Programming using Java

Java Basics

THE JAVA BUZZWORDS

- Simple
- Object oriented
- Network Savvy
- Robust
- Secure
- Architecture Neutral
- Portable
- Interpreted
- High Performance
- Multithreaded
- Dynamic

Simple

system more comprehensible – can be programmed easily

Object Oriented

It focuses on data and the interfaces to that data (object).

Network Savvy

It has extensive library of routines for coping with TCP/IP Protocols like HTTP and FTP. Java applications can open and access objects across net via URLs with the same ease as when accessing a local file system

Robust

It is reliable in many ways. It puts a lot of emphasis on early checking, runtime checking and eliminating situations that are error prone. Java has a pointer model that eliminates overwriting memory and corrupting data.

Secure

Java is used in networked / distributed environments. capability to be virus free / tamper free.

Architecture Neutral

The compiled code is executable on many processors, given the presence of Java Runtime system.

Portable

The libraries are a part of system defined portable interfaces. Example, there is an abstract Window class and its implementations of it for WINDOWS, UNIX and the MACINTOSH.

Interpreted

The Java interpreter can execute Java bytecodes directly on any machine to which the interpreter has been ported.

High Performance

Just in time (JIT) compiler can monitor which code is executed frequently and optimize just that code.

Multithreaded

It is very easy in JAVA. Threads in JAVA can take advantage of multi processor systems if the base operating system does so.

The code for calling threads remains the same across the machines, Java offloads the implementation of multithreading to the thread library.

For server side development it is an appealing language.

Dynamic

It finds out runtime information, it adapts to the evolving environment. Libraries can freely add new methods and instance variables without any effects on clients.

TO BE DISCUSSED ….

- Background
- The Java Virtual Machine
- Applications & Applets
- Classes & Objects
- The 3 Principles of Object Oriented Programming
- Start up Java
- Variables & Assignments
- Strings & Characters
- Arithmetic Operators & Expressions
- Type Conversion
- Comments
- Arrays
- Keywords

Background / History

- Designed by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems in 1991.
- The original motivation is not Internet: platform-independent software embedded in consumer electronics devices.
- With Internet, the urgent need appeared to break the fortified positions of Intel, Macintosh and Unix programmer communities.
- Java as an “Internet version of C++”? No.

Background / History

- Java was not designed to replace C++, but to solve a different set of problems. There are significant practical/ philosophical differences.
- Write once, Run everywhere
- Spread Rapidly through WWW and Internet

Java Technology

There is more to Java than the language.

Java Technology consists of:

1) Java Programming Language
2) Java Virtual Machine (JVM)
3) Java Application Programming Interfaces (APIs)

Execution Platform

What is an execution platform?

- An execution platform is the hardware or software environment in which a program runs, e.g. Windows, Linux, Solaris or MacOS.
- Most platforms can be described as a combination of the operating system and hardware.

Java Execution Platform

1) A software-only platform that runs on top of other hardware-based platforms.

2) Java Platform has two components:

