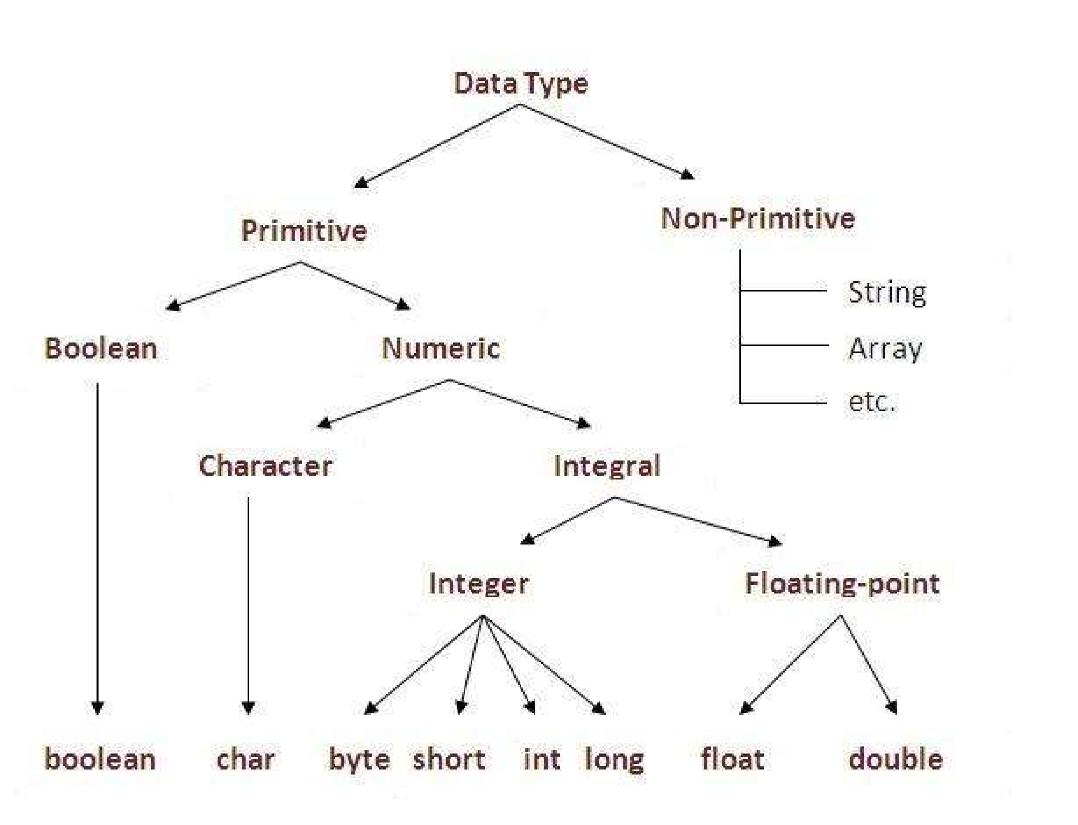

JVM
set of libraries e.g. rt.jar etc.
other files
Development tools e.g. javac,java etc.
JRE
JDK

TYPES OF JAVA VARIABLES

- LOCAL VARIABLES
- INSTANCE VARIABLES
- STATIC VARIABLES

```
class A{
int data=10;//instance variable
static int m=20;//static variable
void method(){
int n=30;//local variable
}
}//end of class
```

