CSSE 220 Day 17

Inheritance

Check out *Inheritance* from SVN

Questions?

Nested classes

- You can define a class inside another class
  - This is called a *nested* class
  - It has access to the outer class' fields and methods
  - Useful if the inside class is a "helper class" of interest only to the outside class
- You can define a class and construct an instance of it inside a method
  - This is called a *local* inner class
  - Useful if the class is small and the object refers to variables in the outside class
- You can even make the inside class anonymous.
  - This is called an *anonymous* inner class
  - Let's do an example

This nomenclature is not universal. See
http://blogs.sun.com/darcy/entry/nested_inner_member_and_top
for more than you could possibly want to know about this subject

Q1

Homework part 1

- LinearLightsOut
- Individual assignment
- Show you internalized what you learned from SwingDemo
- Anonymous listeners could help (but not required)
- A good practice exam question
- Due Tuesday
  - I recommend you complete through stage 5 tonight so you can ask questions tomorrow.

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 instance fields

Q2

Examples

- `class SavingsAccount extends BankAccount`
  - adds interest earning, keeps 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

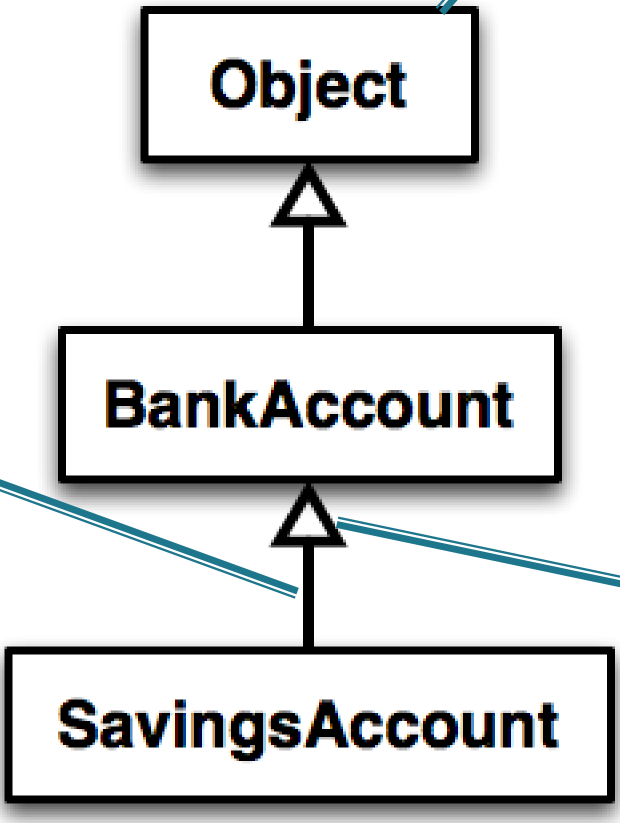
-
  ```
  class SavingsAccount extends BankAccount {
      // added fields
      // added methods
  }
  ```

- Say "SavingsAccount **is a** BankAccount"

- **Superclass**: BankAccount

- **Subclass**: SavingsAccount

Inheritance in UML



The "superest" class in Java

Solid line shows inheritance

Still means "is a"