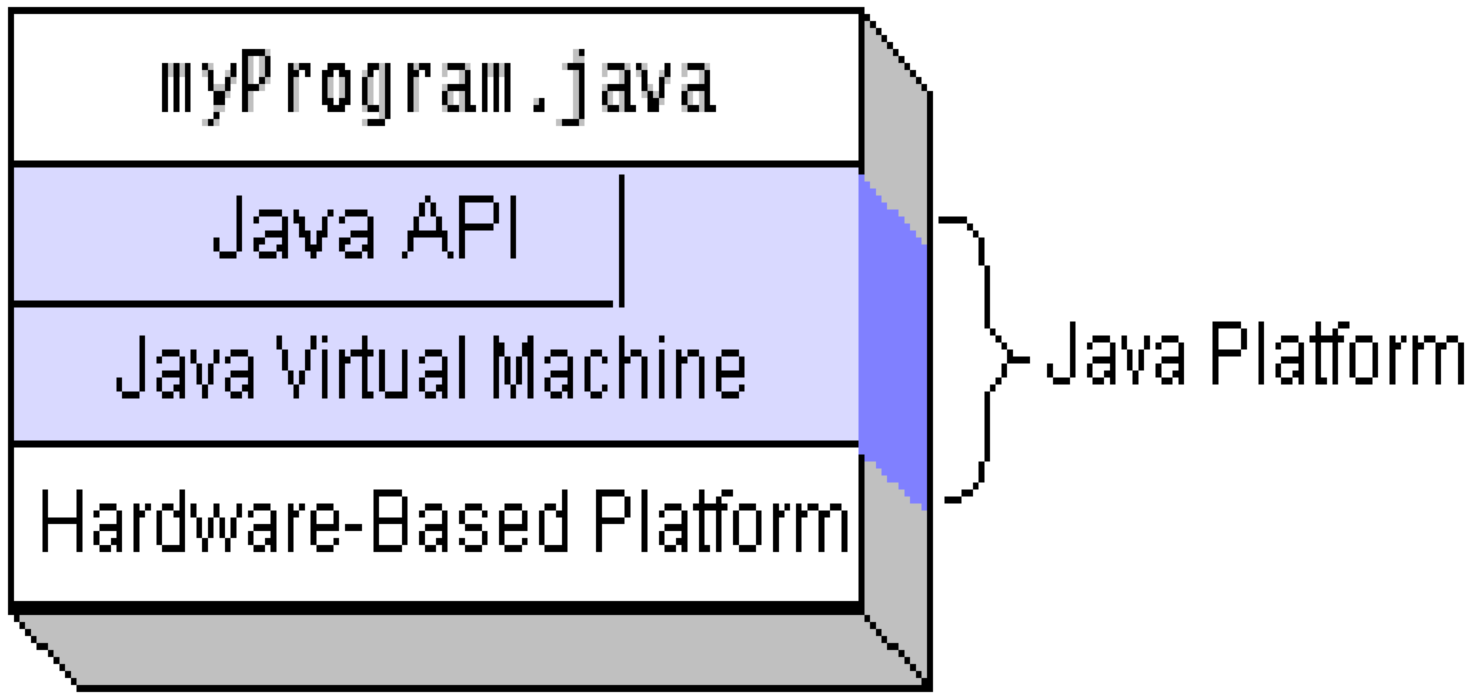
a) Java Virtual Machine (JVM) – interpretation for the Java bytecode, ported onto various hardware-based platforms.

b) The Java Application Programming Interface (Java API)

- a large collection of ready-made software components that provide many useful capabilities, e.g. graphical user interface grouped into libraries (packages) of related classes and Interfaces

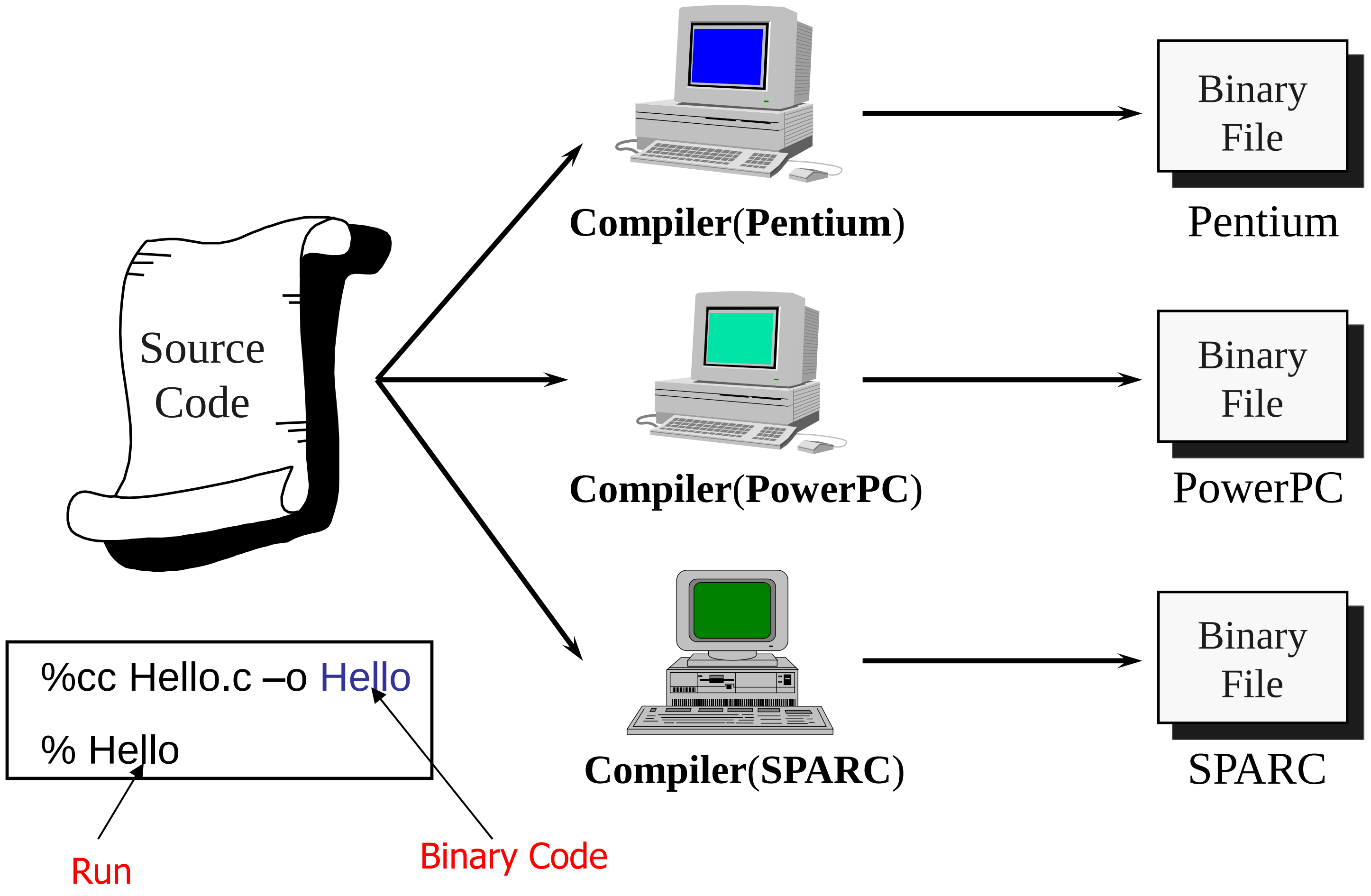
Together with JVM, insulates Java programs from the hardware and operating system variations

