Abstract Method & Abstract Classes

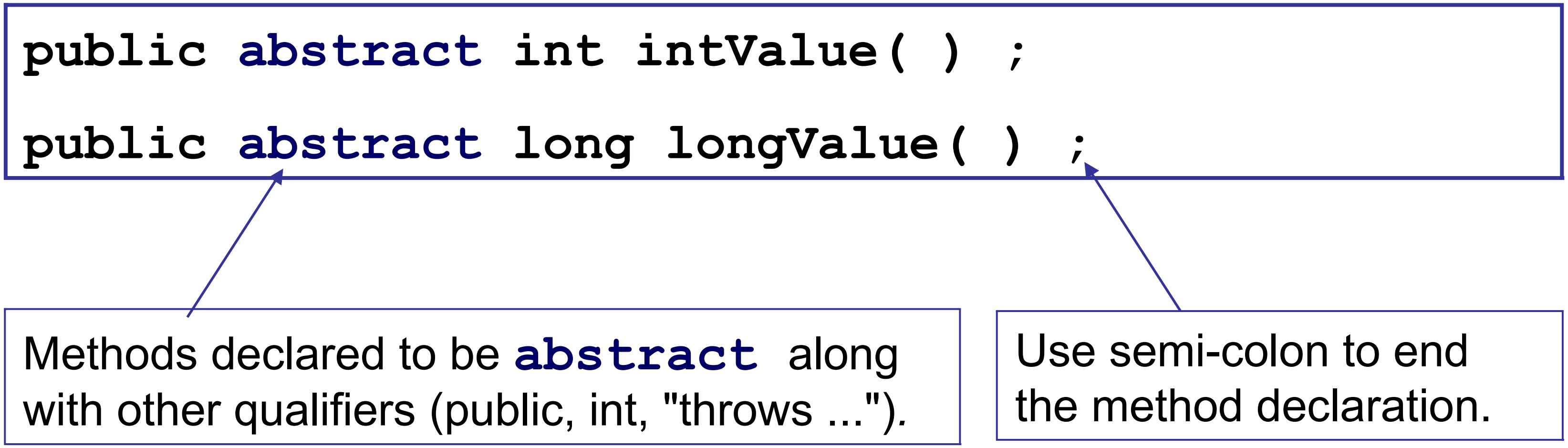
Jim Brucker

What is an Abstract Method?

An *abstract method* is a method declaration without a method body.

An abstract method specifies behavior but no implementation.

Example: In the Number class, `intValue`, `longValue`, ... are abstract.



```
public abstract int intValue( ) ;

public abstract long longValue( ) ;
```

Methods declared to be **`abstract`** along with other qualifiers (public, int, "throws ...").

Use semi-colon to end the method declaration.

Interface Methods are Abstract

All the methods in an interface are abstract:

```
public interface Comparable {
  public int compareTo(Object o);
}
```

is the same as:

```
public interface Comparable {
  public abstract int compareTo(Object o);
}
```

Class with Abstract Method

A class can have abstract methods.

Example:

The `Number` class has. are abstract methods `intValue()`, `longValue()`, and more

```
public abstract class Number {

  public abstract int intValue( ) ;
  public abstract long longValue( ) ;

```

Abstract Classes

A class with an abstract method is an abstract class.

- You must write "abstract class" in declaration.
- You *cannot create objects (instances)* of abstract class.

Error: `Number num = new Number( );` 🚫

```
public abstract class Number {

   public abstract int intValue( ) ;
   public abstract long longValue( ) ;
   ...etc...
```

OK for type declaration

This is OK because Double is a concrete subclass:

```
Number pi = new Double(3.14159);
```

What Can You Put in Abstract Class?

An abstract class can contain anything that a normal class can contain.

```
public abstract class Money
               implements Comparable<Money>
{
  static final String CURRENCY = "Baht";
  public Money( ) { ... }
  public abstract int getValue( );
  // not abstract
  public int compareTo(Money m) { ... }
```

Why Use Abstract Classes?

