

PRAGMATIC

Audacity

# Core Java

Introduction



# Introduction



Java was developed at Sun Microsystems, Inc. later acquired by Oracle Corporation in 2010



James Gosling, Mike Sheridan, Chris Wrath and Patrick Naughton initiated the Java project in June 1991



The language was initially called Oak named after an oak tree that stood outside Gosling's office



However, Oak was trademarked by Oak Technologies; hence, renamed to Java



Oak was originally developed for set-top boxes, interactive TV and PDA.

# Introduction



However, the demise of these devices in 1993 ushered the investors' thoughts to be reinvented.



National Center for Supercomputing Applications (NCSA) unveiled its new commercial web browser for internet in 1992.



The Green team saw an opportunity for the same embedded technology in the web browser space and called it “**HotJava**” that was capable of running “**Applets**”

# Drivers behind Java

- ☕ C and C++ were the dominant programming language before emergence of Java
- ☕ C is structural and the focus is on how to solve a problem.
- ☕ Memory management is developer(s) responsibility using `malloc()`, `calloc()` and `free()` functions defined in `<stdlib.h>`
- ☕ Direct access and manipulation of memory addresses using **pointers**.
- ☕ Though C++ is objected oriented; yet it breaks certain rules of data hiding through **friend functions**.

# Drivers behind Java



Memory Management is developer(s) responsibility



Non Portable. Size of integer variables varied across platforms/architectures.



C/C++ were not designed for embedded systems that require smaller memory footprint.



C++ suffers from the classic **Diamond Inheritance** problem. In order to solve this, one must use virtual methods.

# Drivers behind Java

- ☕ Language should be “simple, objected oriented and familiar”
- ☕ Language should be “robust and secure”
- ☕ Language should be “architecture neutral and portable”
- ☕ Language should execute with “high performance”
- ☕ Language should be “interpreted, multithreaded and dynamic”

# What is Java

☕ Java is a set of several computer **software products** and **specification** from Sun Microsystems now taken over by Oracle that provide a system for developing application software and deploying it in a cross-platform computing environment.

☕ Java is a programming language that has a set of **API**, syntax and data types.

☕ Java borrowed its syntax from **C/C++** that were the most popular programming languages of the time.

☕ Its object-oriented features are modeled after **Smalltalk** and **Objective-C**.



# What is Java



Low-level constructs such as **pointers are eliminated** in Java. It has a very simple memory model where every **object is allocated on the heap**. Also, Java internally uses **soft/smart pointers**.



**Memory management** is handled through integrated automatic **Garbage Collection**.

# Characteristics of Java



Simple



Object-oriented



Distributed



Robust



Secure



Architecture-neutral



Portable



High-performing



Multithreaded



Dynamic


# Java is Simple


- ☕ Most of the language syntax of Java was borrowed from C and C++ that were the most popular and powerful languages of the time and simple to write programs
- ☕ C/C++ had many features like pointers that were quite dangerous in wrong hands.
- ☕ James Gosling said that Java is “**C++ without the knives, guns and clubs.**”
- ☕ One major simplification in Java is the way that memory is managed, using largely automatic processes rather than requiring the programmer to do this.

# Object Oriented

- ☕ In 1990s most of the languages were procedural in nature.
- ☕ Object-oriented approach was regarded as a new approach to programming
- ☕ In an object-oriented language, instead of having data on one hand and processes on the other, the two are “**encapsulated**” together to provide objects that have both **state(data)** and **behavior(methods)**.
- ☕ The behavior of real-world things are easier to model by tying the two.

# Distributed

 Most of the computers are connected to a network and probably to the Internet. Hence, most of the software resources are kept on network or cloud.

 Java is designed for network programming and can easily work with common Internet protocols such as HTTP (Hypertext Transfer Protocol) and FTP (File Transfer Protocol).

 It provides libraries for Socket Communication, Remote Method Invocation, Web Services and various other aspects of distributed computing.

