CSSE 220 Day 11

Inheritance recap
`Object`: the superest class of all
Inheritance and text in GUIs

Check out *Inheritance2* from SVN

Questions?

Project Team Preference Survey

- On ANGEL, under Lessons → Assignments
- Preferences help me to choose teams; I also consider your performance so far in the course
- Complete the survey by **Monday, April 22, 2013, noon**
- Most teams will have 3 students
- Are you willing to be on a team of 2?
- List up to 5 students you'd like to work with, highest preference first.
  - You may not get your first choices, so it's a good idea to list more than two
  - Best to choose partners whose commitment level and current Java coding/debugging ability is similar to yours
- List up to 2 students you'd prefer NOT to work with
  - I'll do my best to honor this, but I must find a team for everyone.

I, Object

- The superest class in Java

Object

- **Every** class in Java inherits from **`Object`**
  - Directly and **explicitly**:
    - `public class String extends Object {…}`
  - Directly and **implicitly**:
    - `class BankAccount {…}`
  - **Indirectly**:
    - `class SavingsAccount extends BankAccount {…}`

Q1

Object Provides Several Methods

- `String toString()`
- `boolean equals(Object otherObject)`
- `Class getClass()`
- `Object clone()`
- …

Often overridden

Sometimes useful

Often dangerous!

Q2

Overriding `toString()`

- Return a concise, human-readable summary of the object state

- Very useful because it's called automatically:
  - During string concatenation
  - For printing
  - In the debugger

- `getClass().getName()` comes in handy here…

Q3

Overriding `equals(Object o)`

- Should return true when comparing two objects of same type with same “meaning”
- How?
  - Must check types—use `instanceof`
  - Must compare state—use **cast**
- Example…

Q4

Polymorphism

- Review and Practice

Polymorphism and Subclasses

- A subclass instance **is a** superclass instance
  - Polymorphism still works!
  - ```
    BankAccount ba = new SavingsAccount();
    ba.deposit(100);
    ```
- But not the other way around!
  - ```
    SavingsAccount sa = new BankAccount();
    sa.addInterest();
    ```
- Why not?

Another Example

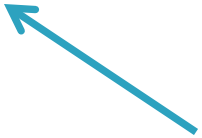
- Can use:
  - ```
    public void transfer(double amt, BankAccount o){
        this.withdraw(amount);
        o.deposit(amount);
    }
    ```
    in BankAccount
- To transfer between different accounts:
  - `SavingsAccount sa = …;`
  - `CheckingAccount ca = …;`
  - `sa.transfer(100, ca);`
    ```

Summary

- If B extends or implements A, we can write

```
A x = new B();
```

Declared type tells which methods x can access. Compile-time error if try to use method not in A.



The actual type tells which class' version of the method to use.

- Can cast to recover methods from B:

```
((B)x).foo()
```

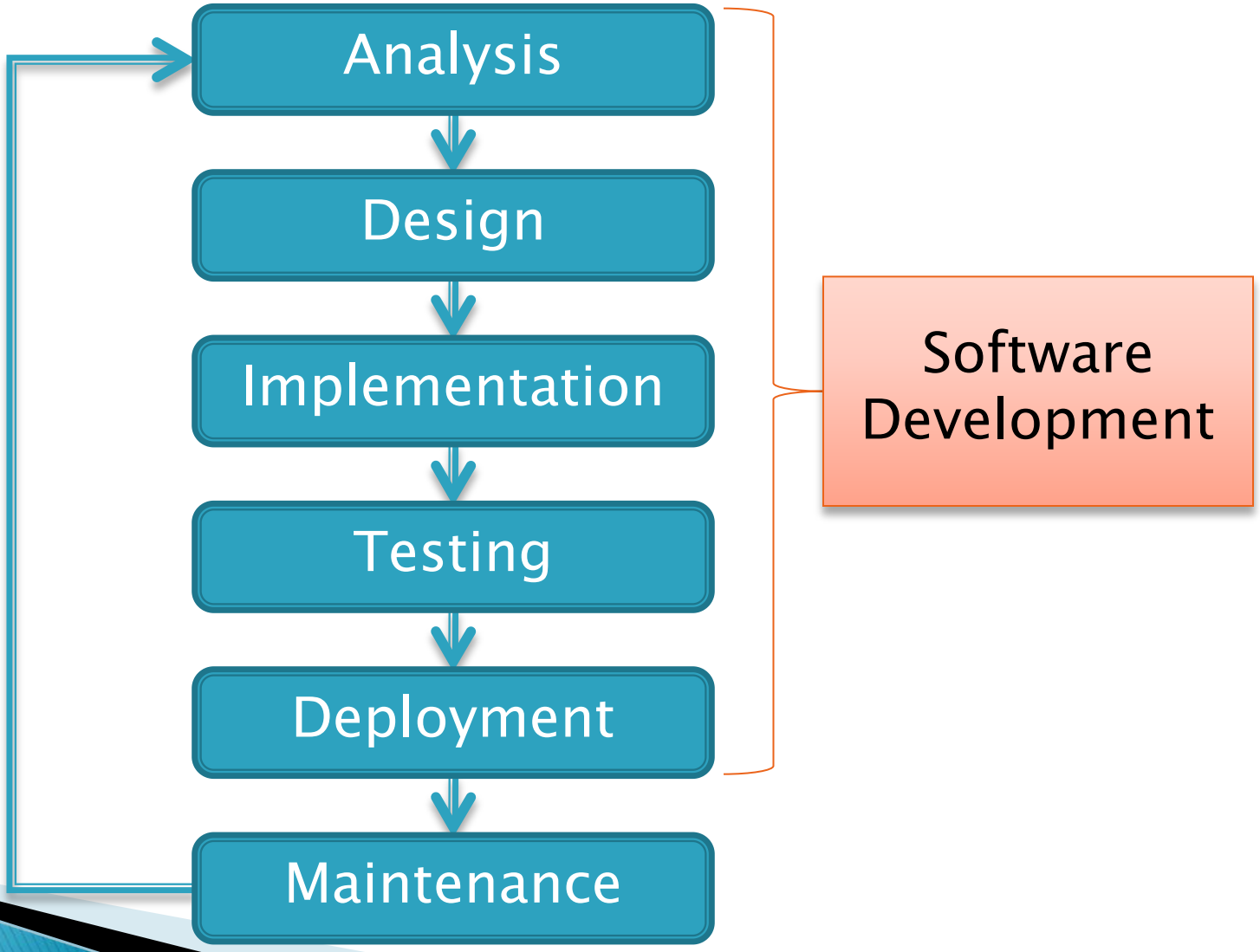
Now we can access all of B's methods too.

If x isn't an instance of B, it gives a run-time error (class cast exception)

Q5

Software Development Methods

Software Life Cycle

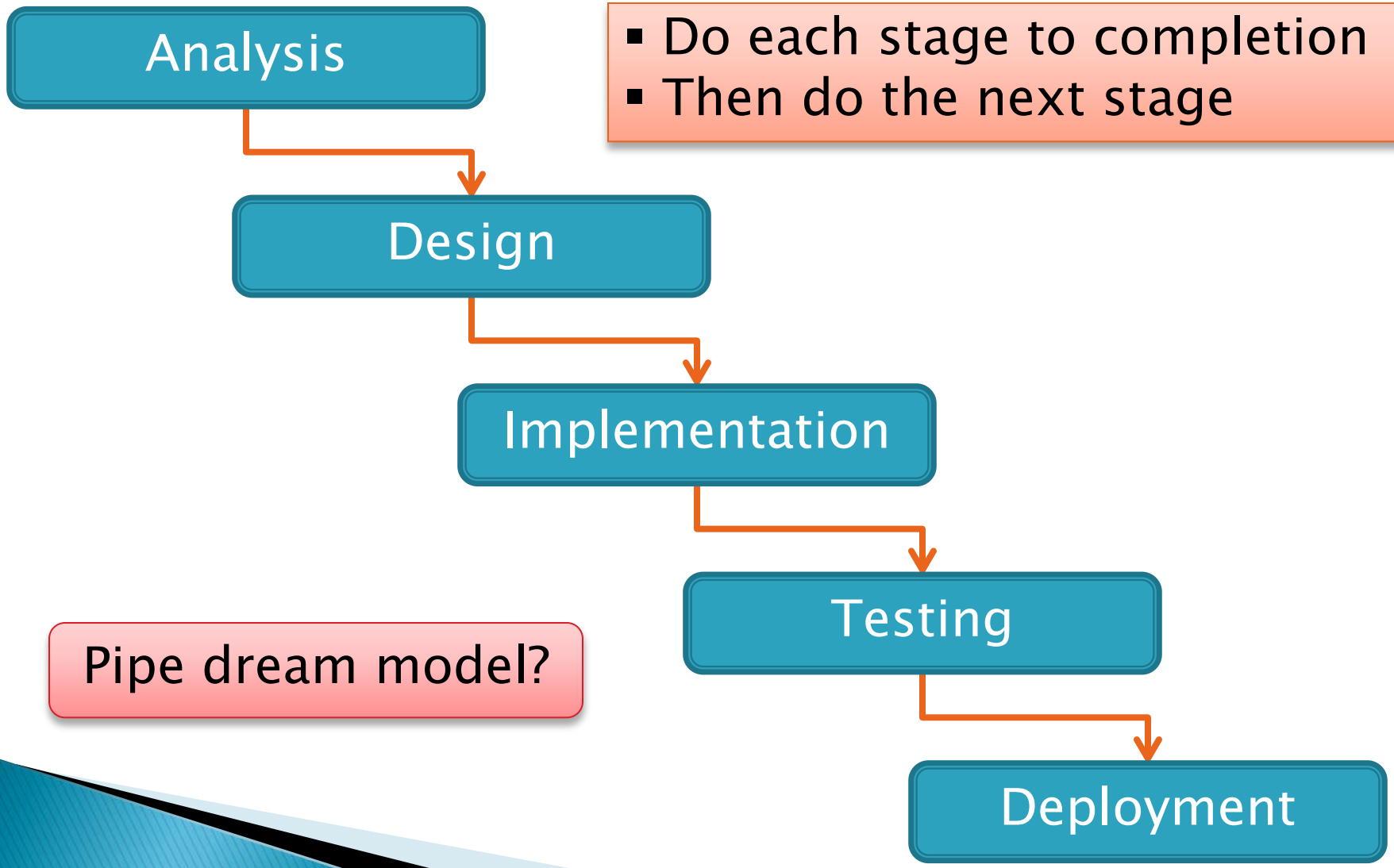


Formal Development Processes

- Standardized approaches intended to:
  - Reduce costs
  - Increase predictability of results

- Examples:
  - Waterfall model
  - Spiral model
  - “Rational Unified Process”

Waterfall Model



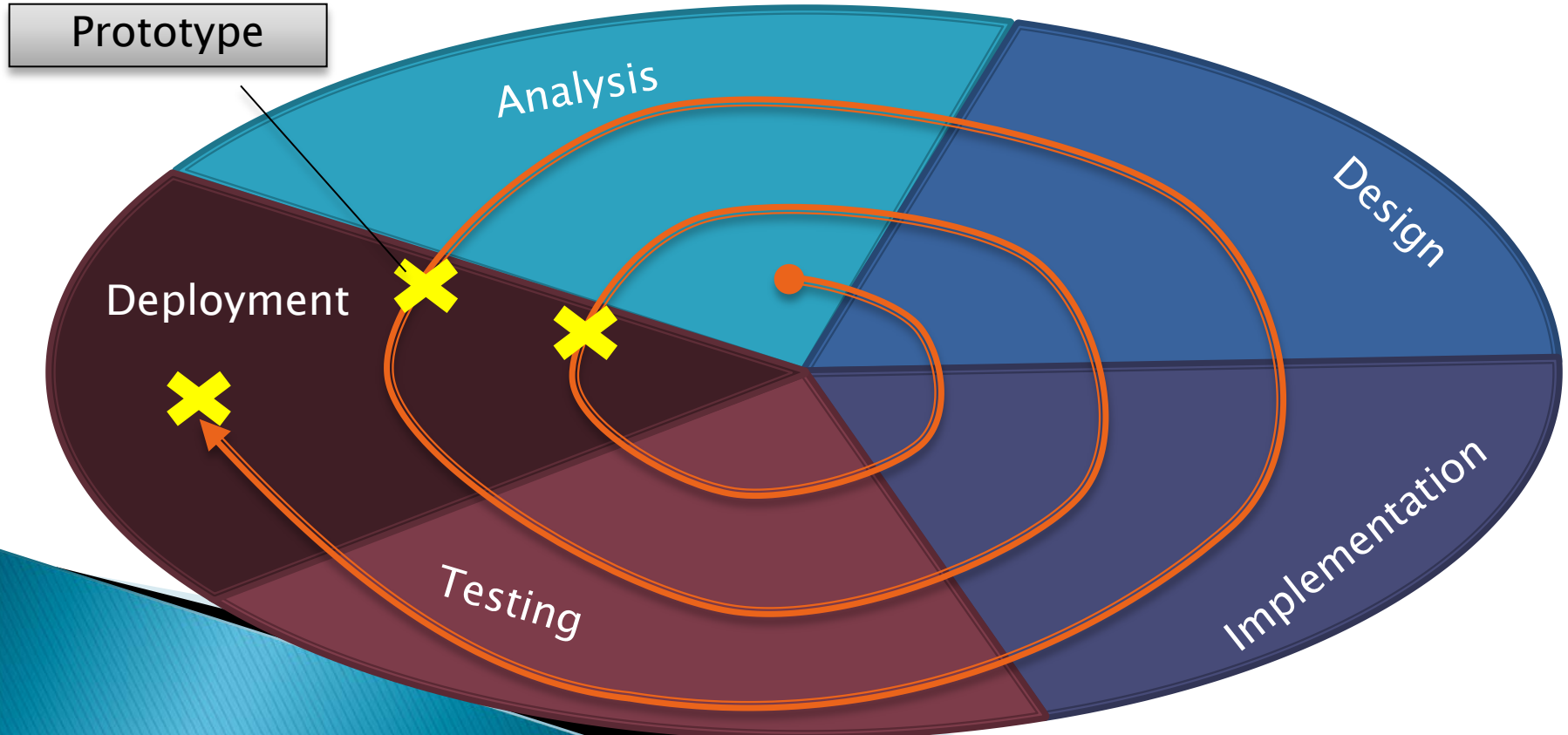
- Do each stage to completion
- Then do the next stage

Pipe dream model?

Spiral Model

- Schedule overruns
- Scope creep

- Repeat phases in a cycle
- Produce a prototype at end of each cycle
- Get early feedback, incorporate changes



Extreme Programming—XP

- Like the spiral model with **very** short cycles
- Pioneered by Kent Beck
