Note: The SP24 live lecture will be different from what these slides show. Any differences will not be in scope for exams, and serve as examples on how to apply the concepts covered in this lecture.

Here are the videos corresponding to these slides.

Here is the repo corresponding to today's new lecture.

Lecture 6

Testing

CS61B, Spring 2024 @ UC Berkeley

Slides credit: Josh Hug

A New Way

Lecture 6, CS61B, Spring 2024

A New Way

Intro to Unit Testing

- Ad-Hoc Tests are Tedious
- Unit Testing Frameworks, Truth

Building Selection Sort

- The Selection Sort Algorithm
- Find Smallest
- Swap
- Correcting a Design Error
- Figuring out the Recursion
- Fixing Another Design Error

Testing Philosophy

How Does a Programmer Know That Their Code Works?

In prior programming classes, you most likely knew your code worked because it passed some autograder tests or local tests provided by an instructor.

In the real world, programmers believe their code works because of **tests they write themselves**.

- Knowing that your code is completely correct is usually impossible.
- But tests can provide strong evidence.

This will be our new way.

How Does a Programmer Know That Their Code Works?

In prior programming cl
passed some autograde

In the real world, progra
write themselves.

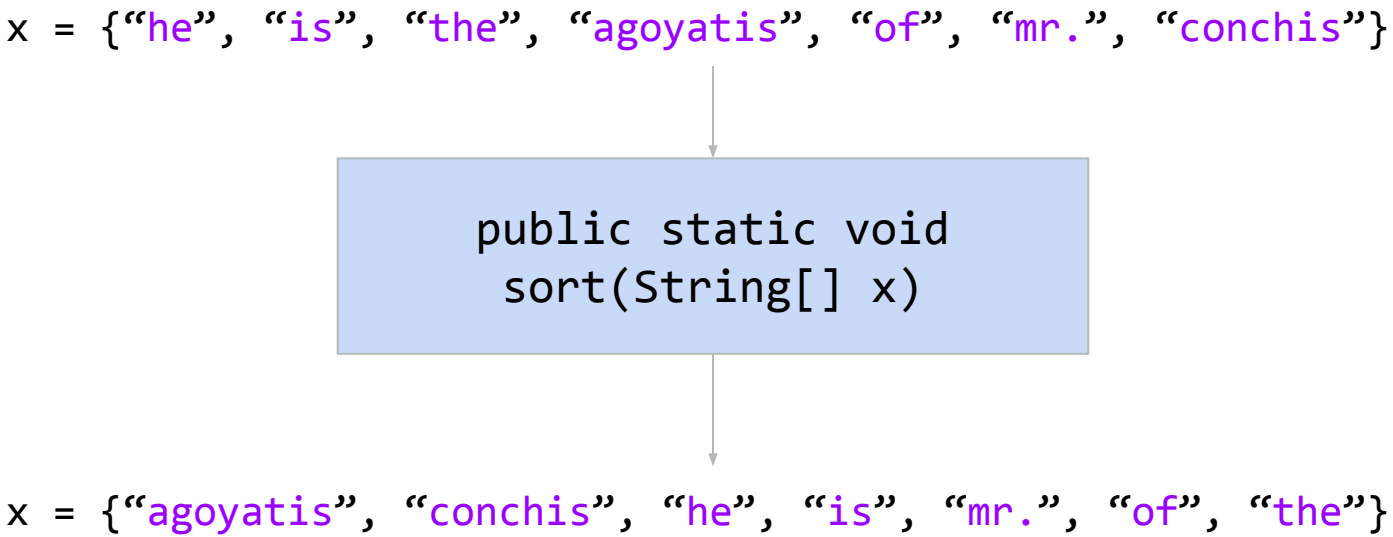
- Knowing that your c
- But tests can provi

This will be our new way

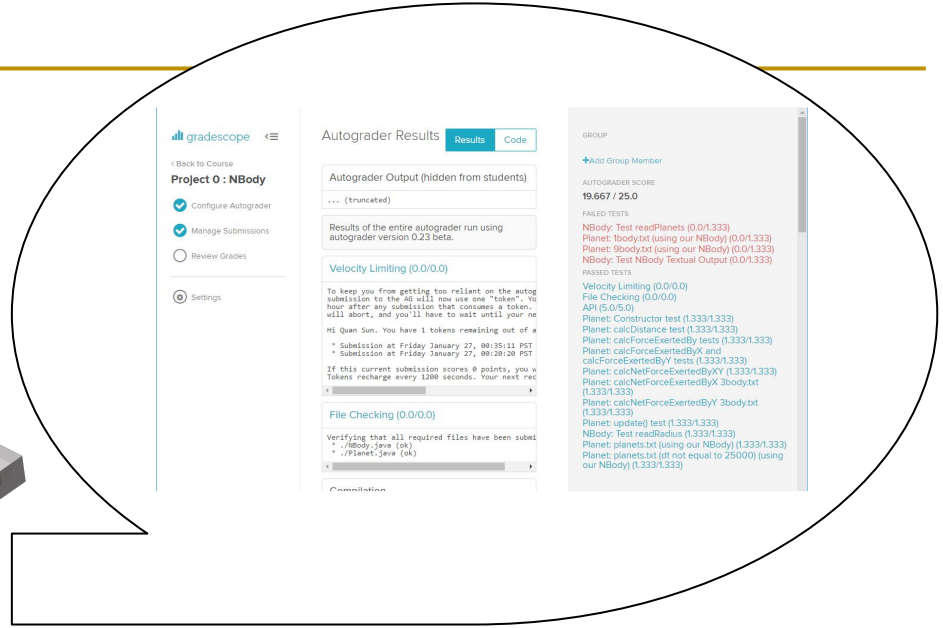
Sorting: The McGuffin for Our Testing Adventure

To try out this new way™, we need a task to complete.

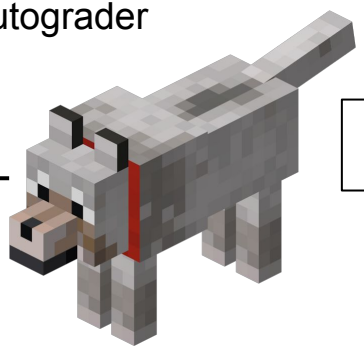
- Let's try to write a method that sorts arrays of Strings.



