Trainer Attendance Marking System

**Grade settings**: Maximum grade: 100  
**Disable external file upload, paste and drop external content**: Yes  
**Run**: Yes **Evaluate**: Yes  
**Automatic grade**: Yes

**Important Instructions:**

●       Please read the document thoroughly before you code.

●       Import the given skeleton code into your Eclipse.

●       Do not change the Skeleton code or the package structure, method names, variable names, return types, exception clauses, access specifiers etc.

●       You can create any number of private methods inside the given class.

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

●       Having completed writing the test methods, execute the main method and identify the result or run the test class using JUnit.

**Assessment Coverage:**

·         JUnit annotations & **exception handling** rules

[***Click here to download the code template***](https://cognizant.tekstac.com/pluginfile.php/54384/mod_vpl/intro/TrainerAttendanceMarkingSystem.zip?time=1614861594075)

***Trainer Attendance Marking System*** is an automated Attendance Marking system. Using the application, the Admin wants to take reports based on the Employee ID for the Particular Month, Generate report those are not marked attendance on particular Date and Employee can mark their attendance.

They have 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.

**Skeleton:**

**TrainerAttendanceMarkingSystemSkeleton**

Functional Requirements:

The application has the below classes and methods implemented.

You are provided with a model class DailyAttendance

Component Specification : DailyAttendance   (Model Class)

|  |  |  |
| --- | --- | --- |
| Type(Class) | Attributes | Methods |
| DailyAttendance | int employeeID,  String subjectExpert,  String topicsHandled,  String asPerSchedule,  LocalDateTime inDatetime,  LocalDateTime outDatetime | Necessary getters and setters are provided.  A six argument constructor in the order EmployeeID, SubjectExpert, TopicsHandled, AsPerSchedule, InDatetime, OutDatetime is provided.  Also the equals and toString methods are overridden. |

**Here the**

✔      subjectExpert -> subject which the trainer is handled on the Particular Date.

✔      topicsHandled -> what topics Trainer is going to delivered on that day.

✔      asPerSchedule ->  Mark “Yes” if topics goes on as per schedule and “No” if topics are delayed

✔      inDatetime and outDatetime is the time which he/she enter in office it can get through system time.

**Component Specification:  InvalidAttendanceMarkingException**

**(This class inherits the Exception Class)**

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

You are also provided with a utility class **MarkAttendance** with business methods.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Component Name** | **Type(Class)** | **Attributes** | **Methods** |
| Mark a Attendance | MarkAttendance | List<DailyAttendance> attendanceList | Getter and setter for the attendanceList  are provided. |

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

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component Name** | **Type(Class)** | **Methods** | **Responsibilities** | **Exception** |
| Validating the schedule | MarkAttendance | public boolean validateSchedule  (String asPerSchedule) | Validate the asPerSchedule. If valid return true else this method should throw a user defined  exception | Throw a user defined exception “InvalidAttendanceMarkingException”   if the asPerSchedule is invalid with a message  “Invalid AsPerSchedule”.  asPerSchedule is valid if its value is “Yes” or “No” (case insensitive) |
| Mark a Attendance | MarkAttendance | public boolean addAttendance(  Int employeeID,  String subjectExpert,  String topicsHandled,  String asPerSchedule,  LocalDateTime inDateTime,  LocalDateTime outDateTime ) | This method should validate the asPerSchedule. If asPerSchedule is valid add to the attendanceList.Else throw a user defined exception. | If asPerSchedule is invalid, throw a user defined exception “InvalidAttendanceMarkingException” if the Attendance is not added. |
| View EmployeeAttendance based on EmployeeID. | MarkAttendance | public DailyAttendance  viewEmployeeAttendanceById  (int employeeID) | This method should retrieve the EmployeeAttendance that holds the employeeID passed as parameter. If DailyAttendance   object exists return that object, Else throw a user defined exception. | Throw a user defined exception “InvalidAttendanceMarkingException”   if a EmployeeID does not exist with a message  “Invalid Employee id” |
| View Attendance Details based on InDatetime and OutDateTime | MarkAttendance | public List<DailyAttendance>   viewAttendanceByDate( LocalDateTime inDatetime, LocalDateTime  outDateTime) | This method takes the InDatetime and OutDateTime as argument. It should return the list of Employee who has marked attendance on particular date. If the list is empty then it will throw a user defined exception | “InvalidAttendanceMarkingException”   if a  list is empty  with a message  “No Matching Found” |

|  |  |  |  |
| --- | --- | --- | --- |
| **Component Name** | **Type(Class)** | **Methods** | **Responsibilities** |
| View Employee based on  SubjectExpert wise | MarkAttendance | public Map<String, List < DailyAttendance >>    viewSubjectExpertWise() | This method should return the list of DailyAttendance for each SubjectExpert.  To do this, iterate the dailyAttendanceList and return a Map that holds subjectExpert as key and the list of DailyAttendance objects pertaining to that subjectExpert as values. |

You need to write Junit test for the MarkAttendance class.

