CSSE 220 Day 13

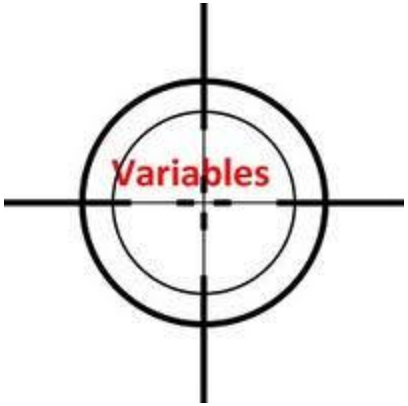
Encapsulation
Coupling and Cohesion
Scoping

Please download EncapsulationExamples from your SVN

The plan

- Test Grading
- Scope
- Encapsulation
- Coupling
- Cohesion

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);
}
```

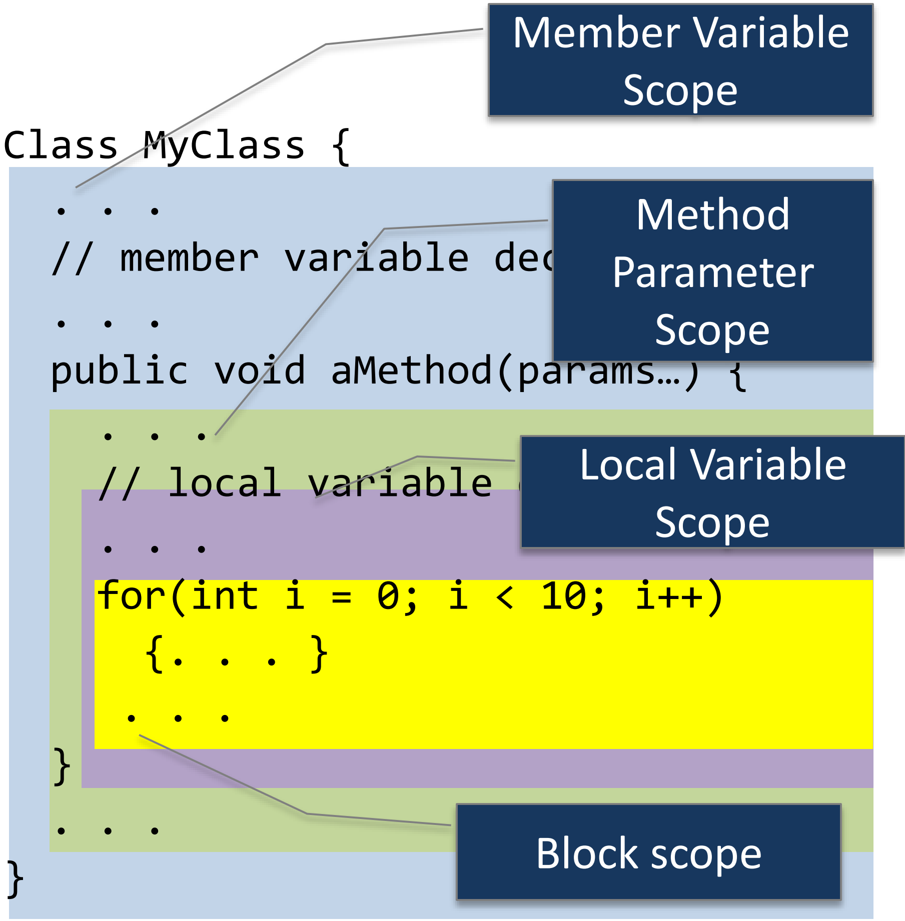
Why do you suppose scoping exists? What happens if two variables have the same name in the same code location?

- 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`



