CD Gallery System

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

[***Click here to download the code template***](https://cognizant.tekstac.com/pluginfile.php/54385/mod_vpl/intro/CdGallerySystem.zip?time=1614945671097)

***CD Gallery System***is an automated movie’s cd collection system. Using the application, you can maintain information of movie’s cds where cd’s working status is checked and information can be accessed for given movie producer also cd information can be seen as per movie release year.

Smith 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 CDInfo

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

|  |  |  |
| --- | --- | --- |
| **Type(Class)** | **Attributes** | **Methods** |
| CDInfo | String cdId;  String movieName;  String producerName;  long releaseYear;  String cdWorkingStatus; | Necessary getters and setters are provided.  An overloaded constructor in the class is provided.  Also toString methods is overridden. |

Here the **cdWorkingStatus**can take a value either “Yes” or “No”(Value is case insensitive).

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

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

You are also provided with an utility class CDGallery with business methods.

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

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component Name** | **Type(Class)** | **Methods** | **Responsibilities** | **Exception** |
| Validating the cd working status | CDGallery | public boolean validateCDWorkingStatus(String cdWorkingStatus) | Validate the cdworking status.. If valid return true , else this method should throw an user defined exception | Throw an user defined exception “InvalidCDInfoException”  if the cdWorkingStatus Is not  “Yes” or “No” .  Irrespecitve of its case (ie "Yes" or "NO" is not case sensitive) |
| Validating the cd release year | CDGallery | public boolean validateCDReleaseYear(long releaseYear) | Validate the cd release year.. If valid return true , else this method should throw an user defined exception | If status is invalid,   throw an user  defined exception “InvalidCDInfoException”  if the release year is not in the range of 1999  to till current year. |
| Count number of working Cds | CDGallery | public int countNoOfWorkingCDs(List<CDInfo> cdList) | This method should return the Number Of working CDs from list of  cdInfo object as argument.  If the list is empty it should throw an user defined exception. | Throw an user defined exception “InvalidCDInfoException”  if the  list of object  is empty |
| View CD information between given release year | CDGallery | public List<CDInfo> viewCDInfoBetweenReleaseYear(List<CDInfo> cdList, long fromReleaseYear, long toReleaseYear) | This method should return the list of CDInfo which release year is between fromReleaseYear and toReleaseYear from list of CDInfo argument (Including the fromReleaseYear and toReleaseYear). If the result list is empty then throw an user defined exception. Validate the  FromReleaseYear and toReleaseYear also. | Throw an user defined exception “InvalidCDInfoException”  if the result list is empty. Validate the fromReleaseYear and toReleaseYear , |
| Count number of movies of each producer | CDGallery | public Map<String,Integer> countNoOfMoviesOfProducer(List<CDInfo> cdList) | This method should return the Map of producername and count of their movies from list of CDInfo argument  If the  CD  list is empty then it will  throw an user defined exception. | Throw an user defined exception “InvalidCDInfoException”  if the list of CD is empty |

**CD Information is maintained from year 1999 till current year;**

 You need to write Junit test for the CDGallery class.

**Testing Scenarios:**

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

**Note:**

To perform testing the cdList should contain objects of CDInfo.

To do this, in CDGalleryTest class you are provided with a setup method.  Use this method to populate the static variable cdList in  CDGalleryTest class .  Create few objects for  CDInfo,  populate the cdList with these objects and  use that list to test the methods  in CDGallery  class that needs a CDInfo list to be passed as attribute.