Java Execution Platform



Traditional programs

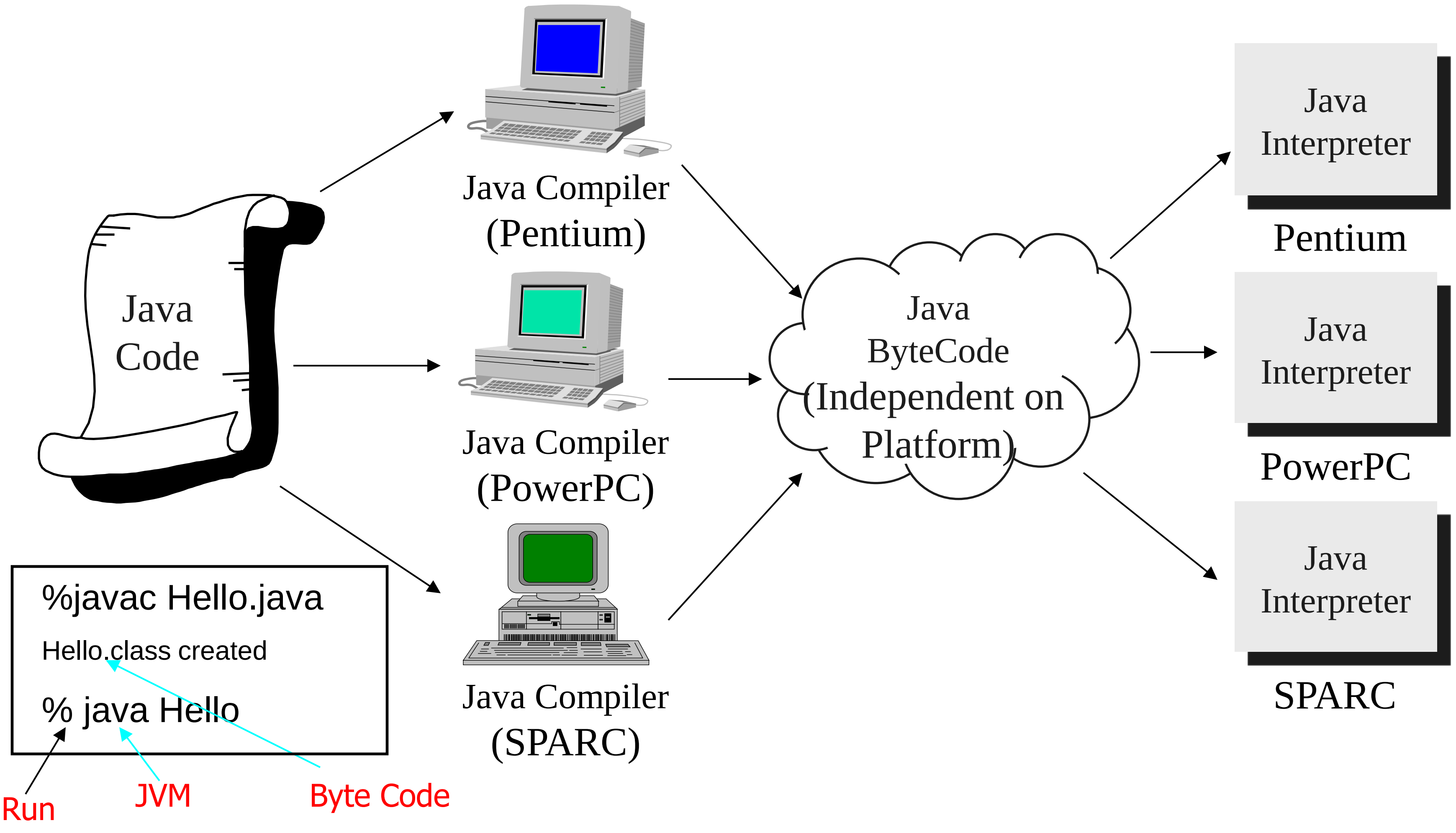
- Existing Development Environment



```
%cc Hello.c –o Hello
% Hello
```

