CSSE 220 Day 13

Encapsulation
Coupling and Cohesion
Scoping

Please download EncapsulationExamples from your SVN

The plan

- Learn 3 essential object oriented design terms:
  - Encapsulation
  - Coupling
  - Cohesion
- Scope (if we have time)

What if there was no String class?

- Instead, what if we just passed around arrays of characters - char[]
- And every String function that exists now, would instead be a function that operated on arrays of characters
- E.g. char[] stringSubstring(char[] input, int start, int end)
- Would things be any different? Discuss this with the person next to you.

The Point of All Program Design

- Say you've somebody has written a program, and it works and it has no bugs, but it is *poorly designed*. What does that mean? Why do we care?
- I think there are two things

Encapsulation

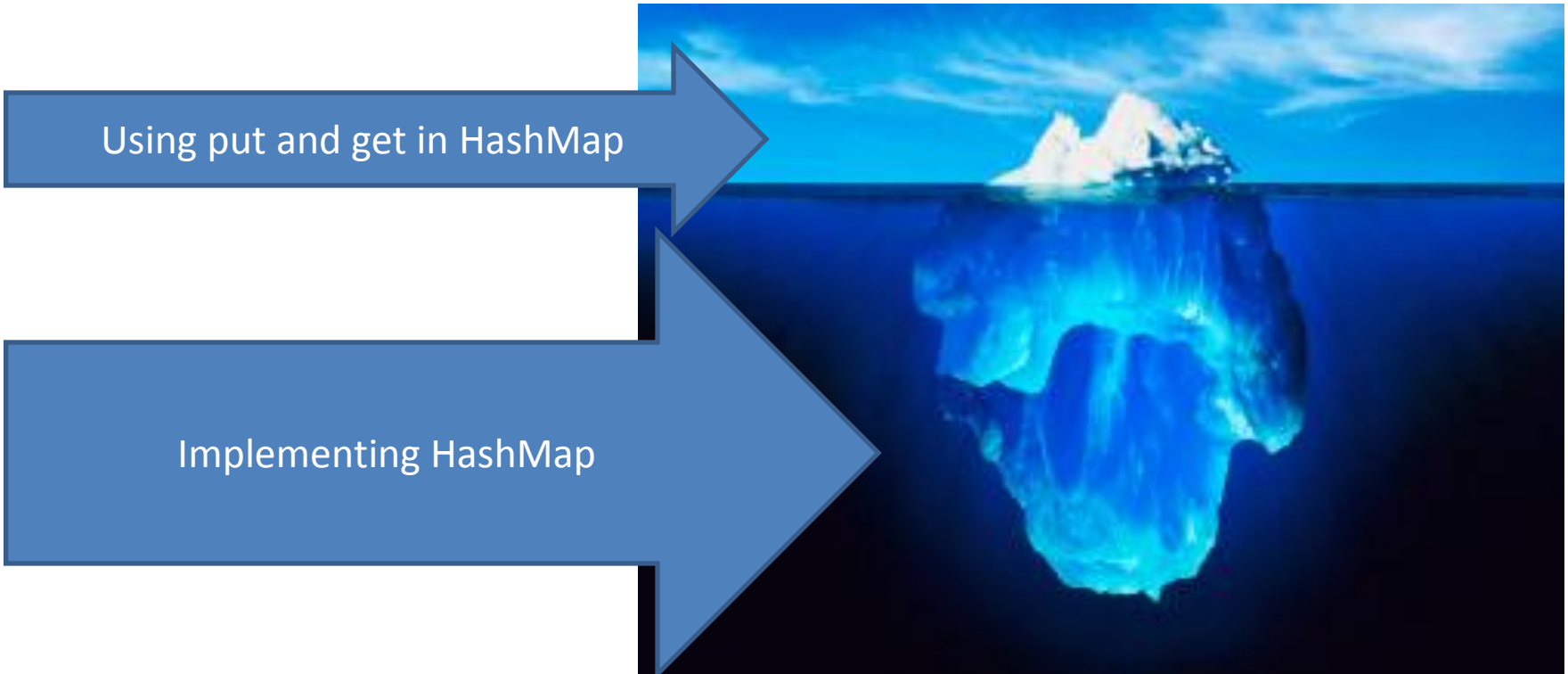
- Mikes definition “grouping some data and the operations that use that data into one thing (an object) and preventing that data from being changed except by using those operations”