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

|  |  |
| --- | --- |
| **Test Method** | **Scenarios / Responsibilities** |
| test11ValidateCDWorkingStatusWhenYes | This method should test the validateCDWorkingStatus method when a valid status  “Yes” is passed as a parameter |
| test12ValidateCDWorkingStatusWhenNo | This method should test the validateCDWorkingStatus method when a valid status  “No” is passed as a  parameter . |
| test13ValidateCDWorkingStatusWhenInvalid | This method should test the validateCDWorkingStatus method when an invalid status  is passed   as a parameter . validateCDWorkingStatus method is expected  to throw InvalidCDInfoException when status is invalid.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block. |
| test14ValidateCDReleaseYearForValidYear | This method should test the validateCDReleaseYear method when a valid year (between 1999 till current year) is passed as parameter to this method. |
| test15ValidateCDReleaseYearForInvalidYear | This method should test the validateCDReleaseYear method when an invalid year is passed as a  parameter. validateCDReleaseYear method is expected  to throw InvalidCDInfoException when year is invalid.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test16CountNoOfWorkingCDs | This method should test the correctness of  countNoOfWorkingCDs method.  Perform testing for the correctness of the value returned. |
| test17ViewCDInfoBetweenReleaseYear | This method should test the correctness of viewCDInfoBetweenReleaseYear method.  Perform testing for the correctness of the value returned. |
| test18CountNoOfMoviesOfProducer | This method should test the correctness of countNoOfMoviesOfProducer method.  Perform testing for the correctness of the value returned. |

Implement the test methods and provide the needed annotation to all the methods in CDGalleryTest 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 CDGalleryTest 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

#### **CdGallerySystem/src/com/cts/cdgallery/exception/InvalidCDInfoException.java**

1 *package* com.cts.cdgallery.exception;

2

3 *public* *class* InvalidCDInfoException *extends* Exception {

4

5 *public* InvalidCDInfoException(String msg) {

6 *super*(msg);

7 }

8 }

9

#### **CdGallerySystem/src/com/cts/cdgallery/model/CDInfo.java**

1 *package* com.cts.cdgallery.model;

2

3 //model class

4 *public* *class* CDInfo {

5 *private* String cdId;

6 *private* String movieName;

7 *private* String producerName;

8 *private* *long* releaseYear;

9 *private* String cdWorkingStatus; // YES, NO

10

11 *public* CDInfo() {

12 // TODO Auto-generated constructor stub

13 }

14

15 *public* CDInfo(String cdId, String movieName, String producerName, *long* releaseYear, String cdWorkingStatus) {

16 *super*();

17 *this*.cdId = cdId;

18 *this*.movieName = movieName;

19 *this*.producerName = producerName;

20 *this*.releaseYear = releaseYear;

21 *this*.cdWorkingStatus = cdWorkingStatus;

22 }

23

24 *public* String getCdId() {

25 *return* cdId;

26 }

27

28 *public* *void* setCdId(String cdId) {

29 *this*.cdId = cdId;

30 }

31

32 *public* String getMovieName() {

33 *return* movieName;

34 }

35

36 *public* *void* setMovieName(String movieName) {

37 *this*.movieName = movieName;

38 }

39

40 *public* String getProducerName() {

41 *return* producerName;

42 }

43

44 *public* *void* setProducerName(String producerName) {

45 *this*.producerName = producerName;

46 }

47

48 *public* *long* getReleaseYear() {

49 *return* releaseYear;

50 }

51

52 *public* *void* setReleaseYear(*long* releaseYear) {

53 *this*.releaseYear = releaseYear;

54 }

55

56 *public* String getCdWorkingStatus() {

57 *return* cdWorkingStatus;

58 }

59

60 *public* *void* setCdWorkingStatus(String cdWorkingStatus) {

61 *this*.cdWorkingStatus = cdWorkingStatus;

62 }

63

64 @Override

65 *public* String toString() {

66 *return* "CDInfo [cdId=" + cdId + ", movieName=" + movieName + ", producerName=" + producerName + ", releaseYear="

67 + releaseYear + ", cdWorkingStatus=" + cdWorkingStatus + "]";

68 }

69

70 }

71

#### **CdGallerySystem/src/com/cts/cdgallery/skeleton/SkeletonValidator.java**

1 *package* com.cts.cdgallery.skeleton;

2

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

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

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

6

7 /\*\*

8 \* @author t-aarti3 This class is used to verify if the Code Skeleton is intact

9 \* and not modified by participants thereby ensuring smooth auto

10 \* evaluation

11 \*/

12

13 *public* *class* SkeletonValidator {

14 *public* SkeletonValidator() {

15 validateClassName("com.cts.cdgallery.util.CDGallery");

16 validateClassName("com.cts.cdgallery.model.CDInfo");

