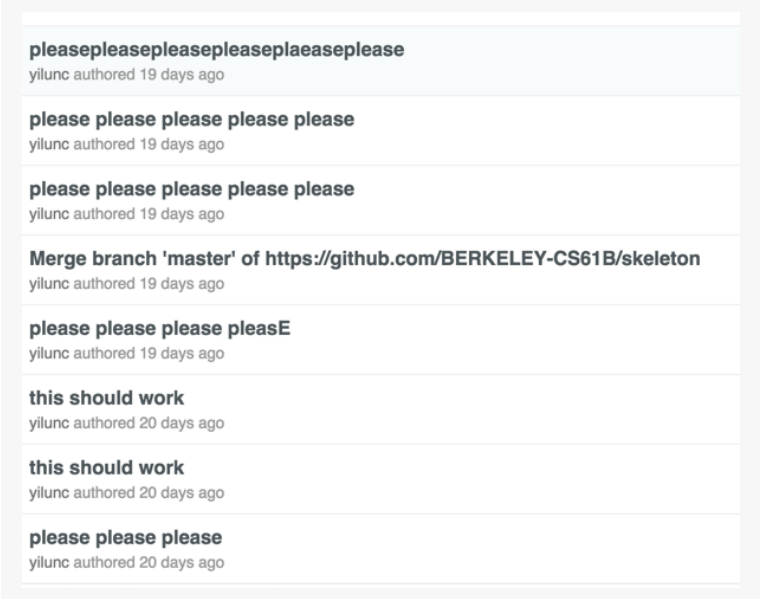
Announcements and Miscellanea

Oases: > 100 people, making the world cooler. Make friends. Improve universe. Info session deets at googling/facebooking “oases at uc berkeley”.

- HW1 due tonight at 10 PM.
- No late homeworks, but:
  - Due to the inevitable technical and administrative glitches, we will have an amnesty period for resubmission later if something goes wrong.
  - Bottom line: If something goes wrong tonight, we’ll be super friendly with this one.
- Post HW1 survey tonight (very short, optional).

Needed today (if you’re willing):

- 5 volunteers (got em)



CS61B, Spring 2015

Lecture 4: Testing and Test-Driven Development

- A Simple JUnit test
- Testing Philosophy
- Selection Sort
- Simpler JUnit Tests

Additional linked data structure material deferred to next week.

How does a Programmer know that their know your code Works?

How did you know that your code was working in 61A?

What evidence do you have on your HW1 for 61B?

The 61B Way

In 61B, we will learn to write our own tests and debug our own code.

- Major focus of our course!

Goal for today, create a class with methods:

See lec4/exercises

- `static void sort(String[] a)`: sorts an array of strings.
- `static void print(String[] a)`: prints an array of strings.
- `static void main(String[] args)`: prints the args in sorted order.

```
$ java Sort he is the agoyatis of mr. conchis
agoyatis conchis he is mr. of the
```

Our approach today:

- Write tests first!

Testing Philosophy

Developing tests

One approach: Write tests first.

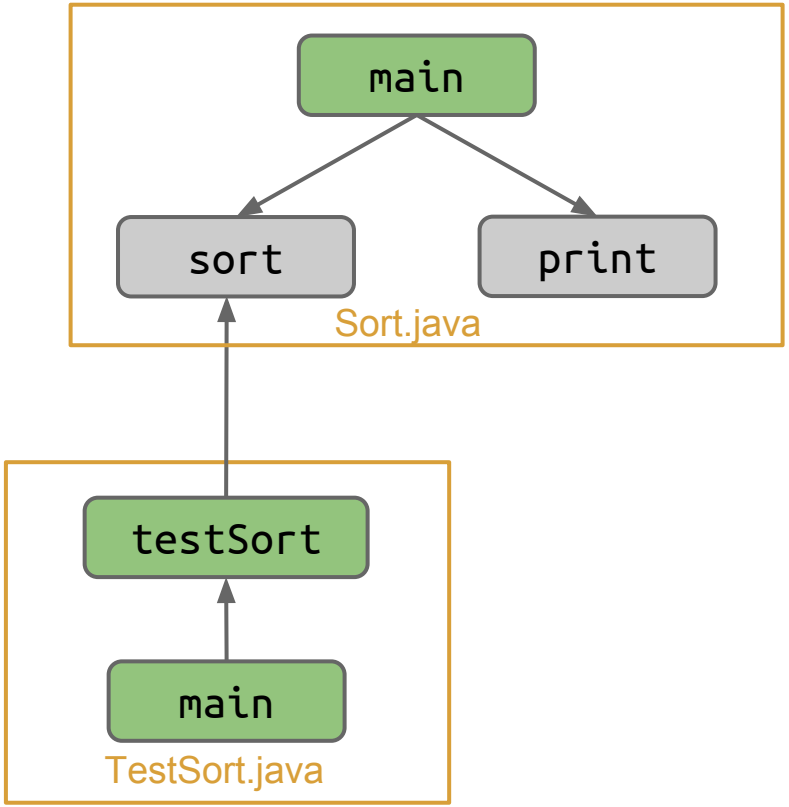
- JUnit makes this easy!

