CSSE 220 Day 11

Software Engineering Techniques
Encapsulation
Coupling and Cohesion
Scoping

Please check out EncapsulationExamples from your SVN

The plan

- Software Engineering Techniques:
  - Pair programming
  - Version Control
- Learn 3 essential object oriented design terms:
  - Encapsulation (today's topic)
  - Coupling
  - Cohesion

What Is Pair Programming?

- Two programmers work side-by-side at a computer, continuously collaborating on the same design, algorithm, code, and/or test
- Enable the pair to produce higher quality code than that produced by the sum of their individual efforts
- Let's watch a video…

Pair Programming

- Working in pairs on a single computer
  - The *driver*, uses the keyboard, talks/thinks out-loud
  - The *navigator*, watches, thinks, comments, and takes notes
  - Person who really understands should start by navigating ☺
- For hard (or new) problems, this technique
  - Reduces number of errors
  - Saves time in the long run

How Does This Work? (1 of 2)

- Pair-Pressure
  - Keep each other on task and focused
  - Don't want to let partner down
- Pair-Think
  - Distributed cognition:
    - Shared goals and plans
    - Bring different prior experiences to the task
    - Must negotiate a common shared course of action
- Pair-Relaying
  - Each, in turn, contributes to the best of their knowledge and ability
  - Then, sit back and think while their partner fights on

Abstracted from: Robert Kessler and Laurie Williams

How Does This Work? (2 of 2)

- Pair-Reviews
  - Continuous design and code reviews
  - Improved defect removal efficiency (more eyes to identify errors)
  - Removes programmers distaste for reviews (more fun)
- Debug by describing
  - Tell it to the "Rosie in the Room"
- Pair-Learning
  - Continuous reviews → learn from partners
  - Apprenticeship
  - Defect prevention always more efficient than defect removal

PAIR PROGRAMMING
100 EYES
010 BRAINS
001 MIND

Abstracted from: Robert Kessler and Laurie Williams

Partnering the Pair

Expert paired with an Expert

Expert paired with a Novice

Novices paired together

Professional Driver Problem

Culture

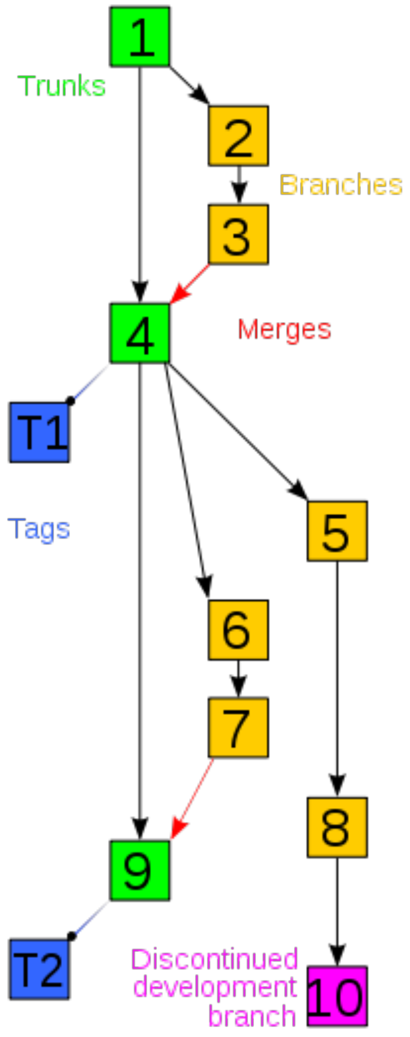
Source: Robert Kessler and Laurie Williams

What can go wrong when you are working with your team on the same system artifacts?

- Take 15 seconds and think about it
- Turn to neighbor and discuss what you think for a minute and list a few examples
- Let’s talk?