17 validateClassName("com.cts.cdgallery.exception.InvalidCDInfoException");

18 validateClassName("com.cts.cdgallery.test.CDGalleryTest");

19

20 validateMethodSignature(

21 "validateCDWorkingStatus:boolean,validateCDReleaseYear:boolean,countNoOfWorkingCDs:int,viewCDInfoBetweenReleaseYear:java.util.List,countNoOfMoviesOfProducer:java.util.Map",

22 "com.cts.cdgallery.util.CDGallery");

23 validateMethodSignature(

24 "test11ValidateCDWorkingStatusWhenYes:void,test12ValidateCDWorkingStatusWhenNo:void,test13ValidateCDWorkingStatusWhenInvalid:void,test14ValidateCDReleaseYearForValidYear:void,test15ValidateCDReleaseYearForInvalidYear:void,"

25 + "test16CountNoOfWorkingCDs:void,test17ViewCDInfoBetweenReleaseYear:void,test18CountNoOfMoviesOfProducer:void",

26 "com.cts.cdgallery.test.CDGalleryTest");

27

28 }

29

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

31

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

33

34 *boolean* iscorrect = *false*;

35 *try* {

36 Class.forName(className);

37 iscorrect = *true*;

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

39

40 } *catch* (ClassNotFoundException e) {

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

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

43

44 } *catch* (Exception e) {

45 LOG.log(Level.SEVERE,

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

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

48 }

49 *return* iscorrect;

50 }

51

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

53 Class cls = *null*;

54 *try* {

55

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

57 *boolean* errorFlag = *false*;

58 String[] methodSignature;

59 String methodName = *null*;

60 String returnType = *null*;

61

62 *for* (String singleMethod : actualmethods) {

63 *boolean* foundMethod = *false*;

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

65

66 methodName = methodSignature[0];

67 returnType = methodSignature[1];

68 cls = Class.forName(className);

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

70 *for* (Method findMethod : methods) {

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

72 foundMethod = *true*;

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

74 errorFlag = *true*;

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

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

77

78 } *else* {

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

80 }

81

82 }

83 }

84 *if* (!foundMethod) {

85 errorFlag = *true*;

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

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

88 }

89

90 }

91 *if* (!errorFlag) {

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

93 }

94

95 } *catch* (Exception e) {

96 LOG.log(Level.SEVERE,

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

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

99 }

100 }

101

102 }

103

#### **CdGallerySystem/src/com/cts/cdgallery/test/CDGalleryTest.java**

1 *package* com.cts.cdgallery.test;

2

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

4 *import* java.util.ArrayList;

5 *import* java.util.List;

6 *import* java.util.\*;

7

8

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

10 *import* org.junit.AfterClass;

11 *import* org.junit.BeforeClass;

12

13 *import* com.cts.cdgallery.model.CDInfo;

14 *import* com.cts.cdgallery.util.CDGallery;

15 *import* com.cts.cdgallery.exception.InvalidCDInfoException;

16 *import* org.junit.Rule;

17 *import* org.junit.Test;

18

19

20 *public* *class* CDGalleryTest {

21

22 @Rule

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

24

25 *private* *static* List<CDInfo> cdList = *new* ArrayList<CDInfo>();

26 *private* *static* CDGallery cdGalleryObj=*null*;

27

28 *static* CDInfo c1;

29 *static* CDInfo c2;

30 *static* CDInfo c3;

31

32 *private* *static* List<CDInfo>cdInfoList=*new* ArrayList();

33

34 @BeforeClass

35 *public* *static* *void* setUp() throws Exception {

36 cdGalleryObj=*new* CDGallery();

37 //Create few CDInfo objects and add to cdList.

38 //Use that list to test all the methods in CDGallery class that requires a list of CDInfo

39

40 // list of CDInfo

41 c1=*new* CDInfo("123","Avengers","marvel",2019,"Yes");

42 c2=*new* CDInfo("789","AP","MRproduction",2018,"No");

43 c3=*new* CDInfo("456","ginee","xyz",2001,"Yes");

44

45 cdInfoList.add(c1);

46 cdInfoList.add(c2);

47 cdInfoList.add(c3);

48

49 // cdGalleryObj.setCDList(cdInfoList);

50

51 }

52

53 @AfterClass

54 *public* *static* *void* tearDown() throws Exception {

55

56 }

57

58 // test the validateCDWorkingStatus method when a valid Status Yes is passed as parameter to this method.