The Old Way



Autograder

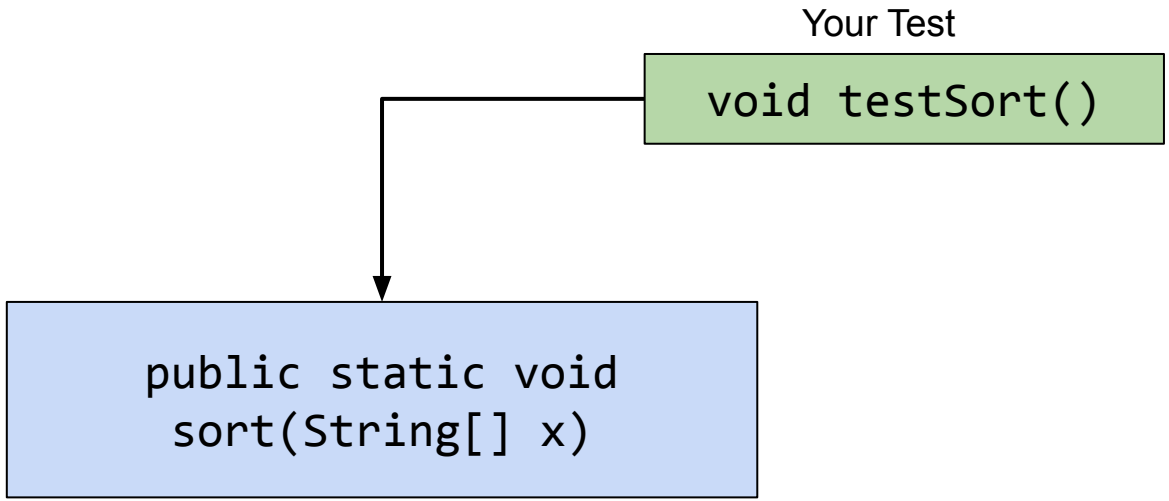


```
public static void
sort(String[] x)
```

The New Way

In this lecture we'll write `sort`, as well as our own `test` for `sort`.

- Even crazier idea: We'll start by writing `testSort` first!



Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      return;
   }

}
```

Ad-Hoc Tests are Tedious

Lecture 6, CS61B, Spring 2024

Writing a Test

If we tried to write a test, the most natural approach would be to start with an input and expected result.

```
public class TestSort {
   /** Tests the sort method of the Sort class. */
   public static void testSort() {
       String[] input =    {"CC", "BB", "DD", "AA"};
       String[] expected = {"AA", "BB", "CC", "DD"};

       ...
   }

   public static void main(String[] args) {
       testSort();
   }
}
```

Writing a Test

If we tried to write a test, the most natural approach would be to start with an input and expected result. Then call sort.

```
public class TestSort {
   /** Tests the sort method of the Sort class. */
   public static void testSort() {
       String[] input =    {"CC", "BB", "DD", "AA"};
       String[] expected = {"AA", "BB", "CC", "DD"};
       Sort.sort(input);
       ...
   }

   public static void main(String[] args) {
       testSort();
   }
}
```

Writing a Test

If we tried to write a test, the most natural approach would be to start with an input and expected result. Then call sort. Then compare the actual result with the expected result.

```
public class TestSort {
   /** Tests the sort method of the Sort class. */
   public static void testSort() {
       String[] input =    {"CC", "BB", "DD", "AA"};
       String[] expected = {"AA", "BB", "CC", "DD"};
       Sort.sort(input);
       ... // see next slide
   }

   public static void main(String[] args) {
       testSort();
   }
}
```

Comparison Code

The code that compares the `input` and `expected` might look something like below:

- Details aren't important, just that it's long and boring code to write.

```
String[] input =    {"CC", "BB", "DD", "AA"};
String[] expected = {"AA", "BB", "CC", "DD"};
Sort.sort(input);

for (int i = 0; i < input.length; i += 1) {
   if (!input[i].equals(expected[i])) {
       System.out.println("Mismatch at position " + i +
                  ", expected: '" + expected[i] +
                  "', but got '" + input[i] + "'");
       return;
   }
}
```

Such Ad-Hoc Testing is Tedious and Repetitive

```
public class TestSort {
    /** Tests the sort method of the Sort class. */
    public static void testSort() {
        String[] input = {"CC" , "BB", "DD", "AA"};
        String[] expected = {"AA" , "BB", "CC", "DD"};
        Sort.sort(input);

        for (int i = 0; i < input.length; i += 1) {
            if (!input[i].equals(expected[i])) {
                System.out.println("Mismatch at position " + i + ", expected: '" + expected[i] +
                        "', but got '" + input[i] + "'");
                return;
            }
        }
    }

    public static void main(String[] args) {
        testSort();
    }
}
```

Code similar to this appears in essentially any test. Tedious to write.

Unit Testing Frameworks, Truth

Lecture 6, CS61B, Spring 2024

Unit Tests

We just wrote a **unit test**:

- Straight from wikipedia: “In computer programming, **unit testing** is a software testing method by which individual units of source code … are tested to determine whether they are fit for use.”

Unit testing frameworks do the hard work for us.

- Example: JUnit (pre-sp23), AssertJ, and **Truth** (sp23 to present).
- Less tedious, even fun.

Let’s try writing a unit test using **Truth**.

Coding Demo

TestSort.java

```
public class TestSort {
   /** Tests the sort method of the Sort class. */
   public static void testSort() {




   }



}
```

Coding Demo

TestSort.java

```
public class TestSort {
   /** Tests the sort method of the Sort class. */
   public static void testSort() {
       String[] input = {"rawr", "a", "zaza", "newway"};




   }



}
```

Coding Demo

TestSort.java

```
public class TestSort {
   /** Tests the sort method of the Sort class. */
   public static void testSort() {
       String[] input = {"rawr", "a", "zaza", "newway"};
       String[] expected = {"a", "newway", "rawr", "zaza"};



   }



}
```

Coding Demo

TestSort.java

```
public class TestSort {
   /** Tests the sort method of the Sort class. */
   public static void testSort() {
       String[] input = {"rawr", "a", "zaza", "newway"};
       String[] expected = {"a", "newway", "rawr", "zaza"};
       Sort.sort(input);


