

# CSSE 220 Day 19

Inheritance

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 instance fields



Q1

## Examples

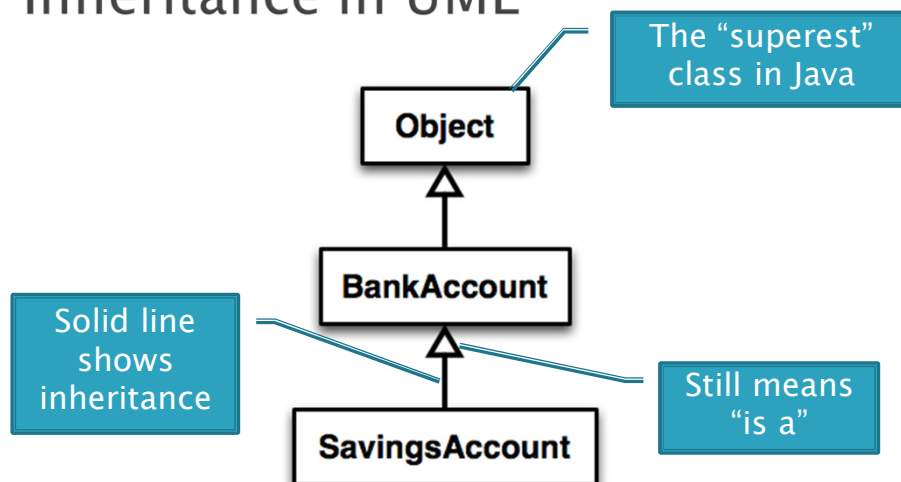
- ▶ **class SavingsAccount extends BankAccount**
  - adds interest earning, keeps other traits
- ▶ **class Employee extends Person**
  - adds pay information and methods, keeps other traits
- ▶ **class Manager extends Employee**
  - adds information about employees managed, changes the pay mechanism, keeps other traits

## Notation and Terminology

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

Q2

## Inheritance in UML



Q3

## Interfaces vs. Inheritance

▶ class ClickHandler **implements** MouseListener

- ClickHandler **promises** to implement all the methods of MouseListener

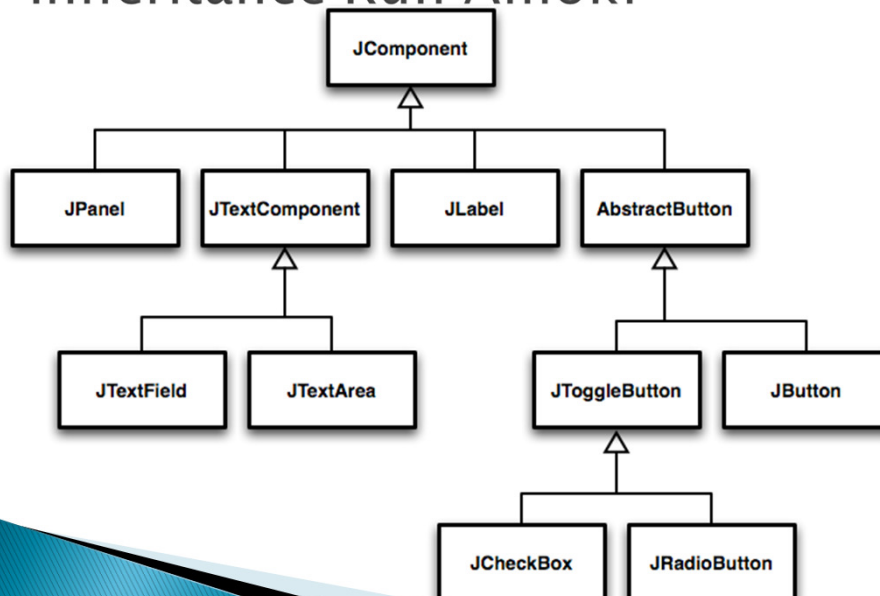
For client code reuse

▶ class CheckingAccount **extends** BankAccount

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

For implementation 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**
- ▶ **Add** entirely new methods not in superclass

Q4

## 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!

Q5

## Super Calls

- ▶ Calling superclass **method**:

- `super.methodName(args);`

- ▶ Calling superclass **constructor**:

- `super(args);`

Must be the first line of the subclass constructor

Q6

## Polymorphism and Subclasses

- ▶ A subclass instance **is** a superclass instance

- Polymorphism still works!
- `BankAccount ba = new CheckingAccount();`  
`ba.deposit(100);`

For client code reuse

- ▶ But not the other way around!

- `CheckingAccount ca = new BankAccount();`  
`ca.deductFees();`

- ▶ Why not?

BOOM!

Q7

## Another Example

▶ Can use:

```
public void transfer(double amt, BankAccount o){
    this.withdraw(amt);
    o.deposit(amt);
}
```

in BankAccount

▶ To transfer between different accounts:

```
SavingsAccount sa = ...;
CheckingAccount ca = ...;
sa.transfer(100, ca);
```

## Abstract Classes

▶ Hybrid of superclasses and interfaces

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

Also look at the code in the shapes package, especially ShapesDemo (during or after class)

▶ 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

Bad  
for  
fields!

Q8

## Work Time

### »» Linear Lights Out

It's a solo project, but feel free to talk with others as you do it.

And to ask instructor/assistants for help

Q9-Q10



# BallWorlds Introduction

» Demo  
UML Design Questions