Testing Scenarios:

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

**Note:**

✔      To perform testing the attendanceList should contain objects of DailyAttendance.

✔      To do this, in MarkAttendanceTest  class you are provided with a setup method.  Use this method to initialize the  attendanceList   attribute in MarkAttendance class.

✔      Create few objects for   DailyAttendance, populate a list with these objects and set the attendanceList to this list using the set DailyAttendanceList method in MarkAttendance class.

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

|  |  |
| --- | --- |
| **Test Method** | **Scenarios / Responsibilities** |
| test11ValidateAsPerScheduleWhenYes | This method should test the validateSchedule method when a valid Schedule “Yes” is passed as parameter to this method. |
| test12 ValidateAsPerScheduleWhenNo | This method should test the validateSchedule  method when a valid Schedule “No” is passed as parameter to this method. |
| test13ValidateAsPerScheduleWhenInvalid | This method should test the validateSchedule   method when an invalid Schedule  is passed to this method.  validateSchedule method is expected  to throw InvalidAttendanceMarkingExceptionwhen Schedule is invalid.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block. |
| test14AddAttendanceForValidSchedule | This method should test the addAttendance method when  valid Schedule is provided for the DailyAttendance .  Test for the success scenario of addition of DailyAttendance object into the list. |
| test15 AddAttendanceForInValidSchedule | This method should test the addAttendance  method when  invalid Schedule is provided for the DailyAttendance. In this case, addAttendance method is expected to throw InvalidAttendanceMarkingException.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block. |
| test16 viewEmployeeAttendanceById ForValidId | This method should test the viewEmployeeAttendanceById method when a EmployeeID is passed as parameter exists in the attendanceList. |
| test17 viewEmployeeAttendanceById ForInValidId | This method should test  the viewEmployeeAttendanceById method when a EmployeeID is passed as parameter does not exist in the attendanceList.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block. |
| test18 viewAttendanceByDateForValid | This method should test the correctness of   viewAttendanceByDate method.  Perform testing for the correctness of the list object returned. |
| test19 viewAttendanceByDateForInValid | This method should test the viewAttendanceByDate method  when inDateTime and outDateTime is not exist in the attendance list  .  Write Junit to test for the exception thrown either by using appropriate annotation or by using try catch block |
| test20 viewSubjectExpertWise | This method should test the correctness of viewSubjectExpertWise method.  Perform testing for the correctness of the map object returned. |

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

Also provide the required 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 MarkAttendanceTest class.

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

Automatic evaluation[[-]](javascript:void(0);)

**Proposed grade: 100.0 / 100**  
**Result Description**  
[[-]](javascript:void(0);)**Grading and Feedback**

*Writing JUnit for a Utility class - 60.0 / 60.0(Success)*

*Test Coverage - 40 / 40.0(Success)*

Test Case Passed

TrainerAttendanceMarkingSystem/src/com/cts/exception/InvalidAttendanceMarkingException.java

1 *package* com.cts.exception;

2

3 *public* *class* InvalidAttendanceMarkingException *extends* Exception

4 {

5 *public* InvalidAttendanceMarkingException(String message)

6 {

7 *super*(message);

8 }

9 }

10

TrainerAttendanceMarkingSystem/src/com/cts/model/DailyAttendance.java

1 *package* com.cts.model;

2

3 *import* java.time.LocalDateTime;

4

5 *public* *class* DailyAttendance {

6

7 *private* *int* employeeID;

8 *private* String subjectExpert;

9 *private* String topicsHandled;

10 *private* String asPerSchedule;

11 *private* LocalDateTime inDatetime;

12 *private* LocalDateTime outDatetime;

13

14 *public* DailyAttendance()

15 {

16

17 }

18

19 *public* DailyAttendance(*int* employeeID, String subjectExpert, String topicsHandled, String asPerSchedule,