   }




}
```

Coding Demo

TestSort.java

```
import static com.google.common.truth.Truth.assertThat;

public class TestSort {
   /** Tests the sort method of the Sort class. */
   public static void testSort() {
       String[] input = {"rawr", "a", "zaza", "newway"};
       String[] expected = {"a", "newway", "rawr", "zaza"};
       Sort.sort(input);

       assertThat(input).isEqualTo(expected);
   }




}
```

Coding Demo

TestSort.java

```
import static com.google.common.truth.Truth.assertThat;

public class TestSort {
   /** Tests the sort method of the Sort class. */
   public static void testSort() {
       String[] input = {"rawr", "a", "zaza", "newway"};
       String[] expected = {"a", "newway", "rawr", "zaza"};
       Sort.sort(input);

       assertThat(input).isEqualTo(expected);
   }

   public static void main(String[] args) {
       testSort();
   }
}
```

Truth: A Library for Making Testing Easier (example below)

```
import static com.google.common.truth.Truth.assertThat;
public class TestSort {
    /** Tests the sort method of the Sort class. */
    public static void testSort() {
        String[] input = {"cows", "dwell", "above", "clouds"};
        String[] expected = {"above", "clouds", "cows", "dwell"};
        Sort.sort(input);

        assertThat(input).isEqualTo(expected);
    }

    public static void main(String[] args) {
        testSort();
    }
}
```

Coding Demo

TestSort.java

```
import static com.google.common.truth.Truth.assertThat;


public class TestSort {
   /** Tests the sort method of the Sort class. */

   public static void testSort() {
       String[] input = {"rawr", "a", "zaza", "newway"};
       String[] expected = {"a", "newway", "rawr", "zaza"};
       Sort.sort(input);

       assertThat(input).isEqualTo(expected);
   }

   public static void main(String[] args) {
       testSort();
   }
}
```

Coding Demo

TestSort.java

```
import static com.google.common.truth.Truth.assertThat;
import org.junit.jupiter.api.Test;

public class TestSort {
   /** Tests the sort method of the Sort class. */
   @Test
   public static void testSort() {
       String[] input = {"rawr", "a", "zaza", "newway"};
       String[] expected = {"a", "newway", "rawr", "zaza"};
       Sort.sort(input);

       assertThat(input).isEqualTo(expected);
   }

   public static void main(String[] args) {
       testSort();
   }
}
```

Coding Demo

TestSort.java

```
import static com.google.common.truth.Truth.assertThat;
import org.junit.jupiter.api.Test;

public class TestSort {
   /** Tests the sort method of the Sort class. */
   @Test
   public static void testSort() {
       String[] input = {"rawr", "a", "zaza", "newway"};
       String[] expected = {"a", "newway", "rawr", "zaza"};
       Sort.sort(input);

       assertThat(input).isEqualTo(expected);
   }




}
```

Coding Demo

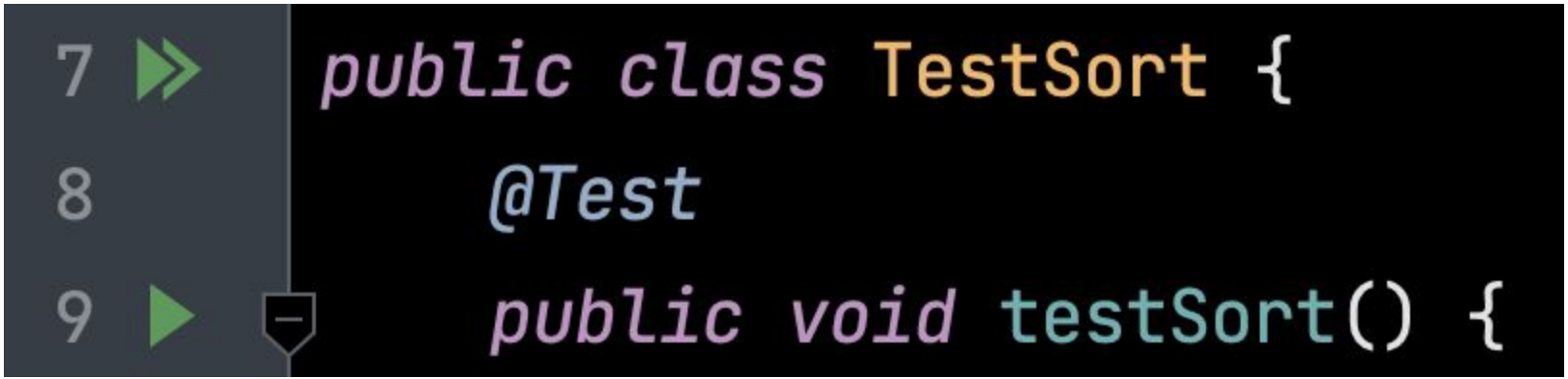
TestSort.java

```
import static com.google.common.truth.Truth.assertThat;
import org.junit.jupiter.api.Test;

public class TestSort {
   /** Tests the sort method of the Sort class. */
   @Test
   public void testSort() {
       String[] input = {"rawr", "a", "zaza", "newway"};
       String[] expected = {"a", "newway", "rawr", "zaza"};
       Sort.sort(input);

       assertThat(input).isEqualTo(expected);
   }




}
```

@Test

If we **add `@Test` before a method** AND **make the function non-static**, green arrows appear.

- The single green arrow by `testSort` means “run this function”.
- The double green arrow means run all tests in this class.



Why non-static? No idea. IMO, weird.

Gamified @Test Output

On added benefit: IntelliJ gamifies bug fixing and design.

- Concrete mini-goals.
- Progress summarized in bottom left.
- You win when you get green checks for every test.

The Selection Sort Algorithm

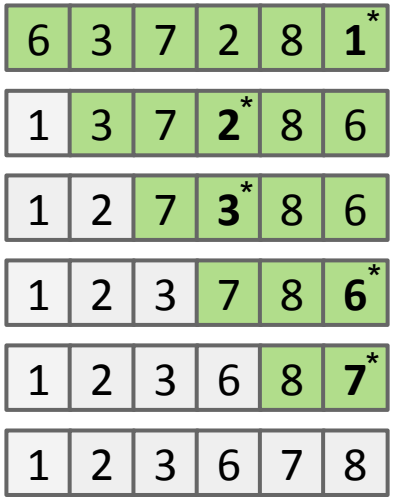
Lecture 6, CS61B, Spring 2024

Example Sorting Algorithm: Selection Sort

Selection sorting a list of N items:

- Find the smallest item.
- Move it to the front.
- Selection sort the remaining N-1 items (without touching front item!).

Move by swapping the smallest item with the front item.



As an aside: Can prove correctness of selection sort by thinking about its invariants.

Back to Sorting: Selection Sort

Selection sorting a list of N items:

- Find the smallest item.
- Move it to the front.
- Selection sort the remaining N-1 items (without touching front item!).

Move by swapping the smallest item with the front item.

Let's try implementing this.

- I'll try to simulate as closely as possible how I think students might approach this problem.
- Along the way I'll show how "test driven development" (TDD) helps avoid major problems.

Find Smallest

Lecture 6, CS61B, Spring 2024

Selection Sort: Find Smallest

Selection sorting a list of N items:

- Find the smallest item (idea: write a `findSmallest` method).
- Move it to the front.
- Selection sort the remaining N-1 items (without touching front item!).

Let's try implementing this.

- I'll try to simulate as closely as possible how I think students might approach this problem.
- Along the way I'll show how "test driven development" (TDD) helps avoid major problems.

Progress Roadmap

Created **testSort**:

```
testSort()
```

Created a **sort** skeleton:

```
sort(String[] inputs)
```

Next up:

Create **testFindSmallest**:

```
testFindSmallest()
```

Create **findSmallest**:

```
String findSmallest(String[] input)
```

Code not shown in slides. See lecture video or github.

Coding Demo

TestSort.java

```
public class TestSort {
   @Test
   public void testFindSmallest() {



   }
}
```

Coding Demo

TestSort.java

```
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};



    }
}
```

Coding Demo

TestSort.java

```
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        String expected = "a";



    }
}
```

Coding Demo

TestSort.java

```java
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        String expected = "a";
        String actual = Sort.findSmallest(input);

    }
}
```

Coding Demo

TestSort.java

```java
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        String expected = "a";
        String actual = Sort.findSmallest(input);
        assertThat(actual).isEqualTo(expected);
    }
}
```

Coding Demo

Sort.java

```
public class Sort {

   public static String findSmallest(String[] x) {









   }
}
```

Coding Demo

Sort.java

```
public class Sort {

   public static String findSmallest(String[] x) {
      return "potato";









   }
}
```

Coding Demo

Sort.java

```
public class Sort {

   public static String findSmallest(String[] x) {
      String smallest = x[0];









   }
}
```

Coding Demo

Sort.java

```
public class Sort {

   public static String findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {









      }



   }
}
```

Coding Demo

Sort.java

```
public class Sort {

   public static String findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {

         if (x[i] < smallest) {

         }


      }



   }
}
```

Coding Demo

Sort.java

```
public class Sort {

   public static String findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {

         if (x[i] < smallest) {
            smallest = x[i];
         }


      }



   }
}
```

Coding Demo

Sort.java

```
public class Sort {

   public static String findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {

         if (x[i] < smallest) {
            smallest = x[i];
         }


      }
      return smallest;


   }
}
```

Coding Demo

Sort.java

```
public class Sort {

   public static String findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(smallest);
         if (x[i] < smallest) {
            smallest = x[i];
         }


      }
      return smallest;


   }
}
```

Coding Demo

Sort.java

```
public class Sort {

   public static String findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(smallest);
         if (cmp < 0) {
            smallest = x[i];
         }


      }
      return smallest;


   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   /** @source https://stackoverflow.com/questions/5153496 */
   public static String findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(smallest);
         if (cmp < 0) {
            smallest = x[i];
         }


      }
      return smallest;


   }
}
```

Progress Roadmap

Created **testSort**:

```
testSort()
```

Created a **sort** skeleton:

```
sort(String[] inputs)
```

Created **testFindSmallest**:

```
testFindSmallest()
```

Created **findSmallest**:

```
String findSmallest(String[] input)
```

Used Google to figure out how to compare strings.

Swap

Lecture 6, CS61B, Spring 2024

A New Way

Intro to Unit Testing

- Ad-Hoc Tests are Tedious
- Unit Testing Frameworks, Truth

Building Selection Sort

- The Selection Sort Algorithm
- Find Smallest
- **Swap**
- Correcting a Design Error
- Figuring out the Recursion
- Fixing Another Design Error

Testing Philosophy

Selection Sort: Swap

Selection sorting a list of N items:

- Find the smallest item (idea: write a `findSmallest` method).
- Move it to the front (idea: write a `swap` method).
- Selection sort the remaining N-1 items (without touching front item!).

Let's try implementing this.

- I'll try to simulate as closely as possible how I think students might approach this problem.
- Along the way I'll show how "test driven development" (TDD) helps avoid major problems.

Progress Roadmap

Created **testSort**:

```
testSort()
```

Created a **sort** skeleton:

```
sort(String[] inputs)
```

Created **testFindSmallest**:

```
testFindSmallest()
```

Created **findSmallest**:

```
String findSmallest(String[] input)
```

Used Google to figure out how to compare strings.

Next up:

Create **testSwap**:

```
testSwap()
```

Create **swap**:

```
swap(String[] input, int a, int b)
```

Coding Demo

TestSort.java

```
public class TestSort {
    @Test
    public void testSort() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        String[] expected = {"a", "newway", "rawr", "zaza"};
        Sort.sort(input);

        assertThat(input).isEqualTo(expected);









    }
}
```

Coding Demo

TestSort.java

```
public class TestSort {
    @Test
    public void testSort() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        String[] expected = {"a", "newway", "rawr", "zaza"};
        Sort.swap(input, 1, 3);

        assertThat(input).isEqualTo(expected);









    }
}
```

Coding Demo

TestSort.java

```
public class TestSort {
    @Test
    public void testSort() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        String[] expected = {"rawr", "newway", "zaza", "a"};
        Sort.swap(input, 1, 3);

        assertThat(input).isEqualTo(expected);









    }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void swap(String[] x, int a, int b) {



   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void swap(String[] x, int a, int b) {
      x[a] = x[b];
      x[b] = x[a];

   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void swap(String[] x, int a, int b) {
      String temp = x[a];
      x[a] = x[b];
      x[b] = temp;
   }
}
```

Correcting a Design Error

Lecture 6, CS61B, Spring 2024

Progress Roadmap

Created **testSort**: `testSort()`

Created a **sort** skeleton: `sort(String[] inputs)`

Created **testFindSmallest**: `testFindSmallest()`

Used Google to figure out how to compare strings.

Created **findSmallest**: `String findSmallest(String[] input)`

Created **testSwap**: `testSwap()`

Used debugger to fix.

Created **swap**: `swap(String[] input, int a, int b)`

Now we have all the **helper methods** we need, as well as **tests** that give proof that they work! All that's left is to write the `sort` method itself.

- Let's start by just doing the first swap as an exploration.

Coding Demo

Sort.java

