

# CSSE 220 Day 21

Object-Oriented Design

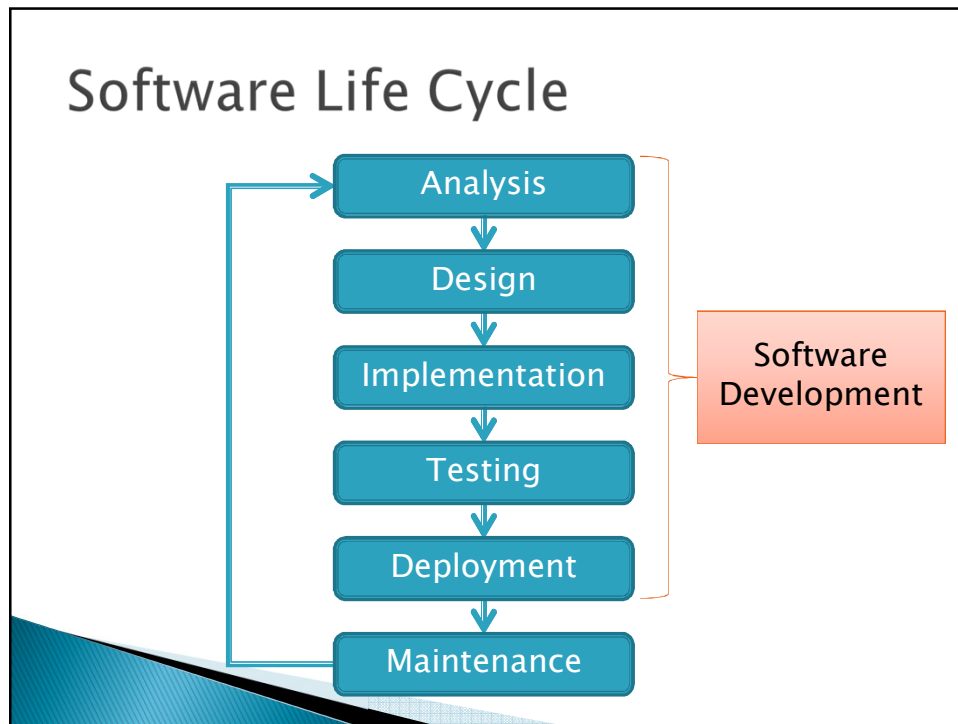
No SVN checkout today

Questions?

## Today's Plan

- ▶ Software development methods
- ▶ Object-oriented design with CRC cards
- ▶ LayoutManagers for Java GUIs
- ▶ BallWorlds work time

