Sterling Hospital

**Grade settings**: Maximum grade: 100  
**Run**: Yes **Evaluate**: Yes  
**Automatic grade**: Yes

[***Click here to download the code template***](https://cognizant.tekstac.com/pluginfile.php/74073/mod_vpl/intro/DoctorDetails.zip)

***Sterling Hospital***is a famous hospital in the city, they automated an application for manipulating details of the Doctors working in the hospital.

They have developed an application for taking various reports based on the Doctor. The details of the various functions supported by the system are provided in this case study.

You are required to write Junit test case and check the correctness of the application developed.

**Functional Requirements:**

The application has the below classes and methods implemented.

You are provided with a model class DoctorInfo

**Component Specification:**DoctorInfo**(Model Class)**

|  |  |  |
| --- | --- | --- |
| **Type(Class)** | **Attributes** | **Methods** |
| DoctorInfo | String doctorId  String doctorName  boolean isSurgeon  String specialization  String availableDays  int yearsOfExperience | Necessary getters and setters are provided.  A constructor is also provided. |

Here the specialization can take a value either “Cardiologist” or “Dentist” or “Neurologist” or “Gynecologist”.

Here the availableDays takes value like Mon:Wed:Fri, where days in the week are separated by :.

[Note: Values are case insensitive].

**Component Specification:**InvalidDoctorInfoException**(This class inherits the Exception Class)**

|  |  |
| --- | --- |
| **Type(Class)** | **Methods** |
| InvalidDoctorInfoException | Provided with a single argument constructor – InvalidDoctorInfoException (String message) |

You are also provided with a utility class DoctorUtility and the below business requirements are implemented in it for which JUnit test cases are to be written and tested.

The below are the requirements  implemented in the Utility class for which JUnit test cases are to be written and tested.

**Component Specification:**DoctorUtility **(Utility Class)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component Name** | **Type (Class)** | **Methods** | **Responsibilities** | **Exception** |
| Validate the event  type | DoctorUtility | public boolean validateSpecialization(String specialization) | Validate the Specialization.  If valid, return true , else this method should throw a user defined exception | Throw a user defined exception “InvalidDoctorInfoException”  if the Specialization is not “Cardiologist” or “Dentist” or “Neurologist”  or “Gynecologist” |
| View Event based on Event Id | DoctorUtility | public DoctorInfo viewDoctorById(List<DoctorInfo> doctorList, String doctorId) | This method should return the Doctor Info object with the Doctor Id passed as parameter from list of doctors, which is also passed as parameter.  If the doctorList is empty or if there is no doctor with the given Doctor Id it should throw a user defined exception | Throw a user defined exception “InvalidDoctorInfoException” if the  doctorList is empty or if  no event exists with the given Doctor Id. |
| View the list of Events for a given Event type | DoctorUtility | public List<DoctorInfo> viewDoctorsAsSurgeon(List<DoctorInfo> doctorList) | This method takes the doctorList as an argument. It should return the list of Doctors who are Surgeon. If the doctorList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidDoctorInfoException” if the  doctorList is empty |
| Count the Events based on date of registration | DoctorUtility | public Map<String,List<DoctorInfo>> viewDoctorsSpecializationWise(List<DoctorInfo> doctorList) | This method takes the doctorList as an argument. It should return the List DoctorInfo Objects based on their Specialization. This method returns a Map with key as Specialization and value as doctor details based on their Specialization. If the doctorList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidDoctorInfoException” if the doctorList is empty. |
| View the event details based on the date of event | DoctorUtility | public Map<String,String> viewDoctorsAvailability(List<DoctorInfo> doctorList, String day) | This method should return the Doctor Names  based on their availability. It takes the doctorList as argument and return a Map with key as Doctor name and value as Specialization based on their availability. If the doctorList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidDoctorInfoException” if the doctorList is empty. |
| Calculate amount based on the date of event | DoctorUtility | public int viewDoctorsByYearsOfExperience(List<DoctorInfo> doctorList, int yearsOfExperience) | This method takes the doctorList and the year of experience as arguments. It should count the doctors based on their experience and return the same. If the doctorList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidDoctorInfoException” if the doctorList is empty. |

You need to write Junit test for the DoctorUtilityclass.

