

Unit 1

Introduction to Java



Introduction and Logistics

Unit 1 – Introduction to Java

- Introduction

- About me
- About you



- Logistics

- Class timings : 9.30 am to 6.00 pm
- Lunch Break : 1pm to 2pm
- Coffee Breaks : 11 - 11.15 am & 3.30 – 3.45 pm

- Please switch off cell phones, IMs

Topics

Unit 1 – Introduction to Java

- History of Java
- Features of Java
- Running a Java Program
- Bytecode
- JIT Compilation
- Java Virtual Machine and JRE
- Java Applications Vs Java Applets
- Security in Java
- Installation of Java
- Simple Java application

History of Java

- Java was developed at **Sun Microsystems**

between 1991 and 1995

- Started off as a small, secret effort called "the Green Project" in 1991
- Spearheaded by James Gosling and Bill Joy



James Gosling



Bill Joy

- The original objective of Java was:**
 - 'To provide a platform-independent programming language and operating system for consumer electronics'



- Java however became popular as the language of the web due to **bytecodes**, **applets** and its platform-independent nature
- However people soon realized that there is more to Java than just Applets!*

Features of Java

- Simple
- Object-Oriented
- Distributed
- Portable
- Interpreted
- High performance
- Robust
- Secure

Buzzwords

Features of Java

- **Simple** →
- Object-Oriented
- Distributed
- Portable
- Interpreted
- High performance
- Robust
- Secure

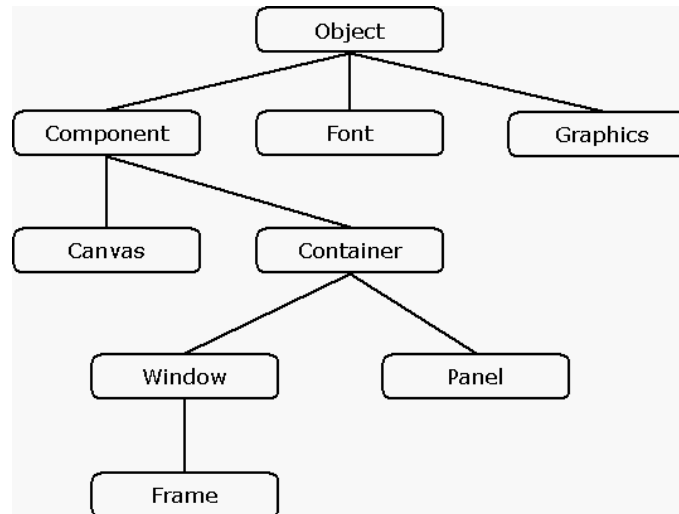
1. Java borrows its style from C++ which, most programmers are familiar with
2. Automatic garbage collection ensures that the focus is on problem-solving
3. Features of C++ that were ambiguous or made applications difficult to maintain were eliminated
Multiple Inheritance
Operator Overloading, etc.

```
6
7     public String numToWords(int number)
8     {
9         int zero = 0;
10        boolean nonzero = true;
11        String words = "";
12
13
14        while (number > 0)
15        {
16            rem = (number % 10);
17            if ((nonzero == true) && (rem == 0))
18            {
19                ++zero;
20            }
21            else
22                nonzero = false;
23
24            rev = (rev * 10) + rem;
25            number = number / 10;
26        }
27    }
```

Features of Java

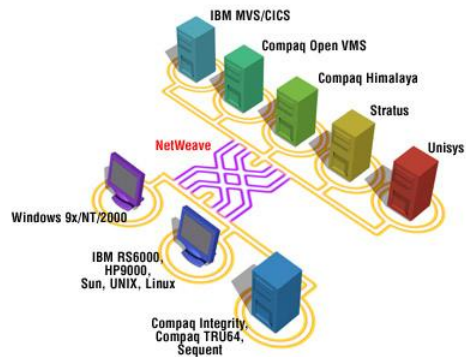
- Simple
- **Object-Oriented** →
- Distributed
- Portable
- Interpreted
- High performance
- Robust
- Secure

1. Everything is an object in Java (even main() method is within a class)
2. Unlike C++ there are no global data or global functions.
3. All of the Java's built-in libraries themselves are fully object-oriented.



Features of Java

- Simple
- Object-Oriented
- **Distributed** →
- Portable
- Interpreted
- High performance
- Robust
- Secure

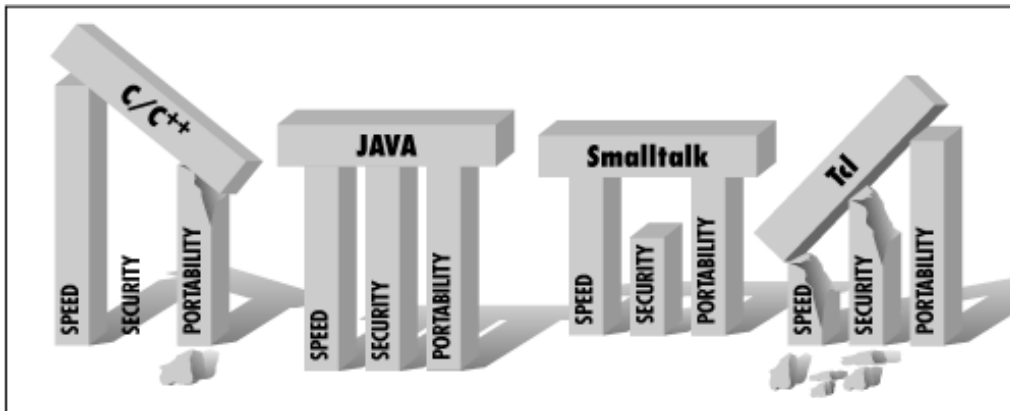


1. Java supports building distributed applications with a rich set of built-in libraries
2. Java has an extensive library of routines for coping easily with TCP/IP protocols like HTTP and FTP.
3. Java's libraries make creating network connections much easier than in C or C++.
4. Java applications can open and access objects across the net in a manner similar to accessing a local file system.