Encapsulation

- Makes your program easier to understand by
  - Grouping related stuff together

Encapsulation

- Makes your program easier to understand by…
  - Saving you from having to think about how complicated things might be



Encapsulation

Makes your program easier to change by…

- Allowing you to change how your data is represented

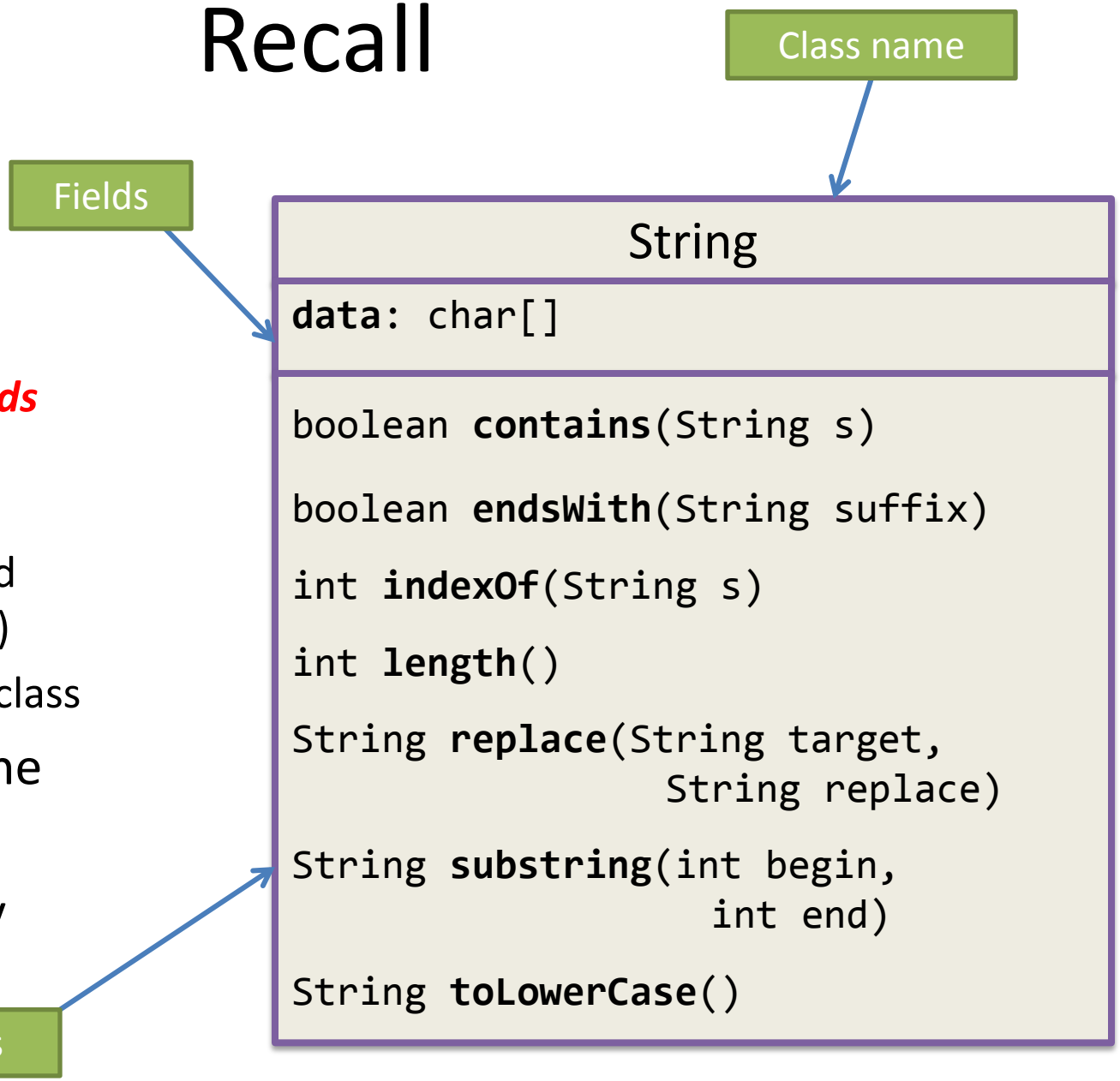
City Temperature Activity

- I will split you into two groups
  - One group will solve the problem by creating a new class (see the Class Section example if you are unsure how to do that)
  - The other group will just write the code in main (see the Letters Example if you are unsure how to do that)
- If you finish early, try to solve it the other way too

