Grand Strand Systems Testing and JUNIT Tests

Jon-Kayla Pointer

Southern New Hampshire University

October 10, 2022

1. **Summary.**

This overall project is a complex application designed for mobile platforms. This application utilizes multiple services functionalities. For this block specifically addresses the packages Contact Service, Task Service, and Appointment Service.

1. **The Contact Service Package**

This package uses memory data structures to support storing contacts and uses the Contact class to invoke data. This package has the Contact class and Contact Service Class.

The package of Contact Service includes the following data:

* Private String firstName (*for last name of contact)*
* Private String lastName *(for last name of contact)*
* Private String id *(for identifier of the contact)*
* Privte string phone *(for phone number)*
* Private String add (*for address)*

The Contact class requires the following data and respective limitations to invoke:

|  |  |
| --- | --- |
| Data | Limitations |
| String firstName | Cannot be *null*  and must be less than10 and have a max of 10 |
| String lastName | Cannot be *null* and must be less than 10 and have a max of 10 |
| String id | Cannot be *null* and must be less than 10 and have a max of 10 |
| String phone | Cannot be *null* and must be 10 |
| String add. | Cannot be *null* and must be less than 30 |

It also obtains the following methods:

* getFirstName() and returns a string called firstName.
* getLastName() and returns a string called lastName.
* getId() and returns a string called id.
* getPhone() and returns a string called phone.
* getAdd() and returns a string called add.

Contact Service includes the data:

* The private ArrayList contactDet (*arraylist of contact details*)
* private Hashtable contacts (hashtable id as key value as contact details)
* private String id

The Contact Service uses the object Contact and uses the Contact Class.

When invoking contactService class it requires no data types as arguments. It simply defines the contacts as a new hash table with string as a key and string arraylist as the value.

It contains the following methods:

* addContact (**contact** value)
* deleteContact(**String** id)
* updateContact(**String** id)

1. **The Task Service Package**

This package uses memory data structures to support storing tasks and uses the Task class to invoke data. This package has the Task class and Task Service Class.

The package of Task Service includes the following data:

* Private String name (*for name of Task)*
* Private String id *(for identifier of the task)*
* Private String desc *(for description of task)*

The Task class requires the following data and respective limitations to invoke:

|  |  |
| --- | --- |
| Data | Limitations |
| String name | Cannot be *null*  and must be less than 10 and have a max of 10 |
| String id | Cannot be *null* and must be less than 20 and have a max of 20 |
| String desc. | Cannot be *null* and must be less than 50 and have a max of 50 |

It also obtains the following methods:

* GetName() and returns a string called name.
* getId() and returns a string called id.
* geDesc() and returns a string called desc.

Task Service includes the data:

* The private ArrayList taskDet (*arraylist of contact details*)
* private Hashtable contacts (hashtable id as key value as task details)
* private String id

The Task Service uses the object Task and uses the Task Class.

When invoking taskService class it requires no data types as arguments. It simply defines the tasks as a new hash table with string as a key and string arraylist as the value.

It contains the following methods:

* addTask(**task** value)
* deleteTask(**String** id)
* updateTasks(**String** id)

1. **The Appointment Services Package**

This package uses memory data structures to support storing appointments and uses the Appointment class to invoke data. This package has the Appointment class and Appointment Service Class. It also uses Java Packages for Date. This is to obtain the current date.

The package of Appointment Service includes the following data:

* Private Date curr (*for current date).*
* Private Date date (*for new date).*
* Private String id *(for identifier of the appointment).*
* Private String desc *(for description of appointment)* .

The Appointment class requires the following data and respective limitations to invoke:

|  |  |
| --- | --- |
| Data | Limitations |
| String id | Cannot be *null*  and must be less than 10 and have a max of 10 |
| Date date | Cannot be *before* the current date. |
| String desc. | Cannot be *null* and must be less than 50 and have a max of 50 |

It also obtains the following methods:

* GetDate() and returns a Date called date.
* getId() and returns a string called id.
* geDesc() and returns a string called desc.
* setDate(**Date** date) and does not return anything but sets the data date to the date that was passed.
* setDesc(String date) and does not return anything but sets the data desc to the string of characters that were passed.

Appointment Service includes the data:

* The private ArrayList apptDet (*arraylist of contact details*)
* private Hashtable appts (hashtable id as key value as appointment details)
* private String id

The Appointment Service uses the object Appointment and uses the Appointment Class.

When invoking AppointmentService class it requires no data types as arguments. It simply defines the Appointment as a new hash table with string as a key and string arraylist as the value.

