CSSE 220 Day 14

Designing Classes

Check out the Static project from SVN

Questions?

What is good object-oriented design?

- It starts with good classes…

Good Classes Typically

- Come from nouns in the problem description
- May…
  - Represent single concepts
    - `Circle`, `Investment`
  - Represent visual elements of the project
    - `FacesComponent`, `UpdateButton`
  - Be abstractions of real-life entities
    - `BankAccount`, `TicTacToeBoard`
  - Be actors
    - `Scanner`, `CircleViewer`
  - Be utilities
    - `Math`

Q1

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**

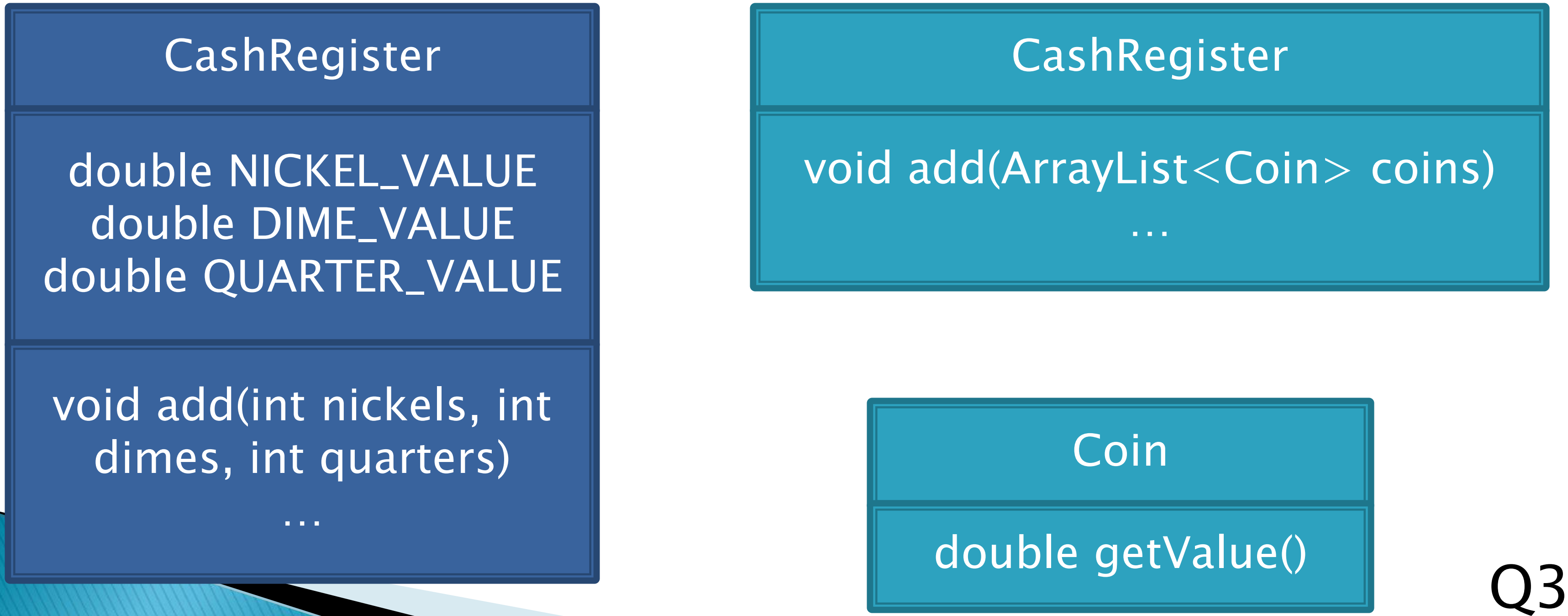
Q2

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?