Features of Java

- Simple
- Object-Oriented
- Distributed
- **Portable** →
- Interpreted
- High performance
- Robust
- Secure

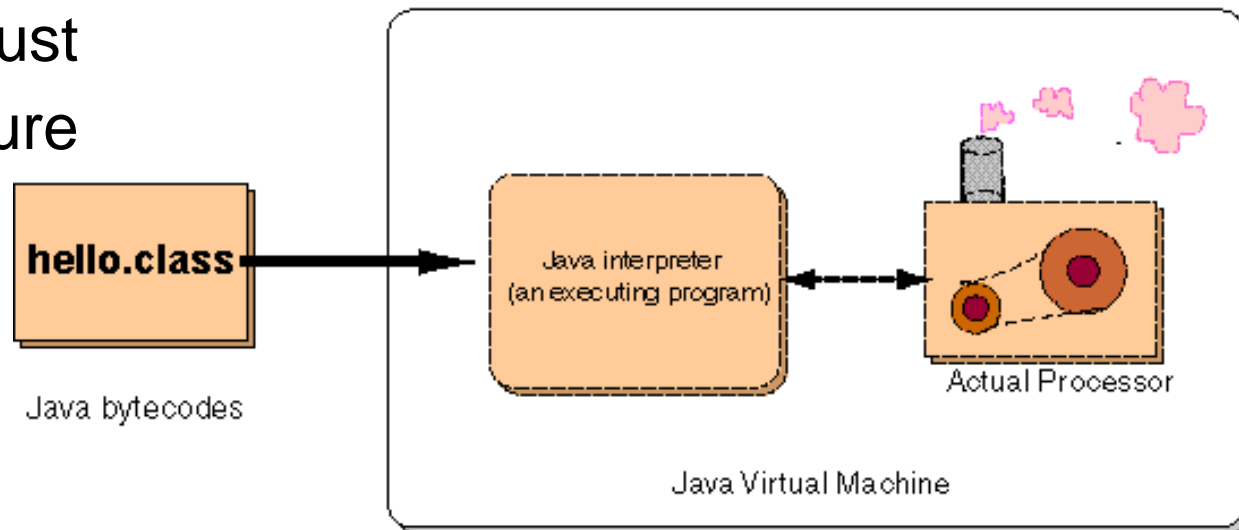
1. Java is platform independent
 1. Java applications can be written and compiled on any platform and can be run on any other platform without requiring recompilation.
 2. 'Write once, run anywhere'
2. No "implementation dependent" aspects of the specifications in Java.
 1. Sizes of the primitive data types are standardized
 2. The libraries that are a part of the system define portable interfaces.



Features of Java

- Simple
- Object-Oriented
- Distributed
- Portable
- **Interpreted** →
- High performance
- Robust
- Secure

1. The Java compiler does not emit platform specific executable code, instead it produces an intermediate code called *Bytecodes*.
2. Java bytecodes are translated on the fly to native machine instructions.



Interpreting Java Bytecode on a Virtual Machine

Features of Java

- Simple
- Object-Oriented
- Distributed
- Portable
- Interpreted
- **High performance**
- Robust
- Secure



High performance. Guaranteed.

1. The Java interpreter makes many optimizations during runtime
2. The bytecode format was designed with generating machine codes in mind, so the actual process of generating machine code is generally simple.
3. The performance of bytecodes converted to machine code is almost similar to native C or C++.

Features of Java

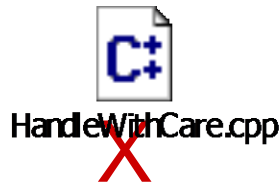
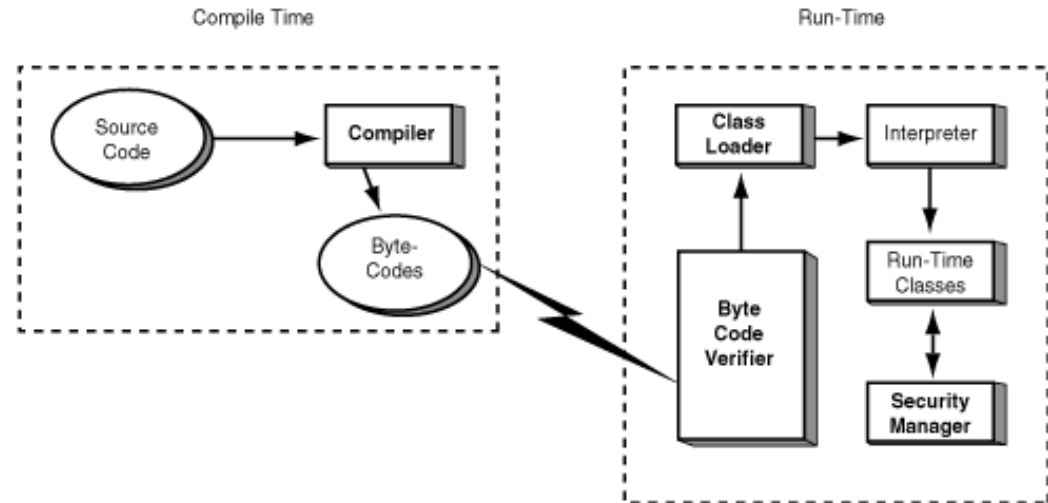
- Simple
- Object-Oriented
- Distributed
- Portable
- Interpreted
- High performance
- **Robust** →
- Secure

1. Robust refers to programs being less error prone and more stable.
2. Java has no pointers and has its own memory handling mechanism to handle memory.
3. Java is strongly typed and disallows improper conversions / casting
4. Provides features like exception handling where the user can be forced to handle exceptions without fail



Features of Java

- Simple
- Object-Oriented
- Distributed
- Portable
- Interpreted
- High performance
- Robust
- **Secure**



Thou shall not do this in JAVA!