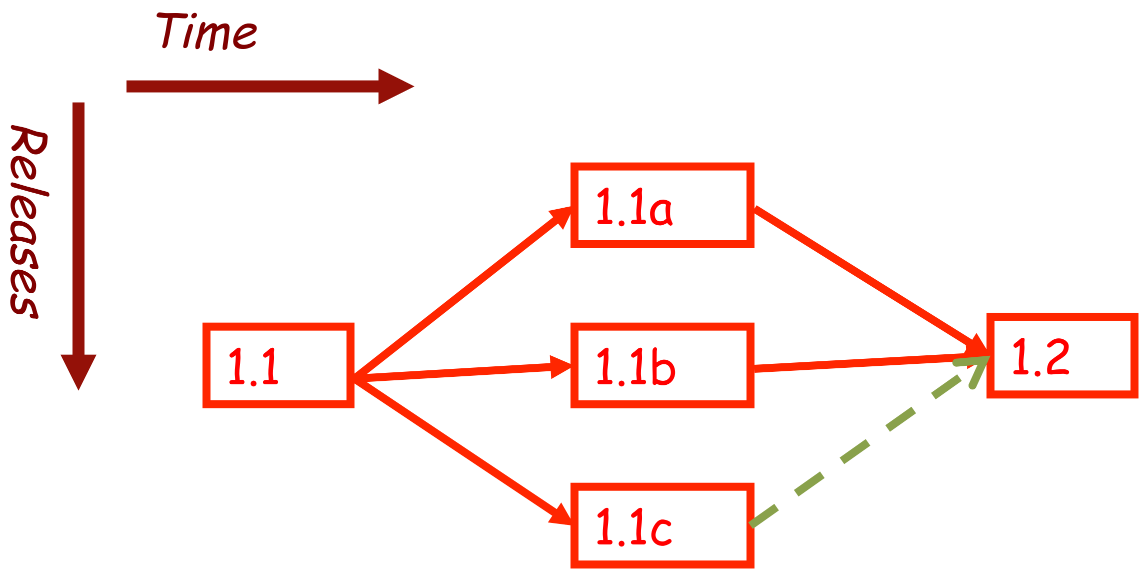
Software Has Multiple Versions

- Why? Again, software is suppose to change …
- Different releases of a product
- Variations for different platforms
  - Hardware and software
- Versions within a development cycle
  - Test release with debugging code
  - Alpha, beta of final release
- Each time you edit a program