CashRegister

double NICKEL_VALUE
double DIME_VALUE
double QUARTER_VALUE

void add(int nickels, int dimes, int quarters)
…

CashRegister

void add(ArrayList<Coin> coins)
…

Coin

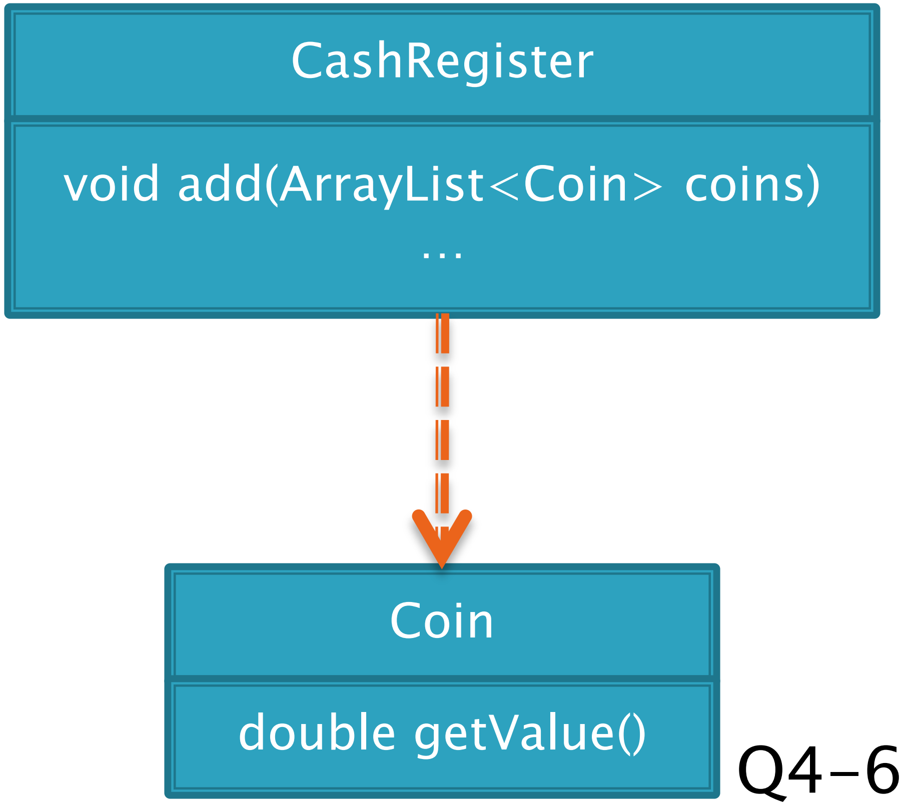
double getValue()

Q3

Dependency Relationship

- When one classes requires another class to do its job, the first class depends on the second

- Shown on UML diagrams as:
  - dashed line
  - with open arrowhead



CashRegister

void add(ArrayList<Coin> coins)
…

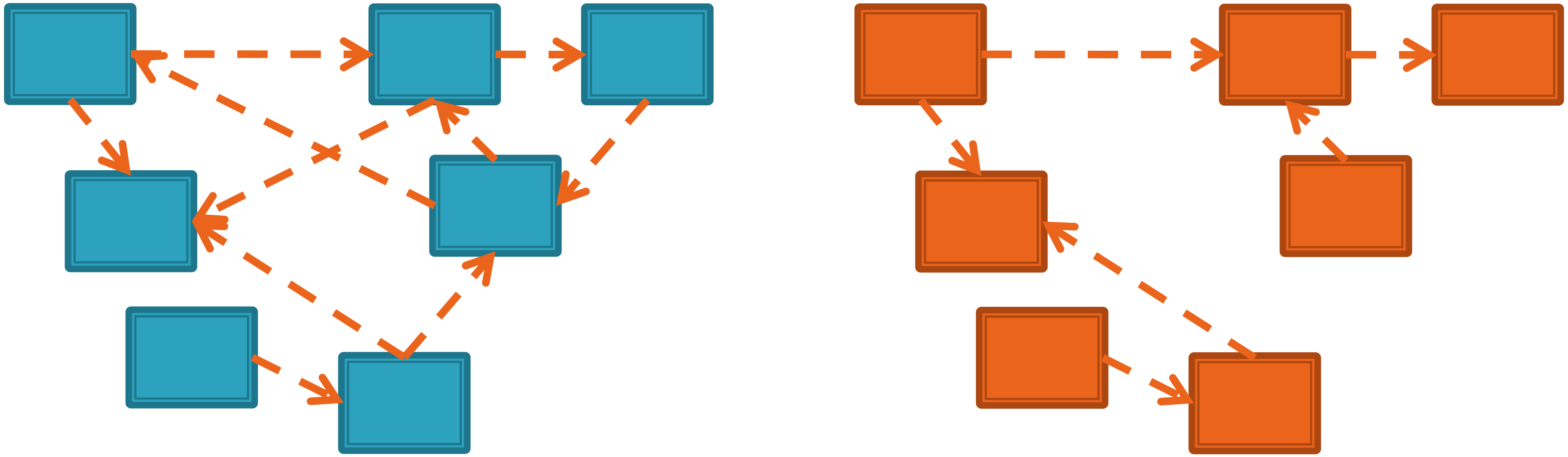
Coin

double getValue()

Q4–6

Coupling

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



- Which is better? Why?