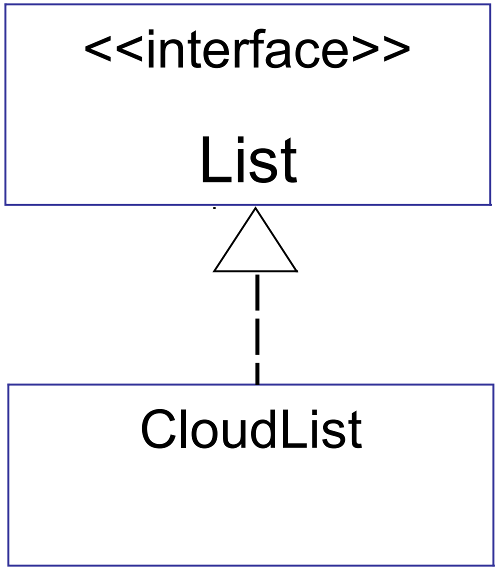
So you don't have to sleep at the office.

Assignment: Write a List

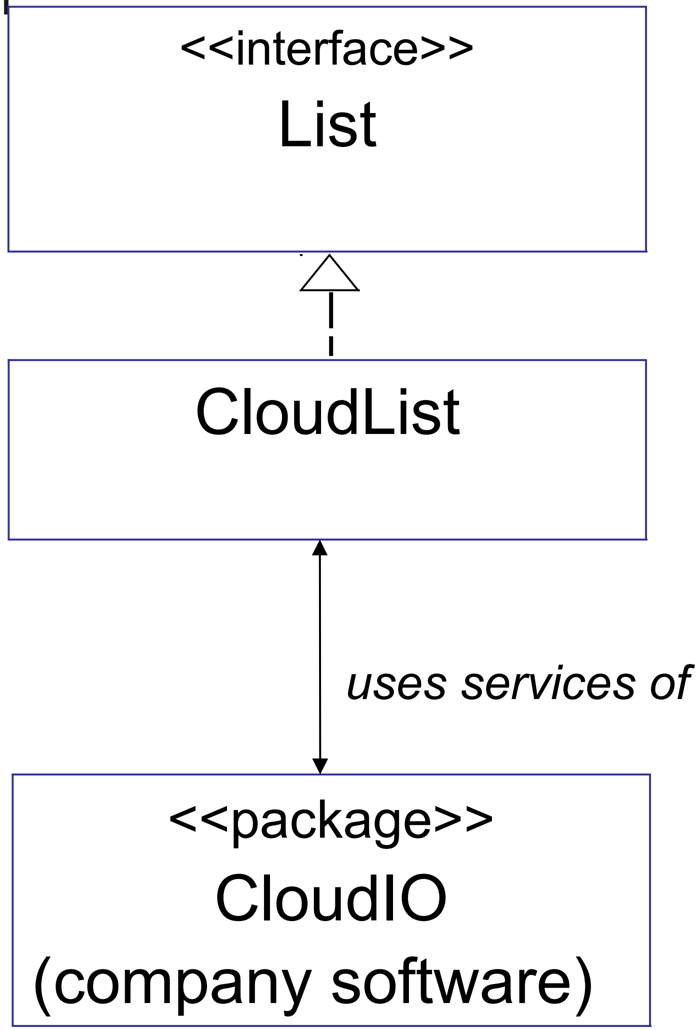
Your Boss: I want you to write a List that stores elements in the Cloud. Call it "CloudList".

You: *No problem.*

Your Boss: We need it *tomorrow*.



At work in your cubicle...



<<interface>>
List

CloudList

uses services of

<<package>>
CloudIO
(company software)