# Robust

- ☕ A robust program does not behave unpredictably due to programmer error.
- ☕ Manipulating memory that has not been correctly allocated can crash a program and can further cause “memory leaks.”
- ☕ Java removes the concept of “pointers” thereby allowing a programmer to write more robust code.
- ☕ Java has only “references” to objects, not pointers to memory they occupy.
- ☕ JVM takes care of **Memory Management** in Java and ‘cos programmers don’t have direct access to memory, they cannot wrongly manipulate it.

# Robust



Exception handling is built-in



Java is strongly-typed language i.e., all variables must be declared an explicit type.



It carries out type checking at both compile-time and run-time ensuring that every data structure has been clearly defined and typed.

# Secure

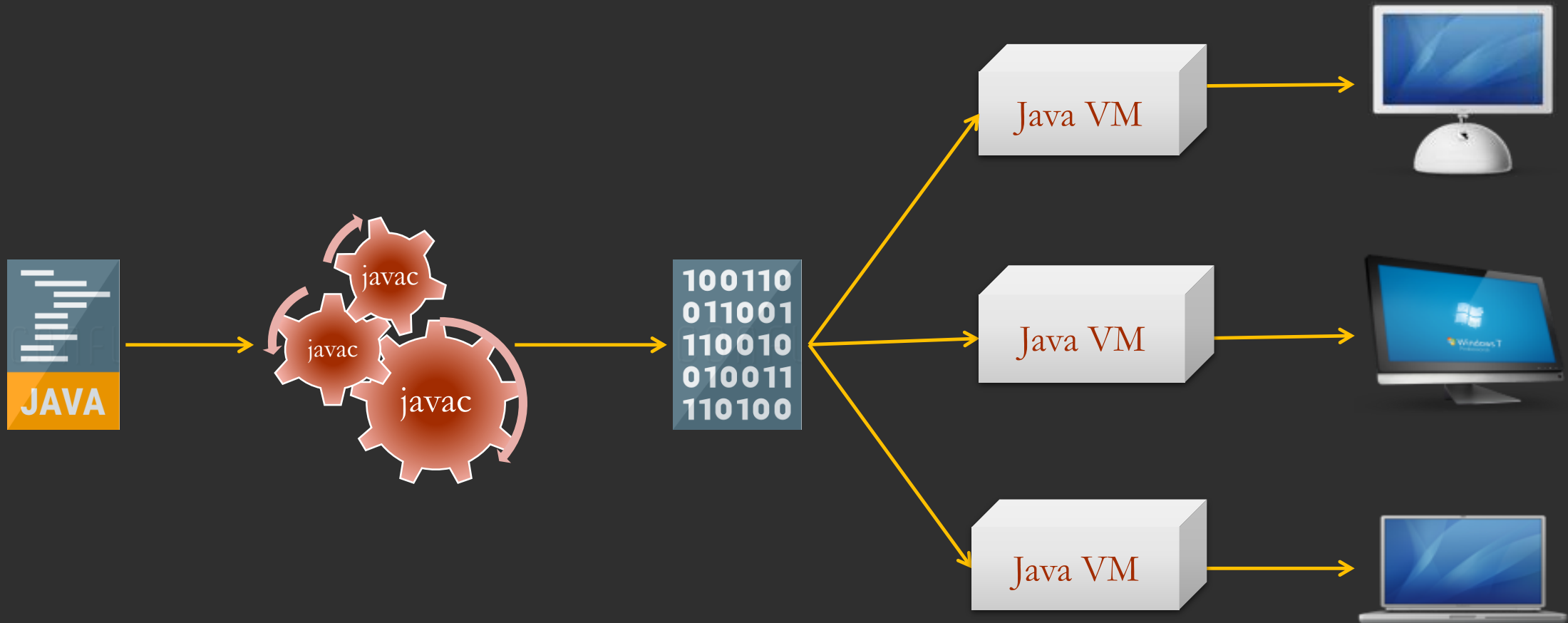
- ☕ Security systems built in Java ensure that code, once written, is not easy to tamper with.
- ☕ All Java programs execute inside the JVM and thus, cannot access system resources directly.
- ☕ Bytecode verifier checks classes after loading
- ☕ Classloader confines objects to unique namespaces.
- ☕ Security Manager determines what resources a class can access such as reading and writing to the local disk.