**Testing Scenarios:**

You are provided with a class “DoctorUtilityTest” to do this testing.

**Note:**

To perform testing, the doctorList should contain objects of DoctorInfo.

To do this, in DoctorUtilityTest class you are provided with a setup method.  Use this method to populate the static variable doctorList in DoctorUtilityTest class.  That is, create few objects for DoctorInfo and populate the doctorList given in DoctorUtilityTest class with these objects and use that list to test the methods in DoctorUtility class that needs a doctorList to be passed as attribute.

The below are the test methods to be implemented in DoctorUtilityTest class.

|  |  |
| --- | --- |
| **Test Method** | **Scenarios / Responsibilities** |
| test11ValidateSpecializationWhenCardiologist | This method should test the validateSpecialization method when “Cardiologist” is passed as parameter |
| test12ValidateSpecializationWhenDentist | This method should test the validateSpecialization method when “Dentist” is passed as parameter |
| test13ValidateSpecializationWhenNeurologist | This method should test the validateSpecialization method when “Neurologist” is passed as parameter |
| test14ValidateSpecializationWhenGynecologist | This method should test the validateSpecialization method when “Gynecologist” is passed as parameter |
| test15ValidateSpecializationWhenInvalid | This method should test the validateSpecialization method when invalid value is passed as parameter  validateSpecialization is expected  to throw InvalidDoctorInfoException when type is invalid.  Write JUnit to test for the exception thrown either by using appropriate annotation or by using try catch block. |
| test16ViewDoctorByIdWhenValid | This method should test the correctness of viewDoctorByIdWhenValid method.  Perform testing for the correctness of the value returned. |
| test17ViewDoctorByIdWhenInvalid | This method should test the correctness of viewDoctorById method for a non existing Doctor Id.  Perform testing for the correctness of the value returned.  viewDoctorById method is expected  to throw InvalidDoctorInfoException when Event Id does not exist.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test18ViewDoctorsAsSurgeon | This method should test the correctness of viewDoctorsAsSurgeon method.  Perform testing for the correctness of the value returned. |
| test19ViewDoctorsSpecializationWise | This method should test the correctness of viewDoctorsSpecializationWise method.  Perform testing for the correctness of the value returned. |
| test20ViewDoctorsAvailability | This method should test the correctness of viewDoctorsAvailability method.  Perform testing for the correctness of the value returned. |
| test21ViewDoctorsByYearsOfExperience | This method should test the correctness of viewDoctorsByYearsOfExperience method.  Perform testing for the correctness of the value returned. |
| test22ViewDoctorsAsSurgeonForEmptyList | This method should test the correctness of viewDoctorsAsSurgeon method for an empty doctorList.  viewDoctorsAsSurgeon method is expected  to throw InvalidDoctorInfoException when doctorList is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test23ViewDoctorsSpecializationWiseForEmptyList | This method should test the correctness of viewDoctorsSpecializationWise method for an empty doctorList.  viewDoctorsSpecializationWise method is expected  to throw InvalidDoctorInfoException when doctorList is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test24ViewDoctorsAvailabilityForEmptyList | This method should test the correctness of viewDoctorsAvailability method for an empty doctorList.  viewDoctorsAvailability method is expected  to throw InvalidDoctorInfoException when doctorList is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test25ViewDoctorsByYearsOfExperienceForEmptyList | This method should test the correctness of viewDoctorsByYearsOfExperience method for an empty doctorList.  viewDoctorsByYearsOfExperience method is expected  to throw InvalidDoctorInfoException when doctorList is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |

Implement the test methods and provide the needed annotation to all the methods in DoctorUtilityTest class.

Also this class is provided with the annotation, so that the test methods are executed in ascending order of the test method names.

You are provided with a Main class with the main method to check the correctness of the test methods written in DoctorUtilityTest class.

Having completed writing the test methods, uncomment the code in Main class and execute the main method.