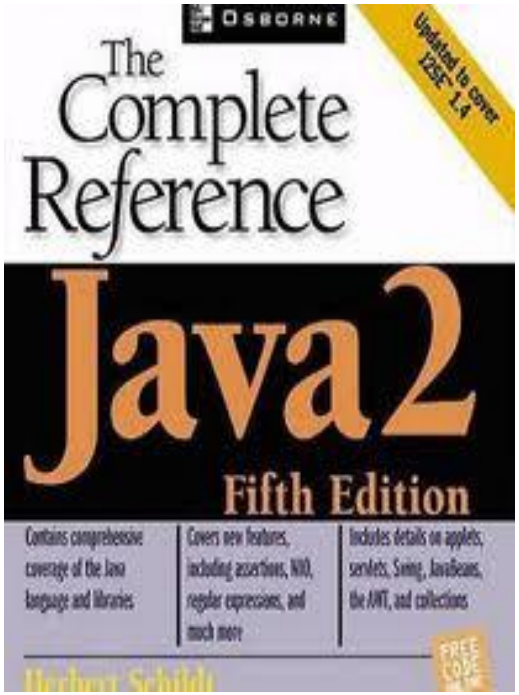
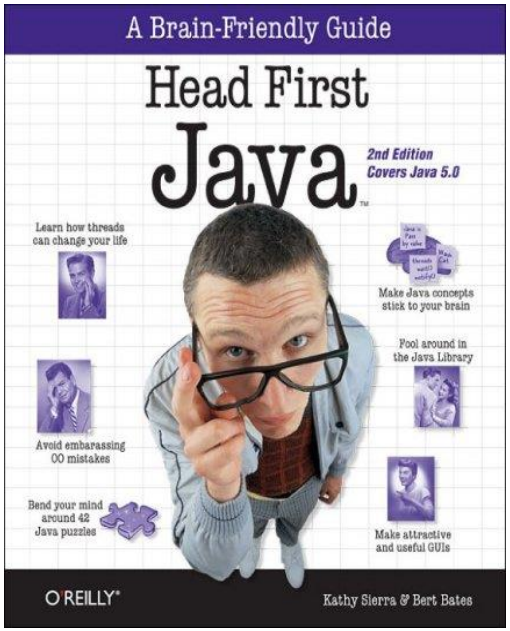
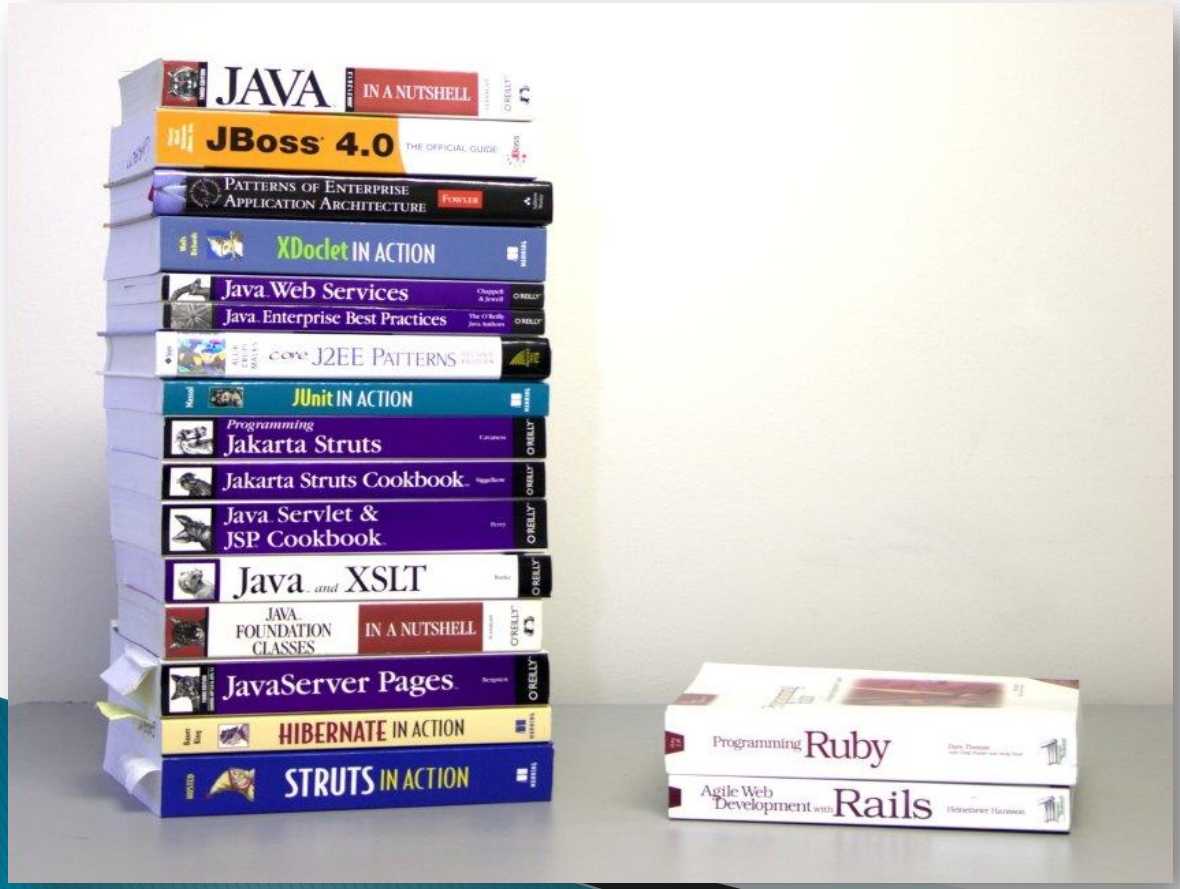
Object Oriented Programming