JAVA DATA TYPES

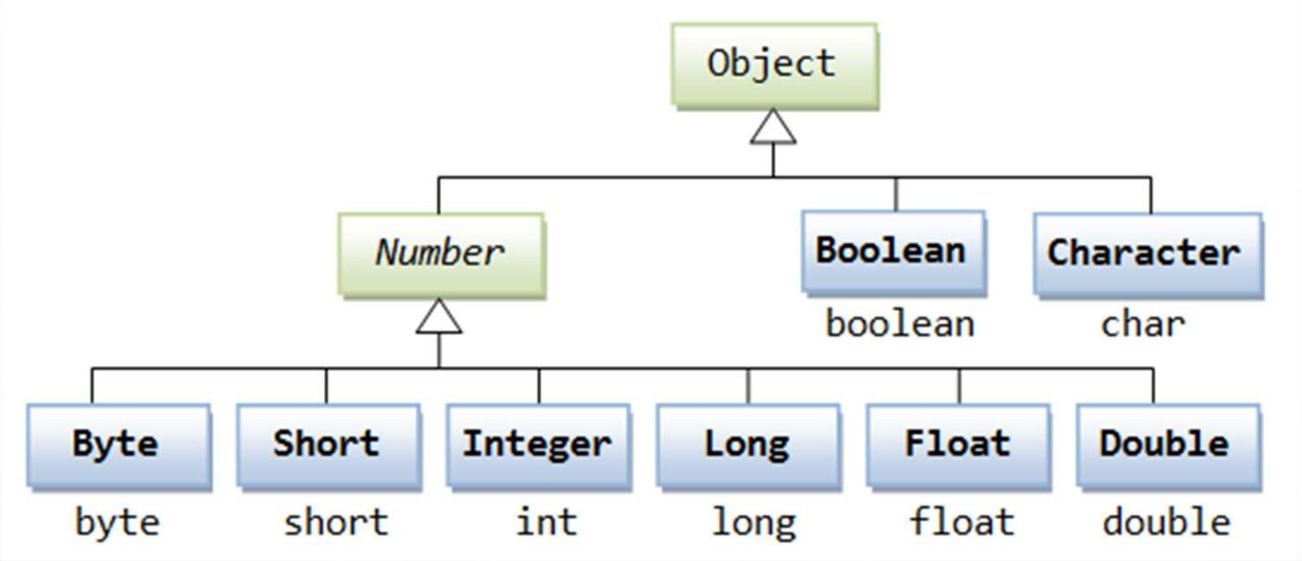


PRIMITIVE TYPES VS OBJECTS 1/2

- SOME SPECIFIC OBJECTS “WRAP” PRIMITIVE TYPES
- PRIMITIVE TYPES SERVE ONLY ONE PURPOSE, CONTAINING PURE, SIMPLE VALUES OF A KIND
- FOR A VARIABLE OF A PRIMITIVE TYPE, THE VALUE OF THE VARIABLE IS STORED IN THE MEMORY LOCATION ASSIGNED TO THE VARIABLE
- A VARIABLE OF A CLASS TYPE ONLY STORES THE MEMORY ADDRESS OF WHERE THE OBJECT IS LOCATED – NOT THE VALUES INSIDE THE OBJECT

PRIMITIVE TYPES VS OBJECTS 2/2

- A BIG DIFFERENCE BETWEEN A PRIMITIVE TYPE AND A CLASS TYPE IS THAT AN OBJECT OF A CLASS TYPE, LIKE AN OBJECT OF THE CLASS STRING, CAN BE OF ANY SIZE
- A COMMON MISTAKE IS USING A == B INSTEAD OF A.EQUALS(B). PEOPLE ARE USED TO DOING A == B WITH PRIMITIVES SO IT'S EASILY DONE WHEN YOU'RE USING THE OBJECT WRAPPERS


Object
Number
Boolean
boolean
Character
char
Byte
byte
Short
short
Integer
int
Long
long
Float
float
Double
double

METHODS OF OBJECT SUPERCLASS

- PROTECTED OBJECT CLONE() THROWS CLONENOTSUPPORTEDEXCEPTION

CREATES AND RETURNS A COPY OF THIS OBJECT.

- PUBLIC BOOLEAN EQUALS(OBJECT OBJ)

INDICATES WHETHER SOME OTHER OBJECT IS "EQUAL TO" THIS ONE.

- PROTECTED VOID FINALIZE() THROWS THROWABLE

CALLED BY THE GARBAGE COLLECTOR ON AN OBJECT WHEN GARBAGE COLLECTION DETERMINES THAT THERE ARE NO MORE REFERENCES TO THE OBJECT

- PUBLIC FINAL CLASS GETCLASS()

RETURNS THE RUNTIME CLASS OF AN OBJECT.