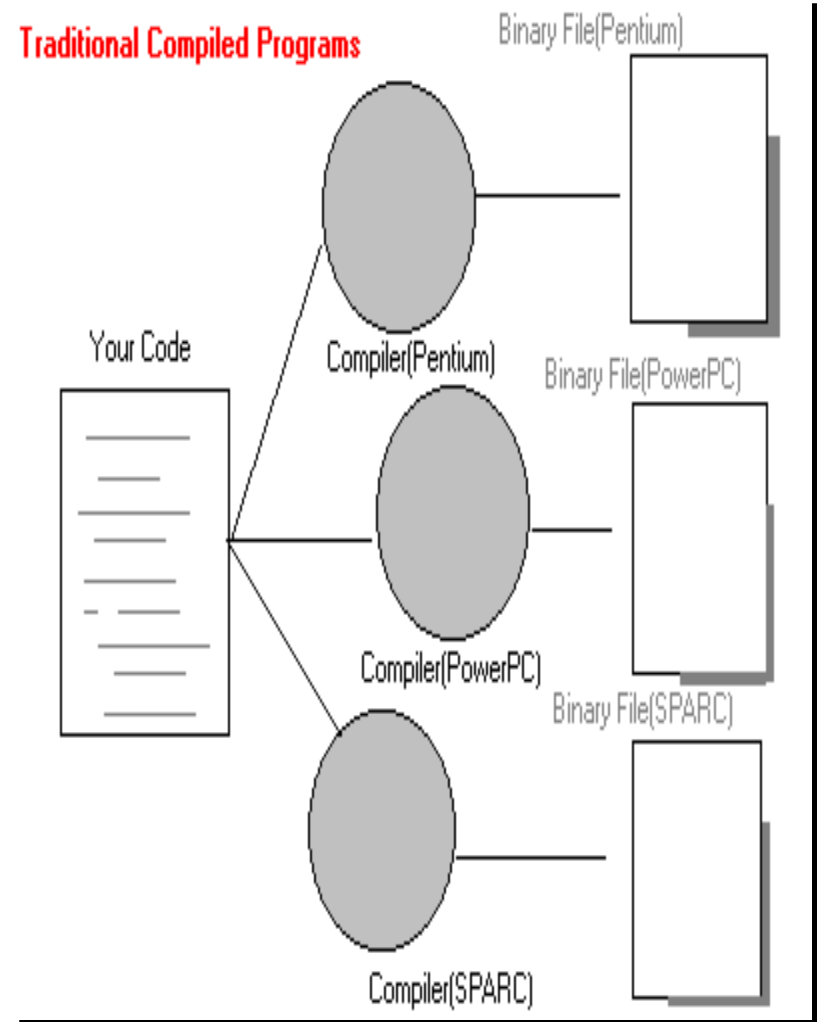


Java™ Security

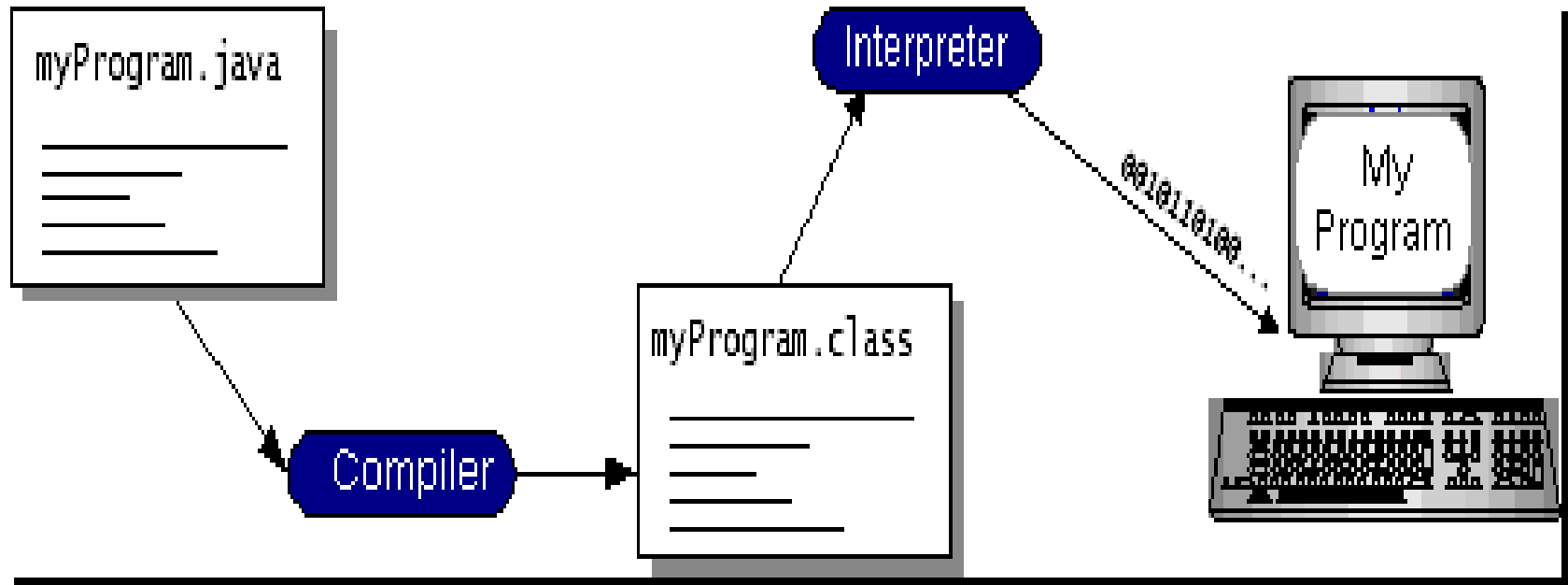
1. Java addresses security in many ways, making it extremely safe for a user to use Java applications.
2. Applications cannot forge access to data structures or to access private data in objects that they do not have access to.
3. Java applets run in a Sandbox and hence cannot access certain types of system resources

How traditional programs run

- In traditional languages such as C/C++ the code had to be
 - written with a specific platform/OS in mind
 - compiled on that platform.
- The compiler would then emit executable code which would execute only on that platform.
- If an application had to run on multiple platforms, the source files had to be customized and compiled using different compilers for each platform.
- This was an expensive proposition if an application had to run on different platforms.