Text Books

- The Complete Reference Java 2 by Herbert Schildt







Grading Criteria

- Quiz 10%
- Assignment (Class + Lab) 10%
- Lab (Lab Performance and Project) 20%
- Mid Exam 20%
- Final Exam 40%

CLASSROOM ETIQUETTE:

- Arrive at class on time
- Do be prepared for class
- Most classes will begin on time and end on time. If you need to know about schedule or assignment changes, kindly ask about them from the class representative. If you have a real need to leave the class early, kindly inform at the beginning of class and then leave quietly.
- Do put cell phones away and on vibrate…Texting should not be going on in class
- Do stay AWAKE and be attentive

Topics To be Covered Today

- What is a Program?
- Evolution of Programming Languages
- Why Java
- Java History
- Versions of Java
- Features of Java
- Java Procedure

What is a Program?

- A computer program (also a software program, or just a program) is a sequence of instructions written to perform a specified task for a computer.

d:\Src\Demo\Demo.java

File Edit Goto Macro New!

```
import java.applet.*;
import java.awt.*;

public class Demo extends Applet {
    Image image;
    int count;

    public void init()
    {
        image = getImage(getDocumentBase(), "World.jpg");
        count = 1;
    }

    public void paint(Graphics g)
    {
        g.drawImage(image, 0, 0, this);
        g.setColor(Color.red);
        for (int y = 15; y < size().height; y += 15) {
            int x = (int) (size().width/2 + 30*Math.cos(Math.PI*y/75));
            g.drawString("Hello", x, y);
        }
        showStatus("Paint called " + count + " time" + ((count > 1) ? "s" : ""));
        count += 1;
    }
}
```

Line 12 Col 6

The Evolution of Programming Languages

- Machine language: 1940's
- Assembly language: early 1950's
- Higher-level languages: late 1950's
  - Fortran: scientific computation
  - Cobol: business data processing
  - Lisp: symbolic computation
- Today: thousands of programming languages

http://www.howtogeek.com/94917/the-evolution-of-computer-programming-languages-infographic/

Machine Languages

- Comprised of 1s and 0s
- The ***"native" language*** of a computer
- Difficult to program – one misplaced 1 or 0 will cause the program to fail.
- Example of code:

```
1110100010101        111010101110
10111010110100       10100011110111
```

Assembly Languages

- Assembly languages are a step towards easier programming.
- Assembly languages are comprised of a ***set of elemental commands*** which are tied to a specific processor.
- Assembly language code needs to be translated to machine language before the computer processes it.
- Example:

```
ADD  1001010, 1011010
```