20 LocalDateTime inDatetime, LocalDateTime outDatetime) {

21

22 *this*.employeeID = employeeID;

23 *this*.subjectExpert = subjectExpert;

24 *this*.topicsHandled = topicsHandled;

25 *this*.asPerSchedule = asPerSchedule;

26 *this*.inDatetime = inDatetime;

27 *this*.outDatetime = outDatetime;

28 }

29

30

31 *public* *int* getEmployeeID() {

32 *return* employeeID;

33 }

34 *public* *void* setEmployeeID(*int* employeeID) {

35 *this*.employeeID = employeeID;

36 }

37 *public* String getSubjectExpert() {

38 *return* subjectExpert;

39 }

40 *public* *void* setSubjectExpert(String subjectExpert) {

41 *this*.subjectExpert = subjectExpert;

42 }

43 *public* String getTopicsHandled() {

44 *return* topicsHandled;

45 }

46 *public* *void* setTopicsHandled(String topicsHandled) {

47 *this*.topicsHandled = topicsHandled;

48 }

49 *public* String getAsPerSchedule() {

50 *return* asPerSchedule;

51 }

52 *public* *void* setAsPerSchedule(String asPerSchedule) {

53 *this*.asPerSchedule = asPerSchedule;

54 }

55 *public* LocalDateTime getInDatetime() {

56 *return* inDatetime;

57 }

58 *public* *void* setInDatetime(LocalDateTime inDatetime) {

59 *this*.inDatetime = inDatetime;

60 }

61 *public* LocalDateTime getOutDatetime() {

62 *return* outDatetime;

63 }

64 *public* *void* setOutDatetime(LocalDateTime outDatetime) {

65 *this*.outDatetime = outDatetime;

66 }

67

68

69 @Override

70 *public* String toString() {

71 *return* "DailyAttendance [employeeID=" + employeeID + ", subjectExpert=" + subjectExpert + ", topicsHandled="

72 + topicsHandled + ", asPerSchedule=" + asPerSchedule + ", inDatetime=" + inDatetime + ", outDatetime="

73 + outDatetime + "]";

74 }

75

76

77

78 @Override

79 *public* *int* hashCode() {

80 *final* *int* prime = 31;

81 *int* result = 1;

82 result = prime \* result + ((asPerSchedule == *null*) ? 0 : asPerSchedule.hashCode());

83 result = prime \* result + employeeID;

84 result = prime \* result + ((inDatetime == *null*) ? 0 : inDatetime.hashCode());

85 result = prime \* result + ((outDatetime == *null*) ? 0 : outDatetime.hashCode());

86 result = prime \* result + ((subjectExpert == *null*) ? 0 : subjectExpert.hashCode());

87 result = prime \* result + ((topicsHandled == *null*) ? 0 : topicsHandled.hashCode());

88 *return* result;

89 }

90

91

92

93 @Override

94 *public* *boolean* equals(Object obj) {

95 *if* (*this* == obj)

96 *return* *true*;

97 *if* (obj == *null*)

98 *return* *false*;

99 *if* (getClass() != obj.getClass())

100 *return* *false*;

101 DailyAttendance other = (DailyAttendance) obj;

102 *if* (asPerSchedule == *null*) {

103 *if* (other.asPerSchedule != *null*)

104 *return* *false*;

105 } *else* *if* (!asPerSchedule.equals(other.asPerSchedule))

106 *return* *false*;

107 *if* (employeeID != other.employeeID)

108 *return* *false*;

109 *if* (inDatetime == *null*) {

110 *if* (other.inDatetime != *null*)

111 *return* *false*;

112 } *else* *if* (!inDatetime.equals(other.inDatetime))

113 *return* *false*;

114 *if* (outDatetime == *null*) {

115 *if* (other.outDatetime != *null*)

116 *return* *false*;

117 } *else* *if* (!outDatetime.equals(other.outDatetime))

118 *return* *false*;

119 *if* (subjectExpert == *null*) {

120 *if* (other.subjectExpert != *null*)

121 *return* *false*;

122 } *else* *if* (!subjectExpert.equals(other.subjectExpert))

123 *return* *false*;

124 *if* (topicsHandled == *null*) {

125 *if* (other.topicsHandled != *null*)

126 *return* *false*;

127 } *else* *if* (!topicsHandled.equals(other.topicsHandled))

128 *return* *false*;

129 *return* *true*;

