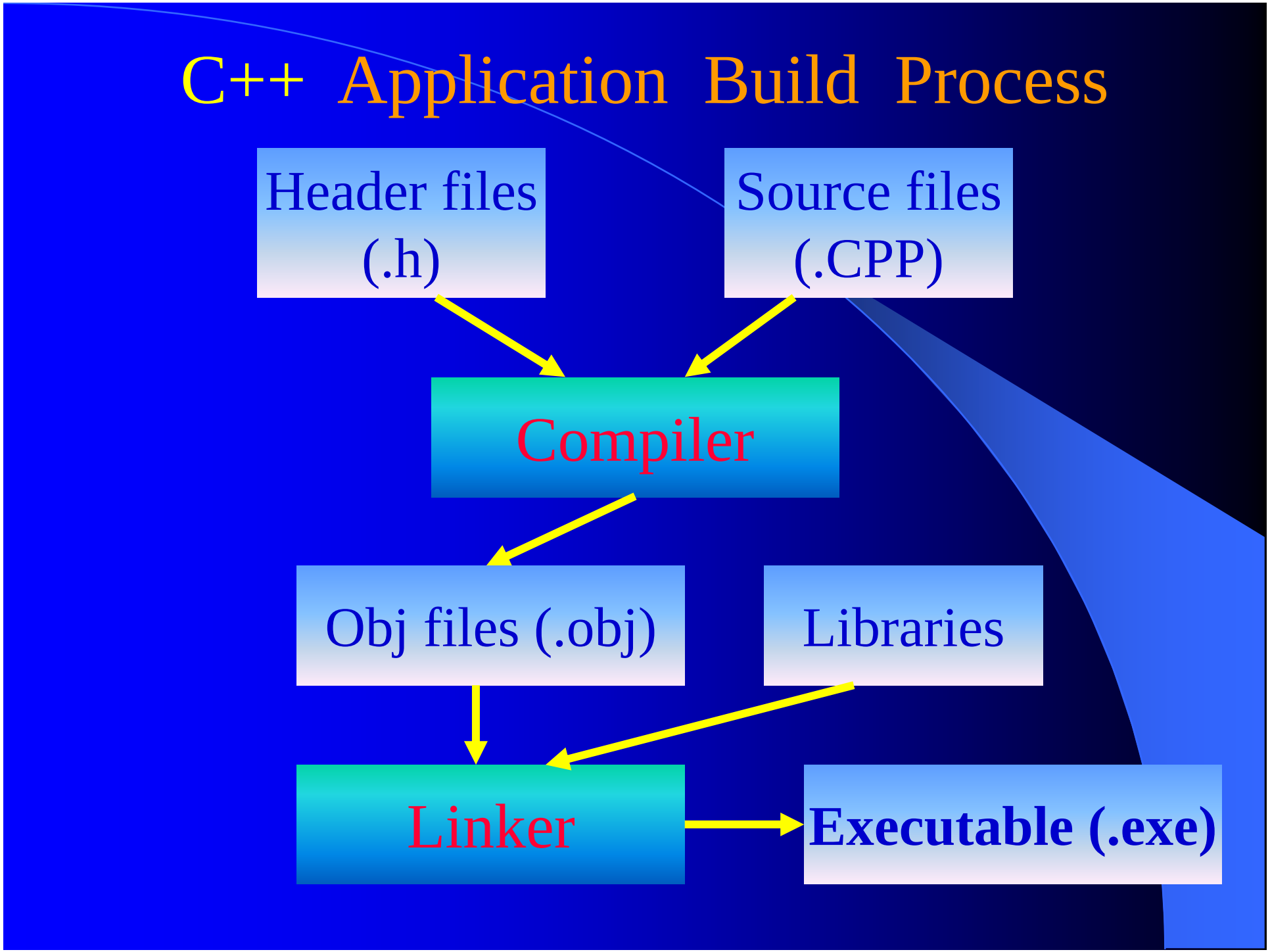
面向对象程序设计（Java 语言）

北京理工大学软件学院　　吕坤

Tel(O): 68918736

Email : kunlv@bit.edu.cn

C++ Application Build Process



Java Program Build Process

Write once

Run anywhere

Source files (.java)