High-Level Languages

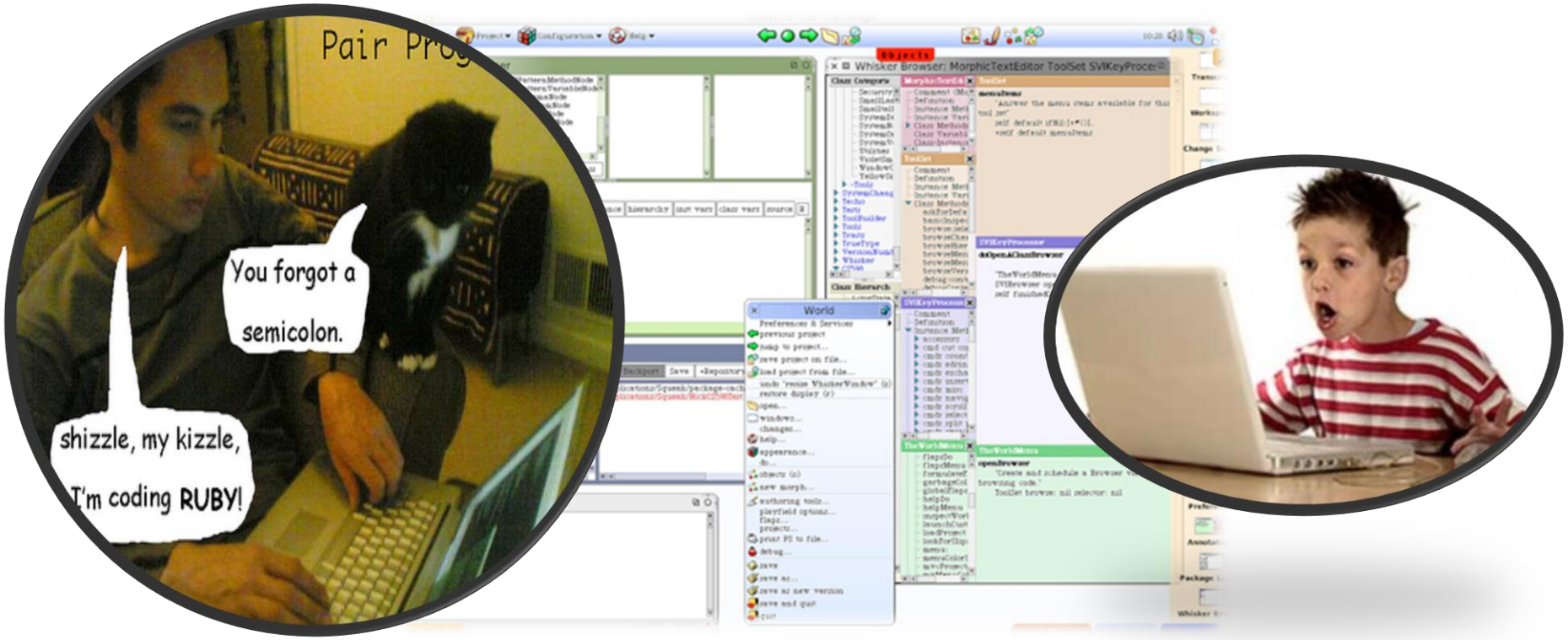
- High-level languages represent a giant leap towards easier programming.
- The syntax of HL languages is similar to English.
  - Example:

    ```
    Student_name = first_name + last_name
    ```

  - Interpreter – Executes high level language programs without compilation.
- Historically, we divide HL languages into two groups:
  - *Procedural languages*
  - *Object-Oriented languages (OOP)*

What is Programming?

- When we say “programming” we are actually referring to the science of transforming our intentions in a high-level programming language.



What are we doing in this course?

- Learn programming in a high-level programming language.
- We will study Object-Oriented Programming using '**Java**', a popular high-level object-oriented programming language.

Why Java?

- It's the current "hot" language
- Java mean **Cup of coffee**
- It's almost entirely object-oriented
- It has a vast library of predefined objects and operations
- It's more platform independent
  - this makes it great for Web programming
- It's more secure
- It isn't C++

Java History

- Java
  - was created in 1991
  - by James Gosling et al. of Sun Microsystems.
  - Initially called **Oak**, in honor of the tree outside Gosling's window, its name was changed to **Java** because there was already a language called **Oak**.

Java History contd. . . . .

- The term *Java* actual refers to more than just a particular language like C or Pascal. Java encompasses several parts, including :
  - **A high level language** ⊕ the Java language is a high level one that at a glance looks very similar to C and C++ but offers many unique features of its own.
  - **Java bytecode** - a compiler, such as Sun's javac, transforms the Java language source code to bytecode that runs in the JVM.
  - **Java Virtual Machine (JVM)** a program, such as Sun's java, that runs on a given platform and takes the bytecode programs as input and interprets them just as if it were a physical processor executing machine code.

Java History contd. . . . .

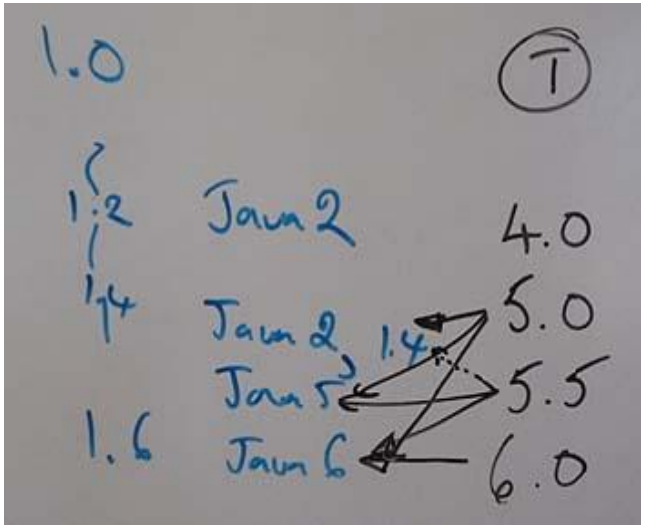
- Sun provides a set of programming tools such as javac, java and others in a bundle that it calls a Java *Software Development Kit* for each version of the language and for different platforms such as Windows, Linux, etc.. Sun also provides a runtime bundle with just the JVM when the programming tools are not needed.