Recall

- Shows the:
  - ***Attributes*** (data, called ***fields*** in Java) and
  - ***Operations*** (functions, called ***methods*** in Java)

  of the objects of a class
- Does *not* show the implementation
- Is *not* necessarily complete



TwoVsTwo

- Look at the code to understand the problem
- Try to solve it using classes and encapsulation
  - Decide what classes/methods you would use (I used two new classes and TwoVsTwo main)
- Draw UML for the classes/methods
- Don’t start coding till I or the TA have looked at your classes!

The plan

- Learn 3 essential object oriented design terms:
  - Encapsulation
  - Coupling
  - Cohesion
- Scope (if we have time)

Coupling and Cohesion

- Two terms you need to memorize
- Good designs have high cohesion and low coupling

At a very high level:

- Low cohesion means that you have a small number of really large classes that do too much stuff
- High coupling means you have many classes which depend too much on each other

Imagine I want to make a Video Game. Here are two classes in my design. Which is more cohesive?

GameRunner
main(args:String) loadLevel(levelName:String) moveEnemies() drawLevel(g:Graphics2D) computeScore():int computeEnemyDamage() handlePlayerInput() doPowerups(…) runCutscene(cutsceneName:String) //some more stuff

Image
loadImageFile(filename:String) setPosition(x:int,y:int) drawImage(g:Graphics2D)

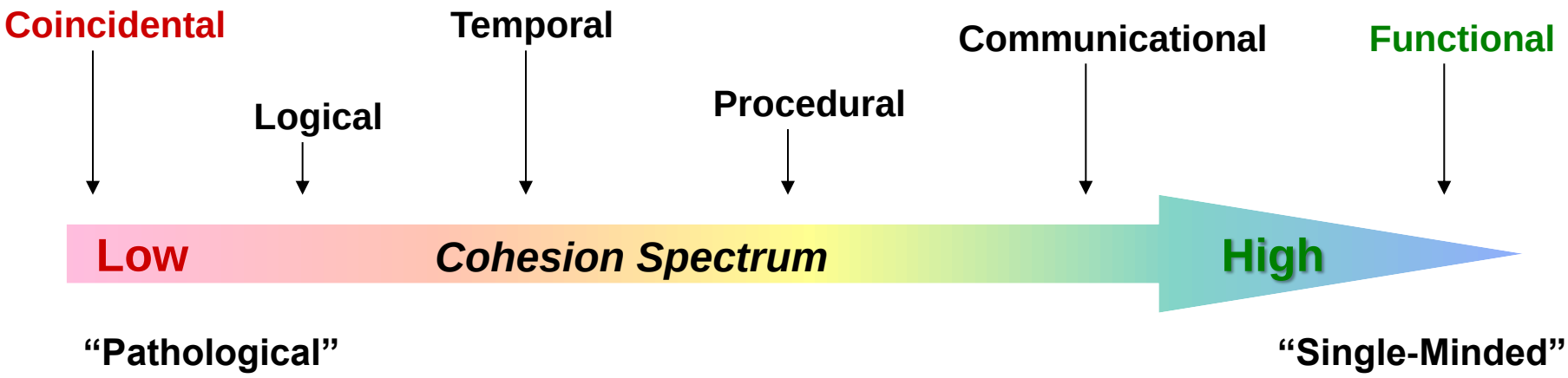
*Note that in both these classes I've omitted the fields for clarity

Cohesion

- A class should represent a single concept. All interface features should be closely related to the single concept that the class represents. Such a class is said to be cohesive.