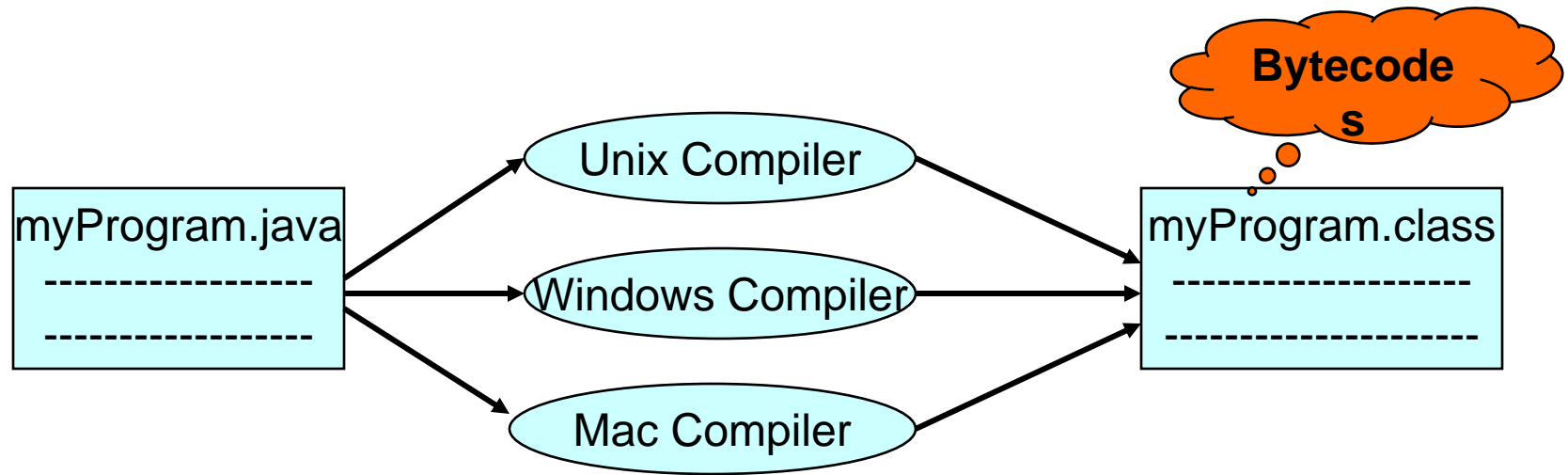


How Java runs



- Each Java Program is both compiled and interpreted.
 - A Java program(*.java file) on compilation is translated into an intermediate language called Java *Bytecodes* which is *platform independent*.
 - These platform independent bytecodes are interpreted by the Java interpreter for the application to run.
- Compilation happens just once, interpretation occurs each time the program is executed.

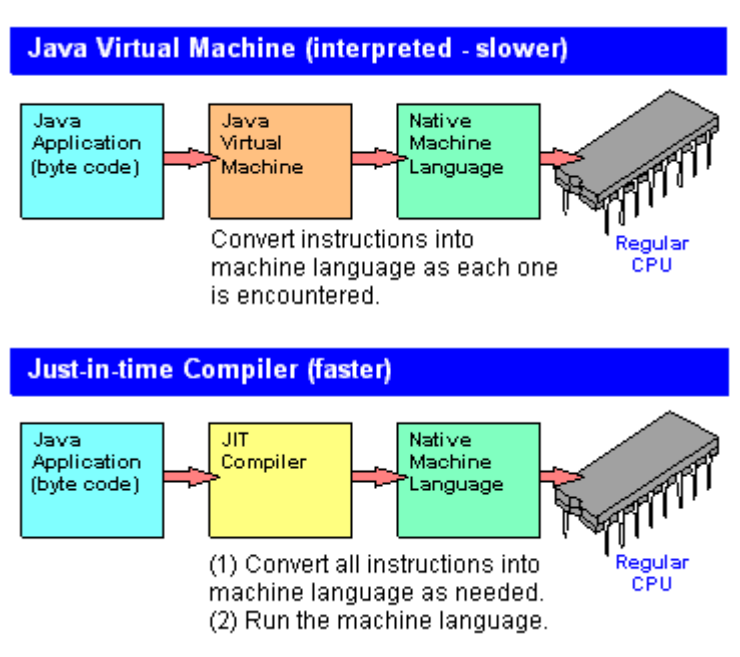
Java Bytecode



- The Java compiler compiles the source program **.java* file into an intermediary *.class* file.
- The *.class* file contains *bytecodes* – the machine language of Java Virtual Machine (JVM).
- Bytecode is the magic behind
 - “*Write Once, Run Anywhere*”tm
- Bytecode is be executed by Java run-time system

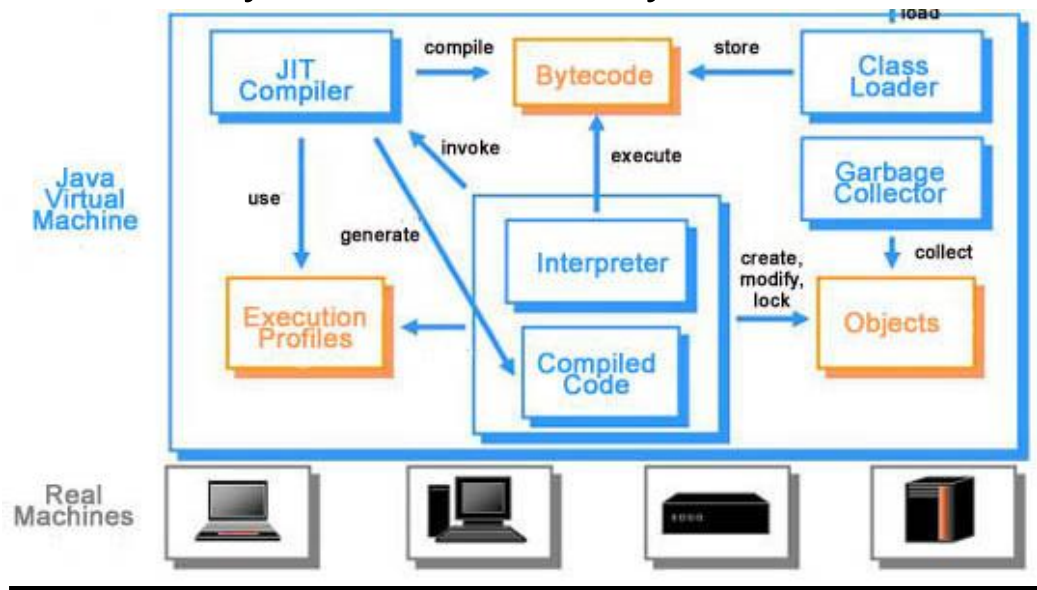
Just In Time (JIT) compilation

- The JIT compiler compiles bytecodes into platform specific instructions.
- The JIT compiler can be used to
 - speed up execution
 - offset the cost of interpretation
- Different implementations of JVM use different algorithms
- Overall, improves performance while using some more memory



The Java Virtual Machine (JVM)

