Overview

° Problem: Can we make inheritance flexible?

° Abstract methods
  • Define methods that will be “filled in” by children

° Interfaces
  • Define common methods which may be implemented by several classes

° Visibility
  • Clever techniques to access hidden data
Interfaces

- A Java interface is a collection of abstract methods and constants
- An abstract method is a method header without a method body
- An abstract method can be declared using the modifier abstract, but because all methods in an interface are abstract, usually it is left off
- An interface is used to establish a set of methods that a class will implement

```
public interface Doable {
    public void doThis();
    public int doThat();
    public void doThis2 (float value, char ch);
    public boolean doTheOther (int num);
}
```

interface is a reserved word

None of the methods in an interface are given a definition (body)

A semicolon immediately follows each method header
Interfaces

- An interface cannot be instantiated
- Methods in an interface have public visibility by default
- A class formally implements an interface by:
  - stating so in the class header
  - providing implementations for each abstract method in the interface
- If a class asserts that it implements an interface, it must define all methods in the interface

```java
public class CanDo implements Doable {
    public void doThis () {
        // whatever
    }

    public void doThat () {
        // whatever
    }

    // etc.
}
```
Interfaces

- A class that implements an interface can implement other methods as well
- In addition to (or instead of) abstract methods, an interface can contain constants
- When a class implements an interface, it gains access to all its constants

```java
//************************************************
//  Complexity.java Author: Lewis/Loftus
//  Represents the interface for an object that can be assigned
//  an explicit complexity.
//************************************************

public interface Complexity
{
    public void setComplexity (int complexity);
    public int getComplexity();
}
```

Note the semicolon after the abstract methods!
Note: public ➔ interface…
Interfaces

- A class can implement multiple interfaces
- The interfaces are listed in the `implements` clause
- The class must implement all methods in all interfaces listed in the header

```java
class ManyThings implements interface1, interface2 {
    // all methods of both interfaces
}
```

Interfaces

- The Java standard class library contains many helpful interfaces
- The `Comparable` interface contains one abstract method called `compareTo`, which is used to compare two objects
- We discussed the `compareTo` method of the `String` class in Chapter 5
- The `String` class implements `Comparable`, giving us the ability to put strings in lexicographic order
The Comparable Interface

- Any class can implement `Comparable` to provide a mechanism for comparing objects of that type

```java
if (obj1.compareTo(obj2) < 0)
    System.out.println("obj1 is less than obj2");
```

- The value returned from `compareTo` should be negative if `obj1` is less than `obj2`, 0 if they are equal, and positive if `obj1` is greater than `obj2`

- When a programmer designs a class that implements the `Comparable` interface, it should follow this intent

The Comparable Interface

- It's up to the programmer to determine what makes one object less than another
- Possible to define the `compareTo` method of an `Employee` class to order employees
  - Ordering by name (alphabetically) or by employee number
- The implementation of the method can be as straightforward or as complex as needed for the situation
Class Hierarchies

° A child class of one parent can be the parent of another child, forming a \textit{class hierarchy}

\begin{center}
\begin{tikzpicture}
  \tikzstyle{level 1}=[sibling distance=100pt]
  \tikzstyle{level 2}=[sibling distance=50pt]
  \tikzstyle{level 3}=[sibling distance=25pt]
  \tikzstyle{every node}=[text width=2cm, text centered]
  \node (business) {Business}
    child {node (retailbusiness) {RetailBusiness}
      child {node (kmart) {KMart}}
      child {node (macys) {Macys}}
    }
    child {node (servicebusiness) {ServiceBusiness}
      child {node (kinkos) {Kinkos}}
    }
\end{tikzpicture}
\end{center}

Class Hierarchies

° Two children of the same parent are called \textit{siblings}

° Common features should be put as high in the hierarchy as is reasonable

° An inherited member is passed continually down the line

° Therefore, a child class inherits from all its ancestor classes

° There is no single class hierarchy that is appropriate for all situations
The Object Class

- A class `Object` is defined in the `java.lang` package of the Java standard class library
- All classes are derived from the `Object` class
- Sometimes a class is not explicitly defined to be the child of an existing class
  - It is assumed to be the child of the `Object` class
- The `Object` class is the ultimate root of all class hierarchies

The Object Class

- The `Object` class contains a few useful methods, which are inherited by all classes
- For example, the `toString` method is defined in the `Object` class
- Every time we define the `toString` method, we are actually overriding an inherited definition
- The `toString` method is in the `Object` class
  - Defined to return a string that contains the name of the object’s class along with some other information
The Object Class

° The equals method of the Object class returns true if two references are aliases
° We can override equals in any class to define equality in some more appropriate way
° The String class defines the equals method
  • Returns true if two String objects contain the same characters
° String class overrides the equals method inherited from Object

Abstract Classes

° An abstract class is a placeholder in a class hierarchy that represents a generic concept
° An abstract class cannot be instantiated
° We use the modifier abstract on the class header to declare a class as abstract:

```java
public abstract class Product
{
    // contents
}
```
Abstract Classes

- An abstract class often contains abstract methods with no definitions (like an interface)
- Unlike an interface, the `abstract` modifier must be applied to each abstract method
- Also, an abstract class typically contains non-abstract methods with full definitions
- A class declared as abstract does not have to contain abstract methods

Abstract Classes

- The child of an abstract class must override the abstract methods of the parent,
  - Otherwise, it too will be considered abstract
- An abstract method cannot be defined as `final` or `static`
- The use of abstract classes is an important element of software design
  - It allows us to establish common elements in a hierarchy that are too generic to instantiate
Interface Hierarchies

- Inheritance can be applied to interfaces as well as classes
- That is, one interface can be derived from another interface
- The child interface inherits all abstract methods of the parent
- A class implementing the child interface must define all methods
  - Includes both the ancestor and child interfaces
- Note that class hierarchies and interface hierarchies are distinct (they do not overlap)

Visibility Revisited

- It’s important to understand one subtle issue related to inheritance and visibility
- ➔ All variables and methods of a parent class, even private members, are inherited by its children
- As we’ve mentioned, private members cannot be referenced by name in the child class
- However, private members inherited by child classes exist and can be referenced indirectly
- The parent can refer to the private member
  - The child can reference it indirectly using its parent’s methods
- The super reference can be used to refer to the parent class, even if no object of the parent exists
Summary

- Inheritance is an important part of object oriented programming

- Interfaces allow us to define common methods which can be implemented by one or more classes

- Abstract classes define a placeholder for methods which will be implemented later

- Most programmers use basic techniques. Too much complexity gets confusing