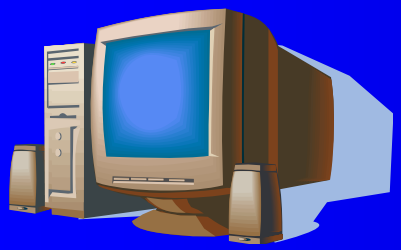
javac.exe

Byte code files (.class)

JVM for Windows

JVM for Linux

JVM for Mac



Reference Books

1. **Thinking in Java (Third Edition)**，**(美) Bruce Eckel**，机械工业出版社，**2005.5**
2. **Java 核心技术 (Core Java) 卷1：基础知识；卷2：高级特性**，**Horstmann & Cornell**，机械工业出版社，**2008.6**
3. Java语言程序设计.基础篇(原书第8版)，(美)Y.Daniel Liang(著) 李娜(译)，机械工业出版社，2011.6
4. **Java语言程序设计（第2版）**，吕凤翥 马皓，清华大学出版社，**2010.9**
5. **Tomcat与Java Web开发技术详解（第2版）**，孙卫琴，电子工业出版社，**2009.1**

About Java Versions

Over time, Sun (Oracle) has released seven major versions of Java: 1.0, 1.1, 2 (which is called version 2 even though the releases of the JDK from Sun continue to use the numbering scheme of 1.2, 1.3, 1.4, etc.) , 5(JDK 1.5), 6(JDK 6.0) , 7(JDK 7.0) and 8(JDK 8.0). Version 2 seems to finally bring Java into the prime time, in particular where user interface tools are concerned. And version 5 brings several amazing features to Java.

Preface

Programming is about managing complexity: the complexity of the problem you want to solve, laid upon the complexity of the machine in which it is solved.

C++ had to be backwards-compatible with C, as well as efficient.

VB was tied to BASIC, which wasn't really designed to be an extensible language.

Perl is often accused of producing "write-only code".

In Java, it seems like an unflinching goal to reduce complexity for the programmer.

Java goes on to wrap all the complex tasks that have become important, such as multithreading and network programming, in language features or libraries that can at times make those tasks trivial.

Java tackles some really big complexity problems: cross-platform programs, dynamic code changes, and even security.

So despite the performance problems we've seen, the promise of Java is tremendous: it can make us significantly more productive programmers.

Introduction

Online documentation

The Java language and libraries (a free download) come with documentation in electronic form, readable using a Web browser.

It's usually much faster if you find the class descriptions with your Web browser than if you look them up in a book (and the on-line documentation is probably more up-to-date).

Chapter 1: Introduction to Objects

This chapter is an overview of what object-oriented programming is all about, including objects, interface vs. implementation, abstraction and encapsulation, messages and functions, inheritance and composition, and the all-important polymorphism.

We'll learn what makes Java special, why it's been so successful, and about object-oriented analysis and design.

Chapter 2: Everything is an Object

This chapter gives an overview of the essentials, including the concept of a reference to an object; how to create an object; an introduction to primitive types and arrays; scoping and the way objects are destroyed by the garbage collector; how everything in Java is a new data type (class) and how to create our own classes.

Chapter 3: Controlling Program Flow

This chapter begins with all of the operators that come to Java from C and C++. In addition, we'll discover common operator pitfalls, casting, promotion, and precedence.

This is followed by the basic control-flow. Note : Java's labeled break and labeled continue (which account for the "missing goto" in Java).

Although much of this material has common threads with C and C++ code, there are some differences.

Chapter 4: Initialization & Cleanup

This chapter begins by introducing the constructor, which guarantees proper initialization.

This chapter also explores the garbage collector and some of its idiosyncrasies. And the chapter concludes with a closer look at how things are initialized.

Chapter 5: Hiding the Implementation

This chapter covers the way that code is packaged together, and why some parts of a library are exposed while other parts are hidden.

Keywords package and import perform file-level packaging and allow you to build libraries of classes.

Chapter 6: Reusing Classes

The concept of inheritance is standard in virtually all OOP languages.

Inheritance is often a way to reuse code by leaving the “base class” the same, and just patching things here and there to produce what you want.

However, inheritance isn’t the only way to make new classes from existing ones. You can also embed an object inside your new class with composition.