# Architecture Neutral

- ☕ Java is a “Write Once, Run Anywhere” language
- ☕ Java is both a compiled and an interpreted language
- ☕ Java source code gets compiled into platform independent “bytecode”
- ☕ The bytecode can run on any hardware that has a Java Virtual Machine

# Architecture Neutral



# Portable

-  In many languages, there's no specification of how much storage a datatype should take.
-  In C/C++ if a program is compiled using 16-bit compiler, then it will produce 16 bits of integers and they might not run on 32 bit architectures. Similarly, a program compiled with 32 bit compiler won't run on 16 bit architecture machine.
-  Part of Java's architecture neutrality is based on portable definitions of how big different datatypes are.
-  For example, in Java the size of integer across platforms is 32 bits long. These are always signed.

# High Performing



Because Java bytecode has to be interpreted, Java programs generally runs more slowly than equivalent programs written in languages that produce native executable binaries.



To overcome this problem, Just-in-time (JIT) compilers have been developed that speed up the interpretation process.



More recent versions of Java include “Hotspot” compiler. This compiler optimized code while it is running.

# Multithreaded

- ☕ By building syntax for multithreading into language, Java makes it easier for programmers to write multithreaded programs.
- ☕ Even where operating system is not multithreaded, Java code can be written that uses multiple threads of control.