```
Class MyClass {
  . . .
  // member variable dec
  . . .
  public void aMethod(params…) {
    . . .
    // local variable
    . . .
    for(int i = 0; i < 10; i++)
      {. . . }
     . . .
  }
  . . .
}
```

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.

Thinking About Design

- Say 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?

Encapsulation

- Makes your program easier to understand by
  - Grouping related stuff together
- Mike’s 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”

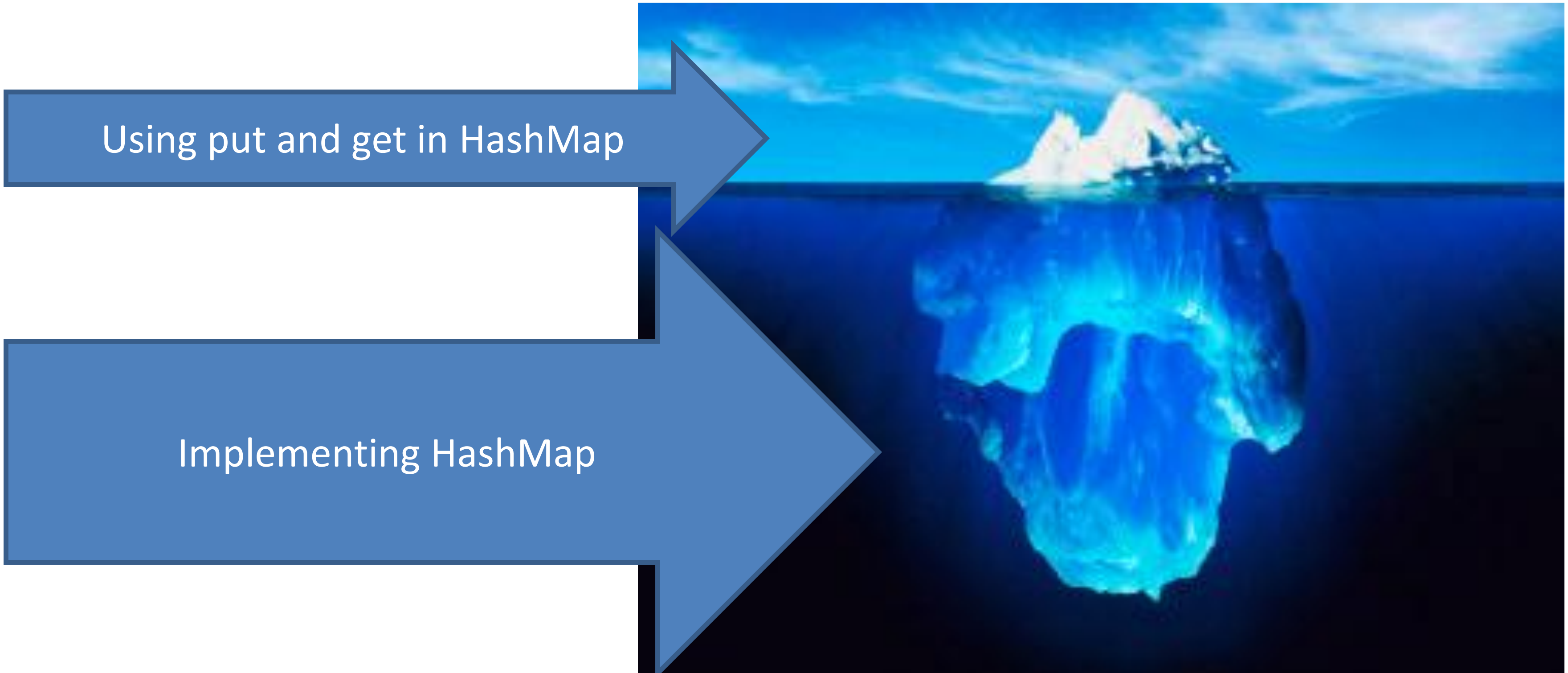
Encapsulation

- Car
  - Could be its own object