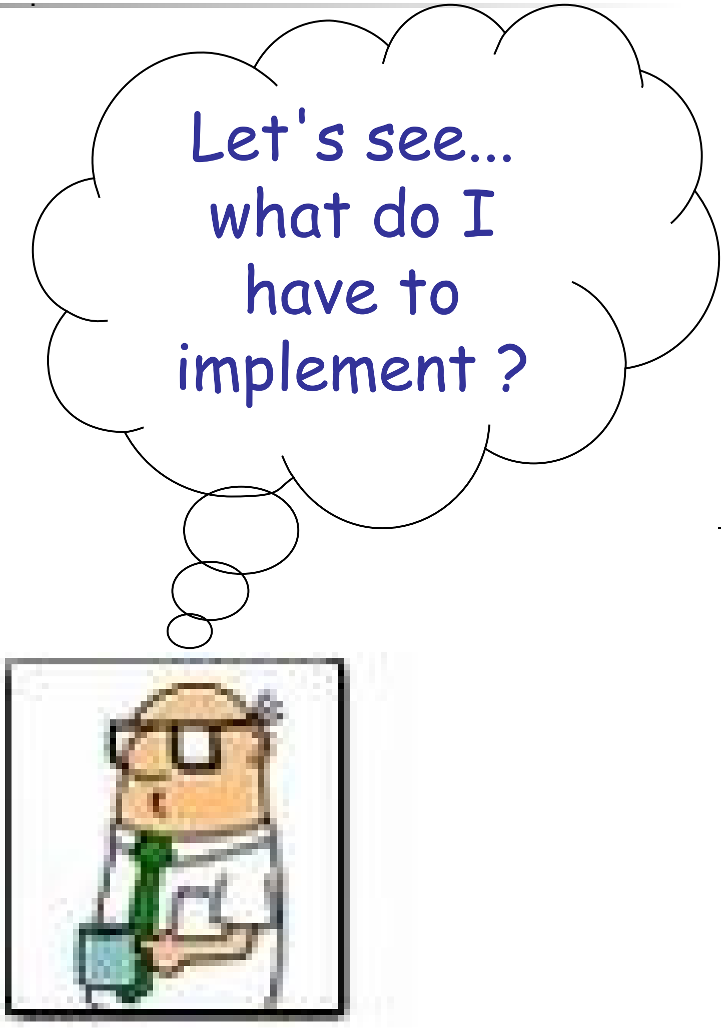
Open up the *List* API doc ...

<<interface>>

List

add(E): bool
add(int, E): void
addAll(Collection)
clear( )
contains(Object)
containsAll(Collection)
equals(Object): bool
get(int): E
hashCode( ): int
indexOf(Object)
isEmpty( )
iterator( ): Iterator<E>
lastIndexOf(Object)
remove(int): E
...

23 Methods



Try it in Eclipse: create a class that implements List, using ***Java language level 7.0*** *(<u>not</u> 8.0)*

Mission IMPOSSIBLE

<<interface>>

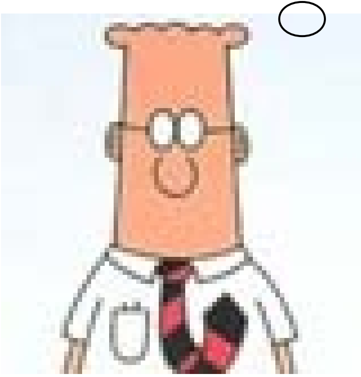
List

add(E): bool
add(int, E): void
addAll(Collection)
clear( )
contains(Object)
containsAll(Collection)
equals(Object): bool
get(int): E
hashCode( ): int
indexOf(Object)
isEmpty( )
iterator( ): Iterator<E>
lastIndexOf(Object)
remove(int): E
...