Versions of Java

- Since its introduction, Sun has released a new version of the Java language every two years or so.
- These new versions brought enhancements, new capabilities and fixes to bugs.
- Until recently, the versions were numbered 1.x, where x reached up till 4. (Intermediate revisions were labeled with a third number - 1.x.y - as in 1.4.2.)
- The newest version, however, is called Java 5.0 rather than Java 1.5.

Versions of Java

- JDK 1.0 (January 23, 1996)
- JDK 1.1 (February 19, 1997)
- J2SE 1.2 (December 8, 1998)
- J2SE 1.3 (May 8, 2000)
- J2SE 1.4 (February 6, 2002)
- J2SE 5.0 (September 30, 2004)

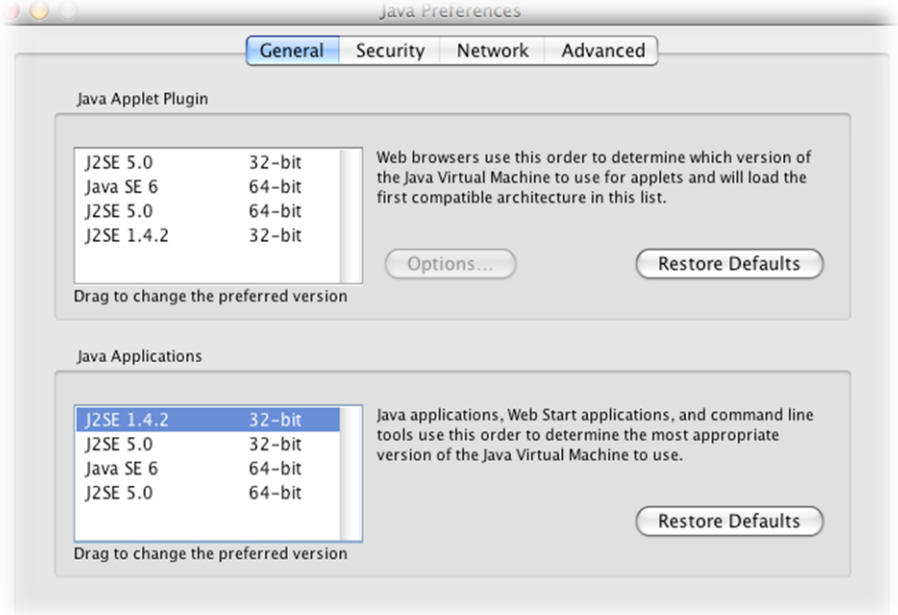


Versions of Java

- **Java SE 6 (December 11, 2006)**
  - Java SE 6 Update 10(Released October 15, 2008)
  - Java SE 6 Update 11(Released December 3, 2008)
  - Java SE 6 Update 12
  - Java SE 6 Update 14(Released May 28, 2009)
  - Java SE 6 Update 16(Released August 11, 2009)
  - Java SE 6 Update 17(Released November 4, 2009)
  - Java SE 6 Update 18(Released January 13, 2010)
  - Java SE 6 Update 19(Released March 30, 2010)
  - Java SE 6 Update 20(Released April 15, 2010)
  - Java SE 6 Update 21(Released July 7, 2010)

Versions of Java

- **Java SE 7.0**
  - Java 7 is a major update to Java (July 2011)
- **Java SE 8.0**
  - Java 7 is a major update to Java (March 2014)



https://en.wikipedia.org/wiki/Java_version_history

Features of Java

- **Simple**

- **object-oriented**
  - Code of the java Language is Written into the classes and Objects So this feature java is most Popular because it also Supports Code Reusability, Maintainability etc.

Features of Java

- **Java is platform Independent**
  - With **Java**, you can compile source code on Windows and the compiled code (bytecode to be precise) can be executed (interpreted) on any **platform** running a JVM. So yes you need a JVM but the JVM can run any compiled code, the compiled code is **platform independent**

Java vs C++

- **C**/C++ is **not platform independent**, because when we compile **C**/C++ source code, it will generate binary code which can be understood by current OS only. When we move this source code to other **platform** and then compile **C**/C++ binary code, the binary code generated cannot be understood by this new **platform**

Features of Java

- **Java is Multi-Threaded**. ... A single **Java** program can have many different threads executing independently and continuously.
  Three **Java** applets on the same page can run together with each getting equal time from the CPU

Features of Java

- **interpreted and high-performance** – Java programs are compiled into an intermediate representation – bytecode:
  - can be later interpreted by any JVM
  - can be also translated into the native machine code for efficiency.