130 }

131

132

133

134 }

135

TrainerAttendanceMarkingSystem/src/com/cts/skeleton/SkeletonValidator.java

1 *package* com.cts.skeleton;

2

3 *import* java.lang.reflect.Method;

4 *import* java.util.logging.Level;

5 *import* java.util.logging.Logger;

6

7 *public* *class* SkeletonValidator {

8

9 *public* SkeletonValidator(){

10

11 validateClassName("com.cts.exception.InvalidAttendanceMarkingException");

12 validateClassName("com.cts.model.DailyAttendance");

13 validateMethodSignature("toString:java.lang.String,hashCode:int,equals:boolean","com.cts.model.DailyAttendance");

14

15 validateClassName("com.cts.utility.MarkAttendance");

16 validateMethodSignature("validateSchedule:boolean,addAttendance:boolean,viewEmployeeAttendanceById:com.cts.model.DailyAttendance,viewAttendanceByDate:java.util.List,viewSubjectExpertWise:java.util.Map,viewNotMarkedAttendanceParticaularDate:java.util.Map","com.cts.utility.MarkAttendance");

17

18 validateClassName("com.cts.test.MarkAttendanceTest");

19 validateMethodSignature("test11ValidateAsPerScheduleWhenYes:void,test12ValidateAsPerScheduleWhenNo:void,test13ValidateAsPerScheduleWhenInvalid:void,test14AddAttendanceForValidSchedule:void,test15AddAttendanceForInValidSchedule:void,test16viewEmployeeAttendanceByIdForValidId:void,test17viewEmployeeAttendanceByIdForInValidId:void,test18viewAttendanceByDate:void,test19viewNotMarkedAttendanceParticularDate:void,test20viewSubjectExpertWise:void","com.cts.test.MarkAttendanceTest");

20

21

22 }

23

24

25 *private* *static* *final* Logger LOG = Logger.getLogger("SkeletonValidator");

26

27 *protected* *final* *boolean* validateClassName(String className) {

28

29 *boolean* iscorrect = *false*;

30 *try* {

31 Class.forName(className);

32 iscorrect = *true*;

33 LOG.info("Class Name " + className + " is correct");

34

35 } *catch* (ClassNotFoundException e) {

36 LOG.log(Level.SEVERE, "You have changed either the " + "class name/package. Use the correct package "

37 + "and class name as provided in the skeleton");

38

39 } *catch* (Exception e) {

40 LOG.log(Level.SEVERE,

41 "There is an error in validating the " + "Class Name. Please manually verify that the "

42 + "Class name is same as skeleton before uploading");

43 }

44 *return* iscorrect;

45

46 }

47

48 *protected* *final* *void* validateMethodSignature(String methodWithExcptn, String className) {

49 Class cls = *null*;

50 *try* {

51

52 String[] actualmethods = methodWithExcptn.split(",");

53 *boolean* errorFlag = *false*;

54 String[] methodSignature;

55 String methodName = *null*;

56 String returnType = *null*;

57

58 *for* (String singleMethod : actualmethods) {

59 *boolean* foundMethod = *false*;

60 methodSignature = singleMethod.split(":");

61

62 methodName = methodSignature[0];

63 returnType = methodSignature[1];

64 cls = Class.forName(className);

65 Method[] methods = cls.getMethods();

66 *for* (Method findMethod : methods) {

67 *if* (methodName.equals(findMethod.getName())) {

68 foundMethod = *true*;

69 *if* (!(findMethod.getReturnType().getName().equals(returnType))) {

70 errorFlag = *true*;

71 LOG.log(Level.SEVERE, " You have changed the " + "return type in '" + methodName

72 + "' method. Please stick to the " + "skeleton provided");

73

74 } *else* {

75 LOG.info("Method signature of " + methodName + " is valid");

76 }

77

78 }

79 }

80 *if* (!foundMethod) {

81 errorFlag = *true*;

82 LOG.log(Level.SEVERE, " Unable to find the given public method " + methodName

83 + ". Do not change the " + "given public method name. " + "Verify it with the skeleton");

84 }

85

86 }

87 *if* (!errorFlag) {

88 LOG.info("Method signature is valid");

89 }

90

91 } *catch* (Exception e) {

92 LOG.log(Level.SEVERE,

93 " There is an error in validating the " + "method structure. Please manually verify that the "

94 + "Method signature is same as the skeleton before uploading");

95 }

96 }

97 }

98

TrainerAttendanceMarkingSystem/src/com/cts/test/Main.java

1 *package* com.cts.test;