  - Your textbook

Types of Cohesion



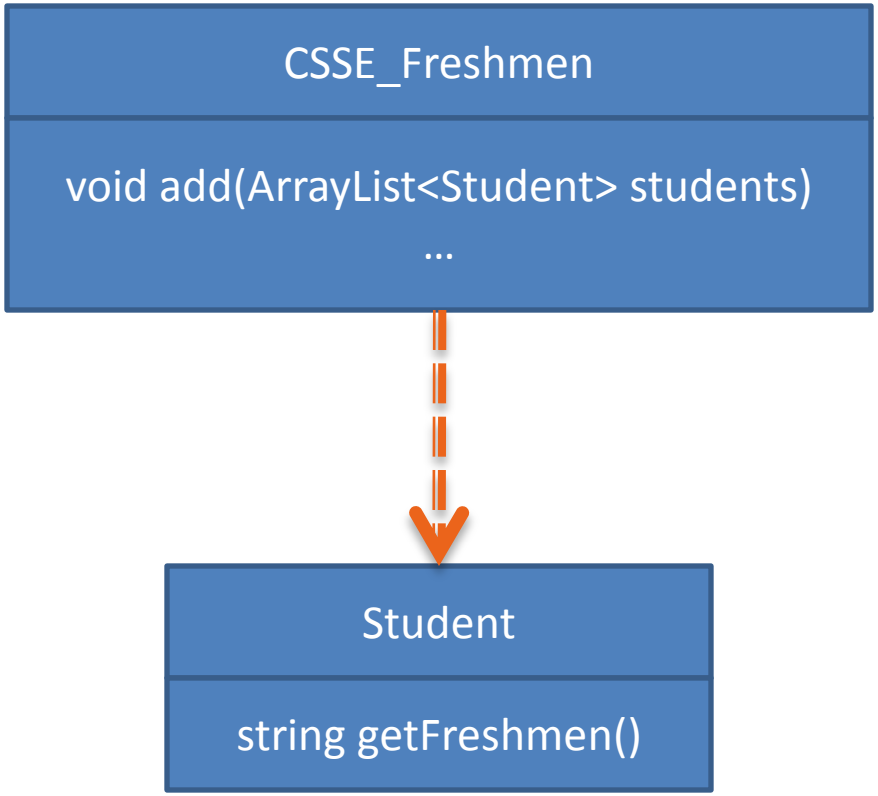
Measure of how related or focused the responsibilities of a single class are

Coincidental: multiple, completely unrelated actions or components
Logical: series of related actions or components (e.g. library of IO functions)
Temporal: series of actions related in time (e.g. initialization modules)
Procedural: series of actions sharing sequences of steps.
Communicational: procedural cohesion but on the same data.
Functional: one action or function

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



The plan

- Learn 3 essential object oriented design terms:
  - Encapsulation
  - Coupling
  - Cohesion
- Scope (if we have time)

Coupling

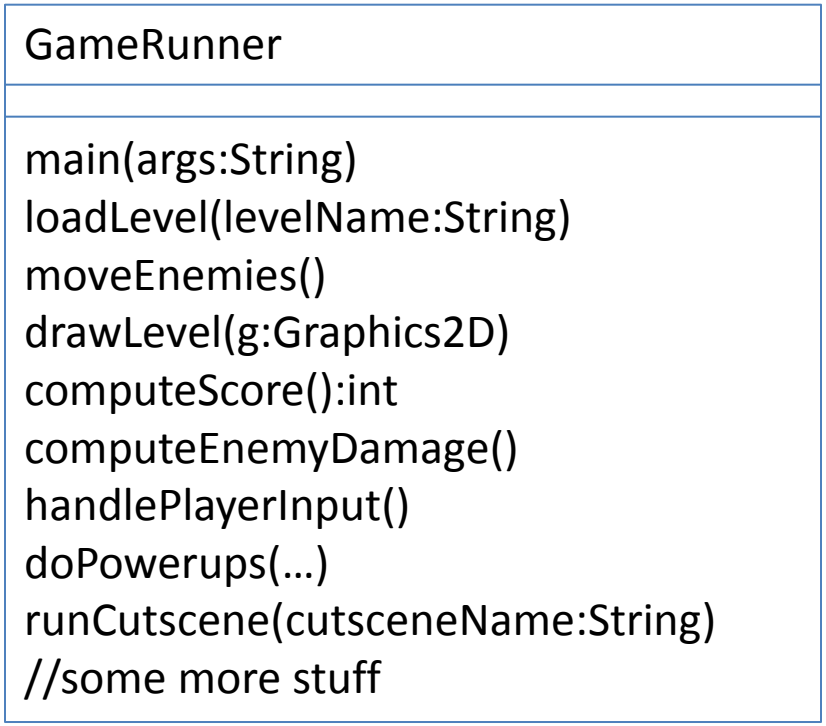
- Coupling is one object depends strongly on another

