CSSE 220 Day 6

Inheritance

Abstract Classes

Check out *Inheritance* from SVN

Questions?

Inheritance

- Sometimes a new class is a special case of the concept represented by another
- Can "borrow" from an existing class, changing just what we need
- The new class inherits from the existing one:
  - all methods
  - all fields
- Can add new fields/methods
- Or override existing methods

Q1

Code Examples

- `class SavingsAccount extends BankAccount`
  - adds interest earning, while keeping other traits
- `class Employee extends Person`
  - adds pay info. and methods, keeps other traits
- `class Manager extends Employee`
  - adds info. about employees managed, changes pay mechanism, keeps other traits

Notation and Terminology

-
  ```
  class SavingsAccount extends BankAccount {
      // added fields
      // added methods
  }
  ```
- Say "SavingsAccount **is a** BankAccount"
- **Superclass**: BankAccount
- **Subclass**: SavingsAccount

Q2

Other natural examples

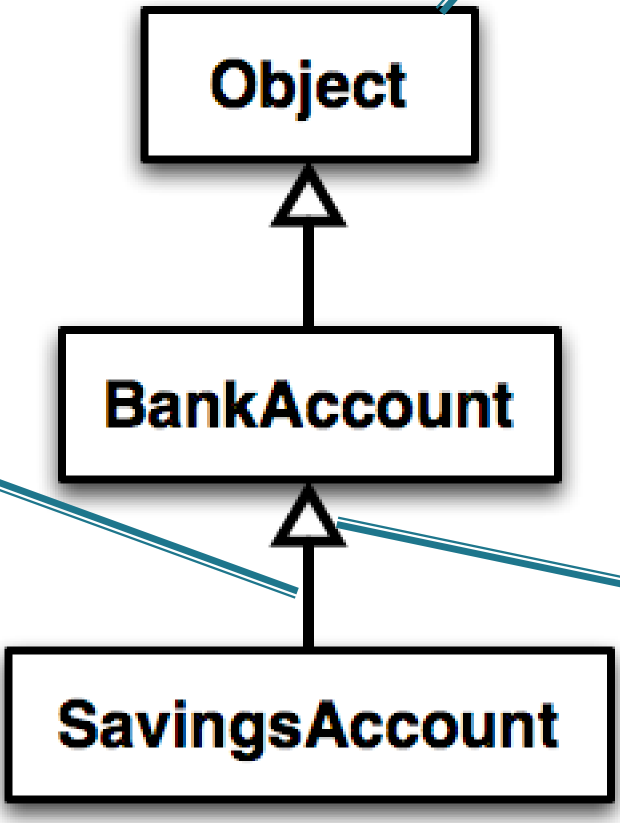
- A Sophomore IS-A Student IS-A **Person.**
- A Continent IS-A LandMass
- An HPCompaqNW8440 IS-A Laptop Computer
- An iPod IS-A MP3Player
- A Square IS-A Rectangle

- It is **not** true that a **Continent** IS-A **Country** or vice-versa.
- Instead, we say that a **Continent** HAS-A **Country.**

Examples From the Java API Classes

String	extends	Object
ArrayList	extends	AbstractCollection
IOException	extends	Exception
BigInteger	extends	Number
BufferedReader	extends	Reader
JButton	extends	JComponent
MouseListener	extends	EventListener
JFrame	extends	Window

Inheritance in UML



The “superest” class in Java

Solid line shows inheritance

Still means “is a”