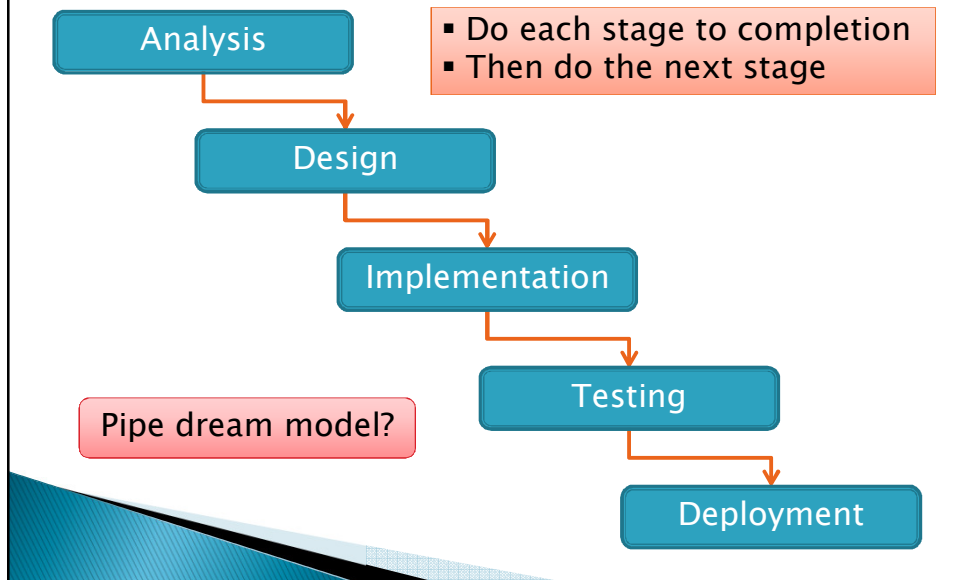
Software  
Development  
Methods



## Formal Development Processes

- ▶ Standardized approaches intended to:
  - Reduce costs
  - Increase predictability of results
- ▶ Examples:
  - Waterfall model
  - Spiral model
  - “Rational Unified Process”

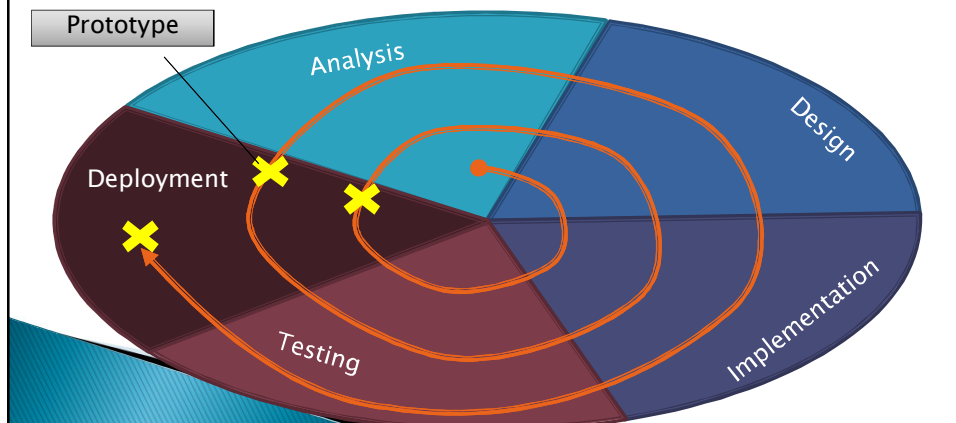
## Waterfall Model



## 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

Q1

# 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

Q2

## 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?

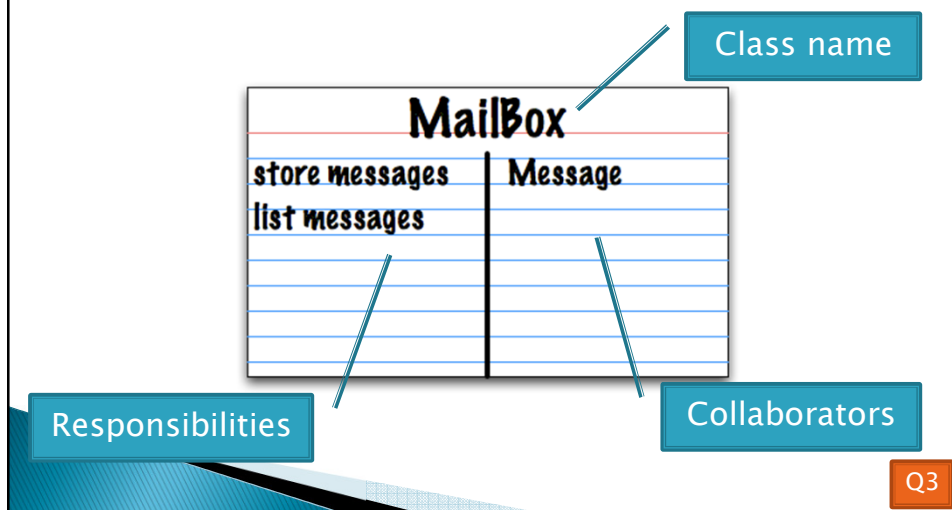
## Determine Responsibilities

- ▶ 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
    - Add additional responsibilities to the collaborators' cards

## 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

Break

» [These go to 11](#)

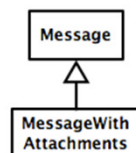
## Describe the Relationships

- ▶ Classes usually are related to their collaborators
- ▶ Draw a UML class diagram showing how
- ▶ Common relationships:
  - **Inheritance**: only when subclass **is a** special case
  - **Aggregation**: when one class **has a** field that references another class
  - **Dependency**: like aggregation but transient, usually for method parameters, **“has a” temporarily**
  - **Association**: any other relationship, can label the arrow, e.g., **constructs**

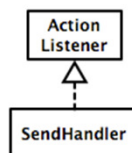
NEW!