23 Methods

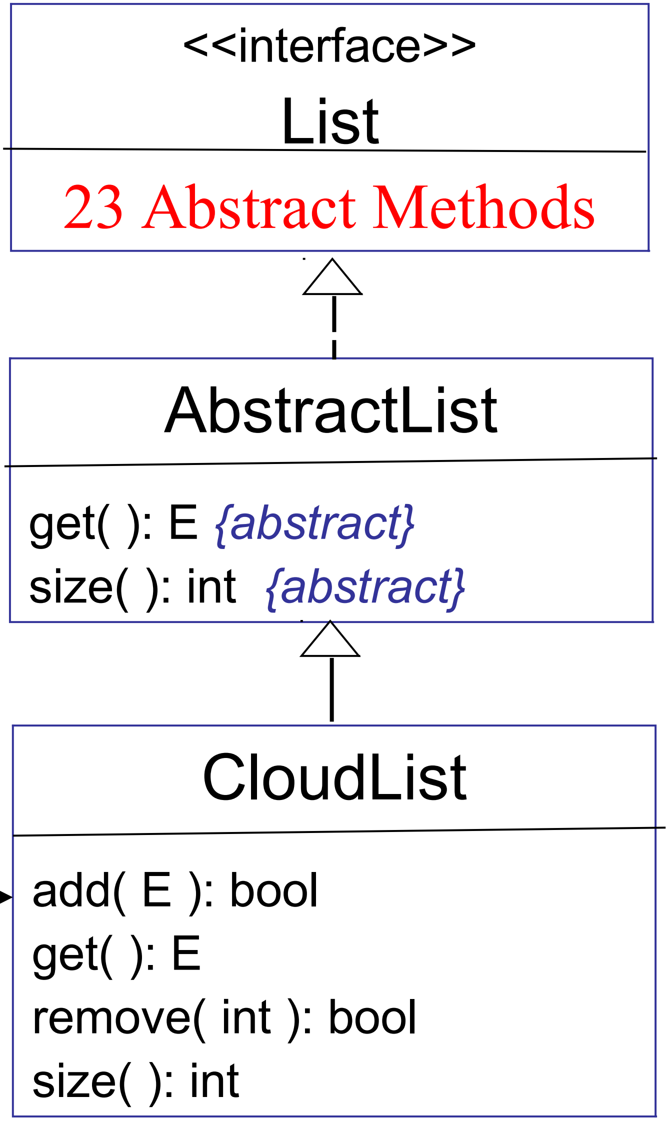
AbstractList to the Rescue

Extend AbstractList. It implements most methods for you.



You should also override a few more, like add( ) and remove( ).

Only 2 abstract methods


<<interface>>
List
23 Abstract Methods

AbstractList
get( ): E {abstract}
size( ): int {abstract}

CloudList
add(E): bool
get( ): E
remove(int): bool
size( ): int

In Java 8, you have to do more work than this.

Other Examples of Abstract Classes

An *interface* specifies required behavior.

An *abstract class* provides a skeleton or convenience class for implementing the interface.

Interface	Abstract Class that implements it...
MouseListener (5 methods)	*MouseInputAdapter* (0 abstract methods)
Set (15 methods)	*AbstractSet* (2 abstract methods)
Action	*AbstractAction*

Interface or Abstract Class?

Q: What is the advantage of using **an interface** instead of an Abstract Class to specify behavior?

```
abstract class AbstractFunction {
   /** function specification: no implementation */
   abstract public double f( double x ) ;

}
```

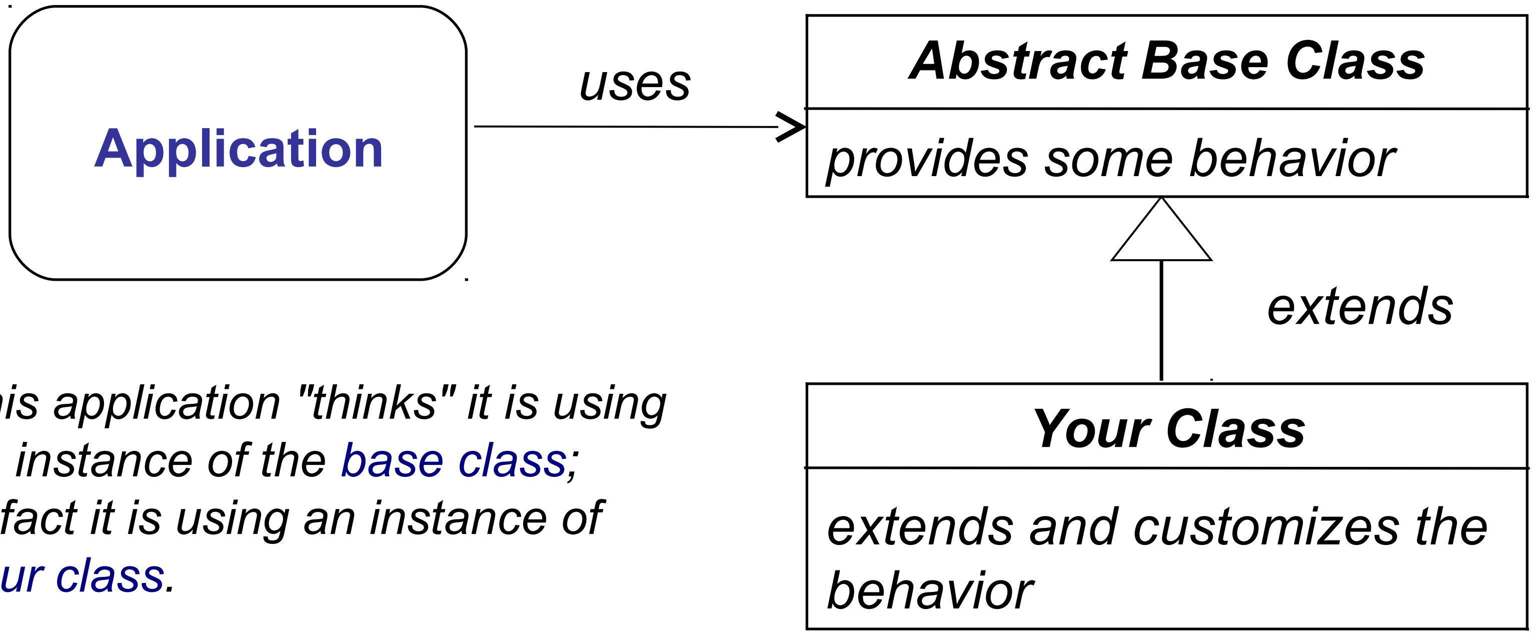
Abstract method does not have a body.

```
public class MyApplication extends AbstractFunction {
   /** implement the method */
   public double f( double x ) { return x/(x+1); }
   ...
}
```