Q7

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

Q8

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”

Q9

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!

Q10

Quality Class Designs

- High cohesion
- Low coupling
- Class names are nouns; Method names are verbs
- Immutable where practical
  - Document where not
- Inheritance for code reuse
- Interfaces to allow others to interact with your code

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

Q11

Discover Classes Based on Requirements

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

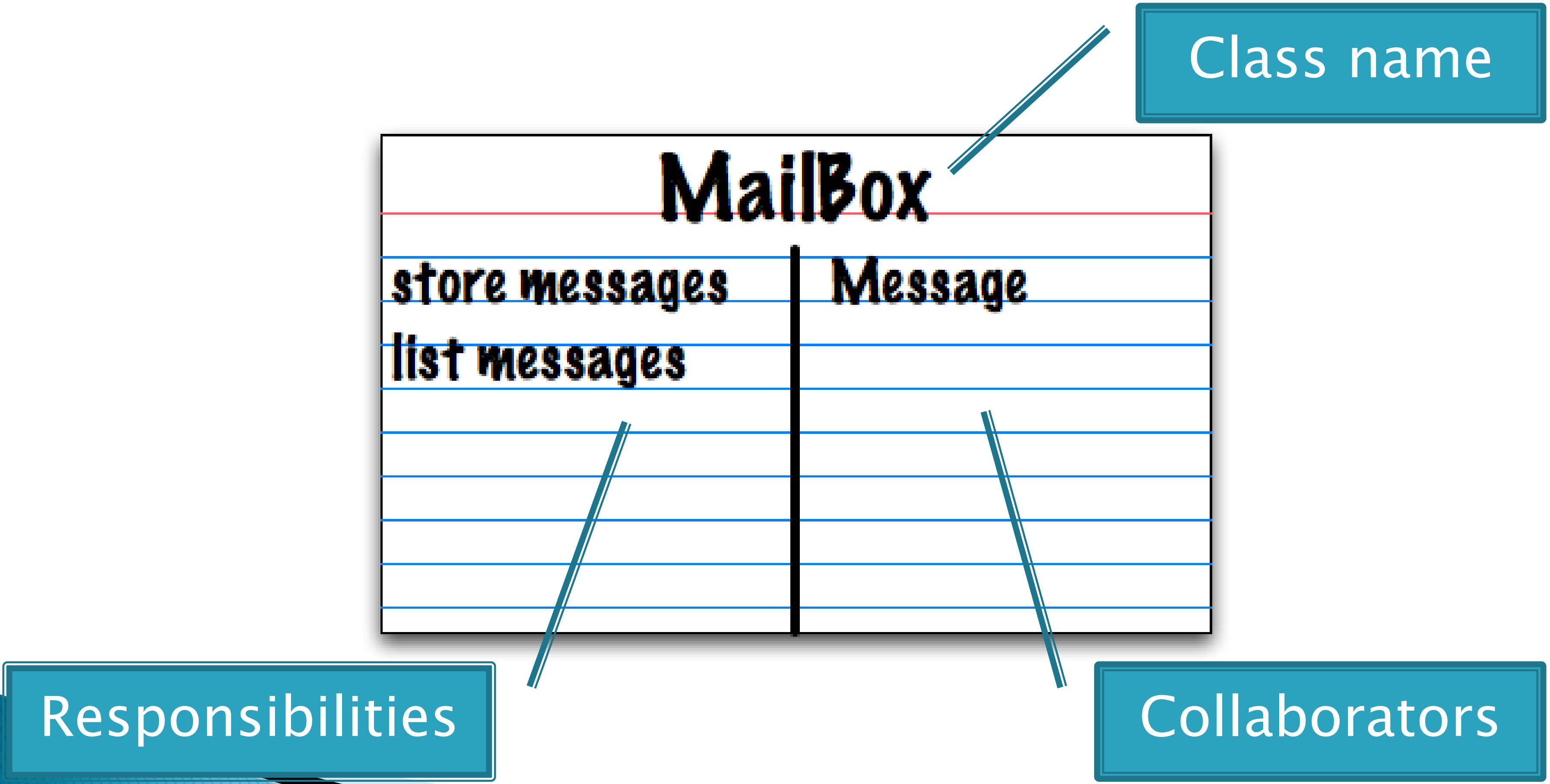
- Look for **verbs** in the requirements to identify **responsibilities** of your system

- Which class handles the responsibility?