Bytecodes and the Java Virtual Machine

- Java Development Environment



```
%javac Hello.java
Hello.class created
% java Hello
```

Java Program Execution

Java programs are both compiled and interpreted:

Steps:

- write the Java program
- compile the program into bytecode
- execute (interpret) the bytecode on the computer through the Java Virtual Machine

Compilation happens once.

Interpretation occurs each time the program is executed.

Compiler vs. Interpreter

- Compiler:

  checks syntax

  generates machine-code instructions

  not needed to run the executable program

  the executable runs faster

- Interpreter:

  checks syntax

  executes appropriate instructions while interpreting the program statements

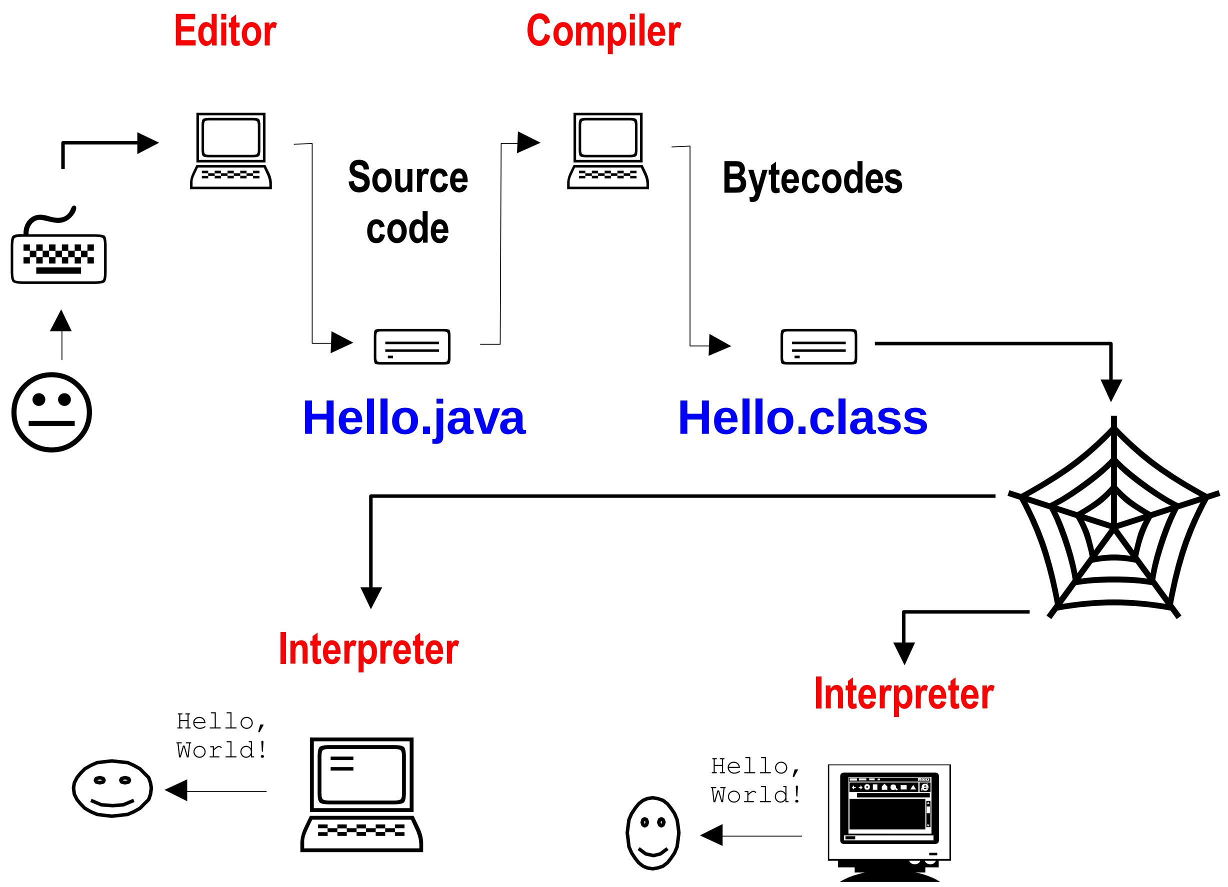
  must remain installed while the program is interpreted