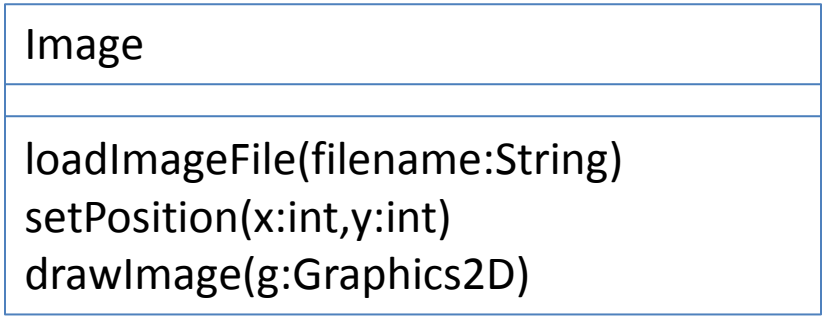
```
//do setup must be called first
this.otherObject.doSetup(var1, var2, var3);

//now we compute the parameter
int var4 = computeForOtherObject(var1,var2);
this.otherObject.setAdditionalParameter(var4);

//finally we display
this.otherObject.doDisplay(this.var5, this.var6);
```

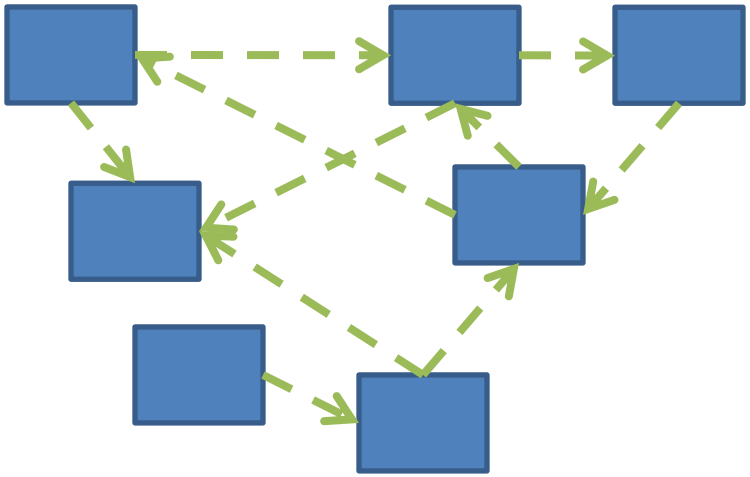
Note that in this design, GameRunner probably had many objects of the image class, but Image does not know the GameRunner class even exists. That's a sign of low coupling between Image and GameRunner.