Why Use Abstract Classes?

Many applications are designed to work with objects of many different classes.

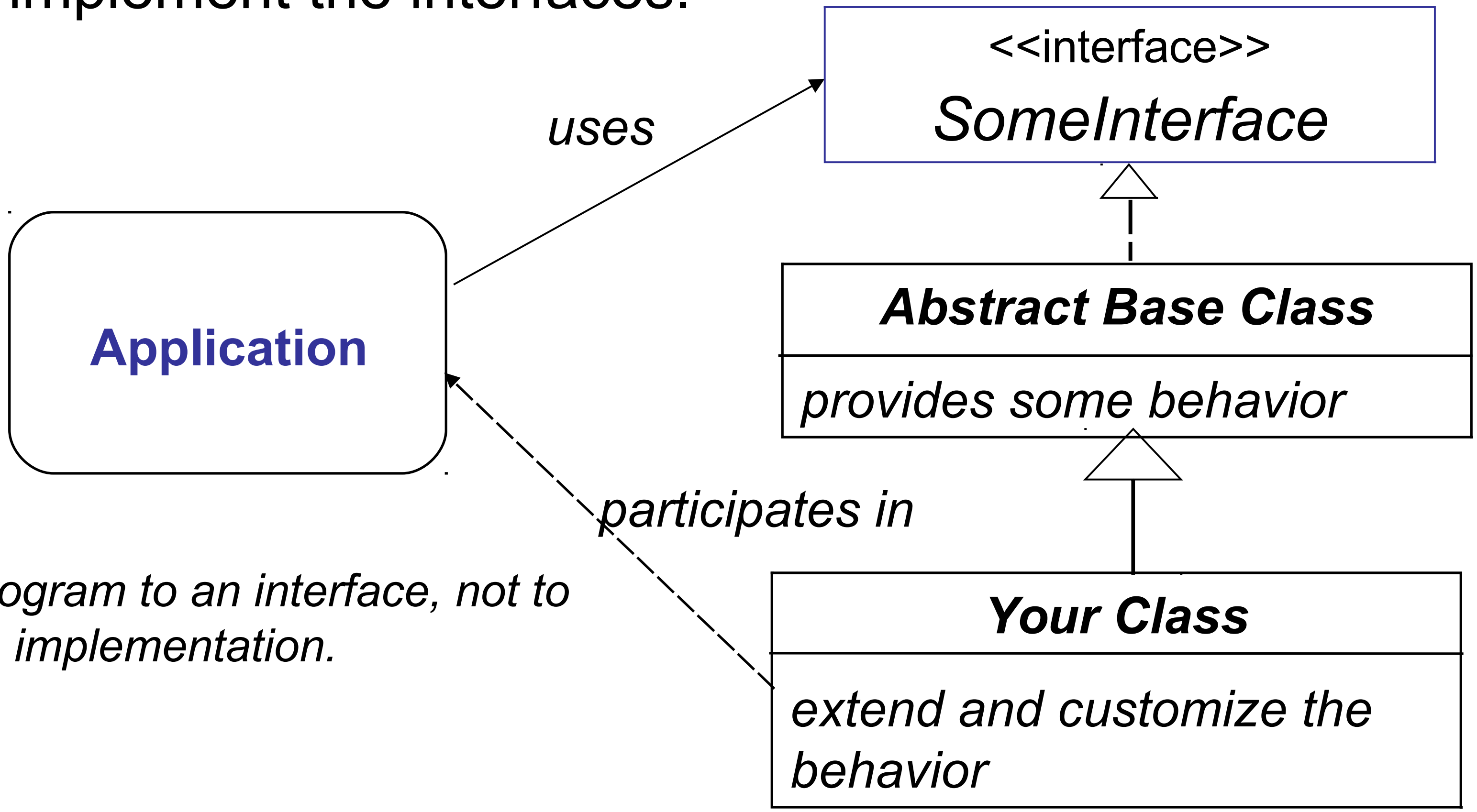
The application (or framework) accepts objects of the base class as parameter.