59 @Test

60 *public* *void* test11ValidateCDWorkingStatusWhenYes() throws InvalidCDInfoException{

61

62 // Code here..

63 assertTrue(cdGalleryObj.validateCDWorkingStatus("Yes"));

64 }

65

66 //test the validateCDWorkingStatus method when a valid Status No is passed as parameter to this method.

67 @Test

68 *public* *void* test12ValidateCDWorkingStatusWhenNo() throws InvalidCDInfoException {

69 // Code here..

70

71 assertTrue(cdGalleryObj.validateCDWorkingStatus("No"));

72 }

73

74 //test the validateCDWorkingStatus method when an invalid Status is passed to this method.

75 @Test

76 *public* *void* test13ValidateCDWorkingStatusWhenInvalid() throws InvalidCDInfoException{

77 // Code here..

78 exceptionRule.expect(InvalidCDInfoException.*class*);

79 exceptionRule.expectMessage("Valid Working Status For CD are - Yes / No");

80 cdGalleryObj.validateCDWorkingStatus("abc");

81 }

82

83 //test the validateCDReleaseYear method when valid Year is provided

84 @Test

85 *public* *void* test14ValidateCDReleaseYearForValidYear() throws InvalidCDInfoException{

86 // Code here..

87 assertTrue(cdGalleryObj.validateCDReleaseYear(2001));

88 }

89

90 //test the validateCDReleaseYear method when invalid Year is provided

91 @Test

92 *public* *void* test15ValidateCDReleaseYearForInvalidYear() throws InvalidCDInfoException {

93 // Code here..

94 exceptionRule.expect(InvalidCDInfoException.*class*);

95 exceptionRule.expectMessage("Valid release year for CD is from 1999 till current year");

96 cdGalleryObj.validateCDReleaseYear(1990);

97 }

98

99 //test the countNoOfWorkingCDs method when a CD List is passed as parameter.

100 @Test

101 *public* *void* test16CountNoOfWorkingCDs() throws InvalidCDInfoException {

102 // Code here..

103 List<CDInfo>tmp=*new* ArrayList<>();

104 tmp.add(c1);

105 tmp.add(c3);

106 assertEquals(2,cdGalleryObj.countNoOfWorkingCDs(tmp));

107 }

108

109 //test the viewCDInfoBetweenReleaseYear method when from release date and to release date is passed as parameter exists in the cdList.

110 @Test

111 *public* *void* test17ViewCDInfoBetweenReleaseYear() throws InvalidCDInfoException{

112 // Code here..

113 List<CDInfo>temp=*new* ArrayList<>();

114 temp.add(c1);

115 temp.add(c2);

116 assertEquals(temp,cdGalleryObj.viewCDInfoBetweenReleaseYear(cdInfoList,2015,2020));

117

118 }

119

120 //test the countNoOfMoviesOfProducer method when a CD List is passed as parameter.

121 @Test

122 *public* *void* test18CountNoOfMoviesOfProducer() throws InvalidCDInfoException {

123 // Code here...

124

125 assertFalse(cdGalleryObj.countNoOfMoviesOfProducer(cdInfoList).isEmpty());

126 }

127

128 }

129

#### **CdGallerySystem/src/com/cts/cdgallery/test/Main.java**

1 *package* com.cts.cdgallery.test;

2

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

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

5

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

7

8 *public* *class* Main {

9

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

11

12 SkeletonValidator validator = *new* SkeletonValidator();

13

14 /\*Result result = JUnitCore.runClasses(CDGalleryTest.class);

15 if (result.getFailureCount() == 0) {

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

17 } else {

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

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

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

21 }

22 \*/

23

24 }

25

26 }

27

#### **CdGallerySystem/src/com/cts/cdgallery/util/CDGallery.java**

1 *package* com.cts.cdgallery.util;

2

3 *import* java.time.LocalDate;

