CSSE 220 Day 16

Designing Classes

Check out *DesigningClasses* 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`

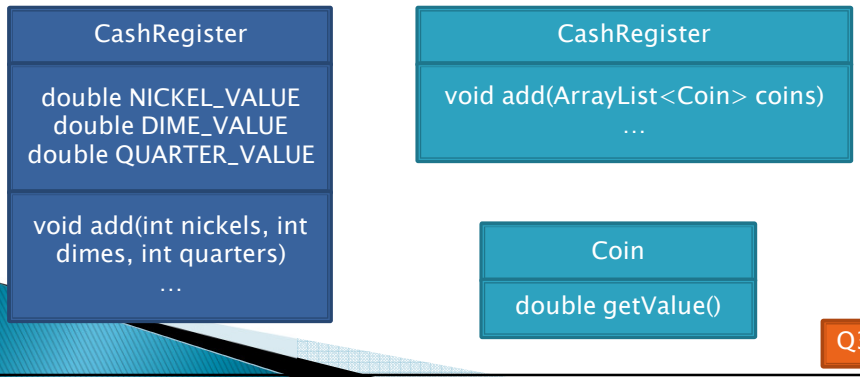
Q2

Analyzing Quality of Class Design

- Cohesion
- Coupling

Cohesion

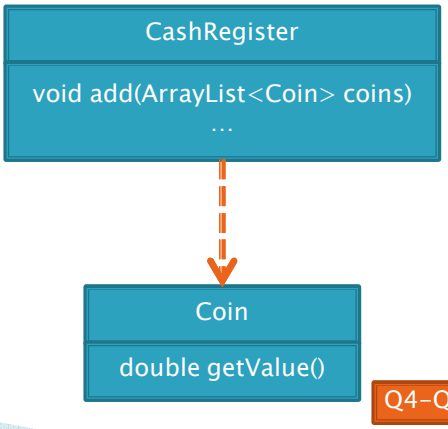
- A class should represent a single concept
- Public methods and constants should be cohesive
- Which is more cohesive?



Q3

Dependency Relationship

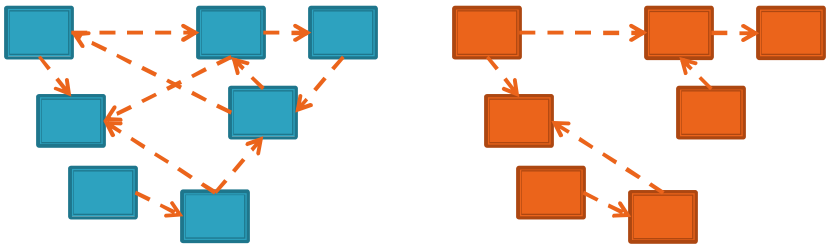
- When one class requires another class to do its job, the first class depends on the second
- Shown on UML diagrams as:
  - dashed line
  - with open arrowhead



Q4–Q6

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

Coming attractions

Class Design Exercise

- See HW16 –Chess exercise
  Work in groups of three or four on the whiteboards

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

- Inside a class, the class name is optional but much clearer to use (just like **this** for instance fields and methods)

When to Declare Static Methods

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

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) {
        ...
    }
}
```

Q11

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 particular method of the a class is called by ANY object of that class

Q12

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");
            // ...
      }

      // …
}
```

Work Time

- Homework 16: Polygon

A Polygon exercise

- Run the program in the polygon package
- 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

Q13-Q14