  - Does it feel right to have turn signals and wind shield washer fluid managed from the same object?
- Possible Objects Inside a Car:
  - Transmission
  - Brakes
  - Tire
  - Seatbelt

Your seatbelt shouldn't have access to your brakes, right?

Encapsulation

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



Using put and get in HashMap

Implementing HashMap

Encapsulation

Makes your program easier to change by…

- Allowing you to change how your data is represented

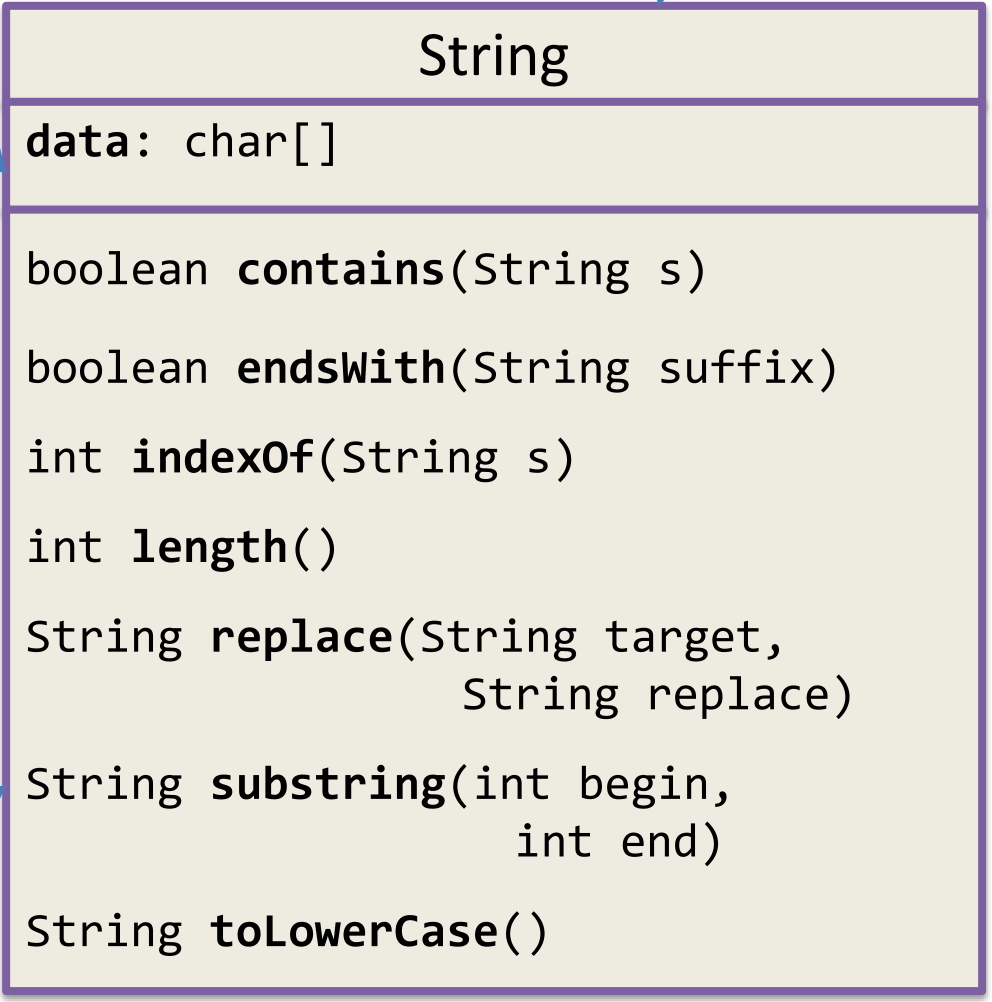
City Temperature Activity

- 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



Class name

Fields

Methods

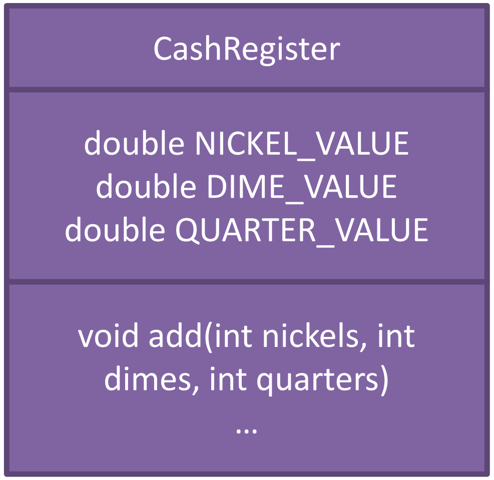
```
String
data: char[]