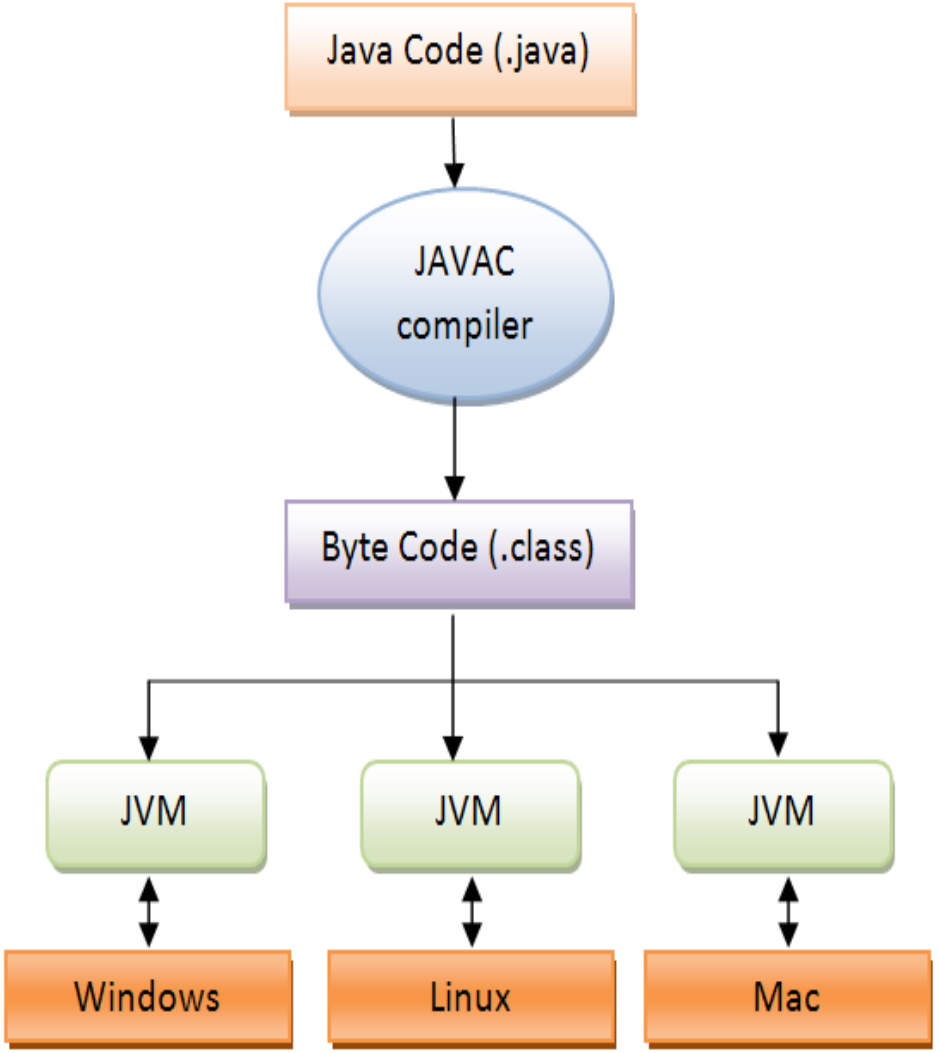
Features of Java

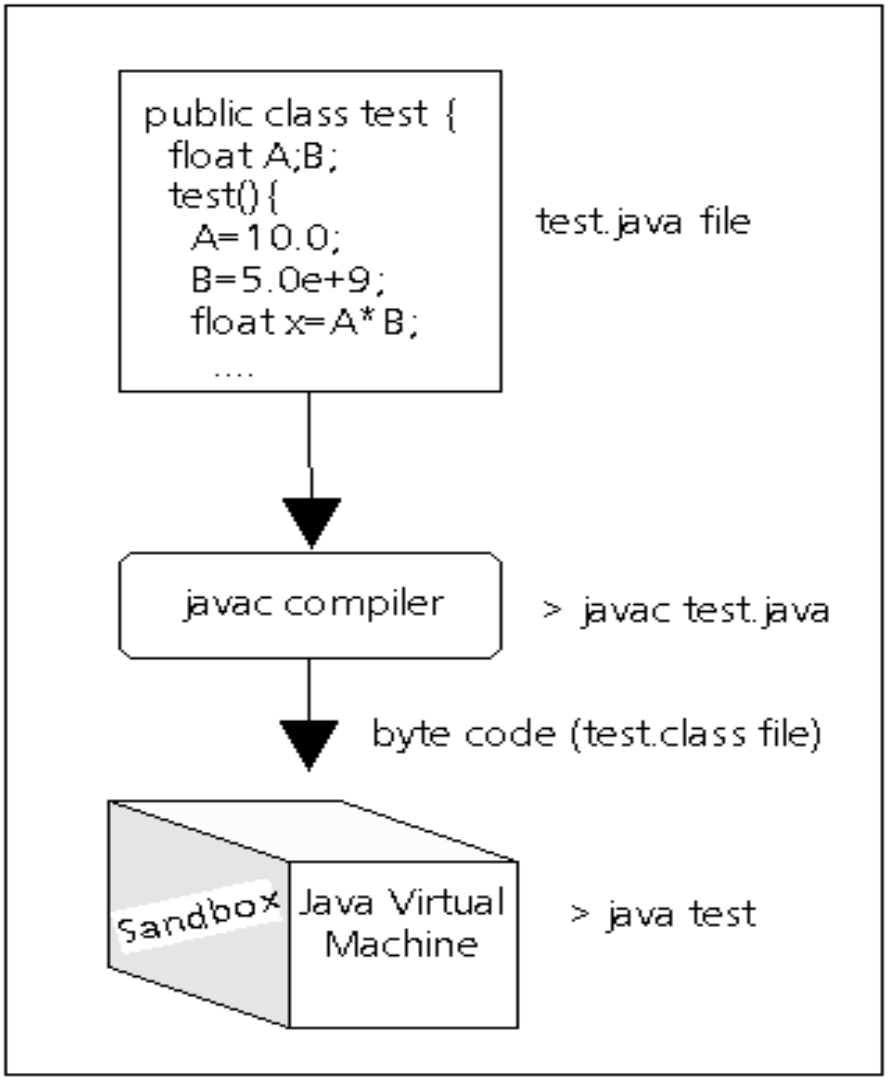
- **dynamic** – The linking of data and methods to where they are located, is done at run-time.
  - New classes can be loaded while a program is running. Linking is done *on the fly*.
  - Even if libraries are recompiled, there is no need to recompile code that uses classes in those libraries. This differs from C++, which uses static binding.

Java Procedure

- The essential steps to creating and running Java programs go as follows:
  - Create a Java source code file
  - Compile the source code
  - Run the compiled code in a Java Virtual Machine.

Steps for creating and running a Java program





Java Procedure Detail

- You create the code with a text editor (for example notepad) and save it to a file with the ".java" suffix.
- All Java source code files must end with this type name.
- The first part of the name must match the class name in the source code.
- In the figure the class name is Test so you must therefore save it to the file name Test.java.

Java Procedure Detail

- With the javac program, you compile this file as follows:

  C:> javac Test.java

- This creates a *bytecode* file (or files if the code file included more than one class) that ends with the ".class" type appended.
  - Here the output is Test.class.