- The Java Runtime Environment consists of the JVM.
- The JVM is an abstract computer on which all Java programs run.
- Bytecodes are the machine language of the JVM
- JVM interprets the Bytecodes to the system

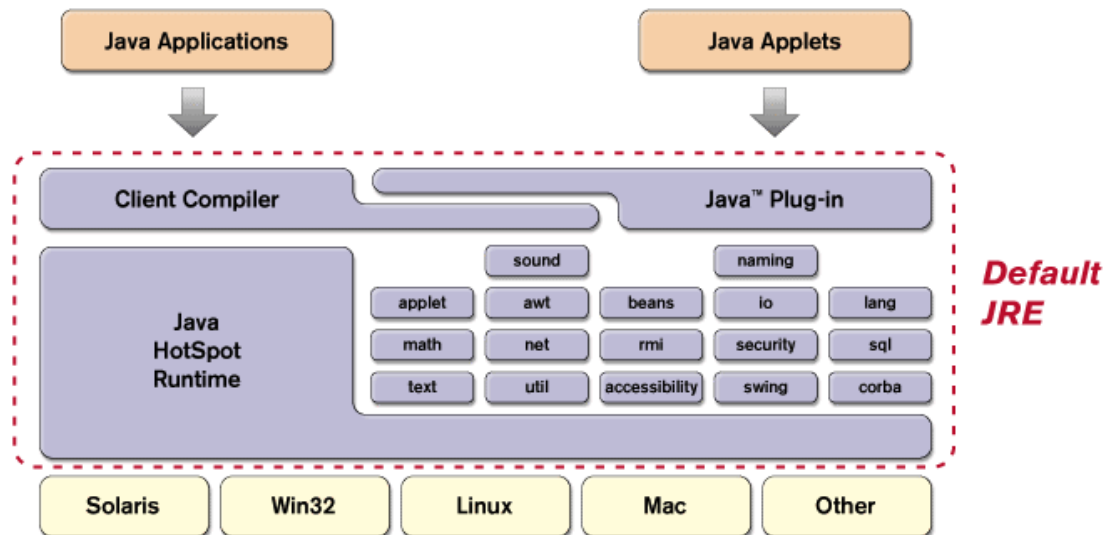


JVM allows

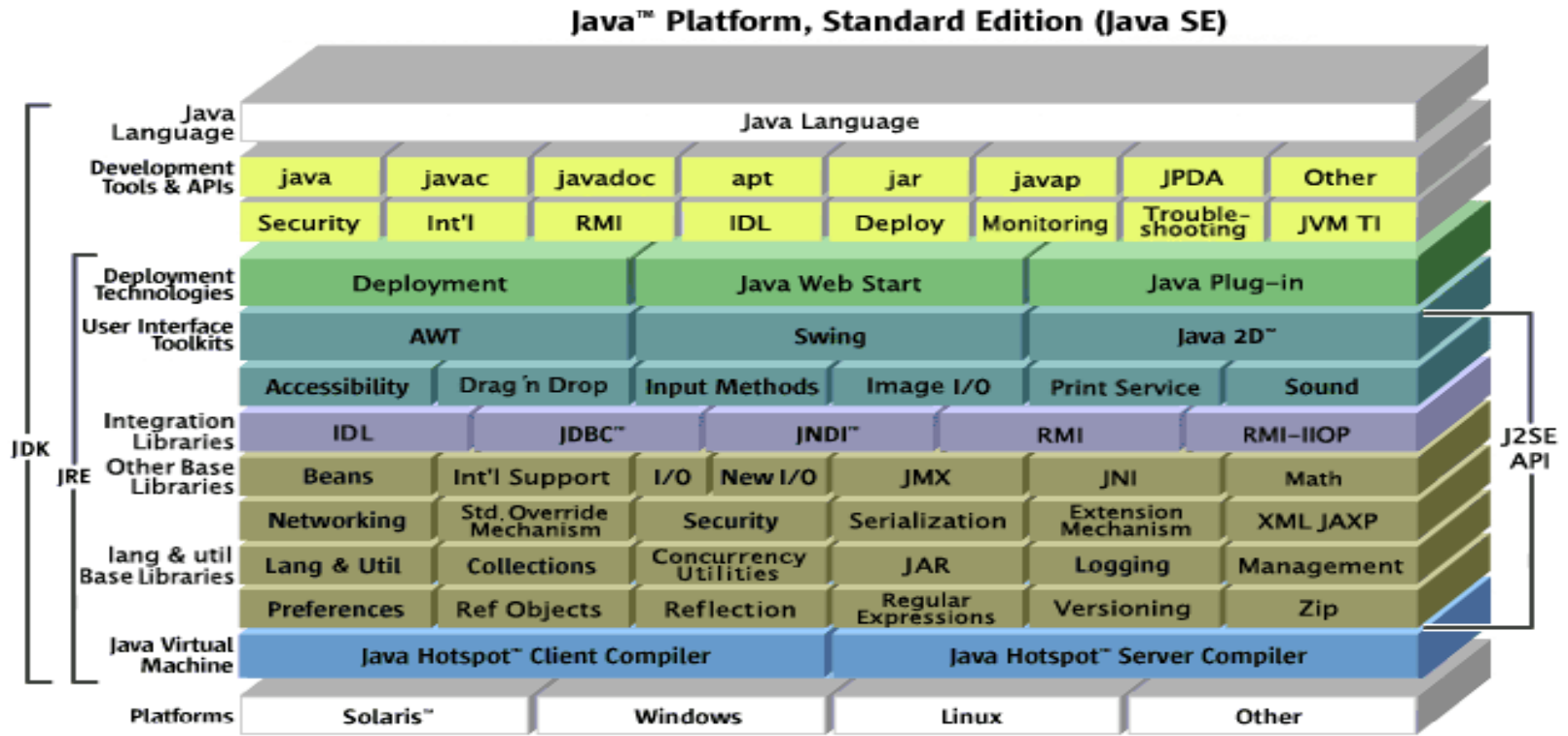
- cross platform delivery
- small size of compiled code
- high level of security

The Java Runtime Environment (JRE)

- The Java Runtime Environment (JRE) is the software that needs to be installed on each machine on which you want to execute Java applications.
- In addition to the JVM, the JRE also has
 - JIT compiler
 - The Java Plug-in
 - The built-in API's
 - The Java API provides the core functionality of the Java programming language.
 - Java API is a large collection of ready-made software components, for use while developing an application.
- For all practical purposes, a lot of people interchangeably use the terms JRE and JVM.