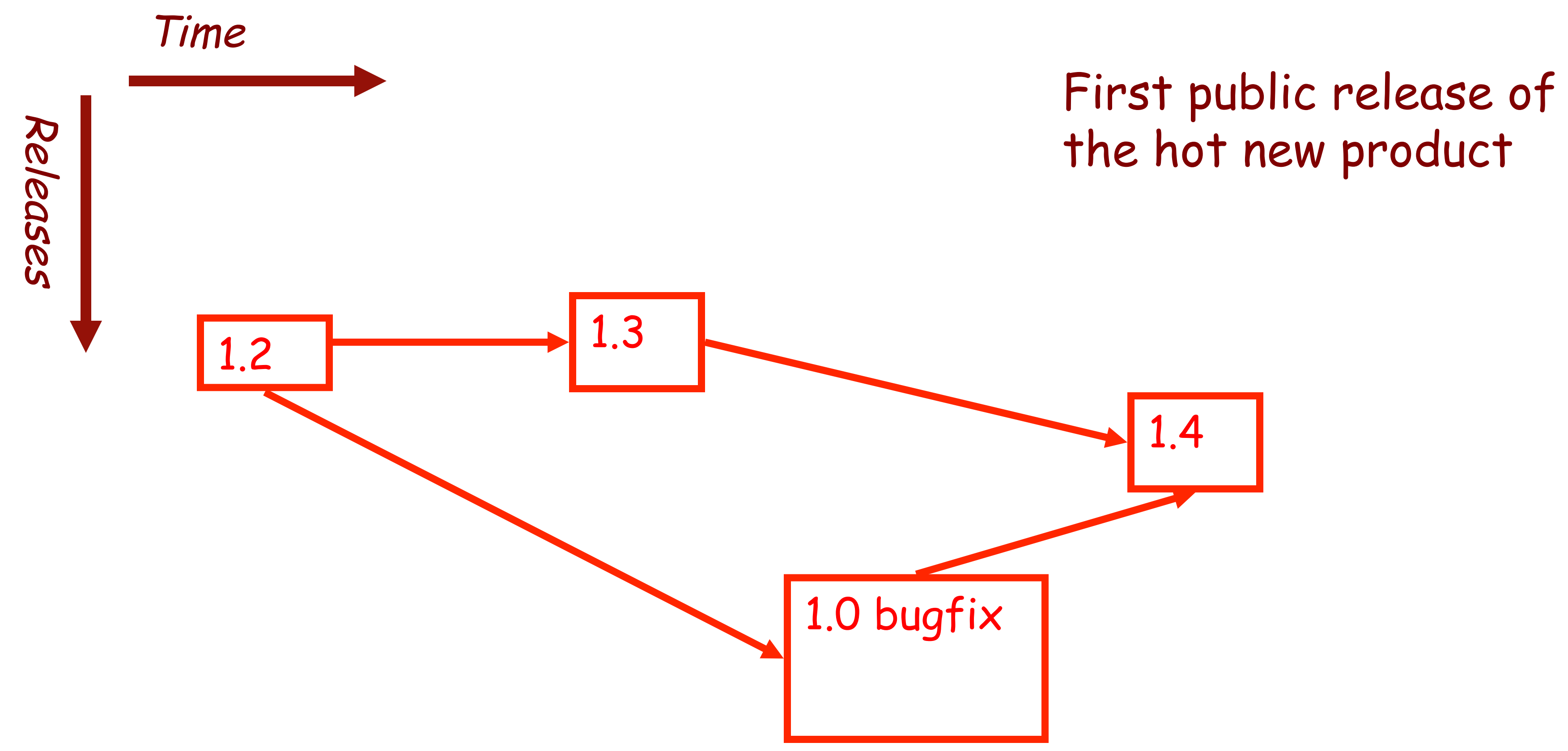
Q8

Scenario I: Normal Development



You are in the middle of a project with three developers named a, b, and c.

Version Control Scenario II: Bug Fix



Team Version Control

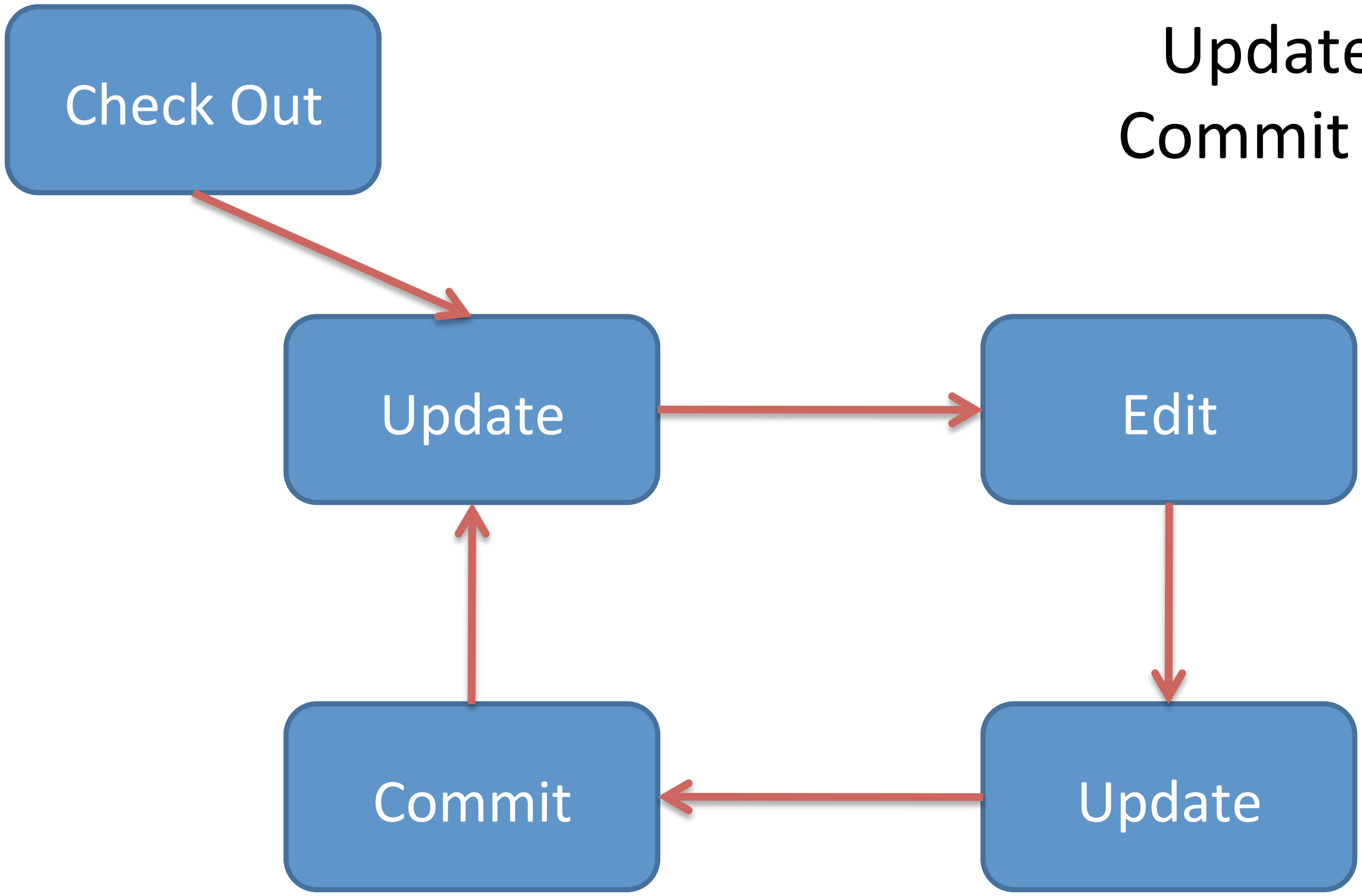
- **Version control tracks multiple versions**
  - Enables old versions to be recovered
  - Allows multiple versions to exist simultaneously