Chapter 7: Polymorphism

In this chapter we’ll see how to create a family of types with inheritance and manipulate objects in that family through their common base class. Java’s polymorphism allows us to treat all objects in this family generically.

Chapter 8: Interfaces & Inner Classes

Java provides a third way to set up a reuse relationship --- the interface.

The interface is *more than* just an abstract class taken to the extreme. It allows us to perform a variation on C++'s *multiple inheritance*, by creating a class that can be upcast to more than one base type.

At first, inner classes look like a simple code hiding mechanism: we place classes inside other classes. However, the inner class does more than that—it knows about and can communicate with the surrounding class.

Chapter 9: Holding our Objects

To solve the general programming problem, you need to create any number of objects, anytime, anywhere. This chapter explores in depth the container library that Java supplies to hold objects while you're working with them.

Chapter 10: Error Handling with Exceptions

The basic philosophy of Java is that badly-formed code will not be run.

Java has exception handling to deal with any problems that arise while the program is running.

This chapter examines when we should throw exceptions and what to do when we catch them.

Chapter 11: The Java I/O System

Theoretically, you can divide any program into three parts: input, process, and output.

In this chapter we'll learn about the different classes that Java provides for reading and writing files, blocks of memory, and the console.

Java's object serialization

Chapter 12: Run-Time Type Identification

Java run-time type identification (RTTI) lets us find the exact type of an object when we have a reference to only the base type.

This chapter explains what RTTI is for, how to use it, and how to get rid of it when it doesn't belong there.

Chapter 13: Creating Windows and Applets

Java comes with the "Swing" GUI library, which is a set of classes that handle windowing in a portable fashion.

These windowed programs can either be applets or stand-alone applications.

"JavaBeans" technology

Chapter 14: Multiple Threads

Java provides a built-in facility to support multiple concurrent subtasks, called threads, running within a single program.

Threads are most apparent when trying to create a responsive user interface.

Chapter 15: Building Java EE Applications

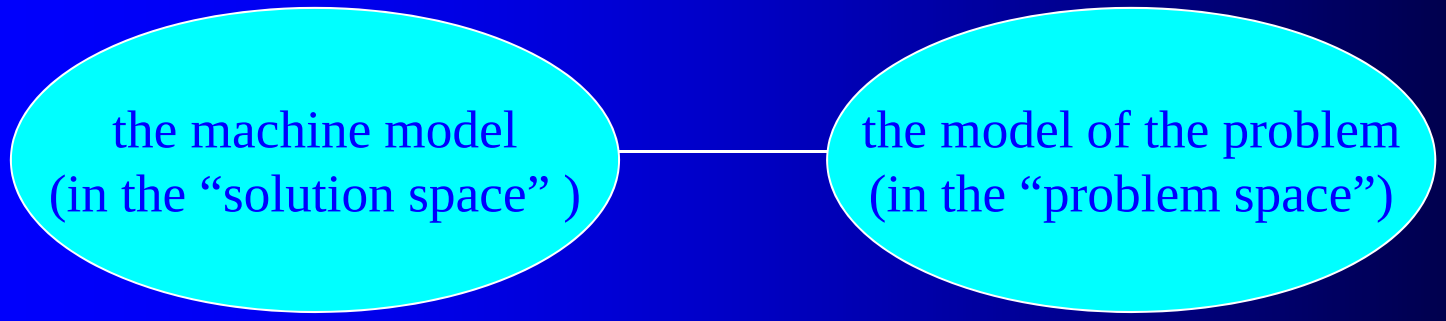
Servlets and JSPs, JDBC, Tomcat, Struts, Spring, Hibernate

Chapter 1: Introduction to Objects

This chapter will introduce us to the basic concepts of OOP, including an overview of development methods.

1.1 The progress of abstraction

All programming languages provide abstractions. It can be argued that the complexity of the problems we're able to solve is directly related to the kind and quality of abstraction.



The alternative to modeling the machine is to model the problem we're trying to solve. Each of these approaches is a good solution to the particular class of problem, but when we step outside of that domain they become awkward.

The object-oriented approach goes a step further by providing tools for the programmer to represent elements in the problem space.