Why?

- Build confidence in basic modules.
- Decrease debugging time.
- Clarify the task.

Why not?

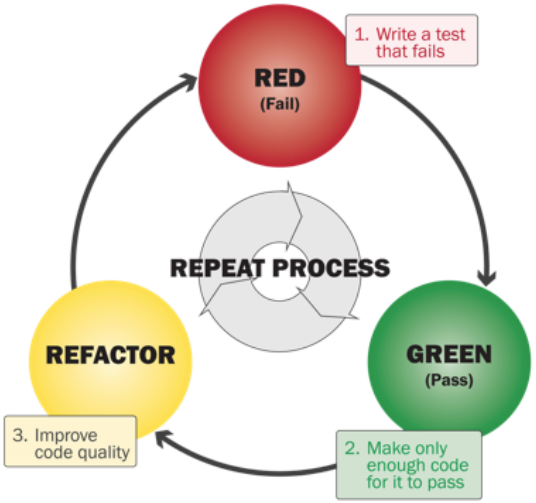
- Building tests takes time.



Test-Driven Development (TDD)

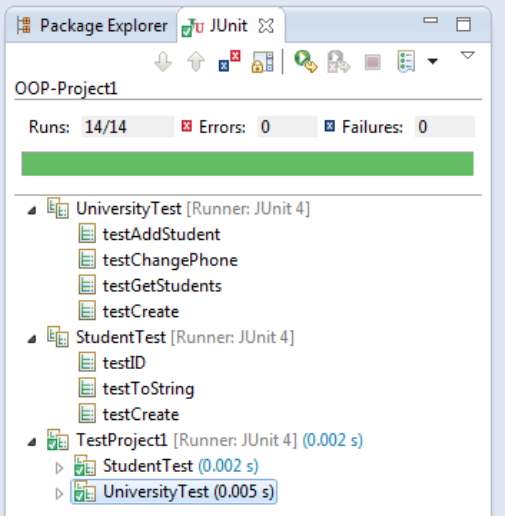
Steps to developing according to TDD:

- Identify a new feature.
- Write a test for that feature.
- Run the test. It should fail. (RED)
- Write code that passes test. (GREEN)
  - Implementation is certifiably good!
- Optional: Refactor code to make it faster, cleaner, etc.



Not required in 61B. You might hate this!

- But testing is a good idea.



The Naive Workflow (that I've observed)

```
$ python sort.py
[63, 12, 91, 5, 0]
got to this spot, lt is: 1
got to this spot, lt is: 2
got here!
[63, 12, 0, 5, 91]
got to this spot, lt is: 3
got to this spot, lt is: 4
got here!
[5, 12, 0, 63, 91]
```

Novice programmers sometimes do this:

- Read and understand the spec.
- Write entire program.
- Compile. Fix all compilation errors.
- Eliminate all bugs by repeatedly:
  - Running top level method (e.g. main)
  - Adding print statements (sometimes even binary searching for the problem).

This workflow is slow and unsafe!