boolean contains(String s)

boolean endsWith(String suffix)

int indexOf(String s)

int length()

String replace(String target,
                String replace)

String substring(int begin,
                  int end)

String toLowerCase()
```

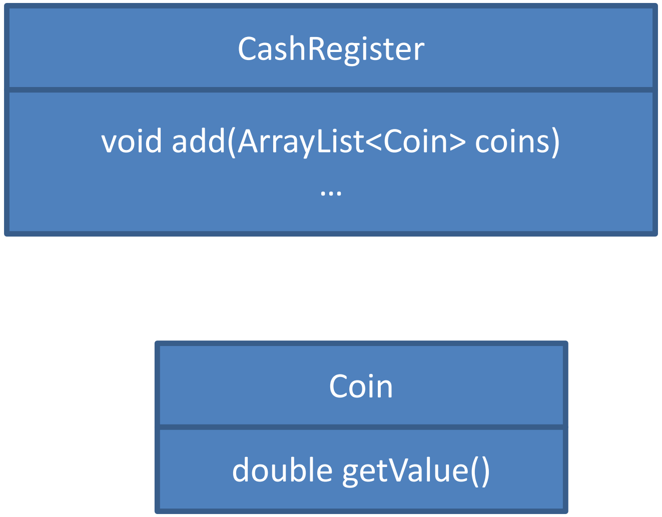
TwoVsTwo

- 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!

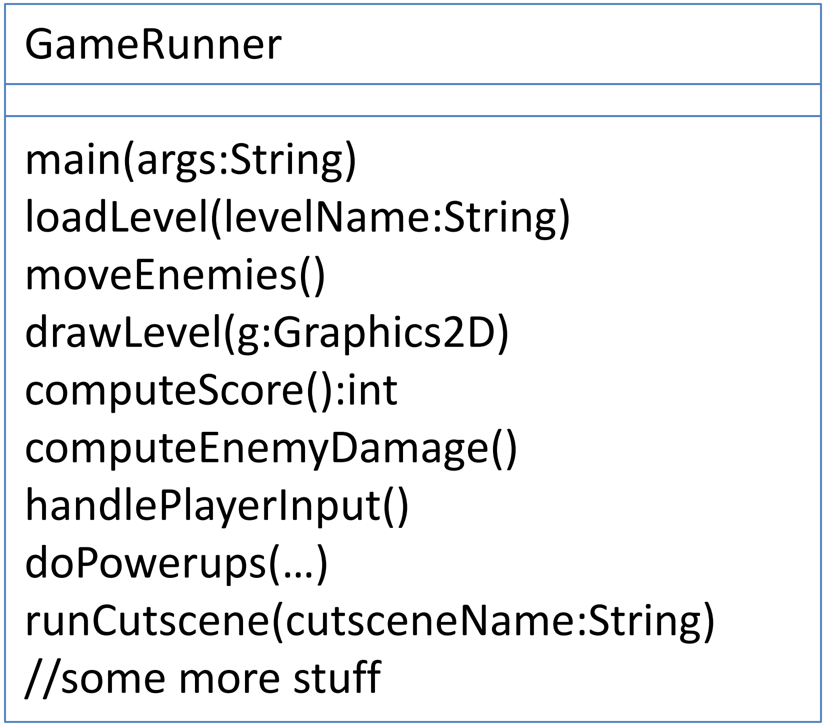
Cohesion

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



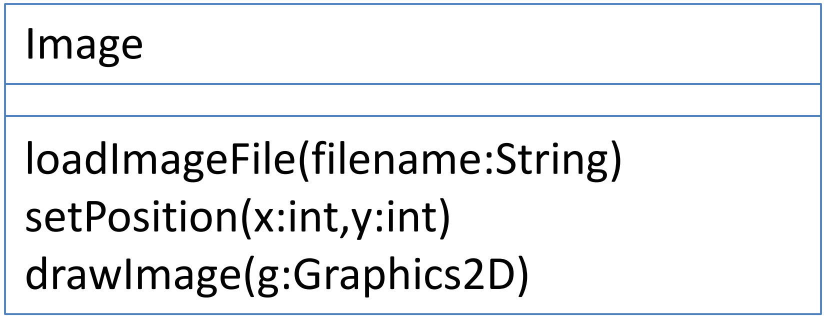