Q4

Interfaces vs. Inheritance

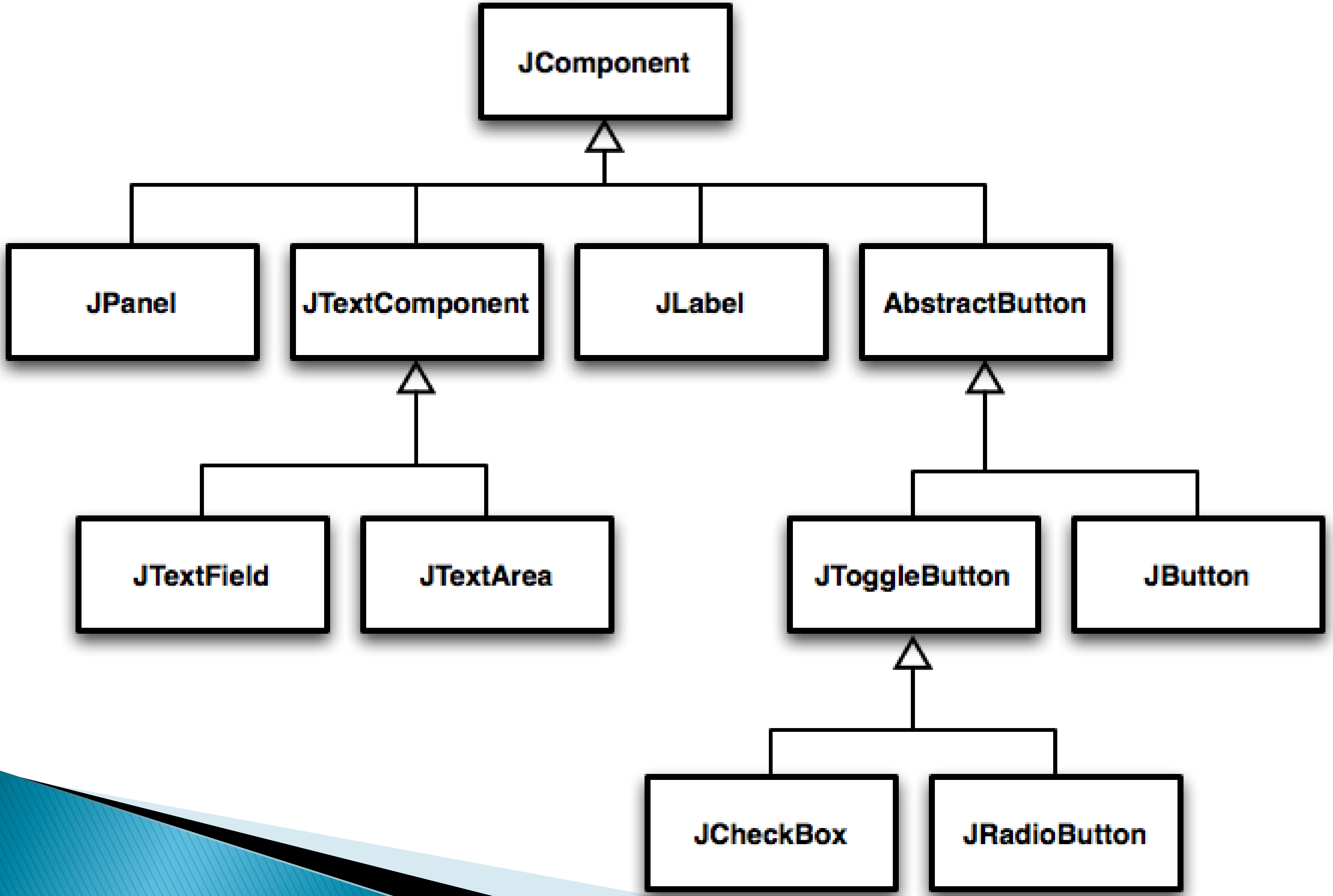
- `class ClickHandler implements MouseListener`
  - ClickHandler **promises** to implement all the methods of MouseListener

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?



With Methods, Subclasses can:

- Inherit methods unchanged
- Override methods
  - Declare a new method with same signature to use instead of superclass method
  - The new method can do completely different behavior from the overridden method, or it can do the overridden behavior plus some new behavior
- 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