- PUBLIC INT HASHCODE()

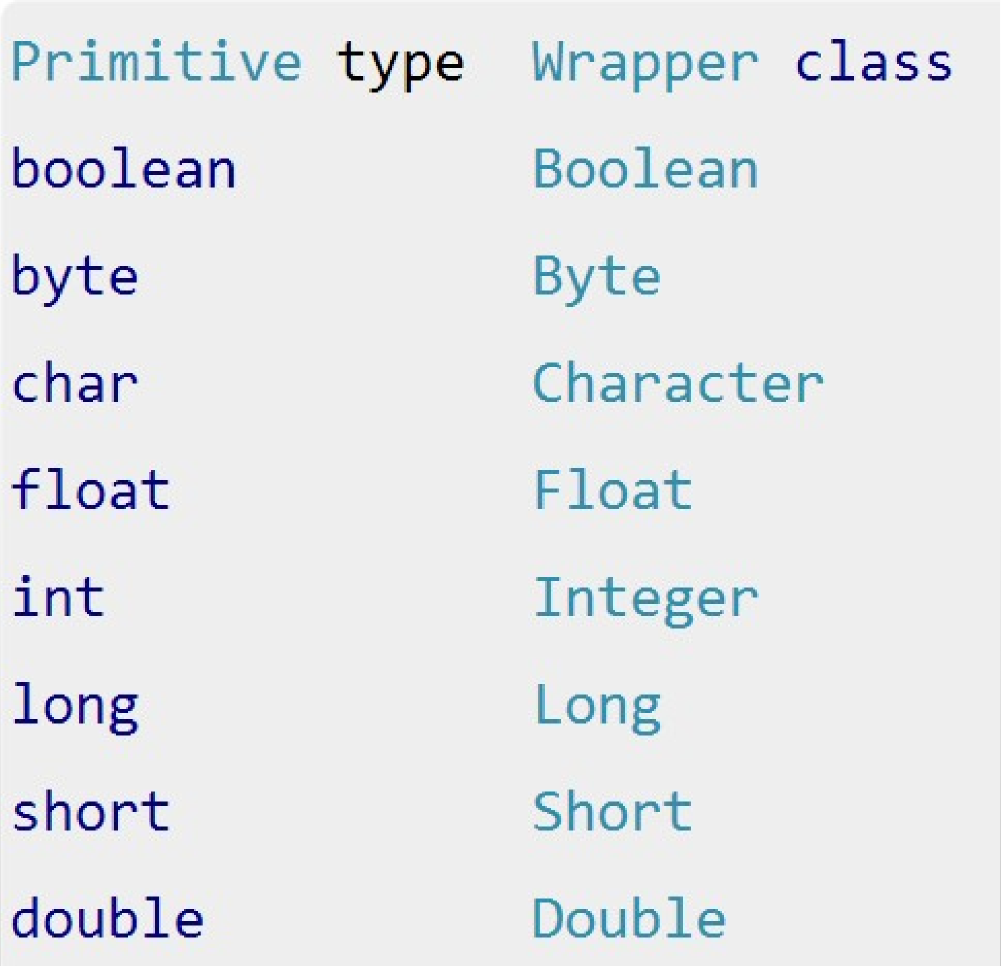
RETURNS A HASH CODE VALUE FOR THE OBJECT.

- PUBLIC STRING TOSTRING()