Imagine I want to make a Video Game. Here are two classes in my design. Which is more cohesive?



GameRunner

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

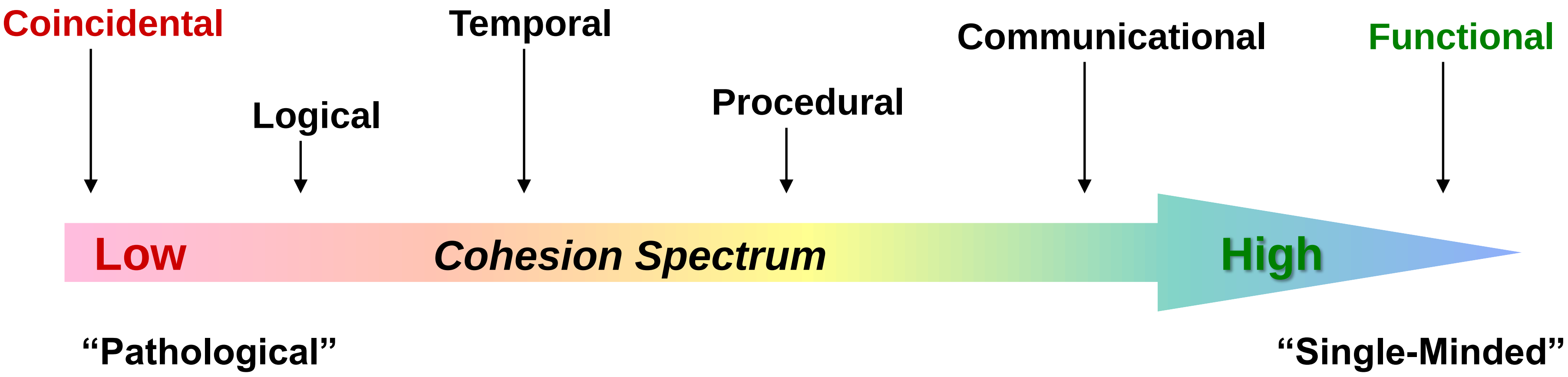


Image

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

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

Types of Cohesion



Coincidental

Logical

Temporal

Procedural

Communicational

Functional

Low

Cohesion Spectrum

High

"Pathological"

"Single-Minded"

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

