Interfaces and Abstraction

COMS W1007
Introduction to Computer Science

Christopher Conway
17 June 2003
Review: Inheritance

- When objects have broadly similar characteristics, but specialized behavior, we can relate them through *inheritance*.
- A *subclass* inherits the public and protected members of its *superclass*.
- The derived class generally has an *is-a* relationship with the parent.
- A class is *assignment-compatible* with any class above it in the *class hierarchy*.
- Inheritance enables *polymorphism*—heterogeneous objects can be treated uniformly.
Abstract Classes

Sometimes a superclass can’t fully define all the behavior of an object—certain behavior only makes sense for a particular subclass.

If a class is too general to be useful on its own, we can declare it **abstract**. You cannot create an instance of an abstract class. It only serves as a base for **concrete** subclasses.

Methods that need to be defined by a subclass are also marked **abstract** and have no body. Any class with an abstract method must also be declared abstract.
Abstract Classes: InputStream

The class InputStream in java.io defines operations on a stream of input data.

- **read** returns the next byte of the input
- **skip(n)** skips the next n bytes of input.

**skip** can be defined using **read**, but **read** depends on what kind of data is being read. **read** is declared abstract; InputStream is an abstract class.
package java.io;

public abstract class InputStream {

  /* Reads the next byte of data from the input stream. Returns -1 if no more data is available. */
  public abstract int read();

  /* Skips the next n bytes of input. Returns the actual number of bytes skipped. */
  public long skip(long n) {
    long i = n;
    while (i > 0 && read() > 0) {
      i--;
    }
    return n - i;
  }

  ...
}
Abstract Classes: InputStream, 2

The concrete subclasses of InputStream define read for the particular kind of data they operate on.
Interfaces

Sometimes we want to define a type by describing its fields and methods without defining any specific behavior. An interface is an extreme form of the abstract class: all of its methods are abstract. An interface defines a external contract that a class agrees to implement.

A class may only extend one superclass, but it may implement any number of interfaces.
Interfaces: Comparable

The interface Comparable in java.lang provides a method for comparing two objects in a standard way. compareTo takes a single Object parameter and returns:

- -1 if the implementing object is less than the parameter.
- 0 if the implementing object is equal to the parameter.
- 1 if the implementing object is greater than the parameter.
package java.lang;

public interface Comparable {

    /* Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object. */
    int compareTo(Object o);
}
Sorting with Comparable

An interface defines a type, just like a class. Any class that implements an interface can be used polymorphically wherever that interface is required.

We can use interfaces to define generic methods that operate on a wide variety of classes polymorphically. Our previous sorting algorithm only worked on type `int`. Now we can define it to operate on any `Comparable` object.
public static void sort(Comparable[] objs) {
    /* iterate over the list and insert the items
     into the sorted list */
    for( int i=1 ; i < objs.length ; i++ ) {
        int n = objs[i] ;

        /* "push back" items that should come
         after n */
        int j = i ;
        while( --j >= 0 && n.compareTo(objs[j]) < 0 ) {
            objs[j+1] = objs[j] ;
        }
        objs[j+1] = n ;
    }
}
public class Point implements Comparable {
    double x, y;

    /* Points are ordered by their distance from
     * the origin. */
    public compareTo(Object obj) {
        if( obj instanceof Point ) {
            Point p = (Point) obj;
            double dist2 = x*x + y*y;
            double p_dist2 = p.x*p.x + p.y*p.y;
            return dist2 < p_dist2 ? -1 : 1;
        } else return -1;
    }
    ...
}
public class Employee implements Comparable {
    String lastName ;
    String firstName ;

    /* Employees are ordered by name. */
    public compareTo(Object obj) {
        if( obj instanceof Employee ) {
            Employee empl = (Employee) obj ;
            int c = lastName.compareTo( empl.lastName ) ;
            /* If the last names match, compare first names. */
            if( c==0 ) {
                c = firstName.compareTo( empl.firstName ) ;
            }
            return c ;
        } else return -1 ;
    }
    ...
}
The Wrapper Classes

Great, now we can sort all kinds of objects. But how do we sort int?

All of the primitive numeric types have wrapper classes that implement Comparable: Integer, Double, Byte, etc.

```
Integer i = new Integer(0) ;
Integer j = new Integer(1) ;
int c = i.compareTo(j) ;
/* c=-1 */
```
The **Number Class**

**Number** is the abstract superclass of all the numeric wrapper classes. It defines abstract methods for converting between types.
The Number Class: 2

Number and its subclasses demonstrate a good programming language design principle: try to implement the features of the language in the language itself.

**Source code:**

```
int i = 2 ;
String s = "i=" + i ;
double x = i ;
```

**Compiler’s translation:**

```
Integer i = new Integer(2) ;
String s = "i=".concat( i.toString() ) ;
Double x = new Double( i.doubleValue() ) ;
```
Summary: Abstract Classes and Interfaces

- A class is a combination of data and operations on that data.
- An abstract class is a class that doesn’t fully define its operations.
- An interface has no data. It only describes operations.

An interface gives you the “what”—it’s up to the implementor to provide the “how”.
Interface Members and Modifiers

- An interface may have fields, but they are all implicitly `public, static and final`. In other words, they must be constants.

- All of the methods in an interface are implicitly `public` and `abstract`.

- An interface has either public or package scope. The default is `public`.

We typically skip the modifiers in an interface definition: we can assume they have the values above.
Extending Interfaces

You can extend an interface, much like a class. The subinterface inherits all of the methods and constants of its superinterface. An interface may extend more than one other interface.

```java
public interface A { void foo() ; }
public interface B { void bar() ; }

public interface C extends A, B {
    void baz() ;
    /* C also contains foo() and bar(). */
}
```