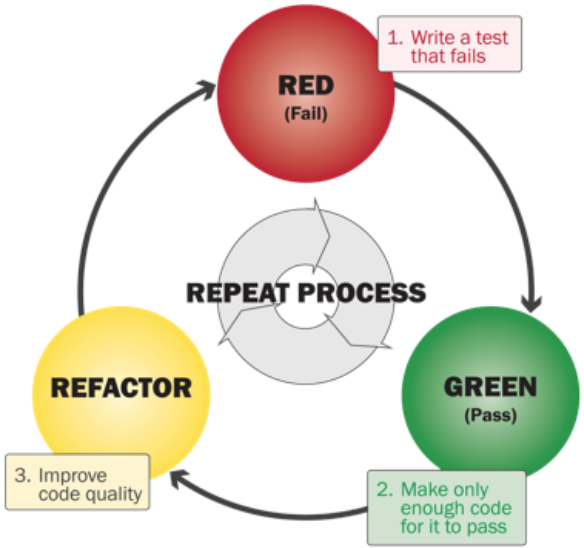
Note: Print statements are not inherently evil. While, they are a weak tool, but they are very easy to use.

A Tale of Two Workflows

TDD is an extreme departure from the naive workflow.

- What's best for you is probably in the middle.

```
$ python sort.py
[63, 12, 91, 5, 0]
got to this spot, lt is: 1
got to this spot, lt is: 2
got here!
[63, 12, 0, 5, 91]
got to this spot, lt is: 3
got to this spot, lt is: 4
got here!
[5, 12, 0, 63, 91]
```



Selection Sort

Back to Sorting: Selection Sort

Selection sorting a list of N items:

- Find the smallest ‘unfixed’ item.
- Move it to the front and ‘fix’ it.
- Selection sort the remaining N-1 unfixed items.

Let’s try it out. Webcast viewers, see: https://www.youtube.com/watch?v=6nDMgr0-Yyo

By fix, I mean “to make firm, stable, or stationary”

Challenge [SortRecursionExercise.java]

method signature

Without changing the signature of `public static void sort(String[] a)`, what would our recursive call look like?

- This is an intentionally vague question (perhaps too vague).
- There is no array slicing in Java (cannot do `a[1:]`, a la Python).

```
/** Puts a in sorted order. */
public static void sort(String[] a) {
    // find the smallest item
    // move it to the front

    // selection sort the rest
    --> ?? recursive call ?? <--
}
```

Challenge [SortSwapExercise.java]

Fill in the swap method and call below (YOUR CODE HERE).

```
/** Sorts A starting from position START. */
private static void sort(String[] a, int start) {
    if (start == a.length)
        return;
    // find the smallest item
    int mindex = indexOfSmallest(a, start);

    // move it to the front (by swapping)
    swap(/* YOUR CODE HERE */)

    // selection sort the rest
    sort(a, start + 1);
}

private static void swap(String[] a, int ix, int iy) {
    /* YOUR CODE HERE */
}
```

Simpler JUnit Tests
(using a new syntax trick and the jh61b library)

JUnit

org.junit.Assert.assertEquals(expected, actual);

- Tests that expected equals actual.
- If not, program terminates with verbose message.

Yes this is weird, as it implies you'd be instantiating TestSort.java. In fact, runClasses does this.

Simplification #1 (just trust me):

- Annotate each test with @org.junit.Test.
- Use a JUnit runner to run all tests and tabulate results.
  - The jh61b.junit.textui runner requires that test methods be non-static.
  - Tabulated output is easier to read, no need to manually invoke tests.

There is a lot of black magic happening here! Just accept it all for now.

