

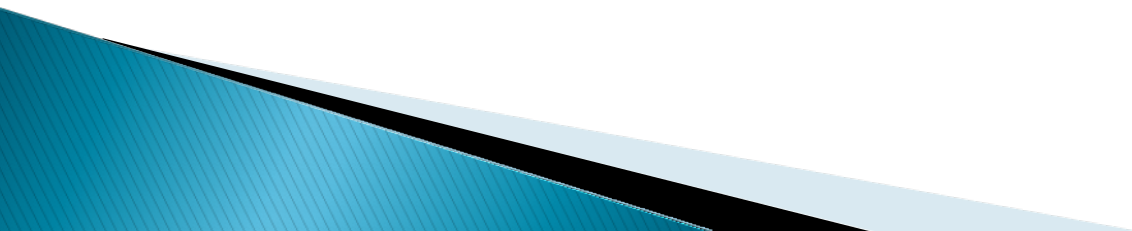
# CSSE 220 Day 11

Exam review  
Designing classes

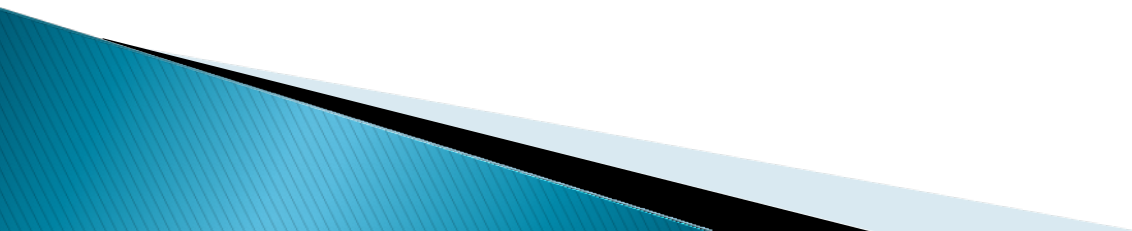
# Exam 1 Details (recap)

- ▶ Chapters 1–8 (I said 1–9 before)
  - Applets are not covered.
  - A [list](#) of textbook items to pay special attention to
    - And some [terminology](#) .
    - Also linked from Day 12 on the schedule page
- ▶ HW 1–10 (except game of Life)
- ▶ If you wish, you can take a whole class period for the written part, and two periods for the programming part. **See the Schedule page**
- ▶ Allowed resources: See Session 8 slides
- ▶ Review in-class Wednesday, Jan 7
  - Bring questions. I won't prepare anything but I am happy to discuss whatever you want, including working examples (you pick them)

# Exam Questions

- ▶ Exam process?
  - ▶ Exam timing?
  - ▶ Material we have covered?
  
  - ▶ Life Questions?
- 

# Today

- ▶ Questions about Exam
  - ▶ Methods with variable number of arguments
  - ▶ Designing Classes
  - ▶ Timer-triggered action events.
  - ▶ Work on Game of Life program
- 

# Methods with a variable number of arguments

- »» A method that returns the maximum of all of its integer arguments

```
public class VariableArgs {
```

```
    public static int max(int ... args){
```

```
        if (args.length == 0)
```

```
            throw new IllegalArgumentException(
```

```
                "Max must take at least one argument");
```

```
        int result = args[0];
```

```
        for (int i=1; i<args.length; i++)
```

```
            result = Math.max(result, args[i]);
```

```
    return result;
```

```
}
```

```
public static void main(String[] args) {
```

```
    System.out.println("max: " + max(2, 5, 7, 3, 4, 6));
```

```
}
```

```
}
```

**args** is an array of integers, containing all of the actual arguments

A call to max (result is 7)

# What is good object-oriented design?

»» It starts with good classes...

# Good Classes Typically

- ▶ Often come from **nouns** in the problem description
- ▶ May...
  - Represent **single concepts**
    - **Circle, Investment**
  - Be **abstractions of real-life entities**
    - **BankAccount, TicTacToeBoard**
  - Be **actors**
    - **Scanner, CircleViewer**
  - Be **utilities**
    - **Math**