```java
public class Sort {
   public static void sort(String[] x) {


   }

   /** @source https://stackoverflow.com/questions/5153496 */
   public static String findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(smallest);
         if (cmp < 0) {
            smallest = x[i];
         }
      }
      return smallest;
   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      String smallest = findSmallest(x);

   }

   /** @source https://stackoverflow.com/questions/5153496 */
   public static String findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(smallest);
         if (cmp < 0) {
            smallest = x[i];
         }
      }
      return smallest;
   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      String smallest = findSmallest(x);
      swap(x, 0, smallest); // ???
   }

   /** @source https://stackoverflow.com/questions/5153496 */
   public static String findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(smallest);
         if (cmp < 0) {
            smallest = x[i];
         }
      }
      return smallest;
   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      String smallest = findSmallest(x);
      swap(x, 0, smallest); // ???
   }

   /** @source https://stackoverflow.com/questions/5153496 */
   public static int findSmallest(String[] x) {
      String smallest = x[0];
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(smallest);
         if (cmp < 0) {
            smallest = x[i];
         }
      }
      return smallest;
   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      String smallest = findSmallest(x);
      swap(x, 0, smallest); // ???
   }

   /** @source https://stackoverflow.com/questions/5153496 */
   public static int findSmallest(String[] x) {
      int smallest = 0;
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(smallest);
         if (cmp < 0) {
            smallest = x[i];
         }
      }
      return smallest;
   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      String smallest = findSmallest(x);
      swap(x, 0, smallest); // ???
   }

   /** @source https://stackoverflow.com/questions/5153496 */
   public static int findSmallest(String[] x) {
      int smallest = 0;
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(x[smallest]);
         if (cmp < 0) {
            smallest = x[i];
         }
      }
      return smallest;
   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      String smallest = findSmallest(x);
      swap(x, 0, smallest); // ???
   }

   /** @source https://stackoverflow.com/questions/5153496 */
   public static int findSmallest(String[] x) {
      int smallest = 0;
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(x[smallest]);
         if (cmp < 0) {
            smallest = i;
         }
      }
      return smallest;
   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      int smallest = findSmallest(x);
      swap(x, 0, smallest);
   }

   /** @source https://stackoverflow.com/questions/5153496 */
   public static int findSmallest(String[] x) {
      int smallest = 0;
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(x[smallest]);
         if (cmp < 0) {
            smallest = i;
         }
      }
      return smallest;
   }
}
```

Coding Demo

TestSort.java

```java
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        String expected = "a";
        String actual = Sort.findSmallest(input);
        assertThat(actual).isEqualTo(expected);
    }
}
```

Coding Demo

TestSort.java

```
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        int expected = 1;
        String actual = Sort.findSmallest(input);
        assertThat(actual).isEqualTo(expected);
    }
}
```

Coding Demo

TestSort.java

```java
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        int expected = 1;
        int actual = Sort.findSmallest(input);
        assertThat(actual).isEqualTo(expected);
    }
}
```

Progress Roadmap

Created **testSort**:

```
testSort()
```

Created a **sort** skeleton:

```
sort(String[] inputs)
```

Created **testFindSmallest**:

```
testFindSmallest()
```

Used Google to figure out how to compare strings.

Created **findSmallest**:

```
String findSmallest(String[] input)
```

Created **testSwap**:

```
testSwap()
```

Used debugger to fix.

Created **swap**:

```
swap(String[] input, int a, int b)
```

Changed **findSmallest**:

```
int findSmallest(String[] input)
```

& modified test

Turns out that we had the wrong abstraction for **findSmallest**!

Figuring out the Recursion

Lecture 6, CS61B, Spring 2024

Progress Roadmap

Created **testSort**:

```
testSort()
```

Created a **sort** skeleton:

```
sort(String[] inputs)
```

Created **testFindSmallest**:

```
testFindSmallest()
```

Used Google to figure out how to compare strings.

Created **findSmallest**:

```
String findSmallest(String[] input)
```

Created **testSwap**:

```
testSwap()
```

Used debugger to fix.

Created **swap**:

```
swap(String[] input, int a, int b)
```

Changed **findSmallest**:

```
int findSmallest(String[] input)
```

& modified test

Turns out that we had the wrong abstraction for **findSmallest**!

- With that design error fixed, let's figure out how to finish our sort method.

Very Tricky Problem

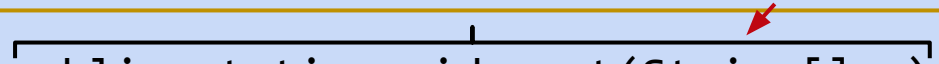
method signature

Without changing the signature of `public static void sort(String[] a)`, how can we use recursion? What might the recursive call look like?

```
public static void sort(String[] x) {
    int smallest = findSmallest(x);
    swap(x, 0, smallest);
    // recursive call??
}
```

(Could also use iteration, but I want to continue practicing recursion)

Very Tricky Problem



Without changing the signature of `public static void sort(String[] a)`, how can we use recursion? What might the recursive call look like?

```
public static void sort(String[] x) {
   int smallest = findSmallest(x);
   swap(x, 0, smallest);
   // sort(x[1:]); ← Would be nice, but not possible!
}
```

Some languages support sub-indexing into arrays. Java does not.

- Bottom line: No way to get address of the middle of an array.
- So what should we do instead?

Very Tricky Problem: Recursive Helper Method

method signature

Without changing the signature of `public static void sort(String[] a)`, how can we use recursion? What might the recursive call look like?