The idea is that the program is allowed to adapt itself to the lingo of the problem by adding new types of objects, so when you read the code describing the solution, you're reading words expressing the problem. OOP allows us to describe the problem in terms of the problem.

The following characteristics represent a pure approach to object-oriented programming:

1. Everything is an object.

2. A program is a bunch of objects telling each other what to do by sending messages.

3. Each object has its own memory made up of other objects.

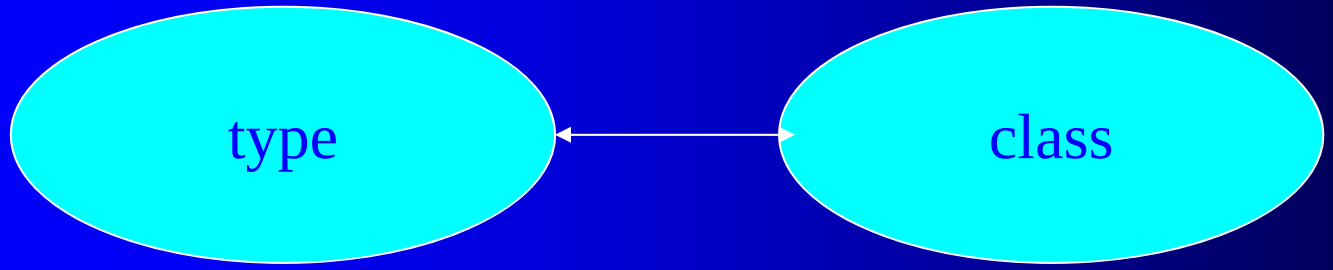
4. Every object has a type.

5. All objects of a particular type can receive the same messages.

1.2 An object has an interface

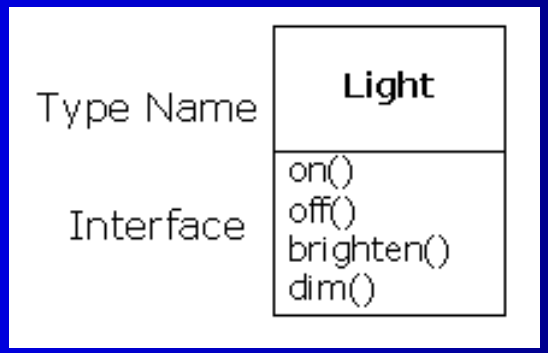
Creating abstract data types (classes) is a fundamental concept in object-oriented programming.

Each object belongs to a particular class that defines its characteristics and behaviors.



Once a class is established, you can make as many objects of that class as you like.

The interface establishes what requests you can make for a particular object. A type has a function associated with each possible request, and when you make a particular request to an object, that function is called.



1.3 The hidden implementation

It is helpful to break up the playing field into class creators and client programmers(the class consumers).

The hidden portion usually represents the tender insides of an object. So hiding the implementation reduces program bugs.

The first reason for access control is to keep client programmers' hands off portions they shouldn't touch. The second reason is to allow the library designer to change the internal workings of the class without worrying about how it will affect the client programmer.

In Java: public , private , protected

Java also has a "default" access --- "friendly" access

Classes can access the friendly members of other classes in the same package, but outside of the package those same friendly members appear to be private.

1.4 Reusing the implementation

Code reuse is one of the greatest advantages that object-oriented programming languages provide.

The simplest way to reuse a class is to just use an object of that class directly, but you can also place an object of that class inside a new class.

Inheritance is often highly emphasized. We should first look to composition when creating new classes, since it is simpler and more flexible.

1.5 Inheritance: reusing the interface

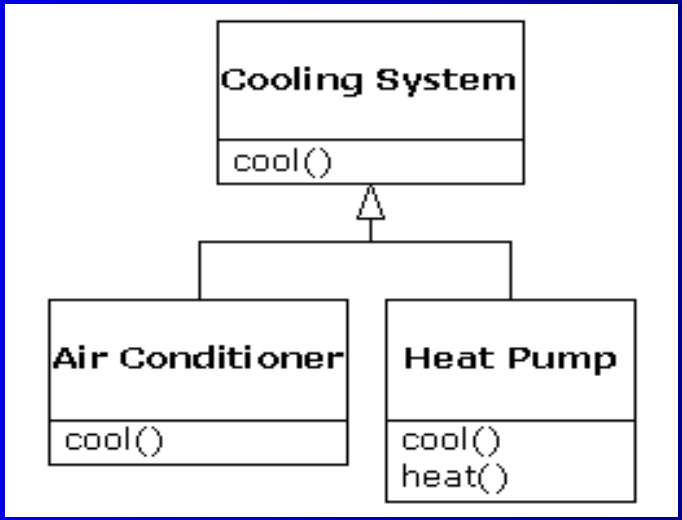
A base type contains all of the characteristics and behaviors that are shared among the types derived from it. From the base type, you derive other types to express the different ways that the core ideas can be realized.