- **Always**:
  - **Update before** working
  - **Update again** before committing
  - **Commit often** and with good messages

- **Communicate** with teammates so you don't edit the same code simultaneously
  - Pair programming ameliorates this issue ☺

Q9

Team Version Control



Update and Commit often!

Why do you keep versions of the test suite under configuration management?

- Take 15 seconds and think about it
- Turn to neighbor and discuss what you think for a minute
- Let’s talk?

Regression Testing

- Keep and run old test cases
- Create test cases for new bugs
  - Like antibodies, to keep a bug from coming back
- Remember:
  - You can right-click the project in Eclipse to run all the unit tests

What if there were 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 someone has written a program that 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

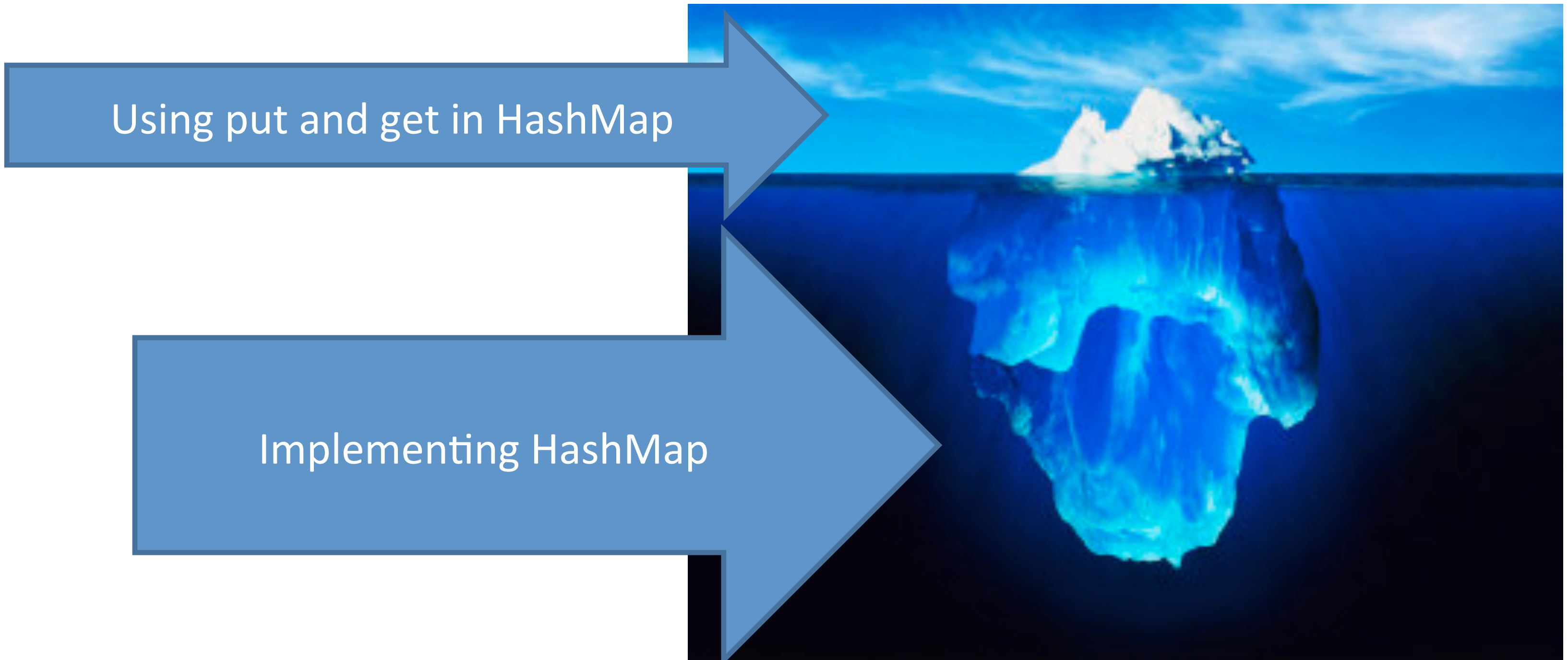
- Mike's definition "grouping some data and the operations that use that data into one thing (an object) and preventing that data from being changed except by using those operations"

Encapsulation

- 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

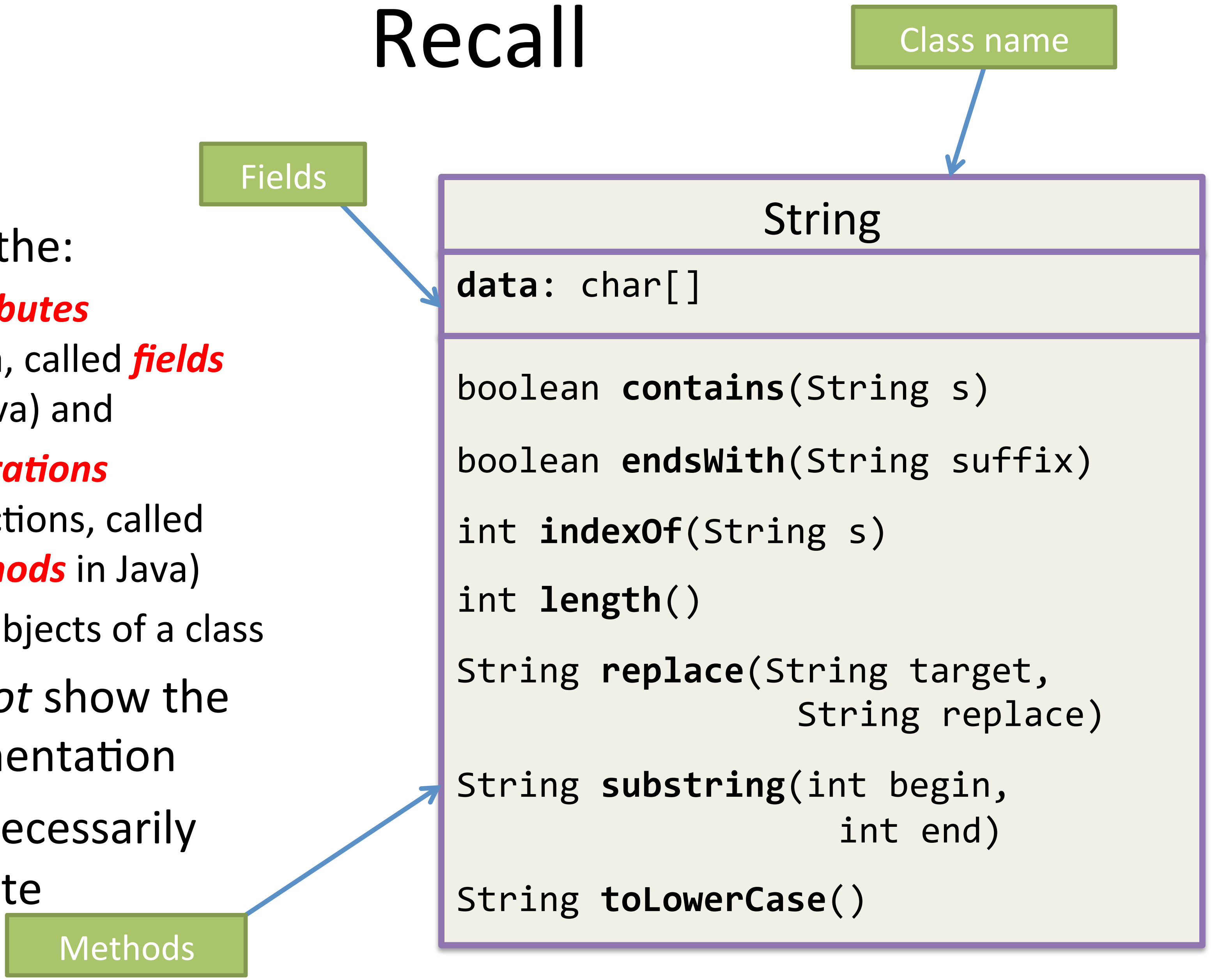
Encapsulation – a good thing?