- One of several “agile” methodologies, focused on building high quality software quickly
- Rather than focus on rigid process, XP espouses 12 key practices…

The XP Practices

- Realistic planning
- Small releases
- Shared metaphors
- Simplicity
- **Testing**
- **Refactoring**
- **Pair programming**
- Collective ownership
- Continuous integration
- 40-hour week
- On-site customer
- **Coding standards**

When you see opportunity to make code better, do it

Use descriptive names

Q6

Object-Oriented Design

- A practical technique

Object-Oriented Design

- We won't use full-scale, formal methodologies
  - Those are in later SE courses
- We will practice a common object-oriented design technique using CRC Cards
- Like any design technique, **the key to success is practice**

Key Steps in Our Design Process

1. **Discover classes** based on requirements

2. **Determine responsibilities** of each class

3. **Describe relationships** between classes

Discover Classes Based on Requirements

- Brainstorm a list of possible classes
  - Anything that might work
  - No squashing

Discover Classes Based on Requirements

- Prompts:
  - Look for **nouns**
  - Multiple objects are often created from each class
    - So look for **plural concepts**
  - Consider how much detail a concept requires:
    - A lot? Probably a class
    - Not much? Perhaps a primitive type
- Don't expect to find them all → add as needed

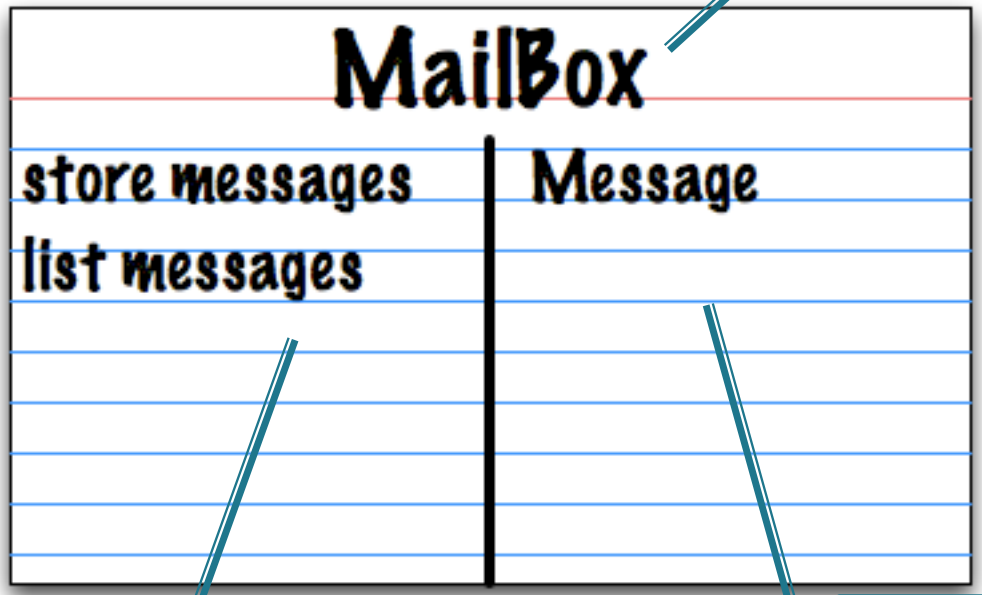
Tired of hearing this yet?

Determine Responsibilities

- Look for **verbs** in the requirements to identify **responsibilities** of your system
- Which class handles the responsibility?
- Can use **CRC Cards** to discover this:
  - **Classes**
  - **Responsibilities**
  - **Collaborators**

CRC Cards

- Use one index card per class



CRC Card Technique

1. Pick a **responsibility** of the program
2. Pick a **class** to carry out that responsibility
   - Add that responsibility to the class's card
3. Can that class carry out the responsibility by itself?
   - Yes → Return to step 1
   - No →
     - Decide which classes should help
     - List them as **collaborators** on the first card
     - `

CRC Card Tips

- **Spread the cards out** on a table
  - Or sticky notes on a whiteboard instead of cards
- **Use a "token"** to keep your place
  - A quarter or a magnet
- **Focus on high-level responsibilities**
  - Some say < 3 per card
- **Keep it informal**
  - Rewrite cards if they get too sloppy
  - Tear up mistakes
  - Shuffle cards around to keep "friends" together

BallWorlds

- Meet your partner (see link in part 3 of spec)
- Carefully read the requirements and provided code
- Ask questions (instructor and TAs).

BallWorlds Worktime

- Pulsar, Mover, etc.