2

3 *import* org.junit.runner.JUnitCore;

4 *import* org.junit.runner.Result;

5

6 *import* com.cts.skeleton.SkeletonValidator;

7

8 *public* *class* Main

9 {

10 *public* *static* *void* main(String[] args) {

11 *new* SkeletonValidator();

12 Result result = JUnitCore.runClasses(MarkAttendanceTest.*class*);

13 *if*(result.getFailureCount() == 0) {

14 System.out.println("All Test cases Cleared");

15 }*else* {

16 System.out.println("One or more test case(s) failed");

17 System.out.println("===============================");

18 result.getFailures().forEach(x -> System.out.println(x.getMessage()));

19 }

20 }

21 }

22

TrainerAttendanceMarkingSystem/src/com/cts/test/MarkAttendanceTest.java

1 *package* com.cts.test;

2

3 *import* java.util.List;

4 *import* java.util.Map;

5 *import* java.time.LocalDateTime;

6 *import* java.util.ArrayList;

7 *import* java.util.\*;

8 *import* org.junit.After;

9 *import* org.junit.Before;

10 *import* org.junit.Test;

11 *import* org.junit.Rule;

12 *import* org.junit.rules.ExpectedException;

13 *import* *static* org.junit.Assert.\*;

14

15 *import* com.cts.model.DailyAttendance;

16 *import* com.cts.utility.MarkAttendance;

17 *import* com.cts.exception.InvalidAttendanceMarkingException;

18

19 *public* *class* MarkAttendanceTest {

20

21 @Rule

22 *public* ExpectedException exceptionRule = ExpectedException.none();

23 *static* MarkAttendance markAttendanceObj;

24 List<DailyAttendance> attendanceList=*new* ArrayList<>();

25 *static* DailyAttendance d1;

26 *static* DailyAttendance d2;

27 *static* DailyAttendance d3;

28 *static* DailyAttendance d4;

29

30 @Before

31 *public* *void* setUp() {

32 markAttendanceObj = *new* MarkAttendance();

33

34 // code here

35 //Create few objects of DailyAttendance and add those objects to attendanceList

36 //Set this list to the attendanceList in MarkAttendance class using setAttendanceList method

37 d1 = *new* DailyAttendance(101,"nisha","java","yes",LocalDateTime.parse("2018-12-30T09:30"),LocalDateTime.parse("2018-12-30T12:30"));

38 d2 = *new* DailyAttendance(102,"isha","spring","no",LocalDateTime.parse("2018-12-30T12:30"),LocalDateTime.parse("2018-12-30T15:00"));

39 d3 = *new* DailyAttendance(102,"isha","java","no",LocalDateTime.parse("2019-12-31T09:30"),LocalDateTime.parse("2019-12-31T12:30"));

40 d4 = *new* DailyAttendance(104,"disha","junit","yes",LocalDateTime.parse("2018-02-03T12:30"),LocalDateTime.parse("2018-02-02T15:00"));

41

42 attendanceList.add(d1);

43 attendanceList.add(d2);

44 attendanceList.add(d3);

45 attendanceList.add(d4);

46 markAttendanceObj.setAttendanceList(attendanceList);

47

48 }

49

50 @After

51 *public* *void* tearDown(){

52

53 //code here

54 }

55

56 // test the validateSchedule method when a valid Schedule Yes is passed as parameter to this method.

57 @Test

58 *public* *void* test11ValidateAsPerScheduleWhenYes()throws InvalidAttendanceMarkingException

59 {

60

61 //code here

62 assertTrue(markAttendanceObj.validateSchedule("yes"));

63 }

64 //test the validateSchedule method when a valid Schedule No is passed as parameter to this method.

65 @Test

66 *public* *void* test12ValidateAsPerScheduleWhenNo()throws InvalidAttendanceMarkingException

67 {

68

69 //code here

70 assertTrue(markAttendanceObj.validateSchedule("no"));

71 }

72

73 //test the validateSchedule method when an invalid Schedule is passed to this method.

