Public-Service Announcement

Interested in Education? Robotics? STEM outreach?

Pioneers in Engineering (PiE) is a student group that provides fun STEM experiences to underrepresented students in the Bay Area. We develop robotic systems and hands-on educational programs that high school students use in a year-long Mentoring Program and an 8-week Robotics Competition hosted by PiE.

If you would like to learn more about what we do, e-mail us at recruiting@pioneers.berkeley.edu, and join us for one of our infosessions on 9/8 (HP Auditorium) and 9/10 (2060 VLSB) @7PM!

CS61B Lecture #3

- **Reading:** Please read Chapter 4 of the reader *A Java Reference* for Wednesday (on Values, Types, and Containers) and Chapter 3 of the textbook.

- **Labs:** We are forgiving during the first week or so, but try to get your lab1 submitted properly. **DBC: Let us know if you can't get something to work!**

- **Homework:** Please see Homeworks #0 (optional) and #1 on the homework/lab page.

More Iteration: Sort an Array

**Problem.** Print out the command-line arguments in order:

```java
% java sort the quick brown fox jumped over the lazy dog
brown dog fox jumped lazy over quick the
```

**Plan.**

```java
public class Sort {
    /** Sort and print WORDS lexicographically. */
    public static void main(String[] words) {
        sort(words, 0, words.length-1);
        print(words);
    }

    /** Sort items A[L..U], with all others unchanged. */
    static void sort(String[] A, int L, int U) {
        /* "TOMORROW" */
    }

    /** Print A on one line, separated by blanks. */
    static void print(String[] A) {
        /* "TOMORROW" */
    }
}
```

How do We Know If It Works?

- **Unit testing** refers to the testing of individual units (methods, classes) within a program, rather than the whole program.

- In this class, we mainly use the JUnit tool for unit testing.

- **Example:** AGTestYear.java in lab #1.

- **Integration testing** refers to the testing of entire (integrated) set of modules—the whole program.

- In this course, we'll look at various ways to run the program against prepared inputs and checking the output.

- **Regression testing** refers to testing with the specific goal of checking that fixes, enhancements, or other changes have not introduced faults (regressions).
Test-Driven Development

- Idea: write tests first.
- Implement unit at a time, run tests, fix and refactor until it works.
- We’re not really going to push it in this course, but it is useful and has quite a following.

Testing sort

- This is pretty easy: just give a bunch of arrays to sort and then make sure they each get sorted properly.
- Have to make sure we cover the necessary cases:
  - Corner cases. E.g., empty array, one-element, all elements the same.
  - Representative “middle” cases. E.g., elements reversed, elements in order, one pair of elements reversed, . . .

Simple JUnit

- The JUnit package provides some handy tools for unit testing.
- The Java annotation @Test on a method tells the JUnit machinery to call that method.
- (An annotation in Java provides information about a method, class, etc., that can be examined within Java itself.)
- A collection of methods with names beginning with assert then allow your test cases to check conditions and report failures.
- [See example.]

Selection Sort

```java
/** Sort items A[L..U], with all others unchanged. */
static void sort (String[] A, int L, int U) {
    if (L < U) {
        int k = indexOfLargest (A, L, U);
        String tmp = A[k];
        A[k] = A[U];
        A[U] = tmp;
        sort (A, L, U-1); // Sort items L to U-1 of A
    }
}
```

Iterative version:

```java
 while (L < U) {
        int k = indexOfLargest (A, L, U);
        String tmp = A[k];
        A[k] = A[U];
        A[U] = tmp;
        U -= 1;
    }
```

And we’re done! Well, OK, not quite.
Really Find Largest

/** k, I0<=k<=I1, such that V[k] is largest element among * V[I0], ... V[I1]. Requires I0<=I1. */
static int indexOfLargest(String[] V, int i0, int i1) {
    if (i0 >= i1)
        return i1;
    else /* if (i0 < i1) */ {
        int k = indexOfLargest(V, i0+1, i1);
        return (V[i0].compareTo(V[k]) > 0) ? i0 : k;
    }
}

Iterative:
int i, k;
k = i1;  // Deepest iteration
for (i = i1-1; i >= i0; i -= 1)
    k = (V[i].compareTo(V[k]) > 0) ? i : k;
return k;

Another Problem

Given an array of integers, A, move its last element, A[A.length-1], to just after nearest previous item that is \( \leq \) to it (shoving other elements to the right). For example, if A starts out as

\{ 1, 9, 4, 3, 0, 12, 11, 9, 15, 22, 12 \}

then it ends up as

\{ 1, 9, 4, 3, 0, 12, 11, 9, 12, 15, 22 \}

If there is no such previous item, move A[A.length-1] to the beginning of A (i.e., to A[0]). So

\{ 1, 9, 4, 3, 0, 12, 11, 9, 15, 22, -2 \}

would become

\{ -2, 1, 9, 4, 3, 0, 12, 11, 9, 15, 22 \}

(Preliminary question: How can I state this without making this last case special?)

Finally, Printing

/** Print A on one line, separated by blanks. */
static void print(String[] A) {
    for (int i = 0; i < A.length; i += 1)
        System.out.print(A[i] + " ");
    System.out.println();
}
/* J2SE 5 introduced a new syntax for the for * loop here: */
for (String s : A)
    System.out.print(s + " ");
/* Use it if you like, but let's not stress over it yet! */

Your turn

public class Shove {
    /** Move A[A.length-1] so that it is just after the nearest * previous item that is <= A[A.length-1], or to A[0] if * there isn't such an item. Move all succeeding items * to the right (i.e., up one index). */
    // BETTER DESCRIPTION?
    static void moveOver(int[] A) {
        // FILL IN
    }
}

Last modified: Wed Sep 9 12:32:07 2015