Joy Kalinowsky  
CS 320  
 Project Two  
 In the past few weeks, I have been developing a mobile application for a customer under Grand Strand Systems. My work on the project constituted developing the contact, task, and appointment services of the mobile application. When building my program, I centered my software design around meeting the customer’s requirements. I used different software testing techniques to write tests to ensure that every service I built for the customer worked as intended across a variety of user inputs, making sure to adhere to good programming practices.  
 My approach was aligned with the software requirements because I made sure to incrementally implement every requirement. I structured the classes that I had to program around the software requirements. For each class, I created a field for each piece of information that needed to be included. For example, in the Appointment class, when one of its requirements was to include an appointment ID, I created a field called appointmentID to contain appointment ID information. I also created methods for other classes to retrieve these fields and modify them when applicable. Using the Appointment class example again, I created the methods getDescription and setDescription to retrieve and modify the description field of an appointment object. In addition to creating classes for the listed objects in the requirements, I’ve also created accompanying service classes to store objects made from these classes. These classes each contain an array field for storing objects created from their associated class and methods to manipulate the array. Objects can be added, removed, and modified inside the array.  
 I know that my JUnit tests were effective based on the coverage percentage because the coverage percentage ensured that every aspect of my program was correctly implemented. I tried to make my coverage percentage as close to 100% as possible, covering both cases of users entering valid and invalid parameters. For all classes, what constituted a valid parameter was determined by the requirements for their object creation. An example of a requirement is for the Contact class’s contactID field, which states that a contact’s ID cannot be null, longer than ten characters, or modified after instantiation. To do this, I created tests for every method in each class and their corresponding service class. Within each test, I created a case that was supposed to succeed, as well as several other cases that were supposed to fail, representing valid and invalid inputs that the user could enter to create an object respectively. This way, I could make sure that my program did not allow invalid inputs, and did not create objects containing invalid inputs.   
 I ensured that my code was technically sound by running multiple tests on it. I did not just create tests that were supposed to pass, for that would’ve prevented me from testing for bad user inputs that would break the behavior of the program. Instead, I’ve added tests that throw exceptions for user inputs that are not allowed, such as creating objects like Appointments that have appointment ID values of null. After adding tests to ensure the program works as expected under normal user behavior and refuses the creation of objects with invalid parameters, I ran the tests to ensure they passed or failed as intended. All of the tests that are meant to pass have passed, and all of the tests that are created to throw exceptions in the case of bad user inputs have also gone through testing and worked correctly. An instance of this is the code line:  
*throw new IllegalArgumentException("Description cannot be null or longer than 50 characters.");*When the user attempts to create a new Appointment object, this line of code throws an exception if the description String that the user has entered for the Appointment object is either null or longer than fifty characters in length.   
 I ensured that my code was efficient by making sure that my methods did not use any more memory than they needed to. This was accomplished by using a type of array that had a very memory-efficient way of accessing its objects. For example, I replaced List array objects that I have previously created with Map objects. Methods take up much more memory when using lists than when using maps. This is particularly seen when creating objects. List array objects require iterating through an array to find an object, whereas Map array objects do not have this same requirement. List array objects need a loop such as a for loop or a for each loop to iterate through the list to locate an object. In contrast, to find an object in a Map array, all the method needs to do is use the key value associated with the Map array, and then if the object exists in the Map array, the object will be returned from the Map array. An example of a Map array object that has been used in a class can be seen in the AppointmentService class:  
 *private HashMap<String, Appointment> appointments = new HashMap<String, Appointment>();*  
This Map array object was created to store Appointment objects. The user has the option of adding and removing Appointments to and from the Map array as well as modifying their Date and Description fields.   
 The software testing techniques that I employed in this project were integration testing, system testing, and boundary value testing.   
 Integration testing is when the software is tested to make sure all of its parts or modules work together to accomplish system requirements. Modules that were recently implemented are combined in testing to make sure that they function properly together. The practical uses of integration testing are making sure that modules that work individually do not cause each other to break, as well as building a significant portion of the project all at once by testing multiple modules at the same time to make sure they work. The main implication of integration testing is that all of the modules of the current working state of the program have been fully tested to be able to be run together. No module causes another to stop working or overwrites the data of another module, causing it to function improperly.  