The derived type contains not only all the members of the existing type, but more important, it duplicates the interface of the base class. All the messages you can send to objects of the base class can also be sent to objects of the derived class.

We have two ways to differentiate the derived class from the original base class. 1. Add brand new functions to the derived class. 2. (more important) Change the behavior of an existing base-class function. ---- Overriding

Pure substitution
is - a

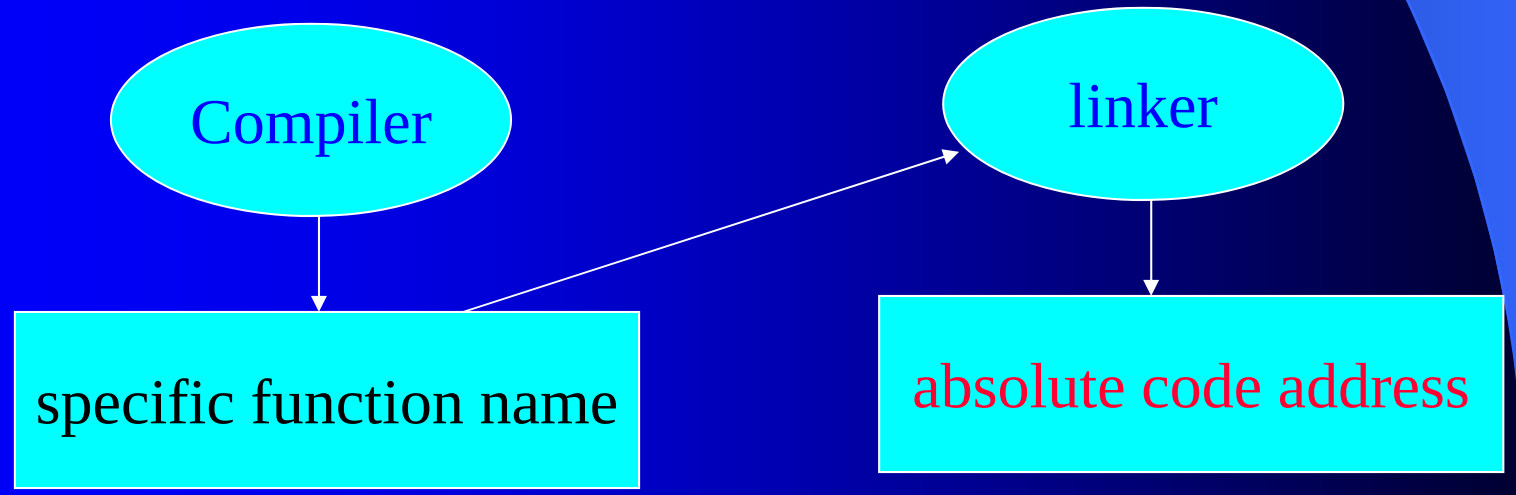
is – like - a



1.6 Interchangeable objects with polymorphism

When dealing with type hierarchies, we often want to treat an object not as the specific type, but instead as its base type. This allows us to write code that doesn't depend on specific types.

The function call generated by a non-OOP compiler causes early binding.