Q4

Interfaces vs. Inheritance

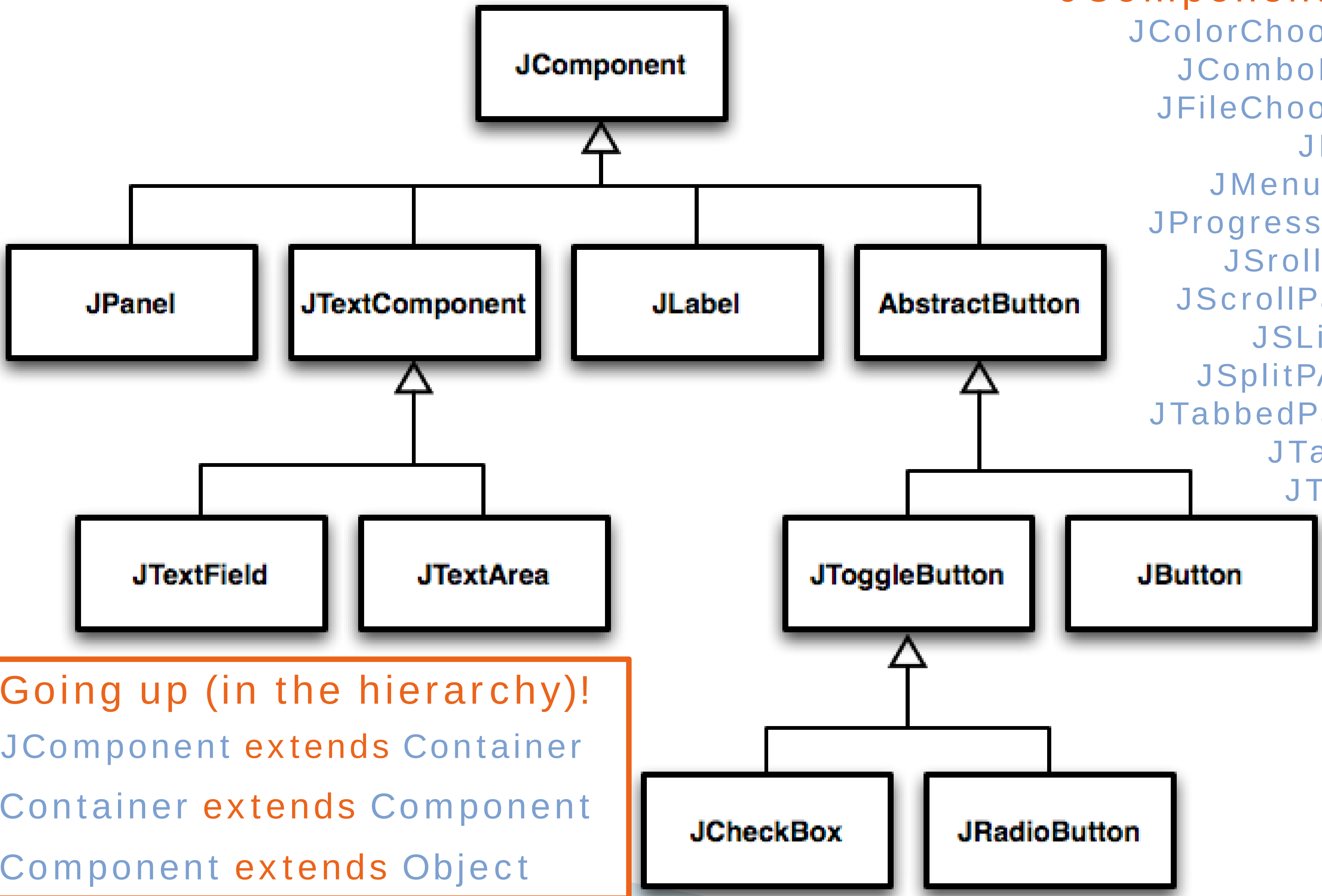
- `class ClickHandler implements MouseListener`
  - ClickHandler **promises** to implement all the methods of MouseListener

For <u>**client**</u> code reuse

- `class CheckingAccount extends BankAccount`
  - CheckingAccount **inherits** (or overrides) all the methods of BankAccount

For <u>**implementation**</u> code reuse

Inheritance Run Amok?



Still more subclasses of JComponent:

JColorChooser
JComboBox
JFileChooser
JList
JMenuBar
JProgressBar
JSrollBar
JScrollPane
JSLider
JSplitPAne
JTabbedPane
JTable
JTree

Going up (in the hierarchy)!

JComponent extends Container

Container extends Component

Component extends Object

With Methods, Subclasses can:

- Inherit methods unchanged
  - No additional code needed in subclass
- Override methods
  - Declare a new method with same signature to use instead of superclass method
- Partially Override methods
  - call super.sameMethod( ), and also add some other code.
- Add entirely new methods not in superclass

Q5

With Fields, Subclasses:

- **ALWAYS inherit** all fields unchanged
- **Can add** entirely new fields not in superclass

DANGER! Don’t use the same name as a superclass field!

Q6

Super Calls

- Calling superclass **method**:
  - `super.methodName(args);`

- Calling superclass **constructor**:
  - `super(args);`

Must be the first line of the subclass constructor. If not present, then **`super()`** is called.

Q7

Abstract Classes

- Halfway between superclasses and interfaces
  - Like regular superclass:
    - Provide implementation of some methods
  - Like interfaces
    - Just provide signatures and docs of other methods
    - Can't be instantiated
- Example:

```
public abstract class BankAccount {
        /** documentation here */
        public abstract void deductFees();
        …
}
```

Elided methods as before

Access Modifiers

- Review
  - **public**—any code can see it
  - **private**—only the class itself can see it
- Others
  - **default** (i.e., no modifier)—only code in the same **package** can see it
    - good choice for classes
  - **protected**—like default, but subclasses also have access
    - sometimes useful for helper methods

Fields should always be private, except possibly for *final* fields. Use a *protected* accessor if your subclass needs access to a field in a superclass

Q9