  the interpreted program is slower

Java's Hybrid Approach: Compiler + Interpreter

- A Java compiler converts Java source code into instructions for the *Java Virtual Machine*.
- These instructions, called *bytecodes*, are the same for any computer / operating system.
- A Java interpreter executes bytecodes on a particular computer.

❑Java's Compiler + Interpreter



Why Bytecodes?

- Platform-independent.
- Load from the Internet faster than source code.
- Interpreter is faster and smaller than it would be for Java source.
- Source code is not revealed to end users.
- Interpreter performs additional security checks, screens out malicious code.

❑Java SDK (a.k.a. JDK) — Software Development Kit

- javac
  - Java compiler
- java
  - Java interpreter
- appletviewer
  - tests applets witho ut a browser
- jar
  - packs classes into jar files (packages)
- javadoc
  - generates HTML do cumentation ("docs ") from source
- jdb
  - command-line deb ugger

All these are

command-line tools, no GUI

Java SDK (cont'd)

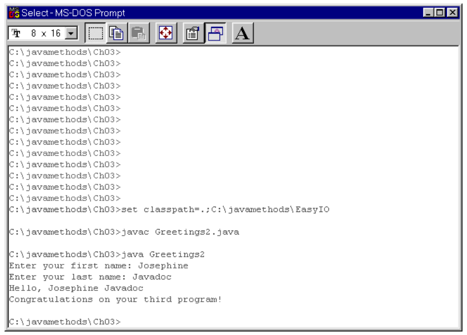
- Available free
- All documentation is online
- Lots of additional Java resources on the Internet

Java IDEs

- GUI front end for SDK
- Integrates editor, javac, java, appletviewer, debugger, other tools:
  - specialized Java editor with syntax highlighting, autoindent, tab setting, etc.
  - clicking on a compiler error message takes you to the offending source code line
- Some IDEs have their own copy of SDK; others need SDK installed separately.

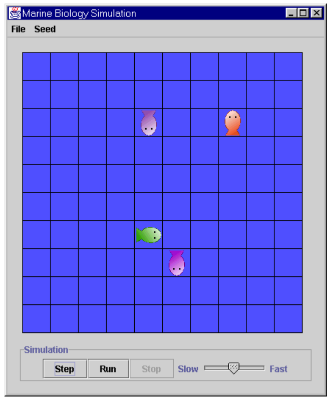
❑Types of Programs:

- Console applications

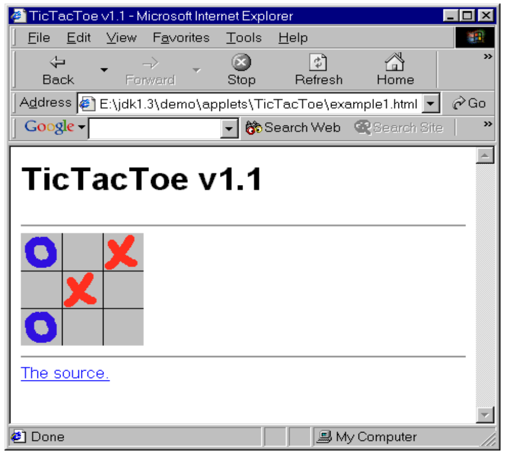


```
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>
C:\javamethods\Ch03>set classpath=.;C:\javamethods\EasyIO

C:\javamethods\Ch03>javac Greetings2.java

C:\javamethods\Ch03>java Greetings2
Enter your first name: Josephine
Enter your last name: Javadoc
Hello, Josephine Javadoc
Congratulations on your third program!

C:\javamethods\Ch03>
```

- GUI applications



- Applets



Java Program

- Java Application Program
  - Application
    - Program written in general programming language
  - Applet
    - Program running in Web Browser Environment
    - Can be viewed by appletviewer or Web browser with JVM

The Java Class Libraries

- java.applet : Applet related
- java.awt : Abstract Window Toolkit
- java.awt.event : Event process from awt component
- java.awt.image : Image processing
- java.beans : JavaBeans Component
- java.io : File or Network I/O Support
- java.lang : Java Language Support
- java.net : Network related functions
- java.util : Utility function

Java 2 SDK

- Java 2 Software Development Kit (Java 2 SDK)
  - Java Application or Applet Development and Running Environment

- Java 2 Runtime Environment (JRE)
  - Java Running Environment

- Java 2 Standard Edition
- Java 2 Micro Edition
- Java 2 Enterprise Edition

Start a Java Application

Java

C

Class Name

Main method

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

Main Function

```
"hello.c"
void main() {
  printf("Hello
  World\n");
}
```

Compile

```
% javac Hello.java
```

```
% cc hello.c –o hello
```

Run

```
% java Hello
```

```
% hello
```

output

```
Hello World
```

Java Virtual Machine

- The .class files generated by the compiler are not executable binaries
    - so Java combines compilation and interpretation
- Instead, they contain “byte-codes” to be executed by the Java Virtual Machine
- This approach provides platform independence, and greater security

HelloWorld program

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

- Note that String is built in
- println is a member function for the System.out class

Comments are almost like C

```
/* This kind of comment can span
   multiple lines
*/

//This kind is to the end of the line

/**
    * This kind of comment is a special
    * ‘javadoc’ style comment
 */
```

Primitive data types are like C

- Main data types are int, double, boolean, char
- Also have byte, short, long, float
- boolean has values true and false
- Declarations look like C, for example,
  - double x, y;
  - int count = 0;

Variables and Assignments

- Variables types

char	16 bits Unicode character data
boolean	Boolean Variable
byte	8 bits signed integer
short	16 bits signed integer
int	32 bits signed integer
long	64 bits signed integer
float	32 bits signed floating point number
double	64 bits signed floating point number

Variables and Assignments

Variable Declaration

```
type varName;
```

float x, y, x;

Value assignments

```
varName = value;
```

int num = 100;

long m = 21234234L

double type : .5   0.8   9e-2 -9.3e-5

float type : 1.5e-3f;

Strings and Characters

- String : sequence of character

  String s = “Enter an integer value: ” ;

  Char c = ‘A’; $\longleftarrow$ Unicode

- Concatenation Operator ‘+’

String s = “Lincoln said: ” + “\” Four score ago\”” ;

Result :

Lincoln said: “Four score ago”

Expressions are like C

- Assignment statements mostly look like those in C
- you can use `=`, `+=`, `*=` etc.
- Arithmetic uses the familiar `+ - * / %`
- Java also has `++` and `--`
- Java has boolean operators `&& || !`
- Java has comparisons `< <= == != >= >`
- Java does *not* have pointers or pointer arithmetic

Arithmetic Operators

- Operators
  - +   -   * / %
  - +=   -=   *=   /=   %=
  - ++   --

    5 / 2 → 2 Why isn't it 2.5 ?

