

# Object-Oriented Programming

*The task of composition of operations is often considered the heart of the art of programming. ... However, it will become evident that the appropriate composition of data is equally fundamental and appropriate.*

— N. Wirth, *Algorithms + Data Structures = Programs*, 1976.

Object-oriented design is based on the following concepts:

**Abstraction:** the decision to concentrate on properties which are shared by many objects or situations in the real world, and to ignore the differences between them.

**Representation:** the choice of a set of symbols to stand for the abstraction (i.e., data structures)

**Manipulation:** the rules for transformation of the symbolic representation as a means of predicting the effect of similar manipulation in the real world (i.e., functions and procedures).

**Axiomatization:** the rigorous statement of those properties which have been abstracted from the real world and which are shared by manipulation of the real world and the symbols which represent it.

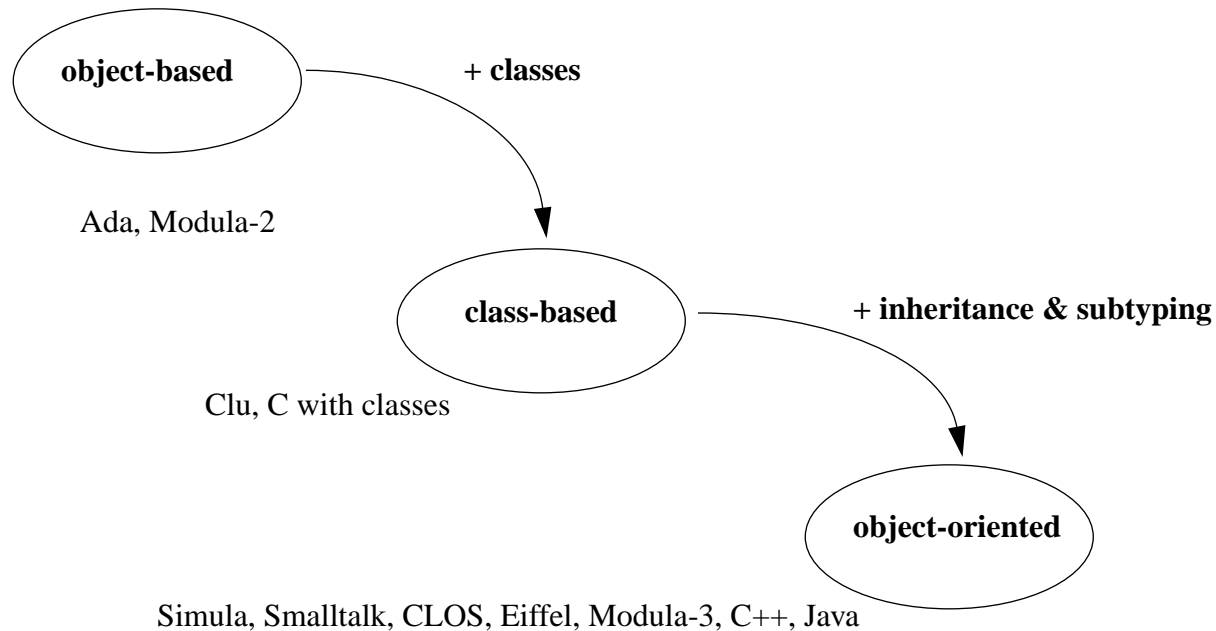
— C. A. R. Hoare, *Notes on Data Structuring*, 1972.

*It has taken 20+ years to bring these ideas into wide-spread use by programmers. Now, everyone wants to do object-oriented programming. Why?*

Object-oriented programming is now common practice among professional software engineers, and programming languages like C++ and Java allow the programmer to express solutions to computational problems in a variety of different ways.

However, the important thing is not the choice of language, but being able to think about a problem in an object-oriented manner, and then use a specific language and its object-oriented features to implement a well-engineered solution that solves the problem such that a program executes **correctly**, **safely**, and **efficiently**.

## Objects + Classes + Inheritance = OO Programs



**Objects** --- provide a conceptual framework for thinking about typed data and typed operations that correspond to a real world object that one wishes to model computationally. *The term 'object' will also be used to refer to the run-time state of a conceptual object.*

**Classes** --- are used to declare a new **type** with compile-time (static) attributes of data and operations (methods) that a set of objects, or *instances* of that class, will have at run-time.

**Inheritance** --- is a mechanism that is used to define new (sub-)types by specializing existing types. Inheritance is a compile-time (static) mechanism in a language like Java.

**Templates** -- allow compile-time (static) parameterization of a class by other types and constant values.

## A Brief History of C++ and Java

C++ was “invented” by Bjarne Stroustrup in the early 1980s at Bell Labs. The original intent of the language was to have the efficiency and portability of C, but borrowing object-oriented ideas from Simula (i.e., the class) and CLU (generic functions and types). The main features were to be strong typing, encapsulation using classes, and inheritance, as improvements on C. Features like templates and exception handling came later.