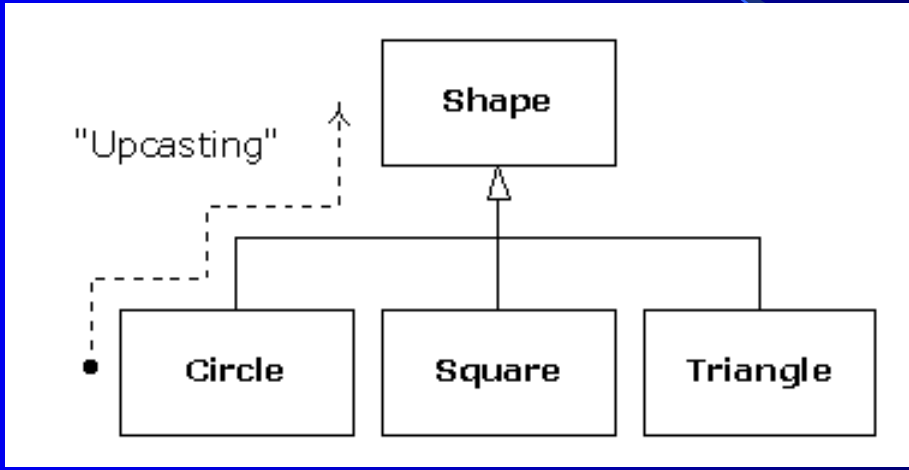


Object-oriented languages use the concept of late binding. When we send a message to an object, the code being called isn't determined until run-time.

To perform late binding, Java uses a special bit of code instead of the absolute call. In Java dynamic binding is the default. ---- no *virtual*

Using polymorphism, the code we write is decoupled from type-specific information, and is simpler to write and easier to understand. Thus, the program is extensible.

We call the process of treating a derived type as though it were its base type upcasting.



You don’t want to create an object of the base class, only to upcast to it so that its interface can be used.

---- abstract class

An abstract method may be created only inside an abstract class. When the class is inherited, that method must be implemented, or the inheriting class becomes abstract as well.

The interface keyword prevent any function definitions at all. ---- one step further

It provides the perfect separation of interface and implementation.

1.7 Object landscapes and lifetimes

1. The way objects are created and destroyed.

C++: For efficiency, it gives the programmer a choice.

Placing the objects on the stack or in the static storage area can get the maximum run-time speed. However, you sacrifice flexibility .

We can also create objects dynamically in the heap. You can simply make an object on the heap when you need it. ---- The greater flexibility

The general assumption : Objects tend to be complicated, so the extra overhead of finding and releasing the storage will not affect the creation of an object heavily.

Java uses the second approach, exclusively. ---- use the *new* keyword to build a dynamic instance of an object.

2.The lifetime of an object.

In C++, you must determine programmatically when to destroy the object, which can lead to memory leaks if you don’t do it correctly.

Java provides a garbage collector that automatically discovers when an object is no longer in use and destroys it.

(1) Collections and iterators

We create another type of object. The new type of object holds references to other objects. ---- Container (or collection)

It will expand itself whenever necessary to accommodate everything you place inside it.

Java also has containers in its standard library.

The container, via the iterator, is abstracted to be simply a sequence. The iterator allows you to traverse that sequence without worrying about the underlying structure—an ArrayList, a LinkedList, a Stack, or something else.

To choose a container: 1. Containers provide different types of interfaces and external behavior. 2. Different containers have different efficiencies for certain operations.

(2) The singly rooted hierarchy

All objects in a singly rooted hierarchy have an interface in common, so they are all ultimately the same type.

All objects in a singly rooted hierarchy can be guaranteed to have certain functionality.

A singly rooted hierarchy makes it much easier to implement a garbage collector (which is conveniently built into Java).

(3) Downcasting vs. templates/generics

It’s not completely dangerous because if you downcast to the wrong thing you’ll get an exception.

The solution is parameterized types, which are classes that the compiler can automatically customize to work with particular types.

Parameterized types are an important part of C++, partly because C++ has no singly rooted hierarchy. Java currently has no parameterized types.

(4) Garbage collectors vs. efficiency and flexibility

The garbage collector is a particular problem because you never quite know when it's going to start up or how long it will take. This means that there's an inconsistency in the rate of execution of a Java program, so you can't use it in real time programs.

Java is simpler than C++, but the trade-off is in efficiency and sometimes applicability. For a significant portion of programming problems, however, Java is the superior choice.

1.8 Exception handling: dealing with errors

An exception is an object that is “thrown” from the site of the error and can be “caught” by an appropriate exception handler. It’s as if exception handling is a different, parallel path of execution that can be taken when things go wrong.