    5 % 2 → 1

    4 / 2 → 2

    4 % 2 → 0
  - count * num + 88 / val – 19 % count
  - j += 6; ➔ j = j + 6;

Operator Precedence

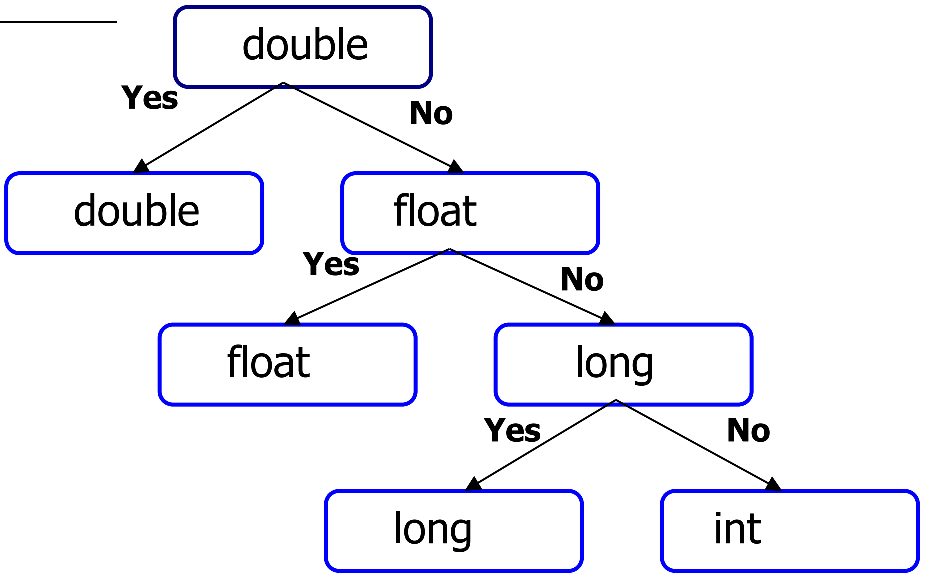
Operator	Association	Precedence
() [] .	Left Assoc.	(High)
! ~ ++ -- + - (Data Type)	Right Assoc.	
* / %	Left Assoc.	
+ -	Left Assoc.	
<< >> >>>	Left Assoc.	
< <= > >= instance	Left Assoc.	
== !=	Left Assoc.	
&	Left Assoc.	
^	Left Assoc.	
|	Left Assoc.	
&&	Left Assoc.	
||	Left Assoc.	
? :	Right Assoc.	
= += -= *= /= %= &= ^= |= <<= >>= >>>=	Right Assoc.	(Low)

Type Conversions in Expression

char ch;   int i;   float f;   double outcome;

ch = ‘0’;   i = 10;   f = 10.2f;

$$\text{outcome} = \underbrace{\underbrace{\underbrace{\text{ch} \;*\; \text{i}}_{\textbf{int}} \;/\; \text{f;}}_{\textbf{float}}}_{\textbf{double}}$$



Type Conversions in Assignment

- Widening Conversion

```
byte b = 127;
int i;
i = b;
```

- Wrong Conversion

```
~~byte b;~~
~~int i = 127;~~
~~b = i;~~
```

- Right Conversion (But data may be lost)

```
byte b;
int i = 127;
b = (byte) i;
```

Type Casting

Increment & Decrement Operator

- Operator
  - ++, --
  - Prefix operator

```
n = 1;
x = ++n;      // x=2, n=2
```

  - Postfix operator

```
n = 1;
x = n++;      // x=1, n=2
```

  - Cannot be use with expressions, only with variables
  - Cannot be applied at the real type

```
(a + b)++    // error
```

Back-Slash Codes

\b	back space
\n	new line
\r	carriage return
\f	form feed
\t	tab
\”	double quotation
\’	single quotation
\0	null
\\	back slash
\uxxxx	Unicode (x: hexa deci mal)

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

System.out.print(“\u00a0
\u00a1 \u00a2 \u00a3”);
  }
}
```

Bitwise Operators

- Operator
  - &, |, <<, >>, >>>, ^, ~
  - Operand should be integer type
  - Precedence

Operator	**Precedence**
~	(H)
<< >> >>>	↕
&	↕
^	↕
|	(L)

Ternary Operator

- Operator
  - Expr1 ? Expr2 : Expr3 (3 Terms Operator)

