**Swap 1 HW**

**CSSE375-02&03, Spring 2014**

This is the first of **multiple “swap” activities** among small teams, doing coding as a HW activity.

The learning goals are as follows:

1. What it’s like exchanging code with others who need to use your code as their next step.
2. Evaluating the “quality” of peers’ code, in specific ways.
3. Applying specific coding skills we’re learning in 375, such as finding smells and refactoring.

Overall, you’ll be adding features and fixing bugs to a system that’s already been started.

At the end of **each week**, you will pass your code to another team. They will then do the following:

1. Grade your code immediately, based on the rubrics of that particular code swap.
2. Add to and bug fix your code, making it “their code” for the next week.

So, for example, next Monday, we’ll do this trade-off, starting with grading what you get from another team.

* **Warning:** We’ll do the “exchange of code” during each Monday’s class, starting next Monday, so it really MUST be done and running by then!

**Where’s the code to start with?**

It’s in “sg15.rar”, in this directory, ready to be un-zipped. It’s a Java program which should run under Eclipse on your machine. (Mine is Indigo Service Release 2, and it runs fine on that.) This is a well-done student Java project, and it already shows very good examples of clearly written (and commented) code.

**What’s it supposed to do?**

This is described in “CSSE 221 – Schedule Generator Project.htm”, also in this directory. As you can see, the program was an assignment for last year’s CSSE221**. Do not** pay attention to the process specifics of that 2012 assignment, such as what svn files to use for turnin! **But do** pay attention to the required functionality of the program! E.g., It’s supposed to generate schedules for a church, with people assigned responsibilities, according to some fair algorithm, at different events. In particular, look at the Required Features, Additional Features, and Bonus Features at the bottom of this 221 assignment. This program DOES do a lot of these functions! (But not all.)

**What do we’ll do during the first week of this HW assignment?**

I. “Today,” on Monday, Feb 17, do the following in class:

1. Preliminaries: You will get a partner to work with. “**Register**” your team immediately on Moodle as shown under Homeworks: “Swap HW 1 - Identify your team!”
2. With your partner, set up whatever system you would like, so that you can share doing the changes. Note that you will be exchanging code with other teams every week, so you should consider that! We will discuss this in class, so you can hear what others are doing. It is strongly recommended that you use a system that lets you revert to an earlier version, just in case some change you are making leads to a mess. See point 0 of the rubric table at the end of these instructions!
3. Bring your laptops: Both partners on a team need to get the program running during class today, and verify what functions you believe it does or doesn’t do, from the list on the 221 assignment. Here’s my own impression: It does the “Required” and “Additional” features described at the bottom of the 221 assignment.

II. During the week, do the following things with your partner, to your copy of this code, verifying after each activity that the system still runs:

1. On every coding change or comment you make, **precede that line or area of code with a comment line** which says: // SWAP 1, TEAM XX where XX is your team number. This way, it’s easy for the next team to find all the changes you made, and use the rubric to grade those.
2. See if you can make the code have the following **good OO qualities**, labeling these with the comment tag QUALITY CHANGES, so they are easy to find. Since we will just be starting on refactoring techniques in class, you are free to invent your own ways to resolve these (but you need to comment how you did them:
   1. The class CalendarGUI.java and the class Config.java both seem to have a lot of *duplicate or very similar code*. In each of these two classes, pick at least two glaring examples of this and get rid of the extra code in elegant ways. Document how you did these, so that whoever is grading your code can tell, easily, what you did.
   2. The class Schedule.java has one *very long method* called calculateNextMonth. Convert this method into smaller methods, each of which have a single purpose and greater cohesion. Explain how you did this.

**On each of these**, comment what additional features could be enabled by the refactoring. Be realistic, but you can use your imagination within that. E.g., if you get rid of separate code for each day of the week, don’t suggest that this would enable adding more days to the week!

1. **Find as many additional code smells as possible:** There are 22 to choose from. With a comment above the smelly code, label each as follows: SMELL: <name of smell> - <explanation>. E.g., in config.java, above the constructor, “SMELL: duplicate code – almost identical code is given for each of the seven days of the week.”  
     
   Your goal for perfection on this: Find a total of 10 of the 22 **different** smells that Fowler identifies, one non-trivial example of each.  
     
   On each SMELL you identify, also **describe the kind of enhancement** that the change would enable. **Don’t do those changes**, but do comment them in enough detail that someone else could do so.
2. **Make changes using refactoring:**   
    **Note:** In doing these changes, you will need to use the “two step process” –
   1. First making the existing code ready for the change. Describe:
      1. What “smell” you overcame, and
      2. How you refactored the code to overcome that. Then,
   2. Make the enhancement without more changes the existing code (“any” is best, “not much” is pretty good)  
        
      In each of the features, *below*, outline, in the tagged comments that go with them, how you used this required methodology!
3. Add **one** additional “**bonus feature**” from the list at the bottom of the 221 assignment. (I believe the program does not do any of these, at present.) Label all the areas where you make related changes to enable this, with the tag BONUS FEATURE. At the main place where you added the enabling code, describe the feature.