# Dynamic

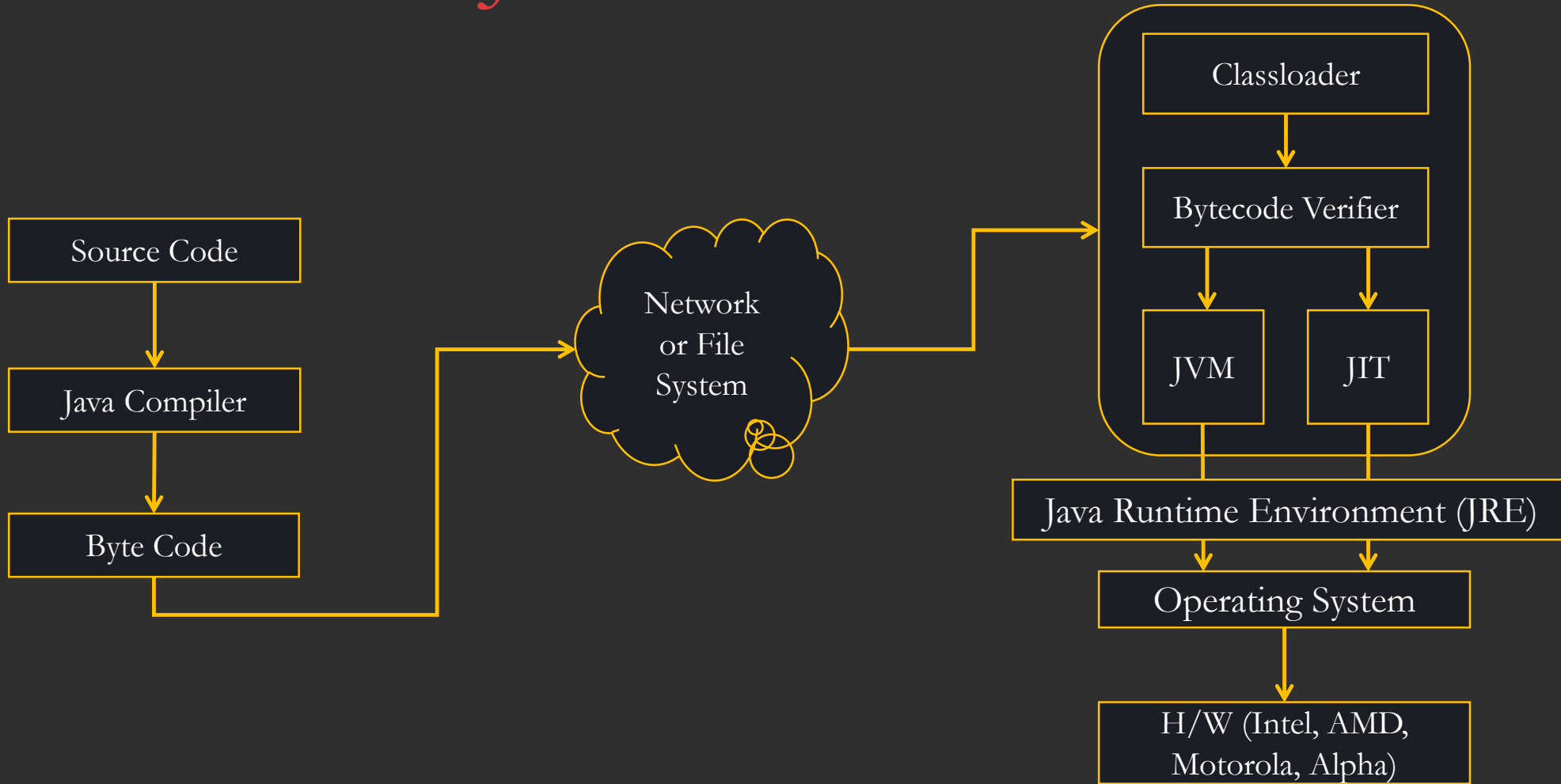


A Java program can dynamically change the resources it is using at runtime



This is useful especially when resources are distributed across networks and clouds. Java can locate them while the program is executing.

# Architecture of Java



# First Program in Java

```
class HelloWorld {  
    public static void main(String [] a) {  
        Console.WriteLine("Welcome to Java!");  
    }  
}
```

Output:

Welcome to Java!



# Setting up environment



Go to Oracle website and download a jdk --

<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

## Java SE Development Kit 8u191

You must accept the [Oracle Binary Code License Agreement for Java SE](#) to download this software.