This application "thinks" it is using an instance of the base class; in fact it is using an instance of your class.

Depend on Interfaces

A better design is for application to depend on interfaces, but also provide abstract base class to help programmer implement the interfaces.



Program to an interface, not to an implementation.

Example of Abstract Classes

A Java GUI application is built using objects of a class named *java.awt.Component*.

- *Component* is an abstract base class
- real components (Buttons, Boxes, ...) are subclasses of *Component*
- Containers that manage components "think" that all components look & behave like *Component*.

```
//API: Container.add( Component c )
       container.add( new JButton("Press me") );
       container.add( new JLabel("Get a life.") );
       container.add( new JComboBox( array ) );
```

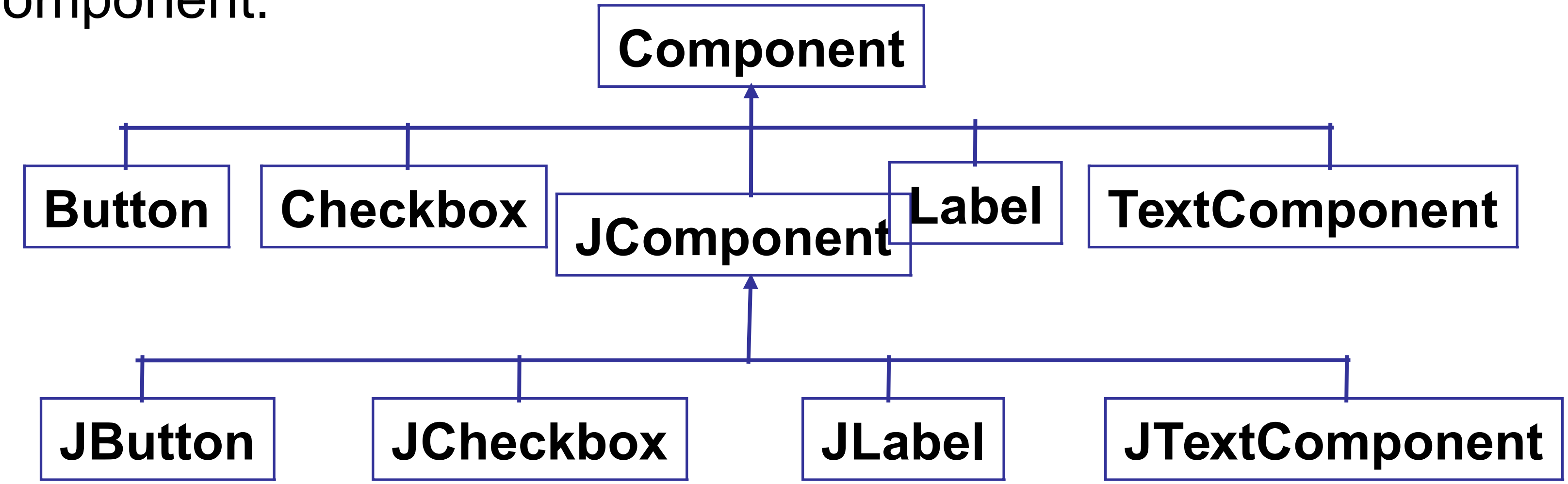
Swing & Abstract Classes

Each real component *extends* Component and overrides the behavior that it wants to *specialize*.

Benefit:

1) any Component can be put in any Container (like JPanel)

2) we can create our own component by *extending* Component. We don't need to rewrite most methods from Component.



Inheritance & Interface for Coin Purse

Discuss and design in class:

We want the Coin Purse to accept many kinds of money, such as Coin, BankNote, Check, and even KU Coupons (from KU Fair).

How can we use interface to make Purse polymorphic?

How can we use abstract classes to reduce coding and duplicate code?