Coupling and Cohesion

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

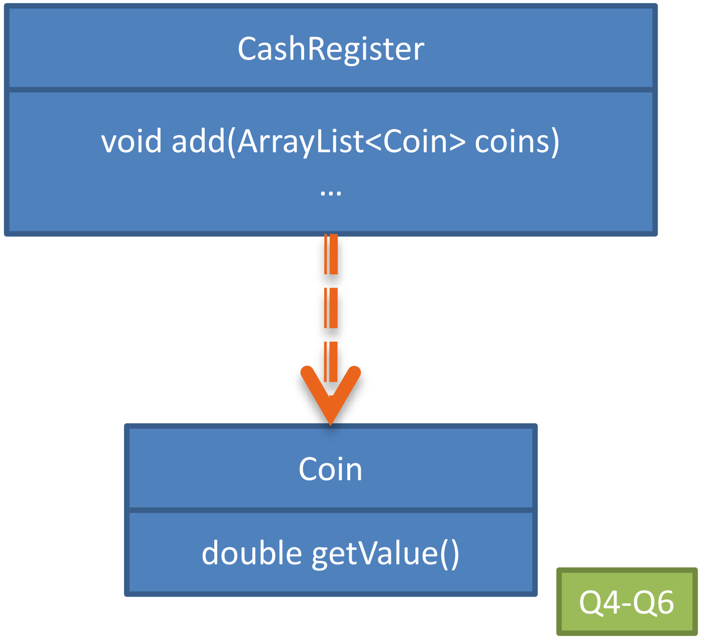
At a very high level:

- 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

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



CashRegister

void add(ArrayList<Coin> coins)
…

Coin

double getValue()

Q4-Q6

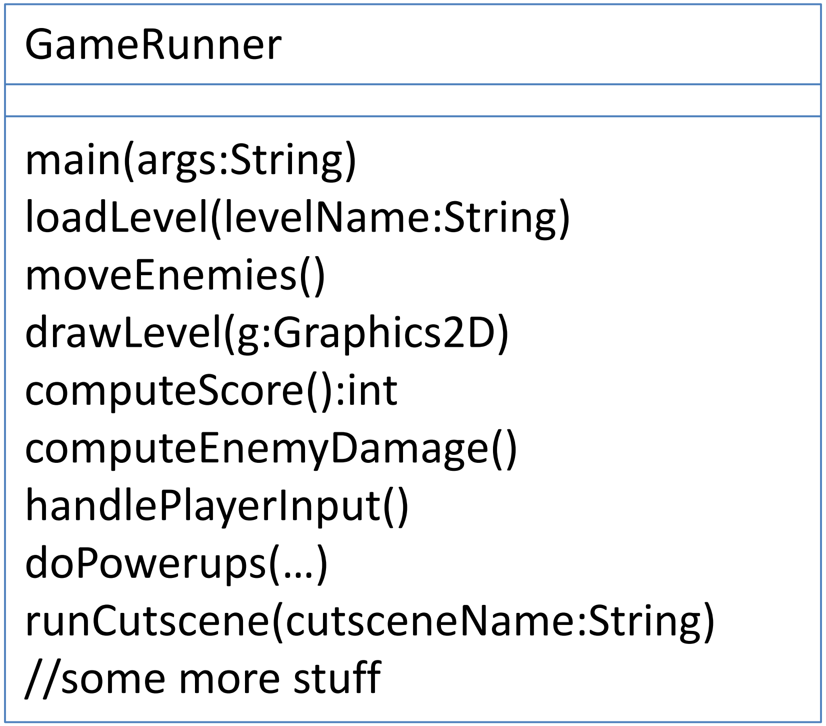
Coupling

- Coupling is one object depending 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);
```