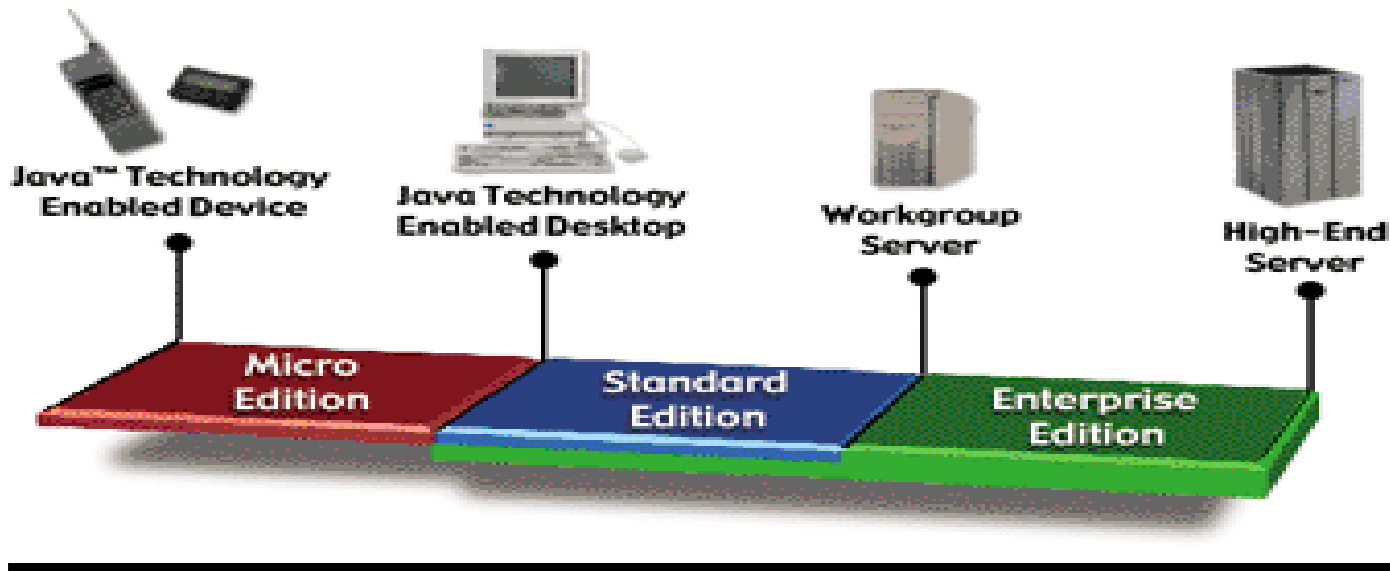


JRE vs. JDK



- The Java Development Kit (JDK) is the software required to develop Java applications.
- This development kit comes with many tools such as the compiler, debugger, libraries etc.
- On installing the JDK, the JRE comes along with it.

Three editions



- Java Micro Edition (JME)
 - An application platform for Java applications to run on mobile phones, PDAs etc.
- Java Standard Edition (JSE)
 - An application platform for Java applications to run on desktops.
- Java Enterprise Edition (JEE)
 - Is the industry standard for developing portable, robust, scalable and secure server-side distributed applications.

Applications vs Applets

Applications

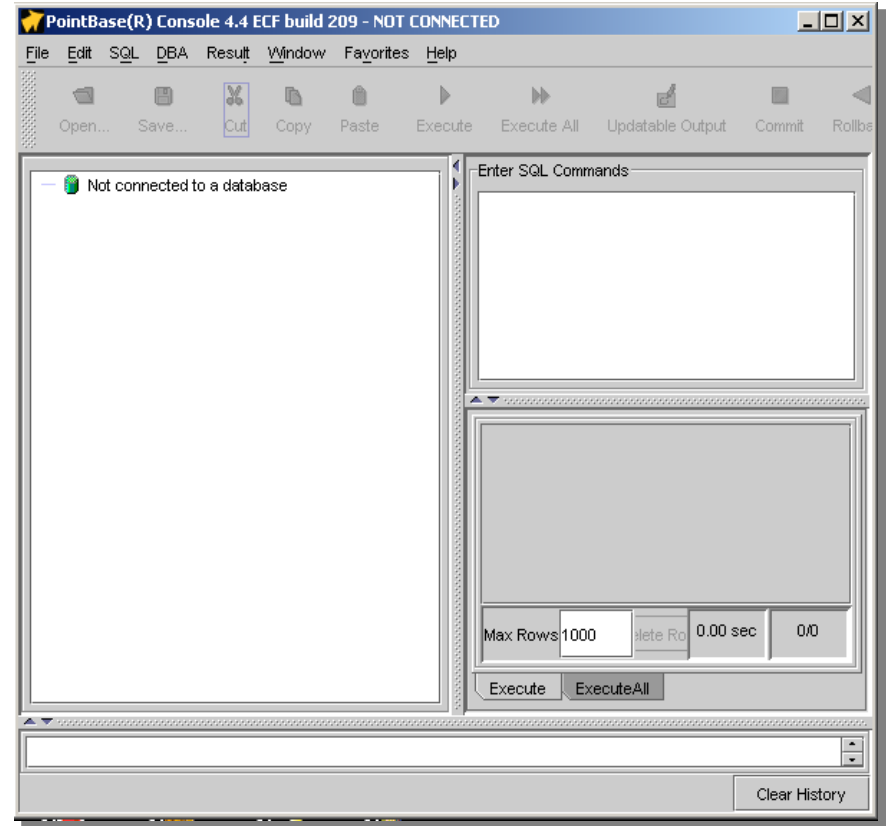
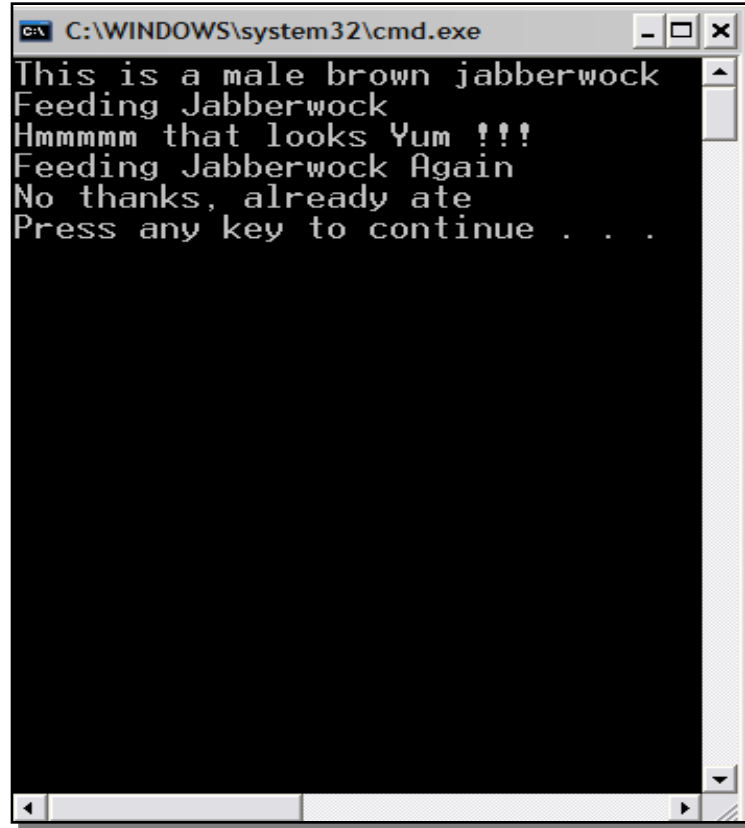
- A Java application is a standalone program that runs directly on the Java VM.
- Executed in command Line by Java Interpreter.
- Do not have any special security restrictions