74 @Test

75 *public* *void* test13ValidateAsPerScheduleWhenInvalid()throws InvalidAttendanceMarkingException

76 {

77

78 //code here

79 exceptionRule.expect(InvalidAttendanceMarkingException.*class*);

80 exceptionRule.expectMessage("Invalid AsPerSchedule");

81 markAttendanceObj.validateSchedule("y");

82 }

83

84 //test the addAttendance method when valid Schedule is provided for the DailyAttendance success

85 @Test

86 *public* *void* test14AddAttendanceForValidSchedule() throws InvalidAttendanceMarkingException

87 {

88

89 //code here

90 assertTrue(markAttendanceObj.addAttendance(104,"disha","junit","yes",LocalDateTime.parse("2018-02-04T09:30"),LocalDateTime.parse("2018-02-03T12:00")));

91 }

92

93 // test the addAttendance method when invalid Schedule is provided for the DailyAttendance. In this case, addAttendance method is expected to throw InvalidAttendanceMarkingException.

94 @Test

95 *public* *void* test15AddAttendanceForInValidSchedule() throws InvalidAttendanceMarkingException

96 {

97

98

99 //code here

100 exceptionRule.expect(InvalidAttendanceMarkingException.*class*);

101 exceptionRule.expectMessage("Invalid AsPerSchedule");

102 markAttendanceObj.addAttendance(105,"disha","junit","ye",LocalDateTime.parse("2018-02-03T09:30"),LocalDateTime.parse("2018-02-03T19:30"));

103

104 }

105

106 //test the viewEmployeeAttendanceById method when a EmployeeID is passed as parameter exists in the attendanceList.

107 @Test

108 *public* *void* test16viewEmployeeAttendanceByIdForValidId() throws InvalidAttendanceMarkingException

109 {

110 //code here

111 assertEquals(d1,markAttendanceObj.viewEmployeeAttendanceById(101));

112 }

113

114 //test the viewEmployeeAttendanceById method when a EmployeeID is passed as parameter does not exist in the attendanceList.

115 @Test

116 *public* *void* test17viewEmployeeAttendanceByIdForInValidId() throws InvalidAttendanceMarkingException

117 {

118 //code here

119 exceptionRule.expect(InvalidAttendanceMarkingException.*class*);

120 exceptionRule.expectMessage("Invalid Employee id");

121 markAttendanceObj.viewEmployeeAttendanceById(110);

122 }

123

124 // test the correctness of the viewAttendanceByDate method. Perform testing for the correctness of the list returned.

125 @Test

126 *public* *void* test18viewAttendanceByDate() throws InvalidAttendanceMarkingException

127 {

128 //code here

129 List <DailyAttendance> d = *new* ArrayList <DailyAttendance>();

130 d.add(d1);

131 assertEquals(d,markAttendanceObj.viewAttendanceByDate(LocalDateTime.parse("2018-12-30T09:30"),LocalDateTime.parse("2018-12-30T12:30")));

132 }

133

134 // test the correctness of the viewAttendanceByDate method. Perform testing for the not existing date and handle the exception

135 @Test

136 *public* *void* test19viewNotMarkedAttendanceParticularDate() throws InvalidAttendanceMarkingException

137 {

138 //code here

139 exceptionRule.expect(InvalidAttendanceMarkingException.*class*);

140 exceptionRule.expectMessage("No Matching Found");

141 markAttendanceObj.viewAttendanceByDate(LocalDateTime.parse("2018-12-30T09:30"),LocalDateTime.parse("2018-12-30T15:30"));

142 }

143

144

145 // test the correctness of viewSubjectExpertWise method. Perform testing for the correctness of the map returned.

146 @Test

147 *public* *void* test20viewSubjectExpertWise()

148 {

149 //code here

150 Map <String, List<DailyAttendance>> myMaps = *new* HashMap<String,List<DailyAttendance>>();

151 List<DailyAttendance> subexp1 = *new* ArrayList<DailyAttendance>();

152 List<DailyAttendance> subexp2 = *new* ArrayList<DailyAttendance>();

153 List<DailyAttendance> subexp3 = *new* ArrayList<DailyAttendance>();

154 subexp1.add(d1);

155 subexp2.add(d2);

156 subexp2.add(d3);

157 subexp3.add(d4);

158 myMaps.put("nisha",subexp1);

159 myMaps.put("isha",subexp2);

160 myMaps.put("disha",subexp3);

161 assertEquals(myMaps,markAttendanceObj.viewSubjectExpertWise());

162 }

163

164 }

165

TrainerAttendanceMarkingSystem/src/com/cts/utility/MarkAttendance.java

1 *package* com.cts.utility;

2

3 *import* java.text.SimpleDateFormat;

4 *import* java.time.LocalDateTime;

5 *import* java.util.ArrayList;