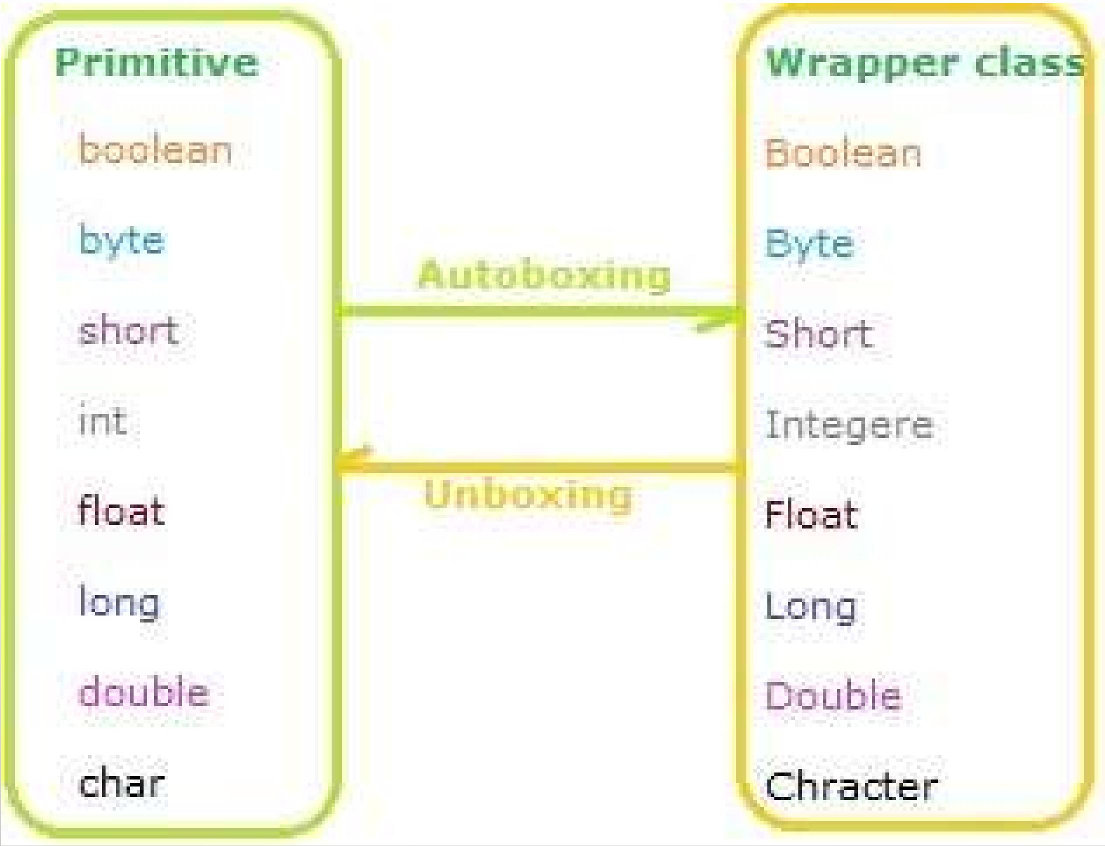
RETURNS A STRING REPRESENTATION OF THE OBJECT

AUTOBOXING AND UNBOXING

- AUTOBOXING AND UNBOXING IS INTRODUCED IN JAVA 1.5
- AUTOBOXING IS THE AUTOMATIC CONVERSION THAT THE JAVA COMPILER MAKES BETWEEN THE PRIMITIVE TYPES AND THEIR CORRESPONDING OBJECT WRAPPER CLASSES
- FOR EXAMPLE – CONVERSION OF INT TO INTEGER, LONG TO LONG, DOUBLE TO DOUBLE
- UNBOXING IS THE REVERSE PROCESS OF AUTOBOXING. AUTOMATICALLY CONVERTING AN OBJECT OF A WRAPPER CLASS TO ITS CORRESPONDING PRIMITIVE TYPE



Primitive type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

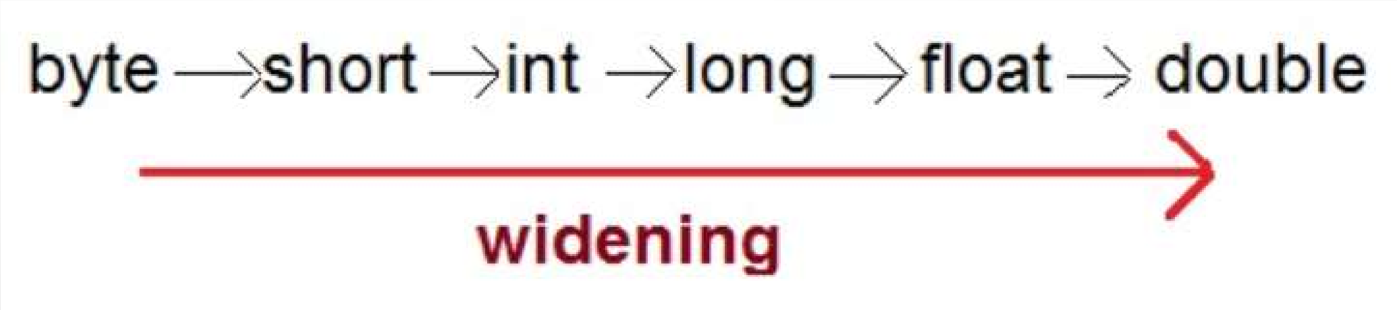


Primitive	Wrapper class
boolean	Boolean
byte	Byte
short	Short
int	Integere
float	Float
long	Long
double	Double
char	Chracter