In Java, exception handling was wired in from the beginning and you’re forced to use it.

Note : Exception handling isn’t an object-oriented feature.

1.9 Multithreading

Java's threading is built into the language. The threading is supported on an object level.

It can lock the memory of any object. --- *synchronized*

Other types of resources must be locked explicitly by the programmer.

1.10 Persistence

Java provides support for *lightweight persistence*.

1.11 Java and the Internet

(1) Client/Server computing

A key to the client/server concept is that the repository of information is centrally located so that it can be changed and those changes will propagate out to the information consumers.

The information repository, the software that distributes the information, and the machine(s) where the information and software reside is called the *server*.

The software that resides on the remote machine and communicates with the server is called the *client.*

Disadvantages:

1. As client software changes, it must be built, debugged, and installed on each client machine.

2. It's especially problematic to support multiple types of computers and operating systems.

3. The all-important performance issue.

The entire client/server problem needs to be solved in a big way.

(2) Client-side programming

The Web's initial server-browser design provided for interactive content, but the interactivity was completely provided by the server.

However, Web sites built on CGI programs can rapidly become overly complicated to maintain, and there is also the problem of response time.

The solution is client-side programming. The Web browser is harnessed to do whatever work it can.

1. Plug-ins

This is a way for a programmer to add new functionality to the browser by downloading a piece of code .

You need to download the plug-in only once.

The value of the plug-in is that it allows an expert programmer to develop a new language and add that language to a browser without the permission of the browser manufacturer. ---- Backdoor

2. Scripting languages

With a scripting language you embed source code for the client-side program directly into the HTML page.

The trade-off is that your code is exposed to everyone. And the scripting languages are really intended to solve specific types of problems.

JavaScript VBScript

3. Java

Java allows client-side programming via the *applet*.

An applet is a mini-program that will run only under a Web browser. The applet is downloaded automatically as part of a Web page. The programmer needs to create only a single program, and that program automatically works with all computers.

One advantage a Java applet has over a scripted program is that it’s in compiled form.

But A scripted program will just be integrated into the Web page as part of its text. This could be important to the responsiveness of your Web site.

4. .NET and C#

For a while, the main competitor to Java applets was Microsoft's ActiveX, although it required that the client be running Windows.

Since then, Microsoft has produced a full competitor to Java in the form of the .NET platform and the C# programming language.

The .NET platform is roughly the same as the Java virtual machine and Java libraries, and C# bears unmistakable similarities to Java.

They had the considerable advantage of being able to see what worked well and what didn't work so well in Java, and build upon that.

5. Internet vs. Intranet

The Web is the most general solution to the client/server problem.

When Web technology is used for an information network restricted to a particular company, it is referred to as an intranet.

If you are involved in such an intranet, the most sensible approach to take is the shortest path that allows you to use your existing code base.

(3) Server-side programming

A complicated request to a server generally involves a database transaction. These database requests must be processed via some code on the server side .

1.12 Analysis and design

Evolution is the point in the development cycle that has traditionally been called “maintenance”. OOP languages are particularly adept at supporting this kind of continuing modification—the boundaries created by the objects tend to keep the structure from breaking down.

1.13 UML (Rational Rose)

1. Use-Case Diagram
2. Static Diagram

3. Behavior Diagram

4. Interactive Diagram

5. Implementation Diagram

1.14 Extreme programming

(1) Write tests first

First, it forces a clear definition of the interface of a class.

Secondly, your tests become an extension of the safety net provided by the language.

(2) Pair programming

The value of pair programming is that one person is actually doing the coding while the other is thinking about it. If the coder gets stuck, they can swap places.

1.15 Why Java succeeds

Java is designed to be practical; Java language design decisions were based on providing the maximum benefits to the programmer.

1. Systems are easier to express and understand
2. Maximal leverage with libraries
3. Error handling
4. Programming in the large

1.16 Java vs. C++

The simplicity of Java over C++ will significantly shorten your development time.

Cross-compiling a program in Java should be a lot easier than doing so in C or C++.

A procedural program is data definitions and function calls. To find the meaning of such a program you have to look through the function calls and low-level concepts to create a model.

A well-written Java program is generally far simpler and much easier to understand than the equivalent C program.