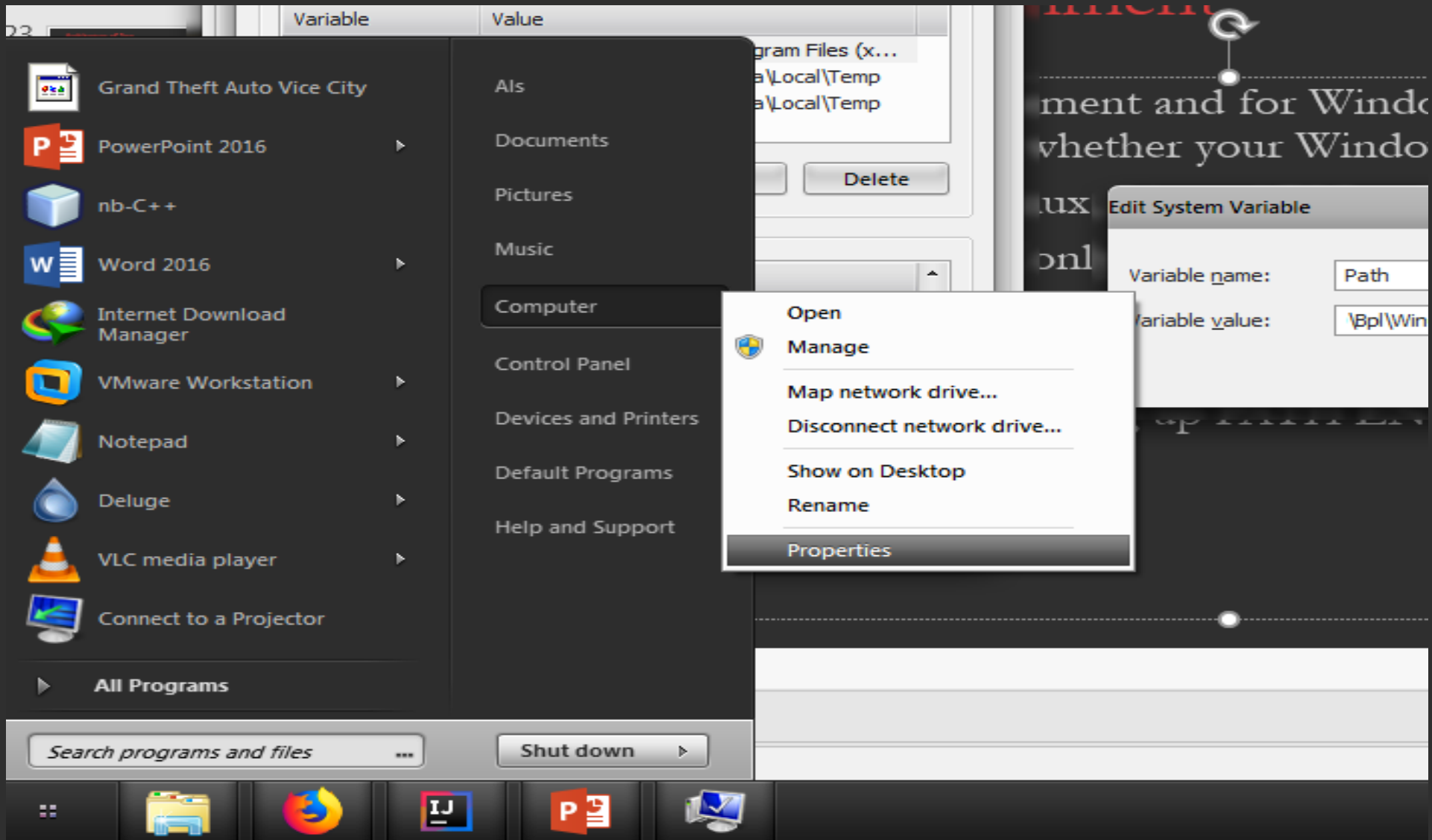
Accept License Agreement  Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.97 MB	<a href="#">jdk-8u191-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	69.92 MB	<a href="#">jdk-8u191-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	170.89 MB	<a href="#">jdk-8u191-linux-i586.rpm</a>
Linux x86	185.69 MB	<a href="#">jdk-8u191-linux-i586.tar.gz</a>
Linux x64	167.99 MB	<a href="#">jdk-8u191-linux-x64.rpm</a>
Linux x64	182.87 MB	<a href="#">jdk-8u191-linux-x64.tar.gz</a>
Mac OS X x64	245.92 MB	<a href="#">jdk-8u191-macosx-x64.dmg</a>
Solaris SPARC 64-bit (SVR4 package)	133.04 MB	<a href="#">jdk-8u191-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	94.28 MB	<a href="#">jdk-8u191-solaris-sparcv9.tar.gz</a>
Solaris x64 (SVR4 package)	134.04 MB	<a href="#">jdk-8u191-solaris-x64.tar.Z</a>
Solaris x64	92.13 MB	<a href="#">jdk-8u191-solaris-x64.tar.gz</a>
Windows x86	197.34 MB	<a href="#">jdk-8u191-windows-i586.exe</a>
Windows x64	207.22 MB	<a href="#">jdk-8u191-windows-x64.exe</a>

# Setting up environment

- ☕ Accept the License Agreement and for Windows select Windows X86 or Windows x64 depending upon whether your Windows OS is 32 bit or 64 bit.
- ☕ Same is applicable for Linux, Linux ARM or Solaris OS.
- ☕ However, for Mac OS X only 64-bit JDK is available.
- ☕ In Windows double click on the downloaded executable file and install the JDK.
- ☕ The setup takes care of setting up PATH ENV Variable in Windows 7 and above for JRE (Java Runtime Environment) only. JDK and tools required for Java development is to be setup manually