  - The bytecode consists of the instructions for the *Java Virtual Machine* (JVM or just VM).

Java Procedure Detail

- The JVM is an interpreter program that emulates a processor that executes the bytecode instructions just as if it were a hardware processor executing native machine code instructions.
  - The Java bytecode can then run on any platform in which the JVM is available and the program should perform the same.
  - This *Write Once, Run Anywhere* approach is a key goal of the Java language.

Another Example with Bytecode Representation

Test.java

```
public class Test
{
   public static void main(String args[])
   {
      int i;
      i = 2;
      i = i + 7;
   }
}
```

Compile it with
> javac Test.java

Command used for reading bytecode file

```
C:\ > javap -c Test
Compiled from Test.java
public class Test extends java.lang.Object {
   public Test(); // a default constructor created
   public static void main(java.lang.String[]);
}

Method Test()
   0 aload_0
   1 invokespecial #3
   4 return

Method void main(java.lang.String[])
   0 iconst_2 // Put integer 2 on stack
   1 istore_1 // Store the top stack value at location 1
   2 iload_1  // Put the value at location 1 on stack
   3 bipush 7 // Put the value 7 on the stack
   5 iadd     // Add two top stack values together
   6 istore_1 // The sum, on top of stack, stored at location 1
   7 return   // Finished processing
```