DATA CONVERSION (CASTING)

- DATA CONVERSION (CASTING) CAN HAPPEN BETWEEN TWO PRIMITIVE TYPES
- THERE ARE TWO KINDS OF CASTING:
  - IMPLICIT: CASTING OPERATION IS NOT REQUIRED; THE MAGNITUDE OF THE NUMERIC VALUE IS ALWAYS PRESERVED. HOWEVER, PRECISION MAY BE LOST WHEN CONVERTING FROM INTEGER TO FLOATING POINT TYPES
  - EXPLICIT: CASTING OPERATION REQUIRED; THE MAGNITUDE OF THE NUMERIC VALUE MAY NOT BE PRESERVED

- WIDENING CASTING(IMPLICIT)



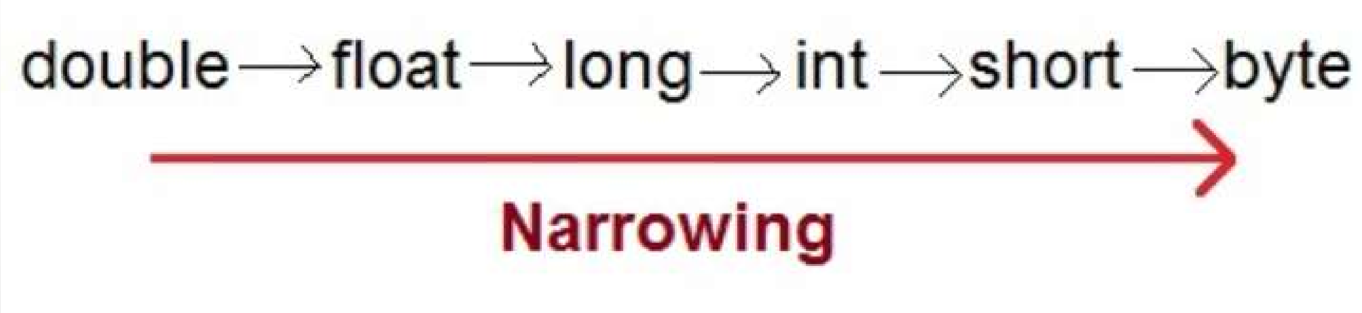
byte → short → int → long → float → double

widening

WIDENING: AUTOMATIC TYPE CONVERSION

- AUTOMATIC TYPE CASTING TAKES PLACE WHEN:
  - THE TWO TYPES ARE COMPATIBLE
  - THE TARGET TYPE IS LARGER THAN THE SOURCE TYPE

