Project Two

Alex Roberts

Summary:

Unit Testing Approach: The system requirements were carefully thought out before I started writing any of the code. It clearly stated what the criteria was and how everything was supposed to work. I planned ahead and made decisions on how I should write the code and then put it all to work. For example, one of the requirements was for the contact object to have a required firstName String field that cannot be longer than 10 characters long and the firstName field cant be null. This makes sure that users are only able to make names a certain length to make things simple and also doesn’t let them submit it if they leave first name blank. The code for this looks like this: if (firstName == null || firstName.isEmpty()) {  
 this.firstName = "NULL"  
else if (firstName.length() > 10) {  
 this.firstName = firstName.substring(0, 10)  
else {  
 this.firstName = firstName;  
}

In the contact service and the appointment service objects are 100% on their Junit tests and the task service got 80%. This tells me that the quality of the code is good but still needs a little work on the task object. This doesn’t completely ensure that the program won’t run without error but it does help minimize errors and makes it easier to pinpoint the problems.

Junit experience: In order to be technically sound you have to write code efficiently and clearly to make sure that your program is running properly and it is easy to read. To do this I used the skills that I have learned throughout my experience coding. I used proper syntax and gave easy to recognize names to everything so I am able to have a good idea about what each object is supposed to do and can connect different parts of the code easier. I also provided comments in the code so either I or somebody else can understand more about what that part of the code is for. For example: public class Task {  
 private final String taskID;  
 private String taskName;  
 private String taskDesc;  
 private static AtomicLong \*idGenerator\* = new AtomicLong();  
  
// CONSTRUCTORS  
/\*  
 \* Constructor takes task ID, task name, and task description as parameters.  
 \* All parameters are checked if null or empty. If either exists, the field is filled  
 \* with the phrase "NULL" to protect data integrity as a placeholder. Task ID is truncated  
 \* to a maximum of 10 characters, task name to 20 characters, and description to 50 characters.  
 \*  
 \*/  
  
 public Task(String taskName, String taskDesc) {  
  
 // TASKID  
 // Task ID is generated when the constructor is called. It is set as a final variable and has  
 // no other getter or setter so there should be no way to change it.  
 // The idGenerator is static to prevent duplicates across all tasks.  
  
 this.taskID = String.\*valueOf\*(\*idGenerator\*.getAndIncrement());  
   
 if (taskName == null || taskName.isEmpty()) {  
 this.taskName = "NULL";  
 } else if (taskName.length() > 20) {  
 this.taskName = taskName.substring(0, 20);  
 } else {  
 this.taskName = taskName;  
 }  
   
 if (taskDesc == null || taskDesc.isEmpty()) {  
 this.taskDesc = "NULL";  
 } else if (taskDesc.length() > 50) {  
 this.taskDesc = taskDesc.substring(0, 50);  
 } else {  
 this.taskDesc = taskDesc;  
 }  
 }

This code is effective at what it is supposed to do but it is not too much which makes it efficient. Code being efficient means that it is simple to understand and read but it gets the job done. The code is commented to give some explanation to what it does so that way if you are trying to fix a problem later it will be much simpler to find.

Reflection: There are several techniques that you can use for testing code. One that I used in this project is Boundary Value Analysis. This technique tests values between two extreme ends. Programmers often use this for password creation to make sure that the users password meets a certain criteria. This was used in the above code snippet for making firstName no longer than 10 characters long. Some other testing techniques are Decision Table Testing which is used to test combinations of conditions and actions and State Transition Testing which tests changes of state, in which the behavior depends on the transition between a current state and a past state.

While working on this project I developed the mindset of being more mindful of how I am writing my code. As I have always thought my coding skills were at least decent, when I ran the tests on them I realized that it could be much better. This project has taught me a lot about writing more efficient code. I have always had the habit of “doing too much” when there are simpler ways that I could write it.

When writing code is important to pay attention to bias. Having bias in your code means that you are comfortable with your skills and are sure that what you are doing is the best way to do it. Avoiding bias means keeping an open mind when writing code and adopting other skills and this will help you write the best code possible especially if you are working with a team.. They may have better ways to do it. If you have bias then you are going to miss a lot of opportunities of making yourself a better programmer. I tried to avoid bias by adopting skills that I have learned from this course instead of just doing what I normaly do. In doing that I have learned a lot about writing more efficient code.

It is very important to be disciplined in your commitment to being a good programmer. If you are going to do this job you need to make sure you are doing it right. If you cut corners you are risking making a janky program and you are not building on your skill level. In the end this could ultimately lead to the downfall of your coding career. On top of that, you could provide people with programs that don’t work properly which could cause all sorts of other problems. It is our responsibility to make sure that we are being the best that we can be.