Bytecode Representation

Programming Tools

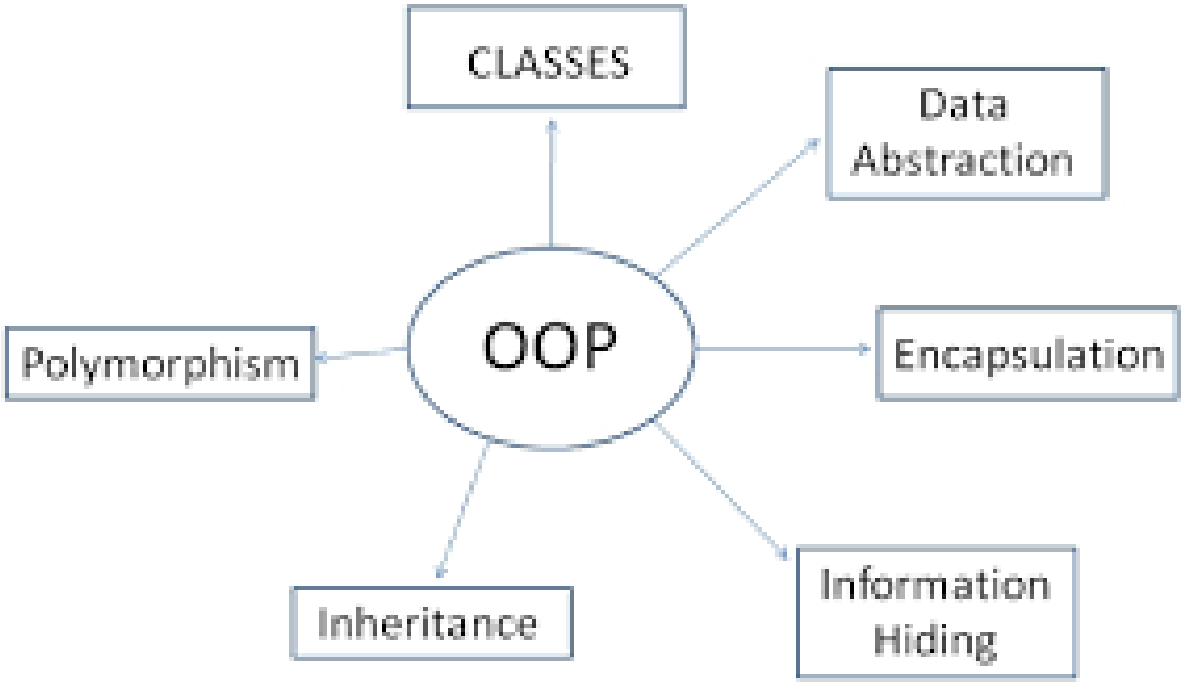
- **Integrated Development Environment** (IDE) - graphical user interface programming environments (often called *GUI Builders*) are elaborate, programs that allow you to interactively build graphical interfaces, edit the code, execute and run the applets and applications all within the IDE system. Example Java IDEs include:
  - **NetBeans**
  - Borland JBuilder
  - **Eclipse**
  - Dr Java

Programming Tools

Borland®
JBuilder®

OOP Principles

- Encapsulation
- Inheritance
- Polymorphism



Getting Started

To begin developing Java programs, follow these steps:

- **Step 1:** Obtain the Software Development Kit (SDK) for J2SE (Java 2 Platform, Standard Edition) or JDK
- **Step 2:** Install the JDK

Simple Java Application

Step 1: *Edit source code file* `HelloWorldApp.java`

```
public class HelloWorldApp
{ public static void main(...
```

Step 2: *Compile the source code file*

```
> javac HelloWorldApp.java
```

This creates bytecode file `HelloWorldApp.class`

Step 3: *Run the application*

```
> java HelloWorldApp
Hello World!
```

Simple Application

- **Step 1:** Use an editor to enter the following code for the HelloWoldApp program:

HelloWorldApp Application

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

- Save this code in a file called HelloWorldApp.java

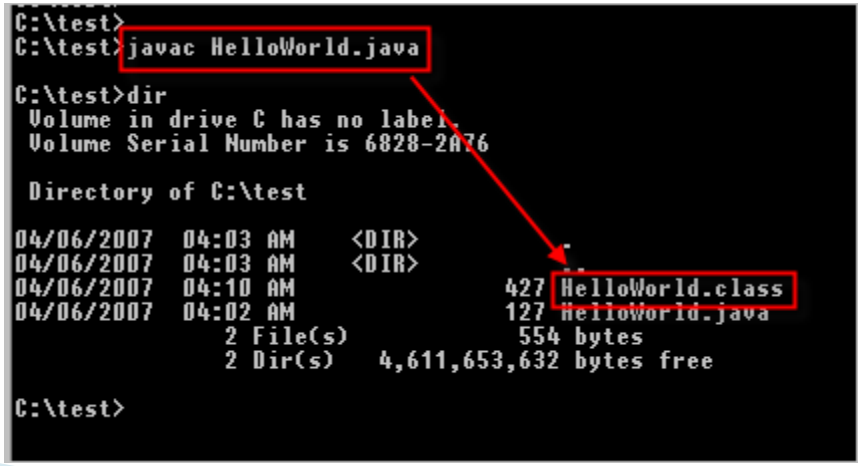
Simple Application

- **Step 2:** Compile the application with the command line:

  > javac HelloWorldApp.java

- This creates the class file (with the bytecode output):

  HelloWorldApp.class



Simple Application

- **Step 3:** Use the java command to run the program:

  > java HelloWorldApp

  Hello World!

  Output

- The output is printed after the command line.

Description

- The keyword **class** is used to declare that a new class is being defined.
- **HelloWorldApp** is an identifier that is the name of the class.
- The entire **class** definition, including all of its members, will be between the opening curly brace ({) and the closing curly brace (}).
- The use of the curly braces in Java is identical to the way they are used in C, C++.
- In Java, all program activity occurs within class. This is one reason why all Java programs are object-oriented.

