Euro Clean

**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/69142/mod_vpl/intro/Euro%20Clean.zip)

***Euro Clean***is a company which manufactures cleaning products, we can order their products by online and by direct ways. They had automated the ordering system, using that application, you can maintain information of the orders placed for their products.

Vivek has developed an application for the above purpose. 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 OrderDetails

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

|  |  |  |
| --- | --- | --- |
| **Type (Class)** | **Attributes** | **Methods** |
| OrderDetails | String orderId  String orderType  String productId  Date dateOfOrder  Date dateOfdelivery  double totalAmount | Necessary getters,setters are provided  A Constructor is also provided |

·         Here, orderType can take a value either “Direct” or “Online” [Note: Values are case insensitive]

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

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

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

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component Name** | **Type (Class)** | **Methods** | **Responsibilities** | **Exception** |
| Validate the order Id | ProductOrder | public boolean validateOrderId(String orderId) | Validate the orderId.  If valid, return true, else this method should throw a user defined exception | Throw a user defined exception “InvalidOrderDetailsException” if the orderId does not contain 4 digits followed by the String “Order”. |
| View Order details based on Order Id | ProductOrder | public OrderDetails viewOrderDetailsByOrderId(List<OrderDetails> orderList, String orderId) | This method should return the Order details object with the order Id passed as parameter from list of orders, which is also passed as parameter.  If the orderList is empty or if there is no order with the given order Id it should throw a user defined exception | Throw a user defined exception “InvalidOrderDetailsException” if the orderList is empty or if no order exists with the given orderId. |
| View the list of Orders based on a specific order type | ProductOrder | public List<OrderDetails> viewOrderDetailsByOrderType(List<OrderDetails> orderList, String type) | This method takes the orderList and order type as arguments. It should return the list of orders for the given type. If the orderList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidOrderDetailsException” if the orderList is empty |
| Count the number of orders on a particular date | ProductOrder | public int countOrdersByDateOfOrder(List<OrderDetails> orderList, Date date) | This method takes the orderList as argument along with the date. It should return the count of orders on that particular date.  If the orderList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidOrderDetailsException” if the orderList is empty. |
| View the list of orders -  Delivery date wise | ProductOrder | public Map<Date, List<OrderDetails>> viewOrdersByDateOfDelivery (List<OrderDetails> orderList) | This method should return the list of Orders  - delivery datewise.  It takes the orderList as argument and returns a Map with key as Delivery date and value as list of orders for that delivery date. If the orderList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidOrderDetailsException” if the orderList is empty. |
| Calculate the total amount based on delivery date | ProductOrder | public double calculateAmountByDateOfDelivery(List<OrderDetails> orderList, Date date) | This method takes the orderList as argument along with the date. It should return the total amount received for the orders on that particular date.  If the orderList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidOrderDetailsException” if the orderList is empty. |

You need to write Junit test for the ProductOrderclass.

**Testing Scenarios:**

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

**Note:**

To perform testing, the orderList should contain objects of OrderDetails.

To do this, in ProductOrderTest class you are provided with a setup method.  Use this method to populate the static variable orderList in ProductOrderTest class.  That is, create few objects for OrderDetails and populate the orderList given in ProductOrderTest class with these objects and use that list to test the methods in ProductOrderclass that needs a OrderDetails list to be passed as parameter.

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

|  |  |
| --- | --- |
| **Test Method** | **Scenarios / Responsibilities** |
| test11ValidateOrderIdForValidOrderId | This method should test the ValidateOrderId method when a valid order Id is passed as parameter. |
| test12ValidateOrderIdForInvalidOrderId | This method should test the validateOrderId method when an invalid order Id is passed as parameter.  validateOrderId is expected to throw InvalidOrderDetailsExceptionwhen order Id is invalid.  Write JUnit to test for the exception thrown either by using appropriate annotation or by using try catch block. |
| test13ValidateOrderIdWithoutStringOrder | This method should test the validateOrderId method when invalid order Id without the String “Order” is passed as parameter  validateOrderId is expected to throw InvalidOrderDetailsExceptionwhen order Id is invalid.  Write JUnit to test for the exception thrown either by using appropriate annotation or by using try catch block. |
| test14ValidateOrderIdWithoutDigits | This method should test the correctness of validateOrderId method when invalid order Id without digits is passed as parameter  validateOrderId is expected to throw InvalidOrderDetailsExceptionwhen order Id is invalid.  Write JUnit to test for the exception thrown either by using appropriate annotation or by using try catch block. |
| test15ViewOrderDetailsByOrderId | This method should test the correctness of viewOrderDetailsByOrderId method for an existing order Id.  Perform testing for the correctness of the value returned. |
| test16ViewOrderDetailsByOrderId | This method should test the correctness of viewOrderDetailsByOrderId method for a non-existing order Id.  Perform testing for the correctness of the value returned.  viewOrderDetailsByOrderId method is expected to throw InvalidOrderDetailsExceptionwhen order Id does not exist.  Write JUnit to test for the exception thrown either by using appropriate annotation or by using try catch block |
| test17ViewOrderDetailsByOrderType | This method should test the correctness of viewOrderDetailsByOrderType.  Perform testing for the correctness of the value returned. |
| test18ViewOrderDetailsByOrderTypeForEmptyList | This method should test the correctness of viewOrderDetailsByOrderType method.  viewOrderDetailsByOrderType method is expected to throw InvalidOrderDetailsExceptionwhen orderList is empty.  Write JUnit to test for the exception thrown either by using appropriate annotation or by using try catch block |
| test19CountOrdersByDateOfOrder | This method should test the correctness of countOrdersByDateOfOrder method for an empty orderList.  Perform testing for the correctness of the value returned. |
| test20CountOrdersByDateOfOrderForEmptyList | This method should test the correctness of countOrdersByDateOfOrder method.  countOrdersByDateOfOrder method is expected to throw InvalidOrderDetailsExceptionwhen orderList is empty.  Write JUnit to test for the exception thrown either by using appropriate annotation or by using try catch block |
| test21ViewOrdersByDateOfDelivery | This method should test the correctness of viewOrdersByDateOfDelivery method.  Perform testing for the correctness of the value returned. |
| test22ViewOrdersByDateOfDeliveryForEmptyList | This method should test the correctness of viewOrdersByDateOfDelivery method for an empty list.  viewOrdersByDateOfDelivery method is expected to throw InvalidOrderDetailsExceptionwhen orderList is empty.  Write JUnit to test for the exception thrown either by using appropriate annotation or by using try catch block |
| test23CalculateAmountByDateOfDelivery | This method should test the correctness of calculateAmountByDateOfDelivery method.  Perform testing for the correctness of the value returned. |
| test24CalculateAmountByDateOfDeliveryForEmptyList | This method should test the correctness of calculateAmountByDateOfDelivery method for an empty list.  calculateAmountByDateOfDelivery method is expected to throw InvalidOrderDetailsExceptionwhen orderList 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 ProductOrderTest 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 ProductOrderTest class.

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