 System testing is when the entire program system is tested to make sure that it meets its system requirements. The main practical use of system testing is that it ensures that the system when it is finally put together from the sum of its modules, functions as expected and meets all of its requirements. The implications of system testing are that every module works as expected and that the entire system runs as expected with no gaps in meeting user requirements.  
 Boundary value testing is when values at the end of the expected user input value boundaries are entered into the software to ensure that the software handles these boundary values as expected. The practical uses of boundary value testing are for testing user inputs that are not expected to be common or likely, but could still show up nonetheless. An example of this would be testing for null values since these values are invalid in creating objects but are still possible for the user to type in. The implications of boundary value testing are that the program can handle unexpected user inputs without deviating from expected behavior or crashing.  
 Some software testing techniques that I did not use for this project are automated testing and security testing.   
 Automated testing is when automated tools are used to test the software with minimal human involvement. The practical use of automated testing is that it saves time for the testers since the testers do not have to input values for them manually. If many tests need to be run, then automated testing would be very helpful for the testers to run them in a short amount of time. The implications of automated testing are that the tests used for the software are easy to automate and therefore likely easy to recreate quickly using slightly different details.   
 Security testing is used to identify potential security flaws and vulnerabilities within a program. The practical uses of security testing are that they can ensure the security system is protected as much as possible from threats and they can be used to detect holes in program security that were not there before. The implications of security testing are that all potential security holes or flaws will be detected by these tests so the developers can then fix them and make their software free of its previous vulnerabilities.   
 I employed a good amount of caution in acting as a software tester. I tried my best to make sure I never had any illegal memory access, such as accessing elements of an array outside of its limit. I also ensured that for my test cases, they did not all access the same test global service array variable. I created one or more test service arrays to store multiple objects for each test, to ensure that no tests overwrote the array data of others. Overall, I tried to ensure that there was no possibility of illegal or unexpected memory access.  
 It was very important to appreciate the complexity and interrelationships of the code I was testing. The interrelationships of the code I was testing make it possible for different code modules to work together to form the final program. These interrelationships can take the form of different methods in a class working together to allow the user to modify the data fields of an object, or one data array field being open to modification by methods of its class to enable the user to store and modify objects within it.  
 I tried to limit bias in the review of my code by making sure to test each case of each test multiple times, particularly every time I made a change. As a biased tester, since I wrote the code, it can be easy to think that my code is infallible and that I don’t need to test it. However, creating tests to test each method of each class proved otherwise, since there ended up being plenty of issues in my code that were not obvious at first glance.  
 I can imagine that bias would be a concern if I were responsible for testing my own code. Bias over the integrity of my code can make it so that I am not as rigorous in testing it as I would be with somebody else’s code. I might just assume that certain methods or data fields work as expected, and not be inclined to make tests for them to ensure that they do. I might be too tired of looking at my code to want to test every detail of it to make sure it works as expected. I might also refuse to test how my program would work regarding edge cases since I would assume that it would handle them correctly. I might end up not devoting any time to making sure that my methods handle edge cases correctly since I could assume they are unlikely user inputs.  
 It’s important not to cut corners when writing or testing code because there cutting corners could result in code that may solve the problem, but is convoluted and difficult to maintain. An example of this includes redundant code, such as code that contains a large amount of copy-pasted code blocks that are virtually identical to each other bar the use of different variables each time. Such redundant code could easily be replaced with a method at first, but as the program grows and includes more lines of code, finding and grouping similar blocks of code into methods becomes much more difficult. This kind of lack of discipline may not prevent the program from working as intended, but instead makes the source code difficult for developers to parse through and understand the inner workings of, slowing product development. To avoid technical debt as a practitioner in the field, I would look for more efficient, readable solutions to blocks of code that are difficult to integrate.