```
public static void sort(String[] x) {
   sort(x, 0);
}
```

```
/** In-place sorts x starting at index k */
public static void sort(String[] x, int k) {
   ...
   sort(x, k + 1);
}
```

Let's try implementing this idea in IntelliJ!

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      int smallest = findSmallest(x);
      swap(x, 0, smallest);
   }









}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      int smallest = findSmallest(x);
      swap(x, 0, smallest);
   }


   private static void sort(String[] x, int start) {









   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      int smallest = findSmallest(x);
      swap(x, 0, smallest);
   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {









   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {


   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {



      int smallest = findSmallest(x);
      swap(x, 0, smallest);


   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);

   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {



      int smallest = findSmallest(x);
      swap(x, 0, smallest);


   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);

   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {



      int smallest = findSmallest(x);
      swap(x, start, smallest);


   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);

   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {



      int smallest = findSmallest(x);
      swap(x, start, smallest);
      sort(x, start + 1);

   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);

   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {
      if (start == x.length) {

      }
      int smallest = findSmallest(x);
      swap(x, start, smallest);
      sort(x, start + 1);

   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);

   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {
      if (start == x.length) {
         return;
      }
      int smallest = findSmallest(x);
      swap(x, start, smallest);
      sort(x, start + 1);

   }
}
```

Fixing Another Design Error

Lecture 6, CS61B, Spring 2024

The Problem

Even after using our clever trick with the recursive helper method, our code is still not working:

What we know:

- Sort's helper methods have evidence of correctness from tests.
- Sort method itself is very simple.

Using the Debugger to Fix Our Code

Let’s try to use the debugger (as seen in labs 2 and 3).

- **IMPORTANT IDEA**: Let’s find the moment when reality diverges from expectation.
  - Don’t just step through the code hoping to see something weird.

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);
   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {
      if (start == x.length) {
         return;
      }
      int smallest = findSmallest(x);
      swap(x, start, smallest);
      sort(x, start + 1);
   }
}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);
   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {
      if (start == x.length) {
         return;
      }
      int smallest = findSmallest(x);
      swap(x, start, smallest);
      sort(x, start + 1);
   }
}

// start:         {"rawr", "a", "zaza", "newway"}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);
   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {
      if (start == x.length) {
         return;
      }
      int smallest = findSmallest(x);
      swap(x, start, smallest);
      sort(x, start + 1);
   }
}

// start:         {"rawr", "a", "zaza", "newway"}
// after 1 swap:  {"a", "rawr", "zaza", "newway"}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);
   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {
      if (start == x.length) {
         return;
      }
      int smallest = findSmallest(x);
      swap(x, start, smallest);
      sort(x, start + 1);
   }
}

// start:         {"rawr", "a", "zaza", "newway"}
// after 1 swap:  {"a", "rawr", "zaza", "newway"}
// after 2 swaps: {"a", "newway", "zaza", "rawr"}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);
   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {
      if (start == x.length) {
         return;
      }
      int smallest = findSmallest(x);
      swap(x, start, smallest);
      sort(x, start + 1);
   }
}

// start:         {"rawr", "a", "zaza", "newway"}
// after 1 swap:  {"a", "rawr", "zaza", "newway"}
// after 2 swaps: {"a", "newway", "zaza", "rawr"}
// but we got:    {"rawr", "a", "zaza", "newway"}
```

Major Design Flaw in findSmallest

Debugger showed us that we didn't properly account for how `findSmallest` would be used.

- Example: Want to find smallest item from among the last 4:

- We need another parameter so that `findSmallest` is useful for sorting.

Progress Roadmap

Created **testSort**:

```
testSort()
```

Created a **sort** skeleton:

```
sort(String[] inputs)
```

Created **testFindSmallest**:

```
testFindSmallest()
```

Used Google to figure out how to compare strings.

Created **findSmallest**:

```
String findSmallest(String[] input)
```

Created **testSwap**:

```
testSwap()
```

Used debugger to fix.

Created **swap**:

```
swap(String[] input, int a, int b)
```

Changed **findSmallest**:

```
int findSmallest(String[] input)
```

Added helper method:

```
sort(String[] inputs, int k)
```

Used debugger to identify another fundamental design flaw in **findSmallest**.

- Let's try to fix it.

Coding Demo

TestSort.java

```
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        int expected = 1;
        int actual = Sort.findSmallest(input);
        assertThat(actual).isEqualTo(expected);









    }
}
```

Coding Demo

TestSort.java

```
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        int expected = 1;
        int actual = Sort.findSmallest(input, 0);
        assertThat(actual).isEqualTo(expected);









    }
}
```

Coding Demo

TestSort.java

```
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        int expected = 1;
        int actual = Sort.findSmallest(input, 0);
        assertThat(actual).isEqualTo(expected);

        expected = 1;
        actual = Sort.findSmallest(input, 0);
        assertThat(actual).isEqualTo(expected);




    }
}
```

Coding Demo

TestSort.java

```
public class TestSort {
    @Test
    public void testFindSmallest() {
        String[] input = {"rawr", "a", "zaza", "newway"};
        int expected = 1;
        int actual = Sort.findSmallest(input, 0);
        assertThat(actual).isEqualTo(expected);

        expected = 3;
        actual = Sort.findSmallest(input, 2);
        assertThat(actual).isEqualTo(expected);




    }
}
```

Coding Demo

Sort.java

```
public class Sort {
   /** @source https://stackoverflow.com/questions/5153496 */
   public static int findSmallest(String[] x) {
      int smallest = 0;
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(x[smallest]);
         if (cmp < 0) {
            smallest = i;
         }
      }
      return smallest;
   }




}
```

Coding Demo

Sort.java

```
public class Sort {
   /** @source https://stackoverflow.com/questions/5153496 */
   public static int findSmallest(String[] x, int start) {
      int smallest = 0;
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(x[smallest]);
         if (cmp < 0) {
            smallest = i;
         }
      }
      return smallest;
   }




}
```

Coding Demo

Sort.java

```
public class Sort {
   /** @source https://stackoverflow.com/questions/5153496 */
   public static int findSmallest(String[] x, int start) {
      int smallest = start;
      for (int i = 0; i < x.length; i += 1) {
         int cmp = x[i].compareTo(x[smallest]);
         if (cmp < 0) {
            smallest = i;
         }
      }
      return smallest;
   }



}
```

Coding Demo

Sort.java

```
public class Sort {
   /** @source https://stackoverflow.com/questions/5153496 */
   public static int findSmallest(String[] x, int start) {
      int smallest = start;
      for (int i = start; i < x.length; i += 1) {
         int cmp = x[i].compareTo(x[smallest]);
         if (cmp < 0) {
            smallest = i;
         }
      }
      return smallest;
   }




}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);
   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {
      if (start == x.length) {
         return;
      }
      int smallest = findSmallest(x);
      swap(x, start, smallest);
      sort(x, start + 1);
   }