4 *import* java.util.ArrayList;

5 *import* java.util.HashMap;

6 *import* java.util.List;

7 *import* java.util.Map;

8

9 *import* com.cts.cdgallery.exception.InvalidCDInfoException;

10 *import* com.cts.cdgallery.model.CDInfo;

11

12 *public* *class* CDGallery {

13

14 // validate cd working status

15 *public* *boolean* validateCDWorkingStatus(String cdWorkingStatus) throws InvalidCDInfoException {

16 *boolean* status = *false*;

17 *if* (cdWorkingStatus.equalsIgnoreCase("Yes") || cdWorkingStatus.equalsIgnoreCase("No")) {

18 status = *true*;

19 } *else* {

20 status = *false*;

21 *throw* *new* InvalidCDInfoException("Valid Working Status For CD are - Yes / No");

22 }

23

24 *return* status;

25 }

26

27 // validate cd release year

28 *public* *boolean* validateCDReleaseYear(*long* releaseYear) throws InvalidCDInfoException {

29 *boolean* status = *false*;

30 LocalDate currentDate = LocalDate.now();

31 *long* currentYear = currentDate.getYear();

32

33 *if* (releaseYear >= 1999 && releaseYear <= currentYear) {

34 status = *true*;

35 } *else* {

36 status = *false*;

37 *throw* *new* InvalidCDInfoException("Valid release year for CD is from 1999 till current year");

38 }

39

40 *return* status;

41 }

42

43 // count number of working cds

44 *public* *int* countNoOfWorkingCDs(List<CDInfo> cdList) throws InvalidCDInfoException {

45 *int* count = 0;

46 *if* (cdList.size() > 0) {

47 *for* (*int* i = 0; i < cdList.size(); i++) {

48 *try* {

49 *if* (validateCDWorkingStatus(cdList.get(i).getCdWorkingStatus())) {

50 *if* (cdList.get(i).getCdWorkingStatus().equalsIgnoreCase("yes")) {

51 count++;

52 }

53 }

54 } *catch* (InvalidCDInfoException e) {

55 // we skip count for invalid cdstatus.

56 }

57 }

58 } *else* {

59 *throw* *new* InvalidCDInfoException("Invalid cd list..");

60 }

61 *return* count;

62 }

63

64 // view cd info released between given years

65 *public* List<CDInfo> viewCDInfoBetweenReleaseYear(List<CDInfo> cdList, *long* fromReleaseYear, *long* toReleaseYear)

66 throws InvalidCDInfoException {

67 List<CDInfo> cList = *new* ArrayList<CDInfo>();

68 *if* (validateCDReleaseYear(fromReleaseYear) && validateCDReleaseYear(toReleaseYear)) {

69 *for* (*int* i = 0; i < cdList.size(); i++) {

70 *long* tempReleaseYear = cdList.get(i).getReleaseYear();

71 *if* (tempReleaseYear >= fromReleaseYear && tempReleaseYear <= toReleaseYear) {

72 cList.add(cdList.get(i));

73 }

74 }

75 *if* (cList.isEmpty()) {

76 *throw* *new* InvalidCDInfoException("No movies releases in given years");

77 }

78 }

79

80 *return* cList; // return null;

81 }

82

83 // count number of CDs of producer

84 *public* Map<String, Integer> countNoOfMoviesOfProducer(List<CDInfo> cdList) throws InvalidCDInfoException {

85 Map<String, Integer> map = *new* HashMap<String, Integer>();

86 *if* (cdList.size() > 0) {

87 *for* (*int* i = 0; i < cdList.size(); i++) {

88 String producerName = cdList.get(i).getProducerName();

89 *int* count = 0;

90 *for* (*int* c = 0; c < cdList.size(); c++) {

91

92 *if* (cdList.get(c).getProducerName().equals(producerName)) {

93 count++;

94 }

95 } // inner for

96 map.put(producerName, count);

97 }

98

99 } *else* {

100 *throw* *new* InvalidCDInfoException("Invalid cd list..");

101 }

102 *return* map;

103 }

104

105 }

106

## Grade

Reviewed on Monday, 3 May 2021, 10:42 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