## Summary of UML Class Diagram Arrows

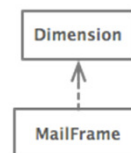
Inheritance  
(is a)



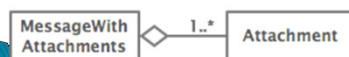
Interface  
Implementation  
(is a)



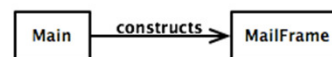
Dependency  
(depends on)



Aggregation  
(has a)



Association



Q4

## Object-Oriented Design



- Draw UML class diagrams based on your CRC cards
- Initially just show classes (not insides of each)
- Add insides for two classes

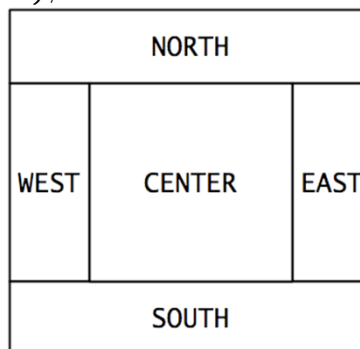
## Some Notes on Layout Managers



- When JFrame's and JPanel's defaults just don't cut it.

## Recall: How many components can a JFrame show by default?

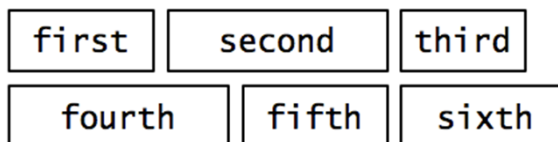
- ▶ Answer: 5
- ▶ We use the two-argument version of `add`:
- ▶ `JPanel p = new JPanel ();`  
`frame.add(p, BorderLayout.SOUTH);`
- ▶ JFrame's default `LayoutManager` is a `BorderLayout`
- ▶ `LayoutManager` instances tell the Java library how to arrange components
- ▶ `BorderLayout` uses up to five components



Q5

## Recall: How many components can a JPanel show by default?

- ▶ Answer: arbitrarily many
- ▶ Additional components are added in a line
- ▶ JPanel's default `LayoutManager` is a `FlowLayout`



...

## Setting the Layout Manager

- ▶ We can set the layout manager of a JPanel manually if we don't like the default:

```
JPanel panel = new JPanel ();
panel .setLayout(new GridLayout(4, 3));
panel .add(new JButton("1"));
panel .add(new JButton("2"));
panel .add(new JButton("3"));
panel .add(new JButton("4"));
// ...
panel .add(new JButton("0"));
panel .add(new JButton("#"));
frame.add(panel);
```



## Lots of Layout Managers

- ▶ A **LayoutManager** determines how components are laid out within a container
  - **BorderLayout**. When adding a component, you specify center, north, south, east, or west for its location. (Default for a JFrame.)
  - **FlowLayout**: Components are placed left to right. When a row is filled, start a new one. (Default for a JPanel.)
  - **GridLayout**. All components same size, placed into a 2D grid.
  - Many others are available, including **BoxLayout**, **CardLayout**, **GridBagLayout**, **GroupLayout**
  - If you use **null** for the LayoutManager, then you must specify every location using coordinates
    - More control, but it doesn't resize automatically

Q6

## Additional Resources on Layout Managers

- ▶ Chapter 18 of Big Java
- ▶ Swing Tutorial
  - <http://java.sun.com/docs/books/tutorial/ui/index.html>
  - Also linked from schedule

Work Time

»» BallWorlds

Q7-Q8