Description

- The next line of code is

```
public static void main(String arg[])
```

- All Java applications begin execution by calling **main( ).**
- The **public** keyword is an access specifier, which allows the programmer to control the visibility of class members. When a class member is preceded by **public,** then that member may be accessed by code outside the class in which it is declared.
- **main( )** must be declared as **public,** since it must be called by code outside of its class when the program is started.

Description

- The keyword **static** allows main( ) to be called without having to instantiate a particular instance of the class. This is necessary since **main( )** is called by the Java interpreter before any objects are made.
- The keyword **void** simply tells the compiler that main( ) does not return a value.

Description

- **main( )** is the method called when a Java application begins. Java is case-sensitive. Thus, **Main** is different from **main.**
- It is important to understand that the Java compiler will compile classes that do not contain a **main( )** method. But the Java interpreter has no way to run these classes. So, if you had typed **Main** instead of **main,** the compiler would still compile your program. However, the Java interpreter would report an error because it would be unable to find the **main( )** method.

Description

- **String arg[ ]** declares a parameter named **arg**, which is an array of instances of the class **String.** *Objects of type* String store character strings.
- In this case, **arg** receives any command-line arguments present when the program is executed. This program does not make use of **String arg[ ].**
- **main( )** is simply a starting place for your program.
- A complex program will have dozens of classes, only one of which will need to have a **main( )** method to get things started.

Description

- The next line of code is shown here

```
System.out.println("Hello World!");
```

- This line outputs string **Hello World!**
- **System** is a predefined class that provides access to the system, and **out** is the output stream that is connected to the console.
- Output is actually accomplished by the built-in **println( )** method**. println( )** displays the string which is passed to it.

Control Statements

- If statement
- For statement

Lexical Issues

Whitespace

- Java is a free-form language.
- For example, the program could have been written all on one line or in any other strange way you felt like typing it.
- In Java, whitespace is a space, tab, or newline.

Lexical Issues

Identifiers

- Identifiers are used for class names, method names, and variable names.
- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- They must not begin with a number.
- Java is case-sensitive, so **VALUE** is a different identifier than **Value.**
- **Some examples of valid identifiers are:**

  AvgTemp count a4 $test this_is_ok
- Invalid variable names include:

  2count high-temp Not/ok

Lexical Issues

Literals

- A constant value in Java is created by using a *literal representation of it.*
- *For example,*here are some literals:

  100 98.6 'X' "This is a test"
- Left to right, the first literal specifies an integer, the next is a floating-point value, the third is a character constant, and the last is a string.
- A literal can be used anywhere a value of its type is allowed.

Lexical Issues

Comments

- There are three types of comments defined by Java. You already know two: single-line and multiline. The third type is called a *documentation comment.*
- This type of comment is used to produce an HTML file that documents your program.
- The documentation comment begins with a ***/**** **and ends with a */.**

Lexical Issues

Separators

- In Java, there are a few characters that are used as separators.
- The most commonly used separator in Java is the semicolon.

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a for statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

Lexical Issues

The Java Keywords

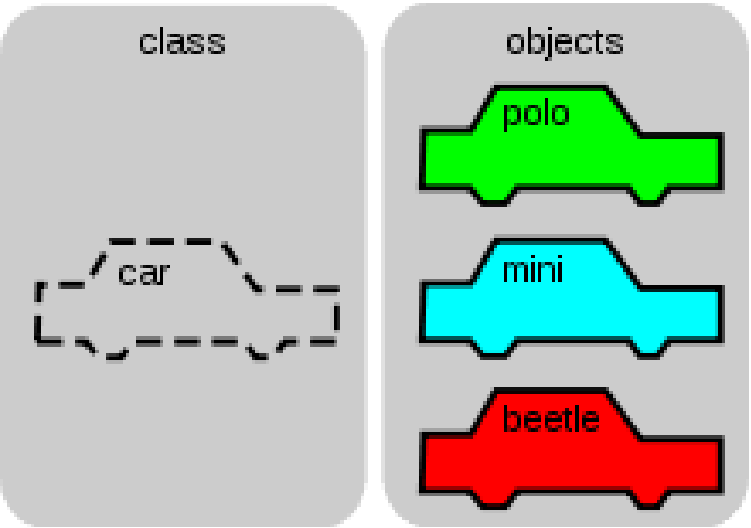
- There are 49 reserved keywords currently defined in the Java language

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

- The keywords **const and goto** are reserved but not used.

Java Class Libraries

- the Java environment relies on several built-in class libraries that contain many built-in methods that provide support for such things as I/O, string handling, networking, and graphics.
- The standard classes also provide support for windowed output.
- Thus, Java as a totality is a combination of the Java language itself, plus its standard classes.


class
car
objects
polo
mini
beetle

Questions?