# What Stinks? **Bad** Class Smells

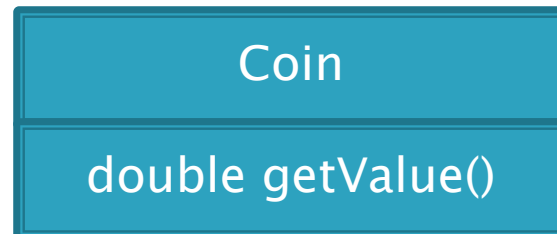
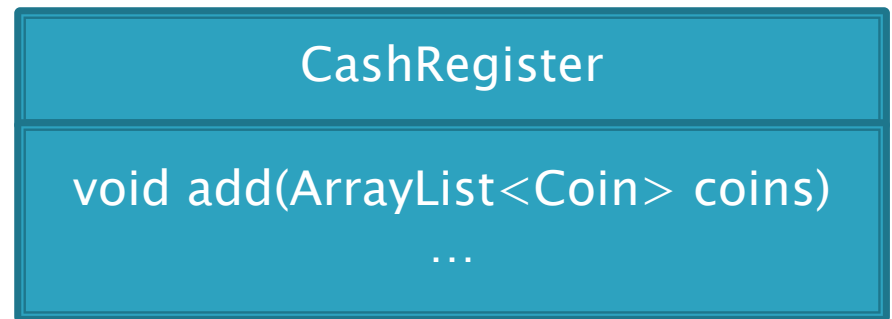
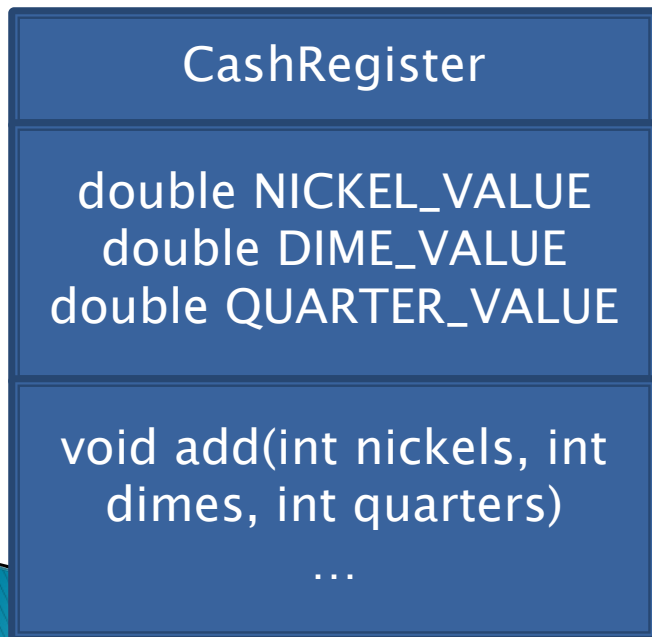
- ▶ Can't tell what it does from its name
  - **PayCheckProgram**
- ▶ Turning a single action into a class
  - **ComputePaycheck**
- ▶ Name isn't a noun
  - **Interpolate, Spend**

# Analyzing Quality of Class Design

- ▶ Cohesion
- ▶ Coupling

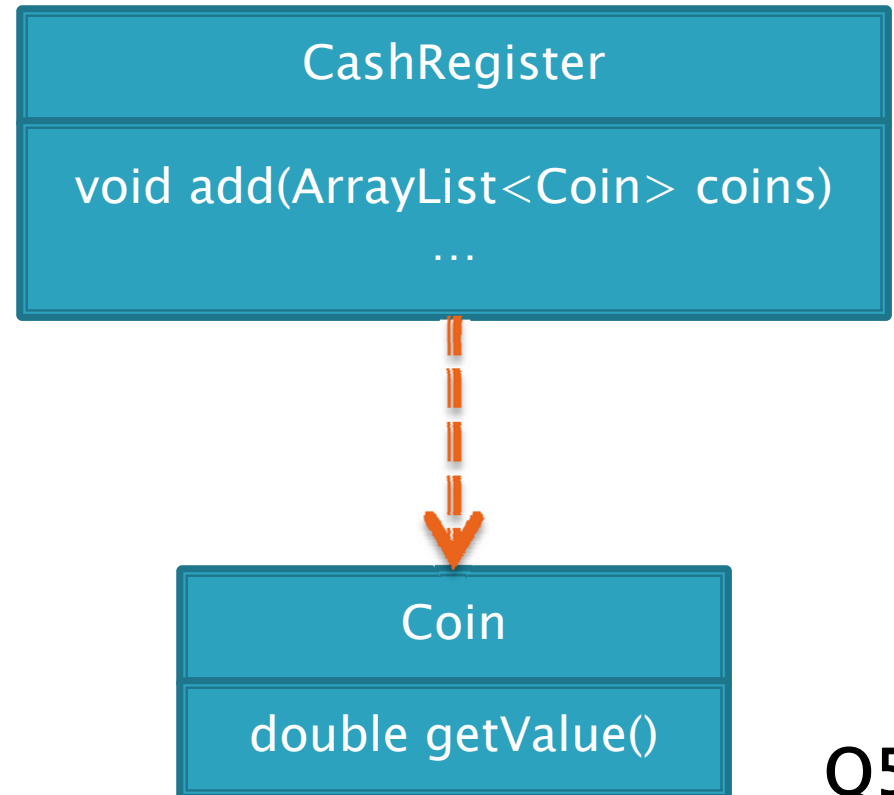
# Cohesion

- ▶ A class should represent a **single concept**
- ▶ Public methods and constants should be **cohesive**
- ▶ Which is more cohesive?