C++ became commercially popular and as things usually go, a committee was formed to standardize the language. C++ is now a complex programming language with lots of features and several “traps and pitfalls” for the novice programmer. It has taken several years for C++ compiler developers to implement the language as defined by the ANSI C++ standard. Today, there are still some C++ compilers that do not yet accept the full language defined by the C++ language standard. There is also a severe lack of useful freely available class libraries, which has been a drawback to using C++. **This is a persistent and frustrating problem that has limited C++’s success, and these problems inspired the inventors of Java to produce a new object-oriented language.**

Having said that, C++ is a very practical and useful programming language in some contexts. In this course, you will learn mostly about its useful features, its traps and pitfalls, and enough about the complex features to make intelligent decisions of when to use or avoid them and how to write professional quality programs.

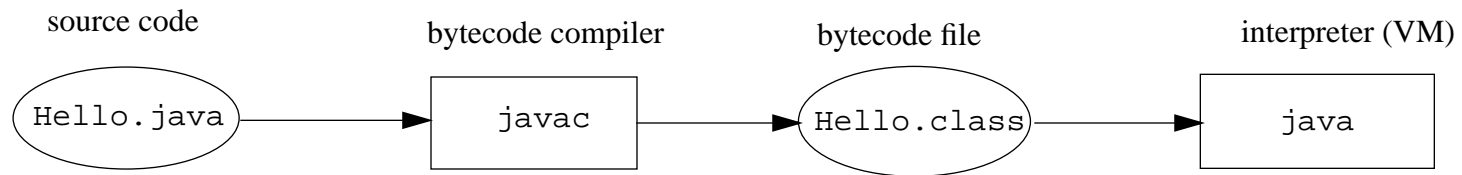
Java, on the other hand, is an interesting “new” approach to object-oriented programming. In fact, Java steals most of its language ideas from other languages. Its object model is very similar to Modula-3 while its syntax looks like C/C++. The hype surrounding this language is unprecedented because of the success of the web. It is a “simpler” language than C++ (and was meant to be), which means it lacks some of the features that C++ programmers find useful. Some people predict the death of C/C++ and the world-wide dominance of Java.

Java was originally called Oak, and was developed by James Gosling (inventor of Gosling Emacs) and others at Sun Microsystems that initially were trying to build an special operating system and programming language for “video on demand” applications for your TV sets in the early 1990s. This technology went no where, and Sun tried to kill off the Oak project. The group survived, with the help of Sun co-founder Bill Joy (author of csh among other things in BSD Unix), and re-targeted Oak at the Internet as a more flexible programming language for “programming the net”. Once it started to take off, Sun renamed Oak to Java, because another company had already trademarked Oak as a name. The popularity and hype over Java, which is afterall just a programming language, is unprecedented. However, the language is allowing programmers to achieve wide-spreadcode reuse, which was not achieved with C++.

## Introduction to Java

Java's basic language syntax is similar to C/C++, but there are many differences too. Java is **strongly typed** like C++, meaning that a Java compiler detects type inconsistencies at compile-time. A Java compiler translates Java source code into Java **bytecodes**, which are instructions to a Java **virtual machine** (Java VM).

A Java VM executes a Java program by *interpreting* the byte codes as virtual machine instructions, which in turn cause the VM to execute real machine instructions. Interestingly, the java interpreter is mostly written in Java! So, when you execute the interpreter, you're running a java program. There is a small run-time part that is written in C, and the rest of it is implemented in Java. For example, the Java compiler is written in Java! This is also the case with other interpreted programming languages, such as Smalltalk, Lisp, Scheme, Dylan, ML, and others.



Java bytecodes are machine independent. This means that you can compile Java source code on one hardware platform and transfer the bytecodes to another platform and execute them using a Java VM that has been ported to that platform. This machine independence of the bytecode instructions is what makes Java suitable for “programming the Internet.” Recently, **native-code** Java compilers have been developed for performance reasons.

You can send Java bytecodes around the Internet as data in the form of an **applet** in a web page or as a **application/java** MIME bodypart in an e-mail message. Sending code around the net in this manner is called providing *executable content*. Java is not the only language that is used for executable content. You can use most interpreted languages in this manner, as long as there is a virtual machine available to interpret the data that is received. In just two years, Java has become language of choice for providing executable content on the Web.

## Running the Java Compiler (javac) and Interpreter (java)

On CS Dept. Solaris systems, the java compiler, interpreter, and class sources are in the directory /usr/java. See the README file in that directory. The directory /usr/java/bin has the java compiler and interpreter, so you need to have that in your PATH. To run the compiler, you just give the compiler the name of a .java file.

```
% javac Hello.java
```

```
% java Hello  
Hello, world
```

**Note that you do not specify the .class extension when you execute a class file. In addition, if you do not name your .java file with the same name as the public class inside the file (i.e., public class Hello), the compiler will complain.**

On Windows 95/NT, you can either run the Sun Java Developer Kit (JDK) for Windows, or you can use a commercial Java development environment (e.g., Visual J++, CodeWarrior). It's up to you to decide which one you like best. You can download the Sun JDK from:

<http://java.sun.com/>

JDK 1.1.8 is a stable version for Solaris, Windows 95 & NT, Linux, Mac, etc. JDK 1.2.1 is very new. If you plan to use Visual J++, check the following URL for up-to-date information on Microsoft Java SDKs:

<http://www.microsoft.com/visualj>

CodeWarrior info is available at the URL: <http://www.metrowerks.com/>

Also see my CS371 webpage for a number of URLs to Java sites. Download a copy of the Java Coding Conventions from:

<http://java.sun.com/docs/codeconv/>

Other documentation is available at: <http://java.sun.com/docs/index.html>

## Java Applications versus Applets

A Java *application* consists of one or more classes, with at least one class containing a **main** method, which is called by the java interpreter when the class is the first one loaded for execution.