It contains the following methods:

* addAppointment (**Appointment** value)
* deleteAppointment (**String** id)
* updateAppointments (**String** id)

It can be assumed true that this mobile application has RESTful Application Programming Interface (API) . This mobile application must modernize operations and use the most current and effective software security. This RESTful API must comply with the set of architectural constraints defined in REST (Representational State Transfer). (A RESTful API is an application programming interface that conforms to the following constraints:

* Stateless client-server communication.
* Client-Server Architecture
* Cacheable data that streamlines client-server interactions
* Uniform interface
* Layered System (To ensure that availability, confidentiality, and integrity is implemented).

REST APIS are:

* Backwards compatible APIS
* Evolvable
* Scalable
* Securable
* Have a spectrum of stateless to stateful services

1. **Testing and JUNIT Tests**

Each package contains a specific test for both of their classes. During the SDLC of this mobile application, dynamic testing was performed. The goal for dynamic testing is to ensure that the product meets all the requirements- the non-functional, functional, and business requirements. Dynamic Testing validates the output of the executing code and can use both approaches of white-box testing and black-box testing. White-box testing was specifically used because the overall structure was known. The Junit test was specifically performed to ensure that the functionality of the methods that were invoked would be true if we passed incorrect data or correct data. For example, in the contact class the contact Test was written and the String for name, "*JaneDoeJaNEDOEEEE*" was passed for Contact(). This test was written to ensure that if the arguments were invalid the program would display it. Junit testing was specifically written to see if the program would react the right way if the given data was passed. However, it does not necessarily determine the functionality of the services like add Contact or update Contact. The same is true for the other tests written for Contact Service Test, Task Test, Task Service Test, Appointment Test, and Appointment Service Test. All of the test that were written utilizing JUNIT performed true. This tells that the functionality of the codes data acceptance and determining if it is invalid is accurate.

Other examples include the following:

In the taskServiceTest the following was written:

**void** testTask() {

task task = **new** task ("0123456789", "task1", "I am a description");

assertTrue(task.getId().equals("0123456789"));

assertTrue(task.getName().equals("task1"));

assertTrue(task.getDesc().equals("I am a description"));

When executing this block this would successfully assert True. It if it did not, than the IDE would include a sign saying it actually asserted false despite the written test saying it would be true.

In the appointment test the following was written:

@DisplayName("Date is Invalid")

**void** testDateInvalid() {

Date date = **new** Date(2004,10,12);

Date curr = **new** Date();

appointment appointment = **new** appointment ("012345678", date, "I am a description");

System.***out***.println("The field for appointment date: " + appointment.getDate() + " date is invalid. ");

assertTrue(appointment.getDate().before(curr));

When executing this block this would successfully identify that this data for appointment is an invalid. The date written was before the current date. If it did not, then the IDE would include a sign saying it asserted false despite the written test saying it would be true.

1. **Reflection**

Looking back at the testing approach that we took, it can be determined that it only really tested to see if the program would behave correctly when a given data was passed. The approach of writing a test immediately after writing the code is a similar approach to the professionals utilizing the ideology of Extreme Programming or XP. The test performed was a unit test. In the XP methodology unit test are automated test written by developers. They test a specific unit or class of the code. JUNIT is a framework for testing. Professionals following XP will use frameworks for unit testing. That is what was conducted during this project.

The techniques we did not use in this project is the other three levels of testing. There are four; these levels include the unit test, the integration test, system testing, and acceptance testing. I think that these tests are all important. We also need to test the overall functionality of the overall system and complete the unit testing.

I think that we must limit bias in coding. For example, assuming that this mobile application is going to be used by good people is bad. We must take the approach that some users may be malicious. To ensure the integrity and security of the code we must be sure to implement secure coding practices. This includes input sanitation and validation. The initial code does not implement this and passing raw data can be dangerous. For example, if the storage of data use SQL, then a failure to implement precautions such as input sanitation/validation and encapsulation will result in SQL Injections! Or Malicious users obtaining data they shouldn’t obtain. We must also be cautious about displaying a system error to the user interface. It is crucial to only implement stack trace errors to the servers on the developmental side. Leaking too much information for the user is dangerous because malicious user could use this data to their advantage. For example when applications use passwords and username as authentication it is good practice when a user enters an invalid password to display ,”invalid username/password combination”, rather than “invalid passwords.” This alerts malicious users that the username was correct, and the password was not. This is why it is important to not cut corners, and follow the secure coding practices. For example, NIST has standards set forth in the Cybersecurity Framework and provides a quick start guide.

References.

Davu, S. (2020, April 13). *The four levels of software testing*. Segue Technologies. Retrieved October 10, 2022, from https://www.seguetech.com/the-four-levels-of-software-testing/

Mahn, A., Topper, D., Quinn, S., &amp; Marron, J. (2022, January 4). Getting started with the NIST Cybersecurity Framework: A quick start guide. NIST. Retrieved October 10, 2022, from <https://www.nist.gov/publications/getting-started-nist-cybersecurity-framework-quick-start-guide>

Valacich, J. S., & George, J. F. (2018). Modern Systems Analysis and Design. Pearson.