Applets

- Applets, in contrary to applications are embedded in Web pages.
- Executed in AppletViewer or a Java-enabled Browser
- Have Security Restrictions

Applications vs Applets

Applications can be GUI based or console based :



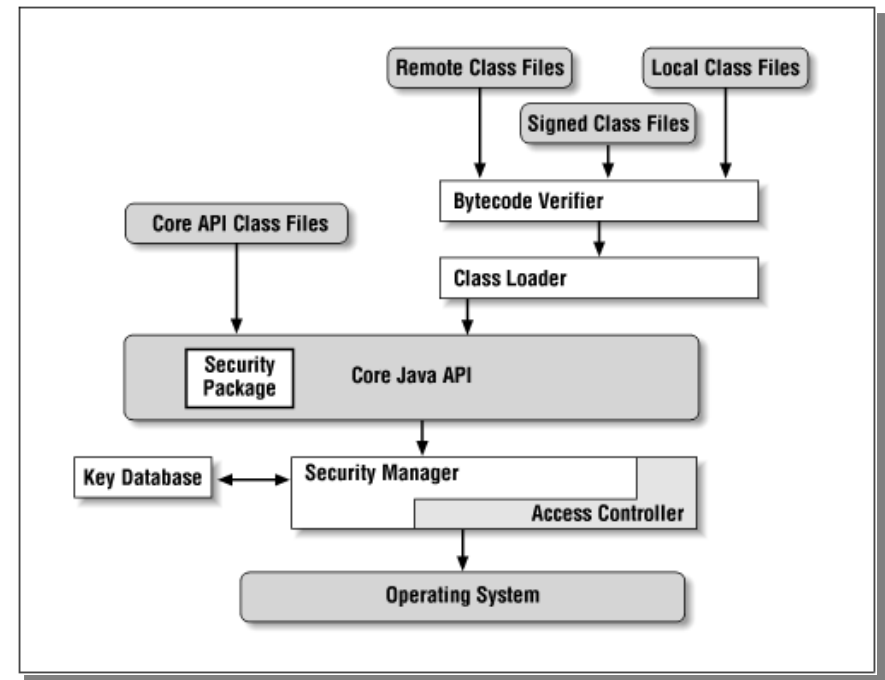
Applications vs Applets

Applets are almost always GUI based



Security in Java

- One of the most important design goals of Java language was to allow creation of powerful applications that could be distributed easily over a network.
- This distribution model had to be achieved without jeopardizing the users local security, wherein any user can download the application and be assured that the code would not be destructive.
- Java's security can be broadly classified into four levels
 - The language and compiler
 - Bytecode Verifier
 - Class loader
 - SandBox



Security in Java

Level 1 :Language and Compiler

- Java language and the compiler are the first line of security.
- Java takes some of its features from C++, it does not have many of the security problems of C++ .
- Unlike C++, Java does not allow forge access to objects that defeat the access controls.
- Unlike C/C++, Java does not directly support pointers to access memory. Java has true arrays and uses named references instead of pointers.

Level 2: Bytecode Verifier

- A Java compiler ensures that Java source code does not violate any of the safety rules.
- But if a Java compiler were written that violates these rules, the bytecode verifier ensures these checks on any code:
 - Does not forge pointers.
 - Does not violate access restrictions.
 - Always uses objects in the way they were intended to be used.
 - Calls methods with appropriate arguments of the appropriate types.
 - Causes no stack overflows.
 - Does not try to execute any privileged or insecure instructions.

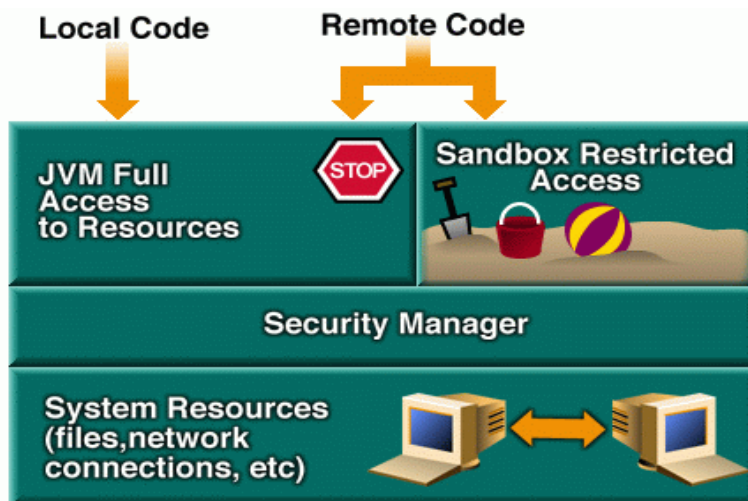
Level 3 : Class Loader

- The class loader allocates memory space for each class and makes sure that the code is not attempting to deceive or dislodge (spoof) a built-in class.
- For example, a programmer could not write his own version of the string class and have it execute instead of the built-in class. The built-in classes are always checked first.