# Setting up environment



# Setting up environment

- ☕ Path is a System/OS environment variable that OS searches for executing commands that are not part of Windows. You have to provide location of the libraries that you want to be include for execution of certain programs.
- ☕ Start → Right Click Computer → Select Properties → Select Advanced System Setting on Left Pane → Click Environment Variables.
- ☕ A modal dialog appears that shows **User variables** and **System Variables**.
- ☕ System variables are setup whenever you install an executable program.
- ☕ It's the **User environment** variable that you should configure.
- ☕ **Caution: Do not temper System Environment Variables**

# Setting up environment

- ☕ Select Path in the **User Variables** section → Click Edit Button
- ☕ A modal dialog appears with **Variable Name**: Path and **Variable value**.
- ☕ In the Variable value append a semicolon at the end of the value;
- ☕ After the semicolon, enter the path where java, javac, javap commands are located.  
Eg: C:\Program Files\Java\jdk1.8.0\_92\bin
- ☕ Push OK button → OK → OK
- ☕ Now open a command prompt and type java -version. You will see output as below:

```
java version "1.8.0_92"  
Java(TM) SE Runtime Environment (build 1.8.0_92-b14)  
Java HotSpot(TM) Client VM (build 25.92-b14, mixed mode)
```

# Setting up environment

- ☕ In the **User Variables** section, you can also declare the classpath variable.
- ☕ The classpath variable will be set to location where your compiled java .class files can be found.
- ☕ In the **User Variables** section, click on new button. A modal dialog appears with the title “**New User Variable**”.
- ☕ Enter the Variable Name → classpath.
- ☕ Enter the Variable Value → Location where .class files or .jar files are kept.
- ☕ Every location entered in the classpath must end with a semicolon;

# Compiling & Execution

- ☕ Once, all your environment variables are set, its time for seeing Java in Action
- ☕ Open a command prompt and change to the directory where you will be creating your Java source code. Eg. `cd C:\Java\Tuts`
- ☕ In a text editor like notepad, notepad++, edit plus or some other editor create a java file with the same content as in “**First Java Program**”
- ☕ Save the file with a meaningful name like `FirstProgram.java`
- ☕ The file name must not contain any spaces or any special symbol other than `_` or `$`.
- ☕ Now compile the `*.java` file using `javac` command → `javac FirstProgram.java`
- ☕ If all goes well then no error is displayed.
- ☕ Now run your program using `java` command → `java FirstProgram`

```
C:\Java\Tuts>java FirstProgram
```

**Error: Could not find or load main class FirstProgram**

# Execution of Java Program

☕ Note that the environment variables are not able to find any class with name `FirstProgram` after compiling file `FirstProgram.java` using `javac`

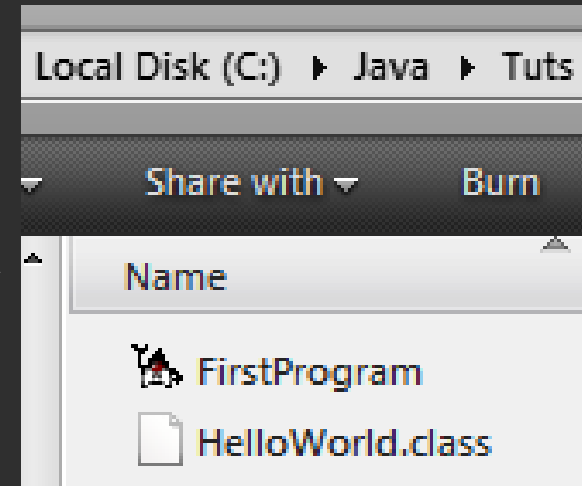
☕ On browsing the directory you'll see that there is no `.class` file with name `FirstProgram` rather there's `HelloWorld.class` file.

☕ The `HelloWorld.class` file is created as the name of the class was `HelloWorld` and not `FirstProgram`.