}
```

Coding Demo

Sort.java

```
public class Sort {
   public static void sort(String[] x) {
      sort(x, 0);
   }

   /** Sorts the array starting at index start. */
   private static void sort(String[] x, int start) {
      if (start == x.length) {
         return;
      }
      int smallest = findSmallest(x, start);
      swap(x, start, smallest);
      sort(x, start + 1);
   }


}
```

Progress Roadmap

Created **testSort**:

```
testSort()
```

Created a **sort** skeleton:

```
sort(String[] inputs)
```

Created **testFindSmallest**:

```
testFindSmallest()
```

Used Google to figure out how to compare strings.

Created **findSmallest**:

```
String findSmallest(String[] input)
```

Created **testSwap**:

```
testSwap()
```

Used debugger to fix.

Created **swap**:

```
swap(String[] input, int a, int b)
```

Changed **findSmallest**:

```
int findSmallest(String[] input)
```

Added helper method:

```
sort(String[] inputs, int k)
```

Used debugger to identify another fundamental design flaw in **findSmallest**.

Modified **findSmallest**:

```
int findSmallest(String[] input, int k)
```

And We're Done!

Often, development is an incremental process that involves lots of task switching and on the fly design modification.

Tests provide stability and scaffolding.

- Provide confidence in **basic units**.
- Ensure that later changes to **basic units** don't break them.
  - All individual pieces of your code are under constant inspection, not just the overall program.
- Help you focus on one task at a time.

In larger projects, tests also allow you to safely **refactor**! Sometimes code gets ugly, necessitating redesign and rewrites (see projects 2B and 3).

Testing Philosophy

Lecture 6, CS61B, Spring 2024

A New Way

Intro to Unit Testing

- Ad-Hoc Tests are Tedious
- Unit Testing Frameworks, Truth

Building Selection Sort

- The Selection Sort Algorithm
- Find Smallest
- Swap
- Correcting a Design Error
- Figuring out the Recursion
- Fixing Another Design Error

Testing Philosophy

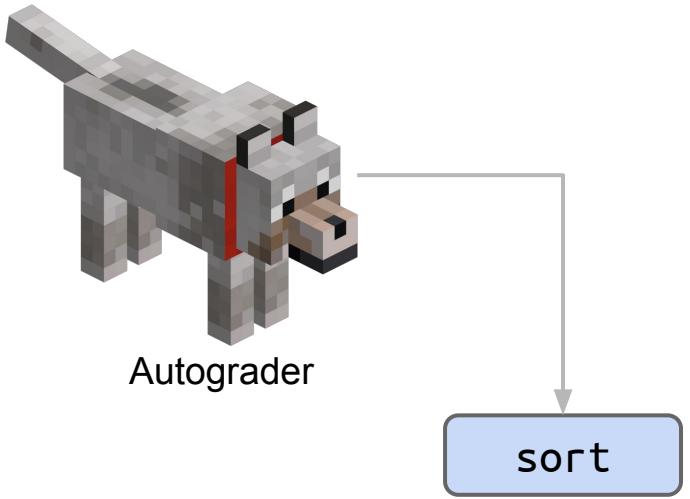
Correctness Tool #1: Autograder

Idea: Magic autograder tells you code works.

- Behind the scenes, we use Truth + JUnit + jh61b libraries.

Downsides:

- Don’t exist in the real world.
- Very slow workflow.



Autograder Driven Development (ADD)

The worst way to approach programming in 61B:

- Write entire program.
- Send to autograder. Get many errors.
- Until correct, repeat:
  - Run autograder.
  - Add print statements to find bug.
  - Make changes to code to try to fix bug.

This workflow is slow and unsafe!