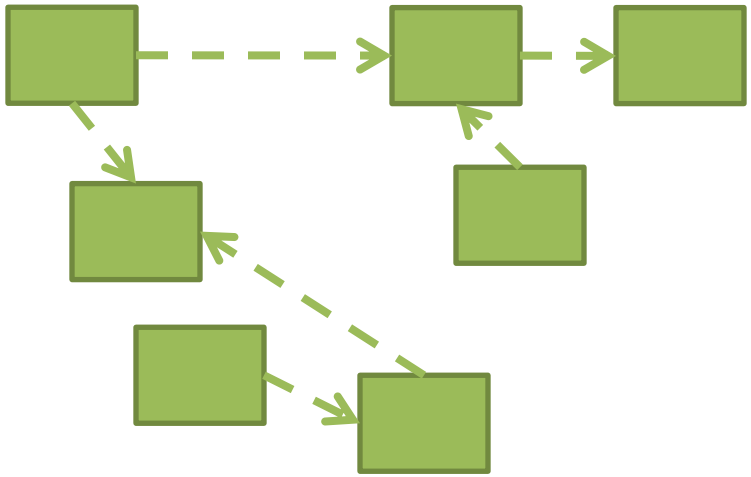




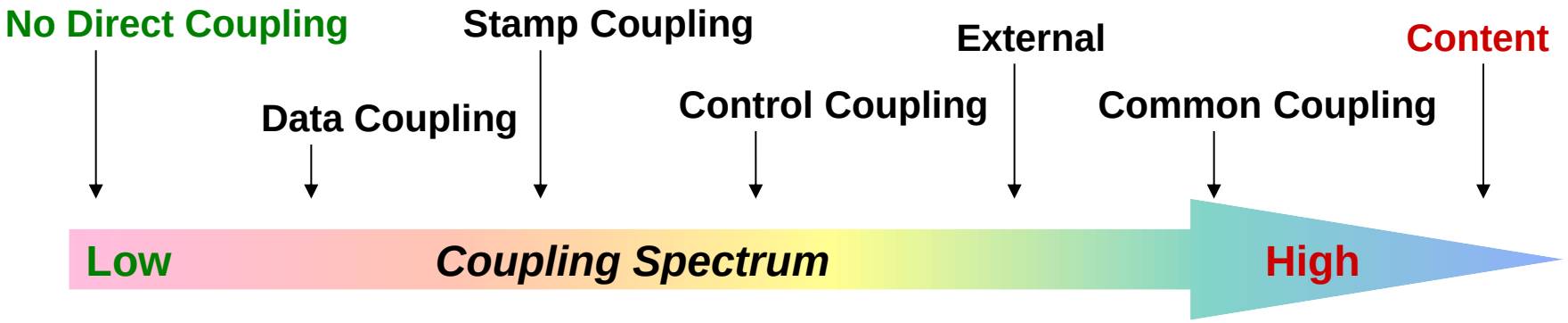
Coupling

- Lot's of dependencies ➔ high coupling
- Few dependencies ➔ low coupling





Types of Coupling



Measure of the interdependence among software components

Content: one component directly references the content of another
Common: both components have access to the same global data
Control: One component passes the element of control to another
Stamp: Two components modify or access data in the same object
Data: One component passes simple data to another as an argument

If we do our design job carefully

- We will break our larger problem into several classes
- Each of these classes will do one kind of thing (i.e. they will have *high cohesion*)
- Our classes will only need to depend on each other in specific, highly limited ways (i.e. they will have *low coupling*). Many classes won't even be aware of most of the other classes in the system.

Imagine that you're writing code to manage a school's students

Things your design should accommodate:

- Handle adding or removing students from the school
- Setting the name, phone number, and GPA for a particular student
- Compute the average GPA of all the students in the school
- Sort the students by last name to print out a report of students and GPA

Discuss and come up with a design with those nearby you. How many classes does you system need?

Note that

- Cohesion will tend to want us to make many smaller classes, each of which will do only one thing
- But if the classes are too small, they'll tend to need to depend on each other to do work, and the coupling will get bad

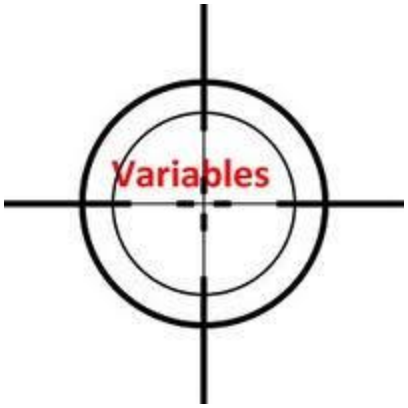
Hints #1 for Designing Objects

- Look for the nouns in your problem, consider making them objects
- Keep any one objects from getting too “fat” – containing too many methods or fields
- Avoid Plural Nouns
- Avoid Parallel Structures