```
if (x > y)  max = x;
else  max = y;
```

```
max = x > y ? x : y ;
```

```
m = a > b ? (c > a ? c : a) : (c > b ? c : b) ;
```

Assignment Operators

Expr **1** = Expr **1** op Expr2 $\longleftrightarrow$ Expr1 op= Expr 2

- Operator
  - Arithmetic operator : + - * / %
  - Bitwise operator : & | ^ << >> >>>

sum = sum + i ; $\rightleftarrows$ sum += i ;

x = x * y + 1; $\not\rightleftarrows$ x *= y + 1;

x = x * (y+1)

Cast Operators

- Data Type Casting Operator

(Data Type) Expression

- Cast operator : ( )

(int) 3.75	===>	3
(float) 3	===>	3.0
(float) (1 / 2)	===>	0.0
(float)1/2	===>	0.5

Control statements are like C

- if (x < y) smaller = x;
- if (x < y){

      smaller = x; sum += x;}
  else { smaller = y; sum += y; }
- while (x < y) {

      y = y - x; }
- do { y = y - x; }

  while (x < y)
- for (int i = 0; i < max; i++)sum += i;
- BUT: conditions must be **boolean** !

Control statements II

```
switch (n + 1) {
  case 0: m = n - 1; break;
  case 1: m = n + 1;
  case 3: m = m * n; break;
  default: m = -n; break;
}
```

- Java also introduces the `try` statement, about which more later

Arrays

Definition of One Dimensional Array

type VarName[]

```
int ia[];
```

Assign Range to One Dimensional Array

varName = new type[size]

```
ia = new int[10];
```

Declaration of One Dimensional Array with Range

type varName[] = new type[size]

```
int ia[] = new int[10];
```

Java Keywords

- 50 Java Keywords

abstract	double	int	super
boolean	else	interface	switch
break	extends	long	synchronized
byte	final	native	this
case	finally	new	throw
catch	float	package	throws
char	for	private	transient*
class	goto*	protected	try
const*	if	public	void
continue	implements	return	volatile
default	import	short	while
do	instanceof	static	strictfp

assert (New in 1.5) enum (New in 1.5)

Simple Java Program

A class to display a simple message:

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

Running the Program

Type the program, save as **Hello**.java.

In the command line, type:

```
> ls
Hello.java
> javac Hello.java
> ls
Hello.java, Hello.class
> java Hello
Hello World!
```

Explaining the Process

1) creating a source file - a source file contains text written in the Java programming language, created using any text editor on any system.

2) compiling the source file Java compiler (javac) reads the source file and translates its text into instructions that the Java interpreter can understand. These instructions are called bytecode.

3) running the compiled program Java interpreter (java) installed takes as input the bytecode file and carries out its instructions by translating them on the fly into instructions that your computer can understand.

Java Program Explained

`Hello` is the name of the class:

class **`Hello{`**

main is the method with one parameter args and no results:

*public static void **main**(String[] args) {*

println is a method in the standard System class:

System.out.println("First Java program.");

}

}

Main Method

The main method must be present in every Java application:

1)*public static void main(String[] args)* where:

a)public means that the method can be called by any object

b)static means that the method is shared by all instances

c)void means that the method does not return any value

2) When the interpreter executes an application, it starts by calling its main method which in turn invokes other methods in this or other classes.

3) The main method accepts a single argument – a string array, which holds all command-line parameters.

Getting Started:

(1) Create the source file:

- open a text editor, type in the code which defines a class (*HelloWorldApp*) and then save it in a file (*HelloWorldApp.java*) file and class name are case sensitive and must be matched exactly (except the `.java` part)

Example Code: HelloWorldApp.java

```
/**
 * The HelloWorldApp class implements an application
 * that displays "Hello World!" to the standard output
*/
public class HelloWorldApp {
  public static void main(String[] args) {
    // Display "Hello World!"
    System.out.println("Hello World!");
   }
}
```

★ `Java is CASE SENSITIVE!`

Getting Started:

(2) Compile the program:

- compile HelloWorldApp.java by using the following command:

```
javac HelloWorldApp.java
```

it generates a file named HelloWorldApp.class

Getting Started:

(3) Run the program:

- run the code through:

```
java HelloWorldApp
```

- Note that the command is `java`, not `javac`,
- and you refer to `HelloWorldApp`, not `HelloWorldApp.java` or `HelloWorldApp.class`

Modify:

(4) Modify the program:

- Declare 2 variables a and b of type int.
- Multiply them
- Print the output as:

```
        a * b = Value obtained
```

- You can copy paste any c program you have made and it will work (by taking care of issues discussed in the previous slides)

Java isn't C!

- In C, almost everything is in functions
- In Java, almost everything is in classes
- There is often only one class per file
- There *must* be only one `public` class per file
- The file name *must* be the same as the name of that public class, but with a `.java` extension