- For the curious and/or masochistic, see: http://goo.gl/tuDGV8

JUnit as we'll use it in 61B

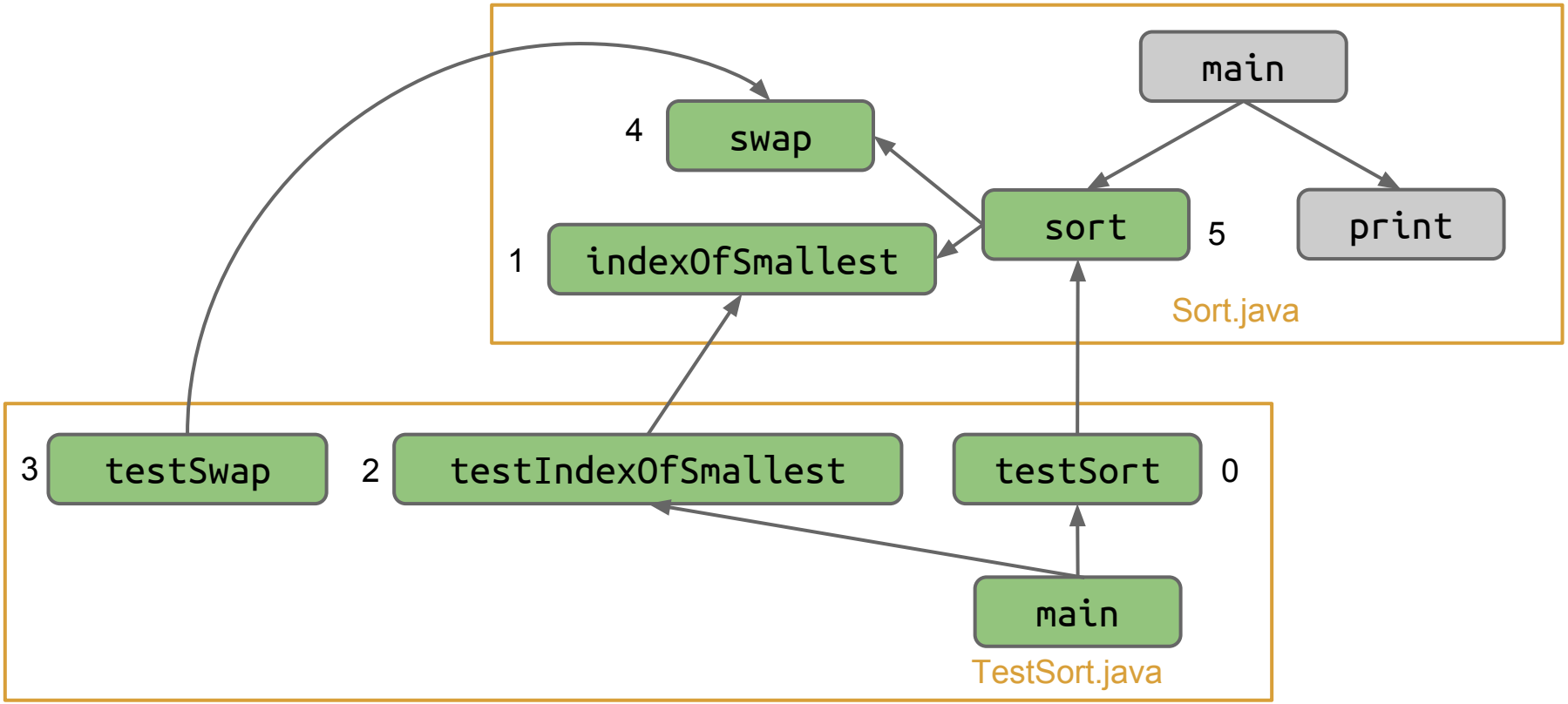
It is annoying to type out the name of the library repeatedly, e.g. **org.junit.**Test and **org.junit.Assert.**assertEquals.

Simplification #2: To avoid this we'll start every test file with:

```
import org.junit.Test;
import static org.junit.Assert.*;
```

This will magically obviate the need to type 'org.junit' or 'org.junit.Assert' (more after the midterm on what these imports really mean).

What we've done so far (print() and main() in lec4/exercises)



Numbers denote order in which classes were fully completed. Note that we attempted indexOfSmallest before writing testIndexOfSmallest, since we thought it would be easy at first.

Parting Thoughts

- JUnit makes testing easy.
- You should write tests.
  - But not too many.
  - Only when they might be useful!
  - Test first when it seems right.
  - HW2 and Lab 2 will give you some practice.
  - Some parts of HW2 are very tough to test, e.g. printing.

- Some people really like TDD. Feel free to use it in 61B.
  - See today's reading for thoughts from the creator of Ruby on Rails.

Citations

Training montage: Wet Hot American Summer

Creepy hand picture (title slide): http://www.automatedtestinginstitute.com/home/images/stories/Functional.jpg

Red-Green-Refactor image courtesy of a guy who has had issues with TDD: http://ryantablada.com/post/red-green-refactor---a-tdd-fairytale