The plan

- Learn 3 essential object oriented design terms:
  - Encapsulation
  - Coupling
  - Cohesion
- Scope (if we have time)

Variable Scope



Scope is the region of a program in which a variable can be accessed

- *Parameter scope:* the whole method body
- *Local variable scope:* from declaration to block end

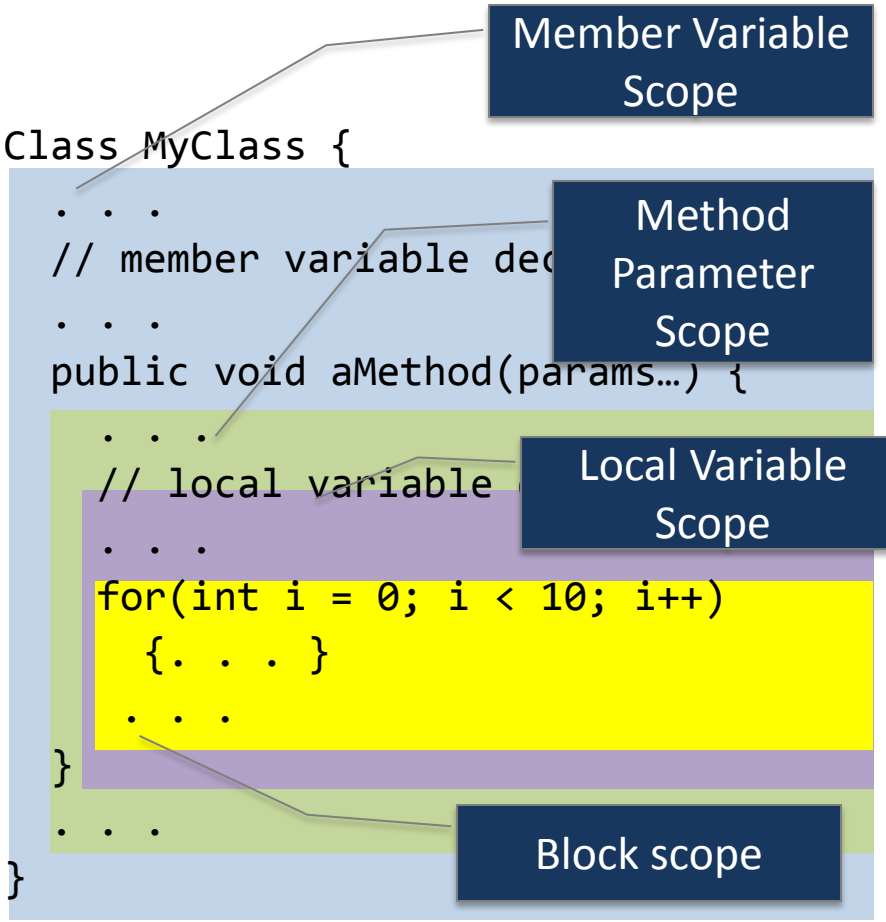
```
public double myMethod() {
    double sum = 0.0;
    Point2D prev = this.pts.get(this.pts.size() - 1);
    for (Point2D p : this.pts) {
      sum += prev.getX() * p.getY();
      sum -= prev.getY() * p.getX();
      prev = p;
    }
    return Math.abs(sum / 2.0);
}
```

Why do you suppose scoping exists? What happens if two variables have the same name in the same code location?

- Please take 15 seconds and think about it
- Turn to neighbor and discuss it for a minute
- Then let's talk?

Member Scope (Field or Method)

- ***Member scope:*** anywhere in the class, including *before* its declaration
  - Lets methods call other methods later in the class
- **`public static`** class members can be accessed from outside with "class qualified names"
  - **`Math.sqrt()`**
  - **`System.in`**



Overlapping Scope and Shadowing

```
public class TempReading {
        private double temp;

        public void setTemp(double temp) {
                this.temp = temp;

        }
        // …
}
```

What does this "temp" refer to?

Always qualify field references with `this`. It prevents accidental shadowing.

What you have learned

- Learn 3 essential object oriented design terms:
  - Encapsulation
  - Coupling
  - Cohesion
- Scope (if we have time)