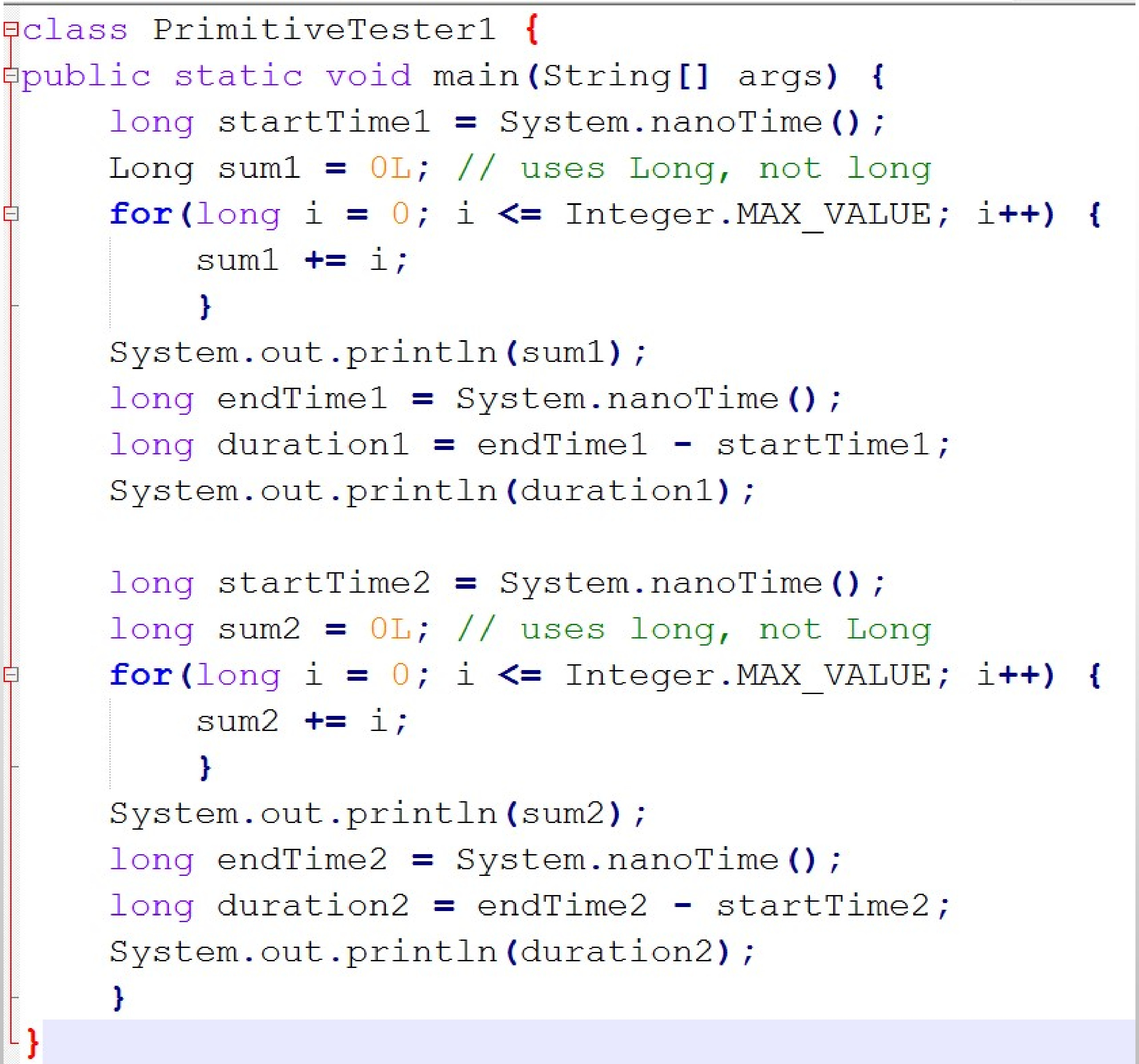
NARROWING CASTING(EXPLICITLY DONE)