Sandbox

Level 4 : The Sandbox Model

- The basic rules of sandbox security determine what a Java applet can not do within a web browser or in any other container that implements this Java security approach.

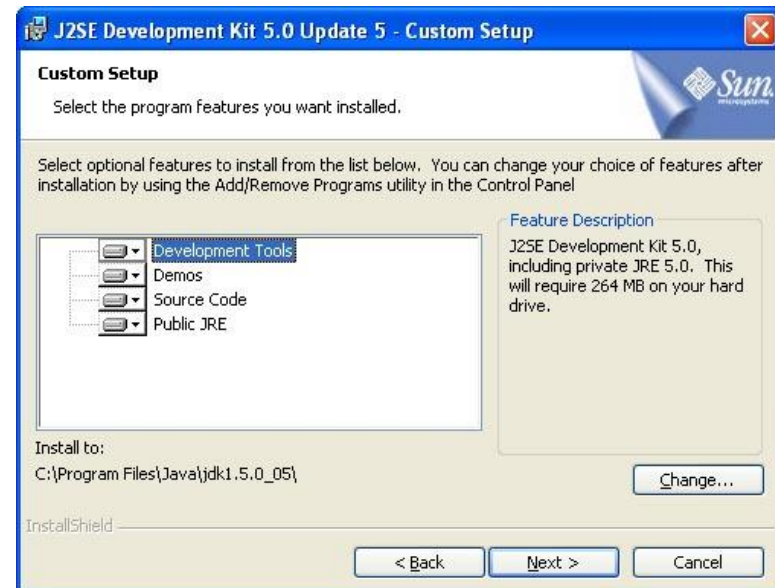


■ Java applets cannot:

- Delete files on the local system.
- Read from or write to local files.
- Create new directories on the local system.
- Inspect directory contents or check various file attributes.
- Execute programs on the local system.
- Call DLLs.
- Create network connections to machines other than the server from which the applet was loaded.
- Create objects from the core packages that manage security, like security manager and class loader.

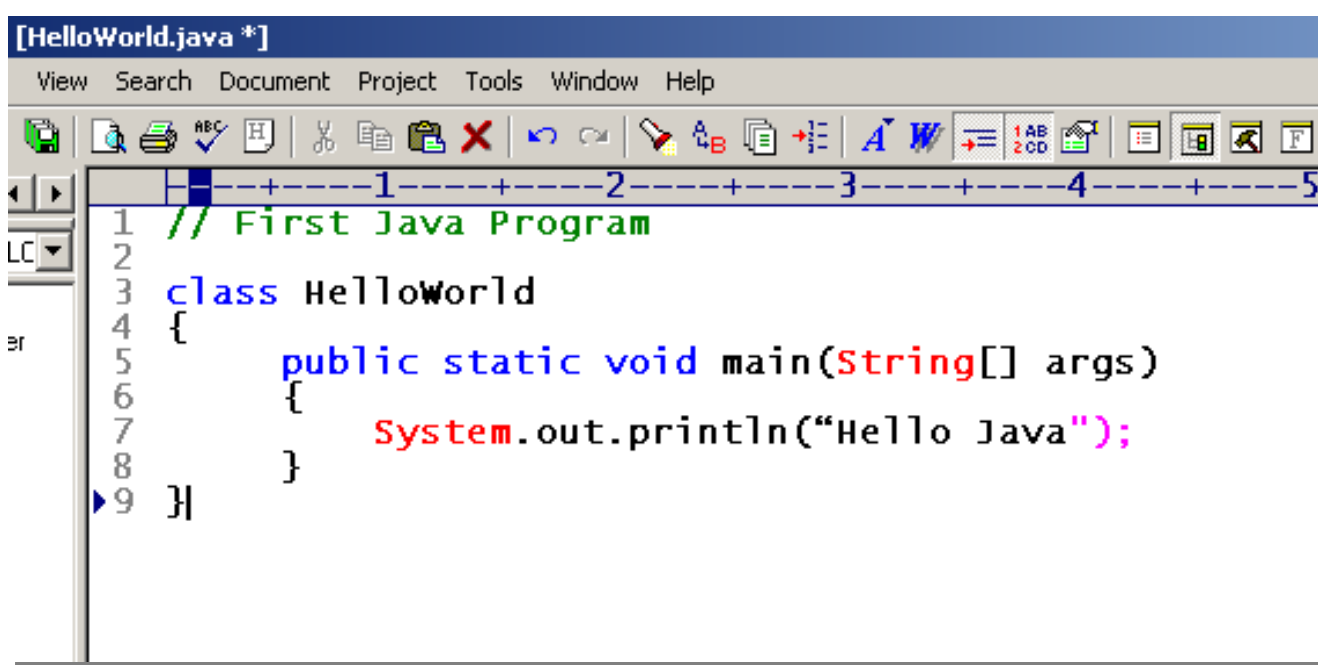
Installing Java

- The Java Development Kit (JDK) can be downloaded from the sun site.
- The JDK comes as an executable file
 - On execution of this, the JDK would be installed on the computer.
- Upon installation add the **bin** directory to the path.
 - Ex : If Java is installed in the folder C:\Program Files, add **C:\Program Files\Java\jdk1.5.0_06\bin** to the environment variable **path**.



Creating a Simple Java Application

- Open a text editor and create the source file with the following code.
- The file should be saved with a .java extension
 - In this case name of the file should be HelloWorld.java.

A screenshot of a text editor window titled "[HelloWorld.java *]". The window has a menu bar with "View", "Search", "Document", "Project", "Tools", "Window", and "Help". Below the menu bar is a toolbar with various icons for file operations and editing. The main text area shows the following code:

```
1 // First Java Program
2
3 class HelloWorld
4 {
5     public static void main(String[] args)
6     {
7         System.out.println("Hello Java");
8     }
9 }
```

Compile the file in the console using

javac HelloWorld.java

Execute the file using

java HelloWorld

Question time



Please try to limit the questions to the topics discussed during the session.

Thank you.