☕ So `javac` compiler will create `.class` files for all the classes declared in the `FirstProgram.java` file.

☕ You can run the `.class` file using `java HelloWorld`

☕ Bingo → `Welcome to Java`





# First Program in Java with Arguments

```
class HelloWorld {  
    public static void main(String [] a) {  
        Console.WriteLine("%s, Welcome to Java!", a[0]);  
    }  
}
```

command

Class name

0<sup>th</sup> Argument

java HelloWorld Students

Students, Welcome to Java!

# Dissecting the `main()` method

- ☕ Every Java application needs a `main()` method in order to execute it.
- ☕ The JVM invokes the `main()` method that has a fixed signature i.e. `public static void main` and the main method takes an array of Strings.
- ☕ `main()` method is invoked by a special thread called “main thread in Java”.
- ☕ `public`. The `main()` can be called by JVM from anywhere outside the application
- ☕ `static`. The method can be accessed without object instance. `static` data and method are loaded into a separate memory inside JVM.
- ☕ `String ... args/String[] args`. This is the default argument to the `main()` method. During execution of a Java program you can pass these arguments with `java` command. Every argument is separated by a space.

# Dissecting the `main()` method

☕ `void`. Unlike C/C++, `main()` method in Java is `void`. When JVM starts, it will run `main()` method, but when `main()` finishes, it doesn't mean that JVM terminates. JVM continues to execute all threads until:

☕ `Runtime.exit()` is called

☕ All normal (not daemon) threads have died. Daemon threads do not count for this second condition. In other words ... if `main()` method spawns some normal threads, JVM will **not** terminate when `main()` finishes. If `main()` doesn't spawn any threads, JVM will terminate. If `main()` spawns only daemon threads, JVM will also terminate when `main()` finishes.

☕ `main()`. This is the name of the method. Notice it is in lowercase because Java is case sensitive. So `Main()` and `main()` are treated as different methods in Java. These are all valid declarations that serves as entry point to JVM :

```
☕ public final synchronized strictfp static void main(String[] args)
☕ public synchronized strictfp static void main(String[] args)
☕ public final strictfp static void main(String[] args)
☕ public strictfp static void main(String[] args)
☕ public static void main(String[] args)
☕ public static void main(String args[])
☕ public strictfp static void main(String ... args)
```

# Command line arguments to java

```
class HelloWorld {  
    public static void main(String [] a) {  
        for(int i=0; i<a.length; i++)  
            Console.WriteLine("%s, Welcome to Java!",a[i]);  
    }  
}
```

command

Class name

Arg 0<sup>th</sup>  
Index

Arg 1<sup>st</sup>  
Index

Arg 2<sup>nd</sup>  
Index

java HelloWorld Tracy Stacey Macy

Tracy, Welcome to Java!

Stacey, Welcome to Java!

Macy, Welcome to Java!

# Online Java IDE & Editors

With the advent of cloud computing, you can also create, store, compile and debug your java programs online using various sites like:

[https://www.onlinegdb.com/online\\_java\\_compiler](https://www.onlinegdb.com/online_java_compiler)

<https://www.compilejava.net/>

Eclipse Che (using Docker). You too can install it on your Linux, OS X and Windows 10 systems.

# Java IDE

☕ As you can see that for compiling and executing very simple Java programs you have to set up various environment variables time-to-time so creating Java programs using editors is not very handy.

☕ Also, large projects that comprises of hundreds of .java source files and thousands of classes, things can go out of hand.

☕ In such cases, use of IDE is advisable. There are number of IDE(s) where you can easily create, compile, debug and execute your programs/projects. These IDE(s) also have AI in form of intelli-sense that suggest improvements in code and index classes and libraries for faster access.

☕ IntelliJ IDEA, eclipse, Netbeans are such IDE(s) which eases Java project development

*Thank You !*

# PRAGMATIC Audacity

You can tolerate us at:



Mohammed Umar Ali Karimi