**Note:** I believe that the last one is tricky: “Accept personal calendars and not schedule individuals when they are not available.” As an example of the format you would see, I included my own upcoming calendar in the file, “Chenoweth Stephen V Calendar.ics”, which is output from MS-Outlook. It’s tricky!

1. Make the program do an **additional feature** of your own invention, something which is difficult to add with one of the bad smells you already identified above, but easy if you fix that smell! Label this one as ADDITIONAL FEATURE, and make your comments that go with it clear, so that the team receiving your code in the next Swap can check it out.

Let me know if you have questions!

**The “turnin” for this assignment**

1. This will be “due” at the start of class, Monday, Mar 24. That’s a hard deadline, because if you miss it, you leave the next team, who will work on your code, with nothing to do! The turn-in will be a zipped version of the Eclipse program files for the program, like the way you received it, but now ready to pass on to the next team.
2. In class, you will find out the names for the team you will send it to. You can email the zipped files to them directly, at the start of class.
3. During class, you also need to turn in the zipped files to me, in a drop box on Moodle! One copy per team.

**The grading**

Here’s the rubric to use in grading the system you receive. You can start grading during class on Monday, March 24. You will need to finish grading, and turn-in your rubric form as a .docx file, in the Moodle drop box for this, by 11:55 pm next Monday night.

SWAP 1 RUBRIC:

CSSE 375 Section # \_\_\_\_ From Team # \_\_\_\_ , Graded by Team # \_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| **Capability / Feature / Assignment** | **Grade (“works perfectly” or “done perfectly” = full credit in each case)** | **Grader Comments – Explain anything that is missing, doesn’t work, or has issues or concerns.** | **Rubric Comments and Examples** |
| **0. It still runs** – Does all the “Required” and “Additional” features for the original CSSE221 assignment, that it did before the team changed it. | \_\_\_\_ out of 30 |  | Take off for things that don’t run. If they don’t have anything to give you next Monday, don’t show up in class, or the whole thing blows up when you try to run it, extend this grade to a 0 on the whole assignment. |
| 0’. They made wholesale changes to the system that were not asked for. | \_\_\_\_ out of -100 |  | This is a negative grade if they did it. **Zero** if they avoided the temptation to do this. -100 if they rewrote the whole program without commenting where they made changes. -30 if they changed key data structures, like how the calendar is represented. |
| 0’’. They didn’t follow the documentation guidelines to make it easy for you to grade this. | \_\_\_\_ out of -20 |  | Also a negative grade. Take off some points if they didn’t put the SWAP 1… comment lines in, or the more specific ones asked for, like SMELL. |
| B 1. They got rid of two different sets of duplicate code in the class CalendarGUI.java, and two such sets in Config.java. | \_\_\_\_ out of 10 |  | Proportional grade if they got part. Full credit if their solutions look understandable and do the job. -2 if they did not also describe sensible features that their refactorings would enable. |
| B 2. They rebuilt the one long method in Schedule.java in an elegant and understandable way. | \_\_\_\_ out of 10 |  | Same as *above*. |
| C. They found 10 additional, different code smells and their examples appear appropriate and non-trivial. | \_\_\_\_ out of 20 |  | Same as *above*. Grade proportionally if they got like 8 out of 10. No credit for duplicate refactoring types, even if these were a good idea. |
| D in general. They documented how they used the two-step process in making changes. | \_\_\_\_ out of 10 |  | This requires them to describe what they did. But, to grade it, you may also need to refer to the original code for comparison. |
| D 1. They added one of the bonus features from the 221 assignment, and it works. | \_\_\_\_ out of 10 |  | Be sure to test that it works, and also the quality of the code that does it, which should be marked so you can find it. |
| D 2. They added one additional feature of their own invention. It is well described, well written, and works. | \_\_\_\_ out of 10 |  | Same as *above*. |
| Total | \_\_\_\_ out of 100 |  |  |

Below the table, in the .docx that you hand in, note additional bugs found that need to be fixed:

1. Description:
2. Etc.

Please observe that I also plan to grade you on the quality of your grading, based on my own review of the same code!

**The next cycle, Swap 2!**

After doing your grading of what you receive, you can start on the next cycle of changes. You will have a new assignment with new things to do, like more refactoring, for Swap 2. I’ll be writing those as you’re doing this assignment!