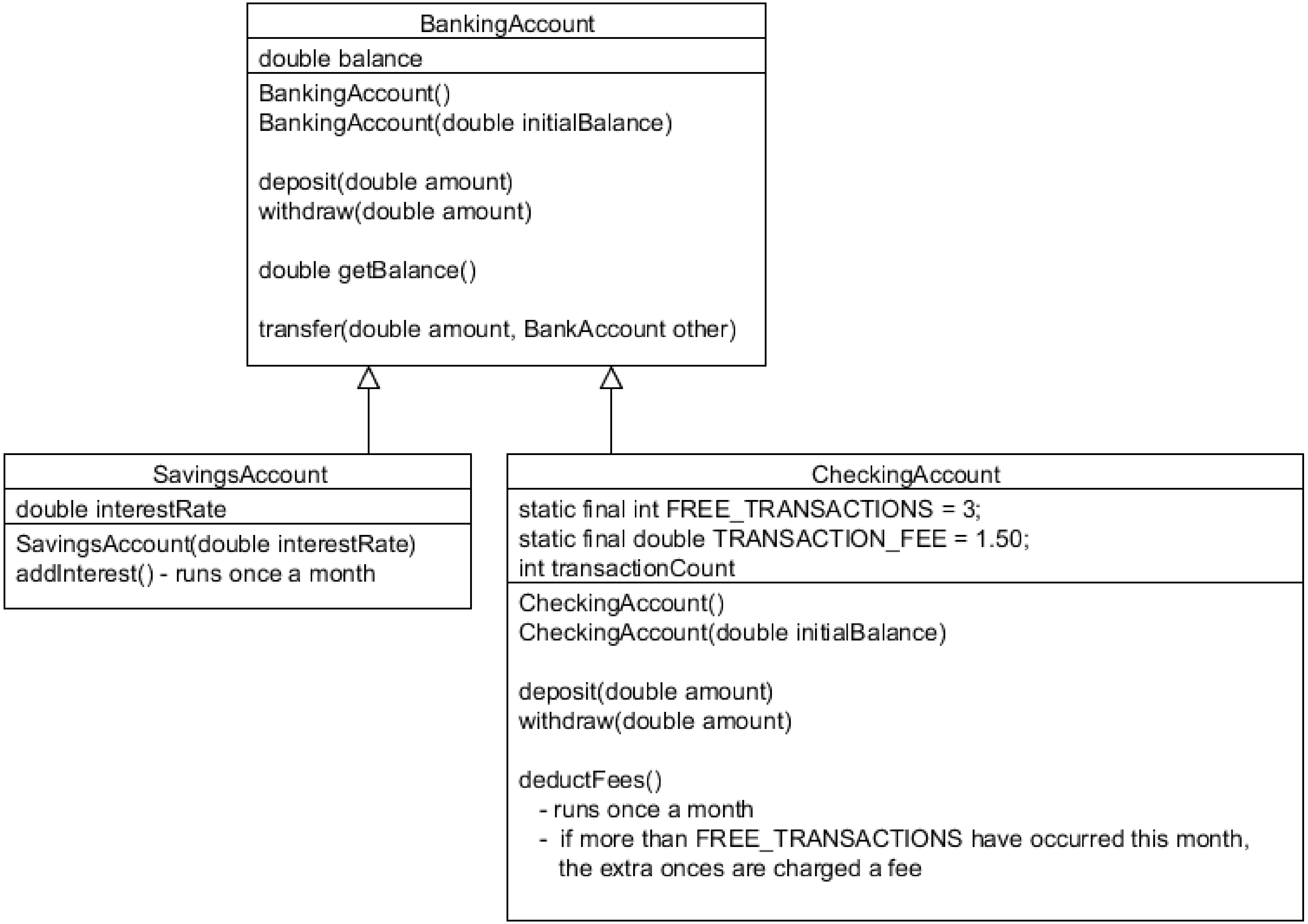
Super Calls

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

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

Must be the first line of the subclass constructor

Q7


BankingAccount
double balance
BankingAccount()
BankingAccount(double initialBalance)
deposit(double amount)
withdraw(double amount)
double getBalance()
transfer(double amount, BankAccount other)
SavingsAccount
double interestRate
SavingsAccount(double interestRate)
addInterest() - runs once a month
CheckingAccount
static final int FREE_TRANSACTIONS = 3;
static final double TRANSACTION_FEE = 1.50;
int transactionCount
CheckingAccount()
CheckingAccount(double initialBalance)
deposit(double amount)
withdraw(double amount)
deductFees()
- runs once a month
- if more than FREE_TRANSACTIONS have occurred this month, the extra onces are charged a fee

Abstract Classes

- Hybrid of 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 related classes
  - **protected**—like default, but subclasses also have access
    - sometimes useful for helper methods

Bad for fields!

Fields should be private

Q9

Break:

- Methods can call super.*methodName*(…)
  - To do the work of the parent class method, plus…
  - Additional work for the child class

```
public class Workaholic extends Worker {
    public void doWork() {
        super.doWork();
        drinkCoffee();
        super.doWork();
    }


}
```

Work Time

BallWorlds »

- Pair programming with a new partner
- Project is in your repository
- Instructions are on course web site,
  under *Programs ~ BallWorlds ~ instructions.htm*
- Your instructor will demo BallWorlds and discuss its UML, especially the Ball interfaces

BallWorlds Teams – Boutell

n	Team
01	krachtkq,davidsac
02	buqshank,kominet
03	beaversr,carvers
04	popenhjc,lemmersj
05	duganje
06	labarpr,parasby
07	weavergg,hannumed
08	runchemr,walthagd
09	smebaksg,amanb
10	mcgeevsa,ngop
11	cheungkt,hugheyjm
12	wanstrnj,macshake
13	shinnsm,eatonmi
14	moravemj,correlbn
15	pedzindm,sheetsjr
16	woodhaal,foltztm
17	breenjw

Check out *BallWorlds* from SVN

Team number used in repository name:
http://svn.csse.rose-hulman.edu/repos/csse220-201030-ballworlds-teamXX