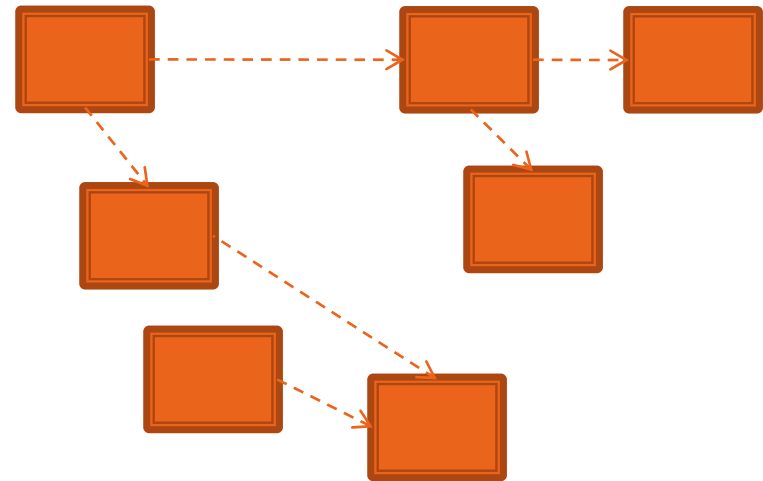
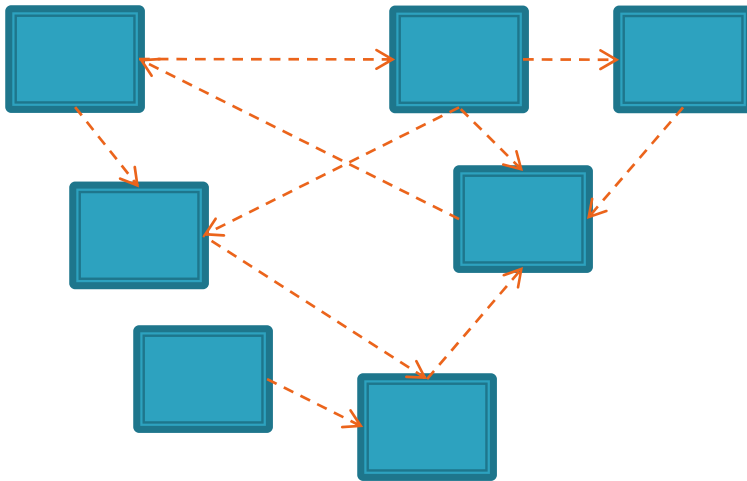
# Dependency Relationship

- ▶ When one class requires another class to do its job, the first class **depends on** the second
- ▶ Shown on UML diagrams as:
  - dashed line
  - with open arrowhead



# Coupling

- ▶ Lots of dependencies == high coupling
- ▶ Few dependencies == low coupling



- ▶ Which is better? Why?


# Quality Class Designs

- ▶ High cohesion
- ▶ Low coupling

# Accessors and Mutators Review

- ▶ **Accessor method**: accesses information *without changing any*
- ▶ **Mutator method**: *modifies* the object on which it is invoked

# Immutable Classes

- ▶ Accessor methods are very predictable
    - Easy to reason about!
  - ▶ **Immutable classes:**
    - Have only accessor methods
    - No mutators
  - ▶ Examples: **String, Double**
  - ▶ Is **Rectangle** immutable?
- 



# Immutable Class Benefits

- ▶ Easier to reason about, less to go wrong
- ▶ Can pass around instances “fearlessly”

# Side Effects

- ▶ **Side effect**: any modification of data
- ▶ **Method side effect**: any modification of data *visible* outside the method
  - Mutator methods: side effect on implicit parameter
  - Can also have side effects on other parameters:
    - ```
public void transfer(double amt, Account other)
{
    this.balance -= amt;
    other.balance += amt;
}
```

Avoid this if you can! Document it if you can't

# Documenting Side Effects

```
/**  
 * Transfers the given amount from this  
 * account to the other account. Mutates  
 * this account and other.  
 *  
 * @param amt  
 *         amount to be transferred  
 * @param other  
 *         receiving account (mutated)  
 */  
public void transfer(double amt, Account other) {  
    this.balance -= amt;  
    other.balance += amt;  
}
```

# Class Design Exercise

- ▶ It's part of HW 11
- ▶ Do this one with your Game of Life Team
- ▶ Due on Monday (as is Life)
- ▶ Don't spend more than a couple of hours on it.

# Timer-triggered Action Events

- ▶ See Big Java Section 9.9
- ▶ Timer constructor takes as arguments:
  - a firing interval time (in milliseconds)
  - an ActionListener object.

```
Timer autoClick = new Timer(AUTO_CLICK_INTERVAL,  
                             updateButton);  
autoClick.start();
```

```
import javax.swing.timer;
```

# Back to Life

- ▶ Work with your partner