```
[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]
Test Failed. Expected: ...
```

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

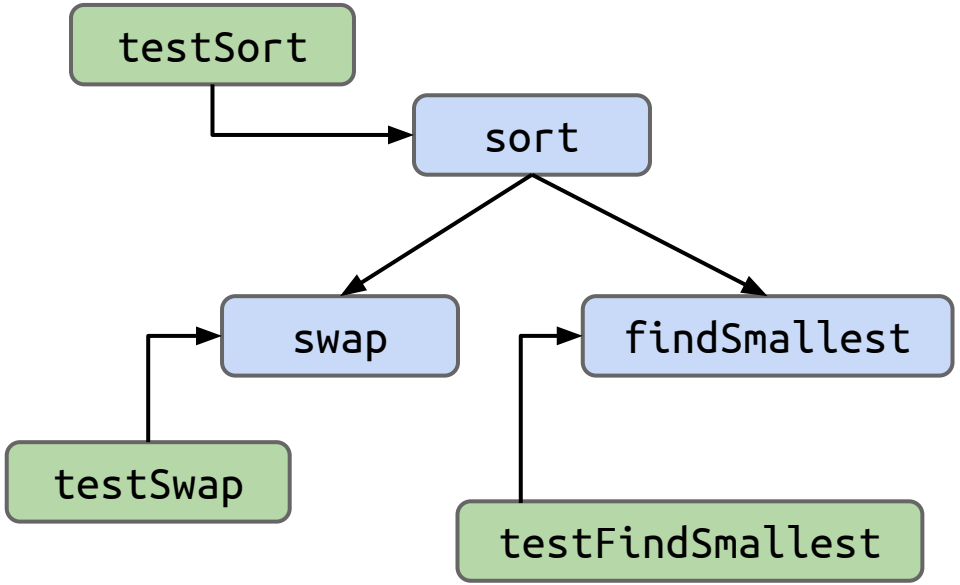
Correctness Tool #2: Unit Tests

Idea: Write tests for every “unit”.

- Truth (and assertJ and JUnit) make this easy!

More up front investment.

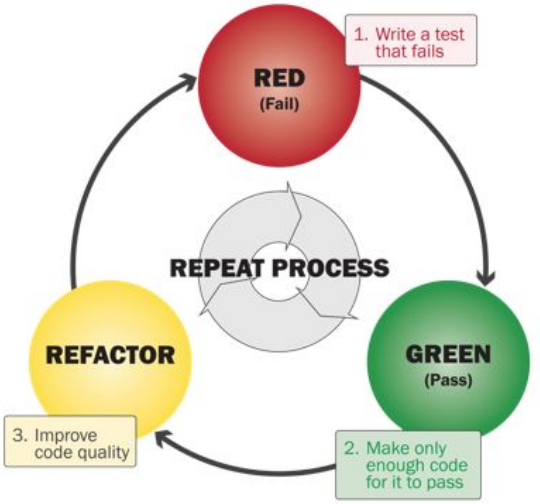
- Saves you time in the long run!



Test-Driven Development (TDD)

Steps to developing according to TDD:

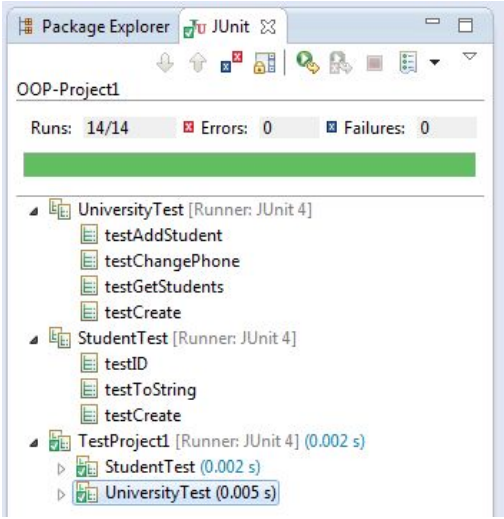
- Identify a new feature.
- Write a unit test for that feature.
- Run the test. It should fail. (RED)
- Write code that passes test. (GREEN)
    - Implementation is certifiably good!
- Optional: Refactor code to make it faster, cleaner, etc.



Not required in 61B. You might hate this!

- Even if you don't use TDD, testing is a good idea.

Interesting perspective: Red-Shirt, Red, Green, Refactor.

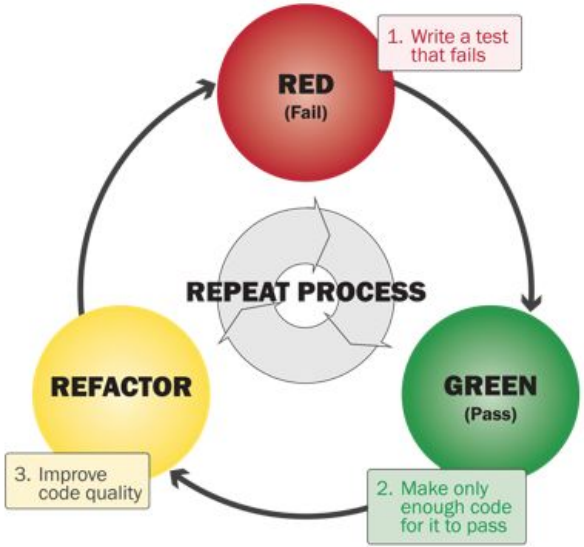


A Tale of Two Workflows

TDD is the opposite of the autograder-with-print-statements workflow.

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

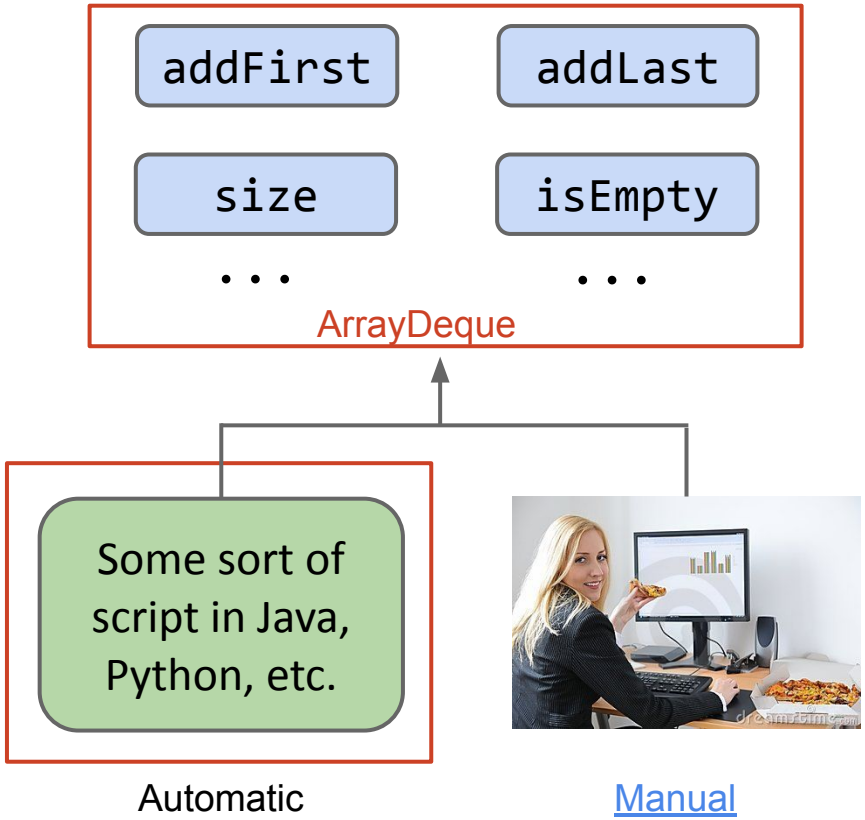
```
$ python sort.py
[63, 12, 91, 5, 0]
got to this spot, lt is: 1
got to this spot, lt is: 2
got here!
[63, 12, 0, 5, 91]
got to this spot, lt is: 3
got to this spot, lt is: 4
got here!
[5, 12, 0, 63, 91]
```



Correctness Tool #3: Integration Testing

Idea: Tests cover interaction of all units at once.

- As seen in project 0.



Why? Unit testing is often not enough to ensure modules interact properly or that system works as expected.

We won't have you build full scale integration tests in our class.

Extra Slides: How Unit Tests are Run

Bonus Slide: What is an Annotation?

Annotations (like @Test) don't do anything on their own.

```
@Test
public void testSort() {
   ...
}
```

Runner uses reflections library to iterate through all methods with "Test" annotation. Pseudocode on next slide.

Sample Runner Pseudocode

Runner uses reflections library to iterate through all methods with “Test” annotation.

```
List<Method> L = getMethodsWithAnnotation(TestSort.class,
                                          @Test);
int numTests = L.size();
int numPassed = 0;
for (Method m : L) {
    result r = m.execute();
    if (r.passed == true) { numPassed += 1; }
    if (r.passed == false) { System.out.println(r.message); }
}
System.out.println(numPassed + “/” + numTests + “ passed!”);
```

How It Usually Goes...