Program #1 Helper

Prof Bill - Jan 2020

Everything for Program #1 is up at the Programs page.

This file will include notes and code snippets to help you in Program #1.

thanks...yow, bill

Quality code

P1 marketing haiku.

> CSC 210. Spring 2020.
> Quality Code. No crap.™

Our CSC 210 coding guidelines are wtkrieger.faculty.noctrl.edu/csc210-spring2020/docs/java_coding_guidelines.pdf

Why are coding conventions important? This is pretty good.

Code conventions are important to programmers for a number of reasons:

- 40%–80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

Source: en.wikipedia.org/wiki/Coding_conventions

Wed Jan 29

Here are some important Program #1 changes:

- New deadline for Program #1: **Sun Feb 2 @ noon**. We'll discuss this in class.
- I'm dropping the "special effects" requirement for the spin command. You still need to add one **creative element** to your P1 solution. Note: This can be special effects at the console, or something else: a command or fancy intro or whatever.
- Reminder: We dropped **read and save commands** earlier. See Jan 24 below.

On my k: drive, I have added a fill-in-the-blanks README.txt for when you're done.

ABC = Always Be Coding/Compiling/Closing

Tue Jan 28

I updated code in my common_area/program01:

- I have updated the **Lis210.java** interface to include the isEmpty() we discussed.
- There are two new files:
    - **List210Tester.java** - test code for your List210
    - **Program01.java** - a sample main to run the list tester

More in class tomorrow…or email me!

thanks...yow, bill

Fri Jan 24

Important: Let's **drop** the two wheel commands related to files: save and read. My rationale: we haven't covered files at all, and our wheel work week is hectic enough.

My after class coding is available on the k: drive...the folder is common_area/program1-inclass.

Current advice:

- ➔ One method/command at a time! ABC = Always be Compiling.
- ➔ Use this interface for your list: List210. Grab it from the k: drive.
- ➔ My classes are:
    - ◆ MyList, MyNode - my linked list (implements List210) and node
    - ◆ Program01 - my main()
    - ◆ Wheel - the wheel where methods should matchup with commands
    - ◆ WheelCommander - get user command and execute it
- ➔ I think I added toString() to MyList to print the items in the wheel. I think I used StringBuilder. I think I added a copy ctor too. Your mileage may vary.

About **reload**...we decided that our wheel alternates between adding items and spinning. On the first spin after adding, save the items in your wheel (to another list). Reload then uses that ist to replace the list of current items. This implies that your Wheel class maintains two things: 1) a reload List210 of items, and 2) a spinning flag to indicate if the wheel is spinning or not. I would implement this command last. Check that. I will implement this command last. (ha)

It's coding weekend. Make it happen! **Email me.** I will be checking email anon.

good luck...yow, bill

Tue Jan 21

Lessons from Homework 01/02:

- ➢ **Consistent** names, spacing, and indentation
- ➢ **Chunks!** Comment and space your code in chunks, not line-by-line:

```
// this chunk does this
if( something) {
  ...
```

```
        }
```

- ➢ **Reader** - the person reading your comments knows Java
- ➢ **Outside help** - comment outside sources of code snippets/ideas you use, include links if any
- ➢ **README.txt** - remember README: name, state of your program, how to run it
- ➢ **Simpler** folder names makes term navigation easier: lower case, no spaces
- ➢ **Done** - as a last step, make sure your program works on the **k: drive**; compile and run

Coding hints:

- ★ Create Program01.java now!
    - It's tiny.
    - It has a pulse: Hello, Program 1
    - Add stub if necessary; something like: WheelCommander wc; wc.run()?
- ★ Create a TODO list; don't turn P1 into an amorphous blob
- ★ ABC - Always Be Compiling; never leave your program in a messy state
    - Create stubs (empty methods) to compile, where necessary
    - Comment out interface methods to compile; code them up one at a time
- ★ Create separate main() methods to test; List210Tester class? We can share code that tests a List210, right?

Wed Jan 15

Design goals: reusable code!

- ➔ Design your linked list so it can be used in other projects.
- ➔ The Wheel in Program #1 is text-based and runs in the console. But the Big Boss indicates that a GUI version may be coming soon. So, design your Wheel so that it can be used in a GUI (or elsewhere) later.

Critical! You must **start working now** on Program #1...to win.

- ★ Start working on P1. Design, then code.
- ★ Start to subdivide this program; identify your classes (and files?)
- ★ Then, run! What tiny portion of Program #1 are you going to work on first? Then second. Rinse and repeat.

Mon Jan 13

We will read text commands from the console. Use Scanner. There are lots of “next” methods to read input: nextLine() reads text, nextInt() reads an integer.

Tested on www.onlinegdb.com/online_java_compiler
Javadoc: docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

```
import java.util.Scanner;

# Works on www.onlinegdb.com/online_java_compiler
public class Main
{
    public static void main(String[] args) {

        System.out.println("Enter your name: ");
        Scanner scanner = new Scanner(System.in);
        String username = scanner.nextLine();
        System.out.println("Hello " + username);

        System.out.println("Enter your age: ");
        int age = scanner.nextInt();
        System.out.println( age + "?!! You're a big boy!");

        scanner.close();
    }
}
```