- Note that we have the ability to change the representation of the CityTemperature class
  - but how important is that?
- Consider adding a bunch more statistics for each city (max, min, mode)
- Consider adding statistics overall (e.g. overall average)

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



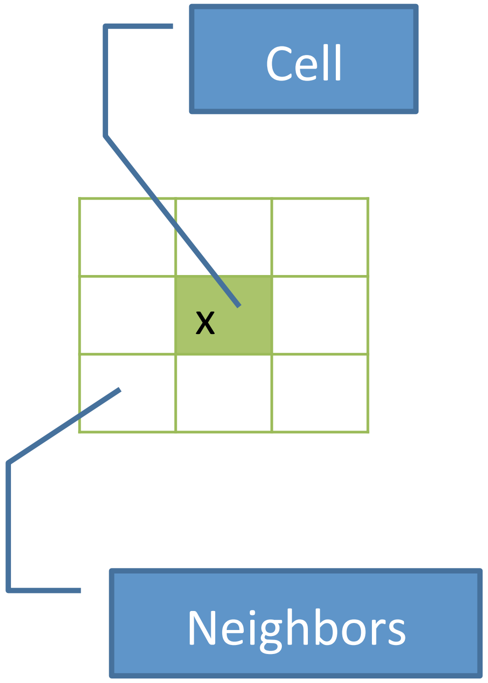
```
String
---------------------------------------------
data: char[]
---------------------------------------------
boolean contains(String s)

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!
- Turn in for extra credit! (10 points; due by Monday Sep. 28… No extensions.)
  - Answer question on Moodle labeled "TwoVsTwo Completed???"

Game of Life

1. A new cell is born on an empty square if it has exactly 3 neighbor cells
2. A cell dies of overcrowding if it is surrounded by 4 or more neighbor cells
3. A cells dies of loneliness if it has just 0 or 1 neighbor cells



Developed by John Conway, 1970

Checkout Game of Life Project

- Go to SVN repository view at bottom of workbench
  - Window➔ show view➔ Other➔ SVN➔ SVN Repositories
- Right click in SVN View, then choose New SVN Repository Location
  - http://svn.csse.rose-hulman.edu/repos/csse220-201610-"your_team_repository"
  - Your team repository will be csse220-201610-gameoflifeXX where XX is the team number
  - On Moodle, click on "Game of Life Team Assignments" to see to what team you have been assigned

Work Time

- Work with your partner on the GameOfLife project
  - Get help as needed
  - The TODOs are numbered – do them in the indicated order.
  - ***Follow the practices of pair programming!***

- ***Don’t do any of the work without your partner!***