6 *import* java.util.HashMap;

7 *import* java.util.Iterator;

8 *import* java.util.LinkedList;

9 *import* java.util.List;

10 *import* java.util.Map;

11

12 *import* com.cts.exception.InvalidAttendanceMarkingException;

13 *import* com.cts.model.DailyAttendance;

14

15 *public* *class* MarkAttendance {

16

17 *private* List<DailyAttendance> attendanceList;

18

19 *public* MarkAttendance() {

20 attendanceList = *new* ArrayList<DailyAttendance>();

21 }

22

23 *public* List<DailyAttendance> getAttendanceList() {

24 *return* attendanceList;

25 }

26

27 *public* *void* setAttendanceList(List<DailyAttendance> attendanceList) {

28 *this*.attendanceList = attendanceList;

29 }

30

31 *public* *boolean* validateSchedule(String asPerSchedule) throws InvalidAttendanceMarkingException {

32 *boolean* isValid = *false*;

33 *if* (asPerSchedule.equalsIgnoreCase("yes") || asPerSchedule.equalsIgnoreCase("no")) {

34 isValid = *true*;

35 } *else* {

36 *throw* *new* InvalidAttendanceMarkingException("Invalid AsPerSchedule");

37 }

38 *return* isValid;

39 }

40

41 *public* *boolean* addAttendance(*int* employeeID, String subjectExpert, String topicsHandled, String asPerSchedule,

42 LocalDateTime inDateTime, LocalDateTime outDateTime) throws InvalidAttendanceMarkingException {

43 *boolean* isAdded = *false*;

44 *if* (validateSchedule(asPerSchedule)) {

45 DailyAttendance dailyAttendance = *new* DailyAttendance(employeeID, subjectExpert, topicsHandled,

46 asPerSchedule, inDateTime, outDateTime);

47 attendanceList.add(dailyAttendance);

48 isAdded = *true*;

49 } *else* {

50 *throw* *new* InvalidAttendanceMarkingException("Invalid AsPerSchedule");

51 }

52 *return* isAdded;

53 }

54

55 *public* DailyAttendance viewEmployeeAttendanceById(*int* employeeID) throws InvalidAttendanceMarkingException {

56 DailyAttendance dailyAttendanceObj = *null*;

57 *for* (DailyAttendance dailyAttendance : attendanceList) {

58 *if* (dailyAttendance.getEmployeeID() == employeeID) {

59 dailyAttendanceObj = dailyAttendance;

60 *break*;

61 }

62 }

63 *if* (dailyAttendanceObj == *null*) {

64 *throw* *new* InvalidAttendanceMarkingException("Invalid Employee id");

65 }

66 *return* dailyAttendanceObj;

67 }

68

69 *public* List<DailyAttendance> viewAttendanceByDate(LocalDateTime inDateTime, LocalDateTime outDateTime) throws InvalidAttendanceMarkingException {

70 List<DailyAttendance> resultList = *new* ArrayList<DailyAttendance>();

71 *for* (DailyAttendance dailyAttendance : attendanceList) {

72 *if* (inDateTime.equals(dailyAttendance.getInDatetime())

73 && outDateTime.equals(dailyAttendance.getOutDatetime())) {

74 resultList.add(dailyAttendance);

75 }

76 }

77 *if* (resultList.isEmpty()) {

78 *throw* *new* InvalidAttendanceMarkingException("No Matching Found");

79 }

80

81 *return* resultList;

82 }

83

84 *public* Map<String, List<DailyAttendance>> viewSubjectExpertWise() {

85 Map<String, List<DailyAttendance>> myMaps = *new* HashMap<String, List<DailyAttendance>>();

86 *for* (DailyAttendance item : attendanceList) {

87 *if* (myMaps.containsKey(item.getSubjectExpert()) == *false*) {

88 myMaps.put(item.getSubjectExpert(), *new* ArrayList<DailyAttendance>());

89 }

90 myMaps.get(item.getSubjectExpert()).add(item);

91 }

92 *return* myMaps;

93 }

94

95 }

96

Grade

Reviewed on Wednesday, 28 April 2021, 10:20 PM by Automatic grade  
**Grade** 100 / 100  
**Assessment report**  
[[-]](javascript:void(0);)**Grading and Feedback**

*Writing JUnit for a Utility class - 60.0 / 60.0(Success)*

*Test Coverage - 40 / 40.0(Success)*

Test Case Passed