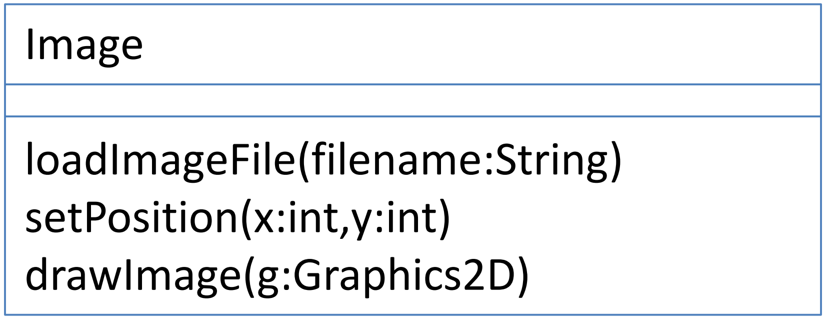
Low Coupling

Game Runner knows about Image, but Image doesn't know about GameRunner.



GameRunner

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

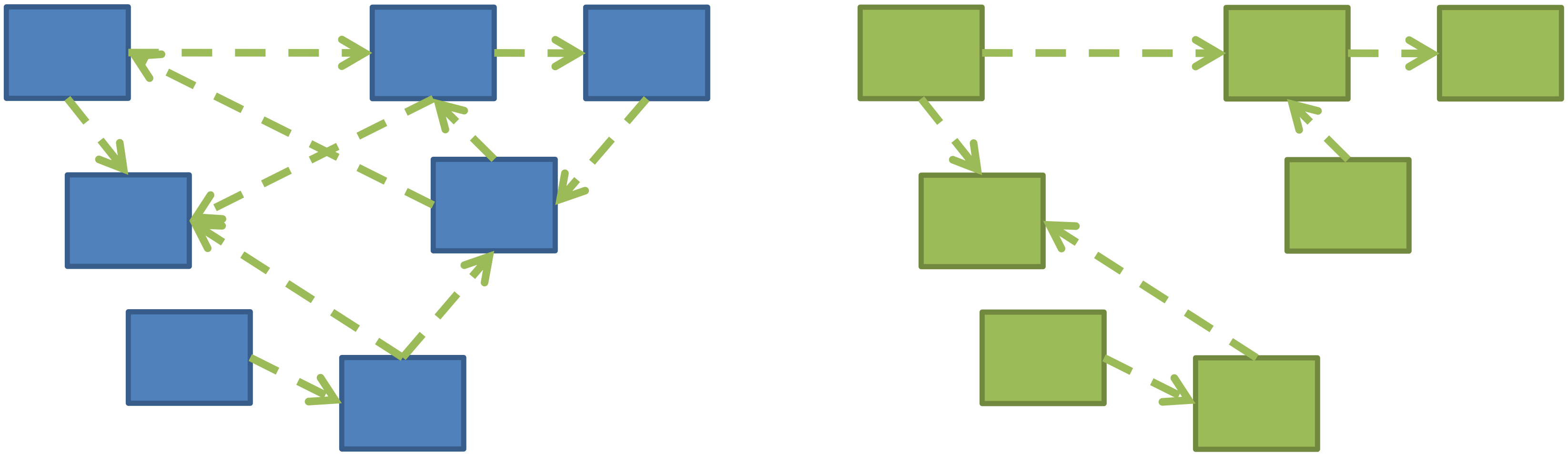


Image

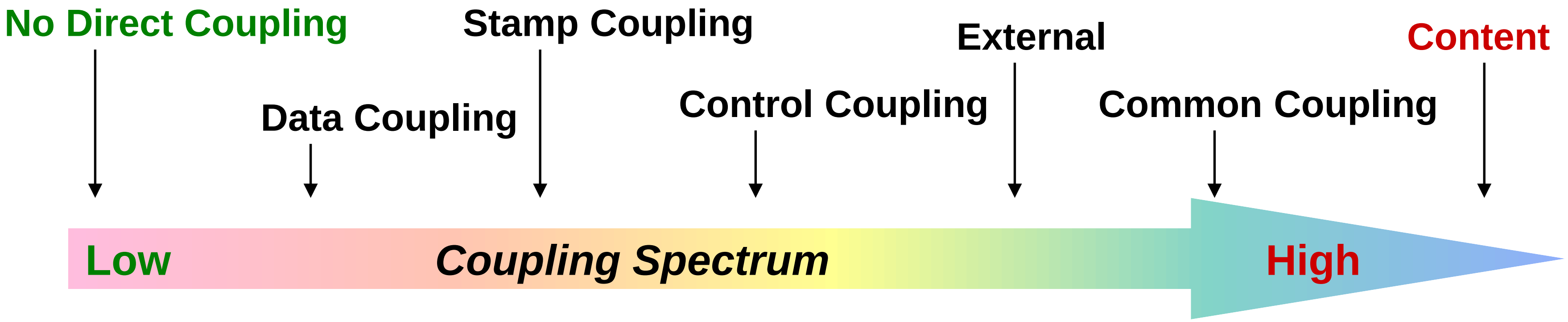
loadImageFile(filename:String)
setPosition(x:int,y:int)
drawImage(g:Graphics2D)

Coupling

- Lots 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

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

Note that

- Cohesion:
  - Can lead to many smaller classes, each of which will do only one thing
- When classes are too small, they'll tend to depend on each other to do work, and the coupling will get high

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 your system need?

Hint #1 for Designing Objects

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

Questions?

- Scope
- Encapsulation
- Coupling
- Cohesion