- Can use **CRC Cards** to discover this:
  - **Classes**
  - **Responsibilities**
  - **Collaborators**

CRC Cards

- Use one index card per class



Class name

MailBox

store messages

list messages

Message

Responsibilities

Collaborators

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 to sloppy
  - Tear up mistakes
  - Shuffle cards around to keep “friends” together

Example: Chess

- High cohesion
- Low coupling
- Immutable where practical
  - Document where not
- Inheritance for code reuse
- Interfaces to allow others to interact with your code

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

Design a program that lets two people play chess against each other.

- Assume a single, shared computer and input via the Console.

Describe the Relationships

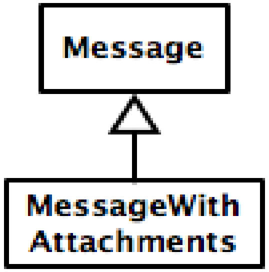
- Classes usually are related to their collaborators
- Draw a UML class diagram showing how
- Common relationships:
  - **Inheritance**: only when subclass **is a** special case
  - **Aggregation**: when one class **has a** **field** that references another class
  - **Dependency**: like aggregation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

Summary of UML Class Diagram Arrows

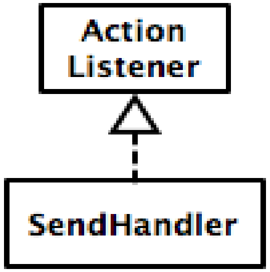
Exercise: Draw a UML class diagram based on your CRC cards or our CRC cards

- Show just classes (not insides of each).
- For homework:
  - Draw using UMLet
  - Add insides for two classes

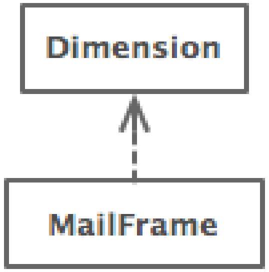
Inheritance (is a)



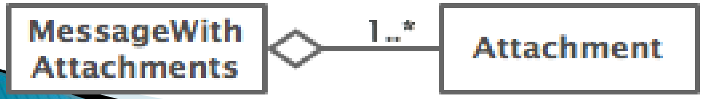
Interface Implementation (is a)



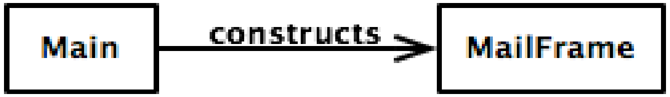
Dependency (depends on)



Aggregation (has a)



Association



Object-Oriented Design

- Very brief demo of UMLet. Show how to:
  - Create a diagram element
  - Type data for that element

What is static Anyway?

- static members (fields and methods)…
  - are **not** part of objects
  - are **part of the class itself**

- Mnemonic: objects can be passed around, but static members stay put

Static Methods

- Cannot refer to **`this`**
  - They aren't in an object, so there is no **`this`**!
- Are called without an implicit parameter
  - **`Math.sqrt(2.0)`**

Class name, not object reference

When to Declare Static Methods

- Helper methods that don't refer to **this**
  - Example: creating list of **`Coordinates`** for glider
- Utility methods like *sin* and *cos* that are not associated with any object
  - Another example:

```
public class Geometry3D {
    public static double sphereVolume(double radius) {
        ...
    }
}
```

- The **`main()`** method is static
  - Why is it static? What objects exist when the program starts?

Q12

Static Fields

- We’ve seen static final fields

- Can also have static fields that aren’t final
  - Should be private
  - Used for information shared between instances of a class
    - Example: the number of times a foo() method of the Blah class is called by ANY object of the Blah class

Q13

Two Ways to Initialize

- private static int nextAccountNumber = 100;

- or use “static initializer” blocks:

```
public class Hogwarts {
    private static ArrayList<String> FOUNDERS;

    static {
        FOUNDERS = new ArrayList<String>();
        FOUNDERS.add("Godric Gryfindor");
        // ...
    }

    // …
}
```

Q14-15

Exercise

- Polygon
  - Run the program
  - Note that the least/most number of sides data is shown but is -1 (not yet implemented)
  - Read all the TODO's in the Polygon class
  - Do and test the TODO's for most number of sides, asking questions as needed
  - Do and test the TODO's for least number of sides
    - You might find Integer.MAX_VALUE helpful

Before you leave today, arrive a time with your partner to complete the Game of Life.