double → float → long → int → short → byte

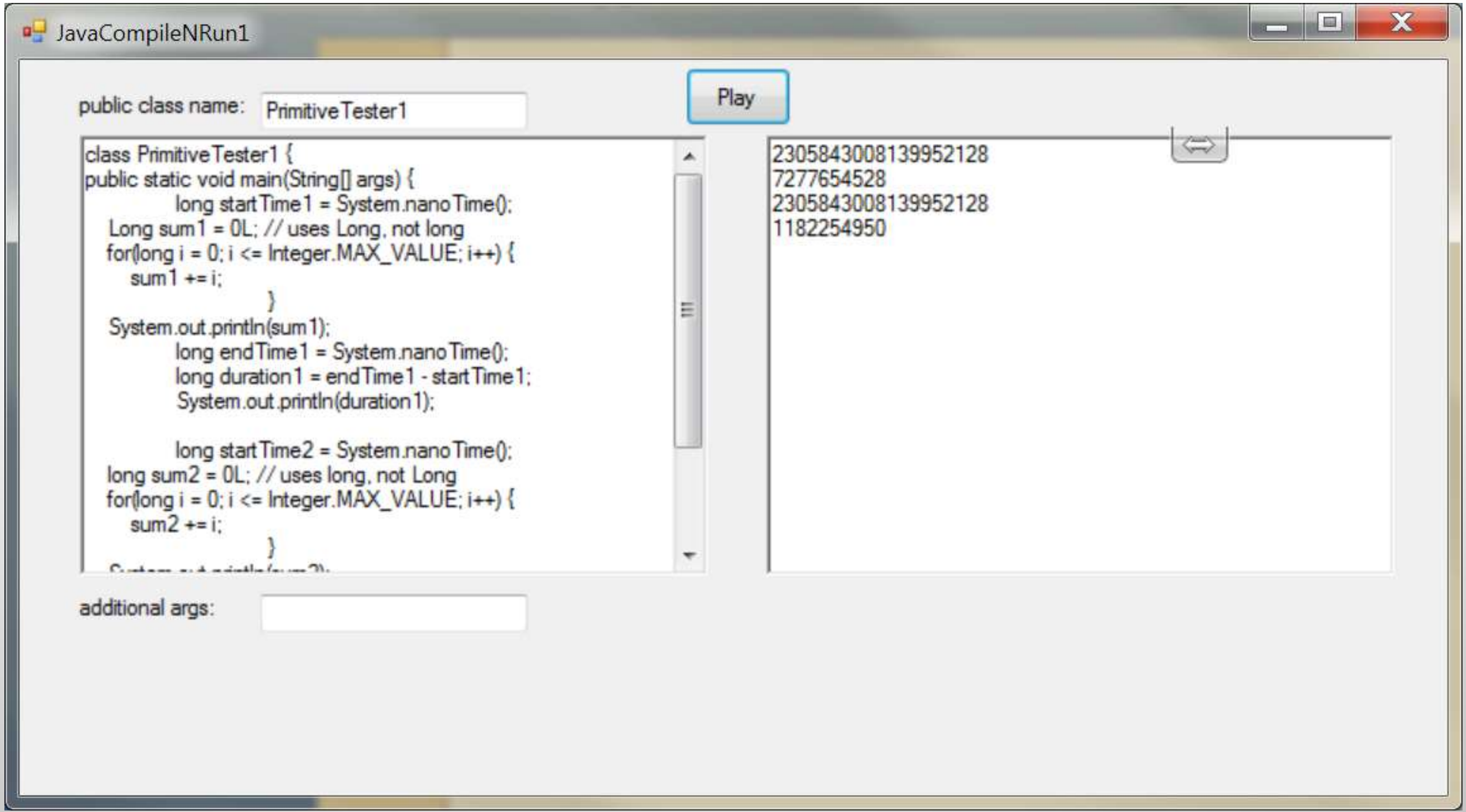
Narrowing

LETS TEST PRIMITIVE TYPE VS OBJECT!



```java
class PrimitiveTester1 {
public static void main(String[] args) {
    long startTime1 = System.nanoTime();
    Long sum1 = 0L; // uses Long, not long
    for(long i = 0; i <= Integer.MAX_VALUE; i++) {
        sum1 += i;
        }
    System.out.println(sum1);
    long endTime1 = System.nanoTime();
    long duration1 = endTime1 - startTime1;
    System.out.println(duration1);

    long startTime2 = System.nanoTime();
    long sum2 = 0L; // uses long, not Long
    for(long i = 0; i <= Integer.MAX_VALUE; i++) {
        sum2 += i;
        }
    System.out.println(sum2);
    long endTime2 = System.nanoTime();
    long duration2 = endTime2 - startTime2;
    System.out.println(duration2);
    }
}
```