You use the Java compiler to compile the application classes, then you run the java interpreter with the class containing the main method. The interpreter calls the main method to start your program. Note that you can write an application that has multiple main methods in different classes. You can load a different class to start your program with different main procedures.

A Java applet consists of one or more classes, with none of them containing a main method. Applets are mostly used in the web as a way to distribute executable content. A class that is an applet inherits from a special Applet class.

As with applications, you use the javac compiler to compile an applet, but instead of running it with the java interpreter, you can either load it into a web browser using the APPLET tags in HTML, or you can run it using a special java application called an Applet Viewer:

Define a simple HTML document (appletdemo.html) in the same directory as your applet as follows:

```
<HTML><HEAD><TITLE>My First Applet</TITLE></HEAD>
<BODY>
<APPLET code="MyApplet.class" width=500 height=300></APPLET>
</BODY>
</HTML>
```

Compile the applet in the same directory as the html file, to produce a MyApplet.class binary file:

```
% javac MyApplet.java
```

Then run Netscape Communicator or Internet Explorer and open the HTML document to watch your applet run. Alternatively, you can run the applet by loading the HTML document into the appletviewer:

```
% appletviewer appletdemo.html
```

## A Simple Java Program

A Java application program consists of one or more classes. One class should have a “main” method, which is the entry point for the Java program. Unlike C/C++, there are no global functions or procedures. All functions/procedures are defined on classes or objects. Note the similarities and differences with C++.

```
package edu.utexas.cs.example; // define class as part of your organization's test package
import java.io.*; // import the java I/O package
public class Hello {
    private static String s = new String("Hello, world");

    public static void main(String[] args) { // note the difference in the array arg
        System.out.println(s);
    }
}
```

1. The **package** statement defines a *namespace*, called edu.utexas.cs.example. The public class Hello belongs to this namespace. The package statement must be the first Java statement in a file. If not specified, the class is put in the default “global package”. Note that package names usually correspond to Internet domain names, for uniqueness.
2. There are no #include directives for including other class definitions, because there is no preprocessor in Java. The **import** statement imports a set of class definitions from another package. In this case, the java.io package.
3. A class may be either public or private, with respect to a package. A public class is visible to users of a package containing the class. A private class is only visible to other classes defined as part of the same package. There are no public/private sections as defined in C++. Instead, each instance variable and method is defined to be **public**, **private**, **protected**, or it has **default access**. Default access is like protected, but restricts access to only those classes in the same package. Default access is also sometimes called “friendly” access.
4. A variable or method declared static is global to all instances of the class, just as in C++. Static methods have no implicit **this reference**, which is how each runtime object is associated with its methods and local data. Note that when you declare a variable, you should initialize it as the same time as it is declared. All objects are created with the **new** operator and the result is assigned to an **object reference variable**. ***There are no pointers in Java!***

## Packages, Classes, Fields and Methods

Java requires that for every class, instance variable, and method you define, that you declare an access qualifier as part of the definition, or accept the default access (which is not private access) Compare this to what you must write in C++. A public Java class must correspond to a file of the same name. So, the Hello class must be defined in a Hello.java file, which is part of the 'test' package. Technically, the main method is called:

```
edu.utexas.cs.example.Hello.main() // a fully qualified method name
```

NOTE: Unlike C++ which uses the '::' operator for scope resolution, Java uses the '.' operator for both scope resolution and class member access.

A *fully qualified name* is of the form <package>.<class>.<field or method>. You might organize your packages into subpackages, in which case you write <package>.<subpackage>.<class>.<field or method>. A fully qualified name corresponds to a directory path in the filesystem, which is how Java is able to guarantee unique naming. So the name test.Hello might correspond to a file:

```
/home/user/edu/utexas/cs/example/Hello.java
```

A single .java file may contain multiple class definitions for a package, but only one may be public. The name of the public class must correspond to the name of the file. The java compiler will compile a .java file with multiple classes into multiple .class files: one for each class.

The CLASSPATH environment variable is used to tell the java interpreter where to look for packages and classes. Rather than always using a fully qualified name all the time, you can "open" a package using the **import** statement. When you specify an import statement, it opens the package, and allows you to omit the package name qualifiers. A package name may be a compound name: For example, the following input/output (io) subpackage is part of the standard Java SDK package:

```
import java.io.*;
```

The import statement causes the java.applet package to be opened, and the '\*' says to import all classes from that package.