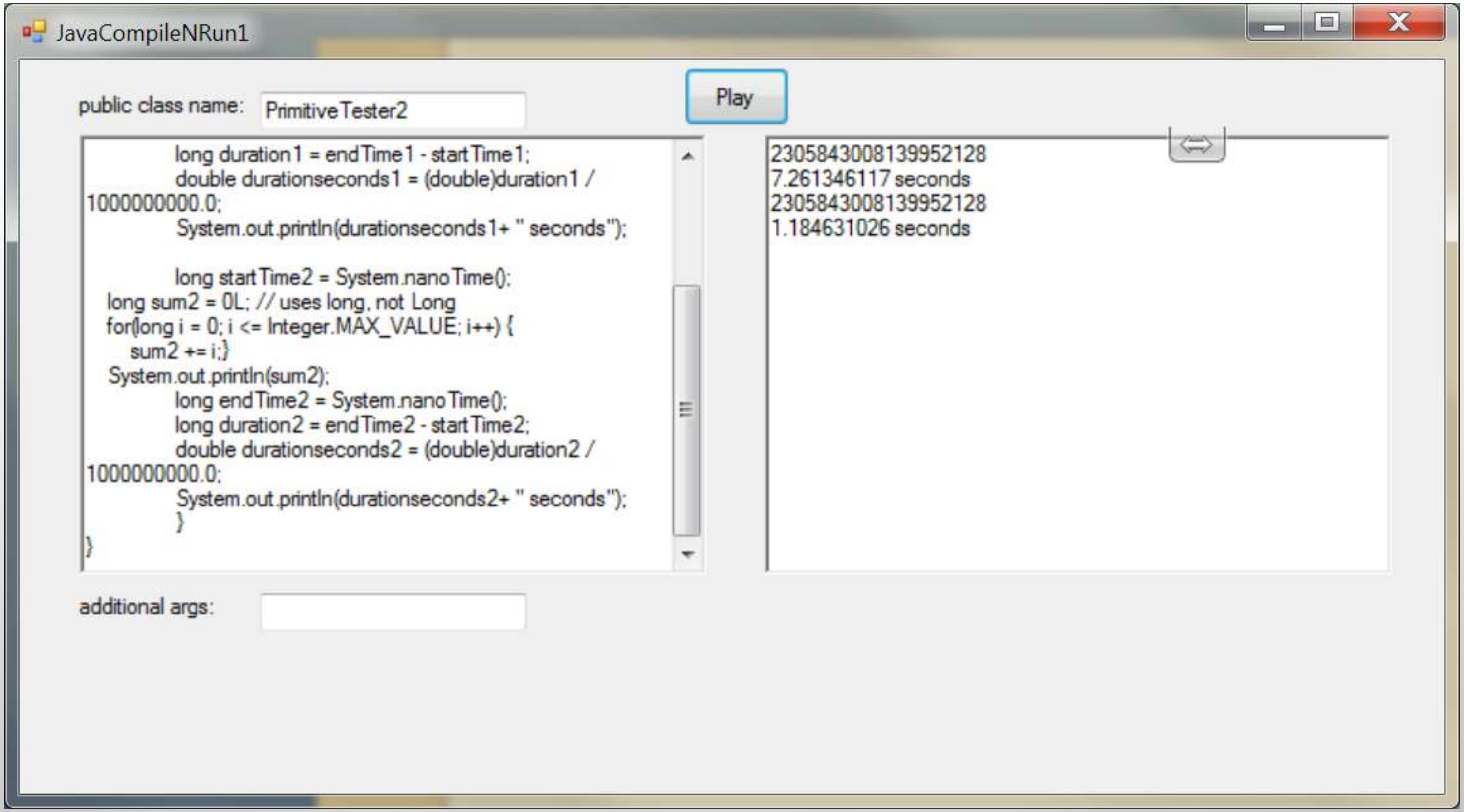
public class name: PrimitiveTester1

```
class PrimitiveTester1 {
public static void main(String[] args) {
            long startTime1 = System.nanoTime();
   Long sum1 = 0L; // uses Long, not long
   for(long i = 0; i <= Integer.MAX_VALUE; i++) {
      sum1 += i;
                         }
   System.out.println(sum1);
            long endTime1 = System.nanoTime();
            long duration1 = endTime1 - startTime1;
            System.out.println(duration1);

            long startTime2 = System.nanoTime();
   long sum2 = 0L; // uses long, not Long
   for(long i = 0; i <= Integer.MAX_VALUE; i++) {
      sum2 += i;
                         }
   System.out.println(sum2);
```

```
2305843008139952128
7277654528
2305843008139952128
1182254950
```

additional args:



public class name: PrimitiveTester2

```
            long duration1 = endTime1 - startTime1;
            double durationseconds1 = (double)duration1 /
1000000000.0;
            System.out.println(durationseconds1+ " seconds");

            long startTime2 = System.nanoTime();
  long sum2 = 0L; // uses long, not Long
  for(long i = 0; i <= Integer.MAX_VALUE; i++) {
     sum2 += i;}
  System.out.println(sum2);
            long endTime2 = System.nanoTime();
            long duration2 = endTime2 - startTime2;
            double durationseconds2 = (double)duration2 /
1000000000.0;
            System.out.println(durationseconds2+ " seconds");
            }
}
```

```
2305843008139952128
7.261346117 seconds
2305843008139952128
1.184631026 seconds
```

additional args: