Aviva Insurance Company

**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/67778/mod_vpl/intro/Aviva%20Insurance%20Company.zip)

***AVIVA Insurance Company***is well known for the excellent services provided by them for various Insurance policies.The company used to motivate the customers by giving additional bonus for their policies. This bonus is applicable for those policies whose expiry date is beyond the date of announcement of the bonus. As the number of transactions related to policy has increased it is tedious to take the reports based on the policy id, type and expiry date etc.  So the management wish to automate the policy details taken by the customers.

Nancy 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 InsurancePolicy

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

|  |  |  |
| --- | --- | --- |
| **Type(Class)** | **Attributes** | **Methods** |
| InsurancePolicy | String policyId  String policyType  String policyName  String customerName  Date policyDate  Date expiryDate  double premiumAmount | Necessary getters,setters are provided  A Constructor is also provided  Also toString method is overridden. |

·         Here, policyType can take a value either “Health Insurance” or “Vehicle Insurance” [Note: Values are case insensitive]

·         There can be different policy names for the given policy type.

For Example:

·         Critical Illness Insurance, Arogya Sanjeevani Policy are few policies of the policy type Health Insurance.

·         Comprehensive Insurance, Third-party Liability Insurance are few policies of the policy type Vehicle Insurance.

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

|  |  |
| --- | --- |
| **Type(Class)** | **Methods** |
| InvalidInsurancePolicyException | Provided with a single argument constructor – InvalidInsurancePolicyException(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:**PolicyManagement**(Utility Class)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component Name** | **Type (Class)** | **Methods** | **Responsibilities** | **Exception** |
| Validate the policy  type | PolicyManagement | public boolean validatePolicyType(String policyType) | Validate the policyType.  If valid, return true , else this method should throw a user defined exception | Throw a user defined exception “InvalidInsurancePolicyException”  if the policyType is neither “Health Insurance” or “Vehicle Insurance” |
| View policy based on Policy Id | PolicyManagement | public InsurancePolicy viewPolicyByInsurancePolicyId(List<InsurancePolicy> policyList, String policyId) | This method should return the InsurancePolicy object with the policy Id passed as parameter from list of policies, which is also passed as parameter.  If the policyList is empty or if there is no policy with the given Policy Id it should throw a user defined exception | Throw a user defined exception “InvalidInsurancePolicyException” if the policyList is empty or if  no policy exists with the given Policy Id. |
| View the list of Policies for a given Expiry date | PolicyManagement | public List<InsurancePolicy> viewPolicyByExpiryDate(List<InsurancePolicy> policyList, Date expiryDate) | This method takes the policyList and the expiry date as an argument. It should return the list of Policies whose expiryDate matches the given expiry date. If the policyList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidInsurancePolicyException” if the  policyList is empty. |
| Calculate the premium amount based on the Expiry date | PolicyManagement | public double calculatePremiumAmountByExpiryDate(List<InsurancePolicy> policyList, Date expiryDate) | This method takes the policyList as argument. It should calculate sum of  premium amount of policies whose expiry date matches the given expiry date and return the same.  If the policyList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidInsurancePolicyException” if the policyList is empty. |
| View the number of policies for each policy type | PolicyManagement | public Map<String, List<InsurancePolicy>> countOfPolicyHoldersBasedOnPolicyType(List<InsurancePolicy> policyList) | This method takes the policyList and the expiry date as argument and returns the count of policy holders with additional bonus. If the expiry date of the policy falls after the date passed as parameter, that policy has to be taken into consideration for count.          If the policyList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidInsurancePolicyException” if the policyList is empty. |
| Count of policies that gets Additional Bonus based on the Expiry date | PolicyManagement | public int countOfPolicyHoldersWithAdditonalBonus(List<InsurancePolicy> policyList, Date expiryDate) | This method takes the policyList and the expiry date as argument and returns the count of policies  with additional bonus. If the expiry date of the policy falls after the date passed as parameter, count of that policy has to be taken into consideration.  If the policyList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidInsurancePolicyException” if the policyList is empty. |

You need to write Junit test for the PolicyManagement class.

**Testing Scenarios:**

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

**Note:**

To perform testing, the policyList should contain objects of InsurancePolicy.

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

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

|  |  |
| --- | --- |
| **Test Method** | **Scenarios / Responsibilities** |
| test11ValidatePolicyTypeWhenHealthInsurance | This method should test the validatePolicyType method when a policy type is  “Health Insurance” is passed as a parameter |
| test12ValidatePolicyTypeWhenVehicleInsurance | This method should test the validatePolicyType method when a policy type is “Vehicle Insurance” is passed as a parameter. |
| test13ValidatePolicyTypeWhenInvalid | This method should test the validatePolicyType method when an invalid type is passed   as a parameter.  validatePolicyType method is expected  to throw InvalidInsurancePolicyException when policy type is invalid.  Write JUnit to test for the exception thrown either by using appropriate annotation or by using try catch block. |
| test14ViewPolicyForValidInsurancePolicyId | This method should test the viewPolicyByInsurancePolicyId method when a list of insurance policy and a policy Id which exists is passed as parameter to this method. |
| test15ViewPolicyForInvalidInsurancePolicyId | This method should test the correctness of viewPolicyByInsurancePolicyId method for a non-existing Policy Id.  Perform testing for the correctness of the value returned.  viewPolicyByInsurancePolicyId method is expected  to throw InvalidInsurancePolicyException for a non-existing Policy Id.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test16ViewPolicyByExpiryDate | This method should test the correctness of viewPolicyByExpiryDate method.  Perform testing for the correctness of the value returned. |
| test17ViewPolicyByExpiryDateForEmptyList | This method should test the correctness of viewPolicyByExpiryDate method for an empty policyList.  viewPolicyByExpiryDate method is expected  to throw InvalidInsurancePolicyException when policyList is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test18CalculatePremiumAmountByExpiryDate | This method should test the correctness of calculatePremiumAmountByExpiryDate method.  Perform testing for the correctness of the value returned. |
| test19CalculatePremiumAmountByExpiryDateForEmptyList | This method should test the correctness of calculatePremiumAmountByExpiryDate method for an empty policyList.  calculatePremiumAmountByExpiryDate method is expected  to throw InvalidInsurancePolicyException when policyList is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test20CountOfPolicyHoldersBasedOnThePolicyType | This method should test the correctness of countOfPolicyHoldersBasedOnThePolicyType method.  Perform testing for the correctness of the value returned. |
| test21CountOfPolicyHoldersBasedOnThePolicyTypeForEmptyList | This method should test the correctness of countOfPolicyHoldersBasedOnThePolicyType method for an empty policyList.  countOfPolicyHoldersBasedOnThePolicyType method is expected to throw InvalidInsurancePolicyException when policyList is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test22CountOfPolicyHoldersWithAdditonalBonus | This method should test the correctness of countOfPolicyHoldersWithAdditonalBonus method.  Perform testing for the correctness of the value returned. |
| test23CountOfPolicyHoldersWithAdditonalBonusForEmptyList | This method should test the correctness of countOfPolicyHoldersWithAdditonalBonus method for an empty policyList.  countOfPolicyHoldersWithAdditonalBonus method is expected to throw InvalidInsurancePolicyException when policyList 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 PolicyManagementTest 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 PolicyManagementTest 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

#### **AvivaInsuranceCompany/src/com/aviva/util/PolicyManagement.java**

1 *package* com.aviva.util;

2

3 *import* java.text.ParseException;

4 *import* java.util.ArrayList;

5 *import* java.util.Date;

6 *import* java.util.LinkedHashMap;

7 *import* java.util.List;

8 *import* java.util.Map;

9

10 *import* com.aviva.exception.InvalidInsurancePolicyException;

11 *import* com.aviva.model.InsurancePolicy;

12

13 *public* *class* PolicyManagement {

14

15 *public* *boolean* validatePolicyType(String policyType) throws InvalidInsurancePolicyException {

16 *boolean* status = *false*;

17 *if* (policyType.equalsIgnoreCase("Health Insurance") || policyType.equalsIgnoreCase("Vehicle Insurance")) {

18 status = *true*;

19 } *else* {

20 status = *false*;

21 *throw* *new* InvalidInsurancePolicyException("Valid Policies are - Health, Life, Vehicle Insurance");

22 }

23

24 *return* status;

25 }

26

27 *public* InsurancePolicy viewPolicyByInsurancePolicyId(List<InsurancePolicy> policyList, String policyId) throws InvalidInsurancePolicyException {

28

29 *if*(policyList.size()==0){

30 *throw* *new* InvalidInsurancePolicyException("Policy List is empty");

31 }

32 *else* {

33 *for*(InsurancePolicy m : policyList){

34 *if*(m.getPolicyId().equals(policyId))

35 *return* m;

36 }

37 *throw* *new* InvalidInsurancePolicyException("Policy Type is invalid");

38 }

39 }

40

41

42 *public* List<InsurancePolicy> viewPolicyByExpiryDate(List<InsurancePolicy> policyList, Date expiryDate)

43 throws InvalidInsurancePolicyException, ParseException {

44 *if*(policyList.size()==0){

45 *throw* *new* InvalidInsurancePolicyException("Policy List is empty");

46 }

47 *else* {

48 List<InsurancePolicy> dateWiseList = *new* ArrayList<InsurancePolicy>();

49

50 *for* (InsurancePolicy i:policyList) {

51 *if*(i.getExpiryDate().compareTo(expiryDate)==0){

52 dateWiseList.add(i);

53 }

54 }

55 *return* dateWiseList;

56 }

57 }

58

59 *public* *double* calculatePremiumAmountByExpiryDate(List<InsurancePolicy> policyList, Date expiryDate)

60 throws InvalidInsurancePolicyException, ParseException {

61 *if*(policyList.size()==0){

62 *throw* *new* InvalidInsurancePolicyException("Policy List is empty");

63 }

64 *else* {

65 *double* sum=0;

66 *for* (InsurancePolicy i:policyList) {

67 *if*(i.getExpiryDate().compareTo(expiryDate)==0){

68 sum=sum+i.getPremiumAmount();

69 }

70 }

71

72 *return* sum;

73 }

74 }

75

76

77 *public* Map<String, List<InsurancePolicy>> countOfPolicyHoldersBasedOnPolicyType(List<InsurancePolicy> policyList) throws InvalidInsurancePolicyException {

78 *if*(policyList.size()==0){

79 *throw* *new* InvalidInsurancePolicyException("Policy List is empty");

80 }

81 *else* {

82 Map<String,List<InsurancePolicy>> result = *new* LinkedHashMap<>();

83

84 *for*(InsurancePolicy t : policyList){

85 *if*(!result.containsKey(t.getPolicyType())){

86 result.put(t.getPolicyType(),*new* ArrayList<InsurancePolicy>());

87 }

88 List<InsurancePolicy> temp=result.get(t.getPolicyType());

89 temp.add(t);

90 result.put(t.getPolicyType(), temp);

91 }

92 *return* result;

93 }

94 }

95

96 *public* *int* countOfPolicyHoldersWithAdditonalBonus(List<InsurancePolicy> policyList, Date expiryDate) throws InvalidInsurancePolicyException {

97 *if*(policyList.size()==0){

98 *throw* *new* InvalidInsurancePolicyException("Policy List is empty");

99 }

100 *else* {

101 *int* count=0;

102 *for* (InsurancePolicy i: policyList) {

103

104 *if* (i.getExpiryDate().compareTo(expiryDate)==1) {

105 System.out.println(i.getPolicyDate());

106 count++;

107 }

108 }

109

110 *return* count;

111 }

112 }

113

114 }

115

#### **AvivaInsuranceCompany/src/com/aviva/exception/InvalidInsurancePolicyException.java**

1 *package* com.aviva.exception;

2

3 *public* *class* InvalidInsurancePolicyException *extends* Exception{

4

5 *public* InvalidInsurancePolicyException(String msg) {

6 *super*(msg);

7 }

8

9 }

10

#### **AvivaInsuranceCompany/src/com/aviva/model/InsurancePolicy.java**

1 *package* com.aviva.model;

2

3 *import* java.util.Date;

4

5 *public* *class* InsurancePolicy {

6

7 *private* String policyId;

8 *private* String policyType;

9 *private* String policyName;

10 *private* String customerName;

11 *private* Date policyDate;

12 *private* Date expiryDate;

13 *private* *double* premiumAmount;

14

15 *public* InsurancePolicy() {

16

17 }

18

19 *public* InsurancePolicy(String policyId, String policyType, String policyName, String customerName, Date policyDate,

20 Date expiryDate, *double* premiumAmount) {

21 *super*();

22 *this*.policyId = policyId;

23 *this*.policyType = policyType;

24 *this*.policyName = policyName;

25 *this*.customerName = customerName;

26 *this*.policyDate = policyDate;

27 *this*.expiryDate = expiryDate;

28 *this*.premiumAmount = premiumAmount;

29 }

30

31 *public* String getPolicyId() {

32 *return* policyId;

33 }

34

35 *public* *void* setPolicyId(String policyId) {

36 *this*.policyId = policyId;

37 }

38

39 *public* String getPolicyType() {

40 *return* policyType;

41 }

42

43 *public* *void* setPolicyType(String policyType) {

44 *this*.policyType = policyType;

45 }

46

47 *public* String getPolicyName() {

48 *return* policyName;

49 }

50

51 *public* *void* setPolicyName(String policyName) {

52 *this*.policyName = policyName;

53 }

54

55 *public* String getCustomerName() {

56 *return* customerName;

57 }

58

59 *public* *void* setCustomerName(String customerName) {

60 *this*.customerName = customerName;

61 }

62

63 *public* Date getPolicyDate() {

64 *return* policyDate;

65 }

66

67 *public* *void* setPolicyDate(Date policyDate) {

68 *this*.policyDate = policyDate;

69 }

70

71 *public* Date getExpiryDate() {

72 *return* expiryDate;

73 }

74

75 *public* *void* setExpiryDate(Date expiryDate) {

76 *this*.expiryDate = expiryDate;

77 }

78

79 *public* *double* getPremiumAmount() {

80 *return* premiumAmount;

81 }

82

83 *public* *void* setPremiumAmount(*double* premiumAmount) {

84 *this*.premiumAmount = premiumAmount;

85 }

86

87 @Override

88 *public* String toString() {

89 *return* "InsurancePolicy [policyId=" + policyId + ", policyType=" + policyType + ", policyName=" + policyName

90 + ", customerName=" + customerName + ", policyDate=" + policyDate + ", expiryDate=" + expiryDate

91 + ", premiumAmount=" + premiumAmount + "]";

92 }

93

94

95

96 }

97

#### **AvivaInsuranceCompany/src/com/aviva/skeleton/SkeletonValidator.java**

1 *package* com.aviva.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.aviva.util.PolicyManagement");

16 validateClassName("com.aviva.model.InsurancePolicy");

17 validateClassName("com.aviva.exception.InvalidInsurancePolicyException");

18 validateClassName("com.aviva.test.PolicyManagementTest");

19

20 validateMethodSignature(

21 "validatePolicyType:boolean,viewPolicyByInsurancePolicyId:com.aviva.model.InsurancePolicy,viewPolicyByExpiryDate:java.util.List,calculatePremiumAmountByExpiryDate:double,countOfPolicyHoldersBasedOnPolicyType:java.util.Map,countOfPolicyHoldersWithAdditonalBonus:int",

22 "com.aviva.util.PolicyManagement");

23 validateMethodSignature(

24 "test11ValidatePolicyTypeWhenHealthInsurance:void,test12ValidatePolicyTypeWhenVehicleInsurance:void,test13ValidatePolicyTypeWhenInvalid:void,test14ViewPolicyForValidInsurancePolicyId:void,test15ViewPolicyForInvalidInsurancePolicyId:void,test16ViewPolicyByExpiryDate:void,test17ViewPolicyByExpiryDateForEmptyList:void,"

25 + "test18CalculatePremiumAmountByExpiryDate:void,test19CalculatePremiumAmountByExpiryDateForEmptyList:void,test20CountOfPolicyHoldersBasedOnThePolicyType:void,test21CountOfPolicyHoldersBasedOnThePolicyTypeForEmptyList:void,test22CountOfPolicyHoldersWithAdditonalBonus:void,test23CountOfPolicyHoldersWithAdditonalBonusForEmptyList:void",

26 "com.aviva.test.PolicyManagementTest");

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

#### **AvivaInsuranceCompany/src/com/aviva/test/Main.java**

1 *package* com.aviva.test;

2

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

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

5 *import* org.junit.runner.notification.Failure;

6

7 *import* com.aviva.skeleton.SkeletonValidator;

8

9 *public* *class* Main {

10

11 *public* *static* *void* main(String args[])

12 {

13 SkeletonValidator s=*new* SkeletonValidator();

14

15 //Uncomment these lines after completing the JUnit Code

16

17 Result result=JUnitCore.runClasses(PolicyManagementTest.*class*);

18

19 *if*(result.getFailureCount()==0)

20 {

21 System.out.println("No Failures");

22 }

23 *else*

24 {

25 *for*(Failure failure: result.getFailures())

26

27 {

28 System.out.println(failure.toString());

29 }

30 }

31 System.out.println("Result "+result.wasSuccessful());

32

33 }

34

35 }

36

#### **AvivaInsuranceCompany/src/com/aviva/test/PolicyManagementTest.java**

1 *package* com.aviva.test;

2

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

4 *import* *static* org.junit.Assert.assertTrue;

5

6 *import* java.text.ParseException;

7 *import* java.text.SimpleDateFormat;

8 *import* java.util.ArrayList;

9 *import* java.util.List;

10 *import* java.util.Map;

11 *import* java.util.Arrays;

12

13 *import* org.junit.BeforeClass;

14 *import* org.junit.FixMethodOrder;

15 *import* org.junit.Test;

16 *import* org.junit.runners.MethodSorters;

17

18 *import* com.aviva.exception.InvalidInsurancePolicyException;

19 *import* com.aviva.model.InsurancePolicy;

20 *import* com.aviva.util.PolicyManagement;

21

22 @FixMethodOrder(MethodSorters.NAME\_ASCENDING)

23 *public* *class* PolicyManagementTest {

24

25 *private* *static* List<InsurancePolicy> policyList = *new* ArrayList<InsurancePolicy>();;

26 *private* *static* PolicyManagement policy;

27 *private* *static* SimpleDateFormat s=*new* SimpleDateFormat("yyyy-dd-MM");

28

29 @BeforeClass

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

31 InsurancePolicy ip=*new* InsurancePolicy("100","Health Insurance","Arogya Sanjeevani Policy","abc",s.parse("2021-23-01"),s.parse("2021-23-03"),200.0);

32 InsurancePolicy ip1=*new* InsurancePolicy("101","Health Insurance","Crirical Illness Insurance","xyz",s.parse("2020-25-10"),s.parse("2021-25-10"),600.0);

33 InsurancePolicy ip2=*new* InsurancePolicy("102","Vehicle Insurance","Comprehensive Insurance","def",s.parse("2021-23-05"),s.parse("2021-23-06"),800.0);

34

35 policyList.add(ip);

36 policyList.add(ip1);

37 policyList.add(ip2);

38 //Create few objects for InsurancePolicy class and add to policyList.

39 //Use that list to test all the methods in PolicyManagementTest class that requires a list of InsurancePolicy

40 }

41

42 // Test validatePolicyType method when the Insurance type is Health Insurance

43 @Test

44 *public* *void* test11ValidatePolicyTypeWhenHealthInsurance() throws InvalidInsurancePolicyException {

45 // Fill the code

46 policy=*new* PolicyManagement();

47 assertTrue(policy.validatePolicyType("Health Insurance"));

48 }

49

50 // Test validatePolicyType method when the Insurance type is Vehicle Insurance

51 @Test

52 *public* *void* test12ValidatePolicyTypeWhenVehicleInsurance() throws InvalidInsurancePolicyException {

53 // Fill the code

54 policy=*new* PolicyManagement();

55 assertTrue(policy.validatePolicyType("Vehicle Insurance"));

56 }

57

58 // Test validatePolicyType method when the Insurance type is Invalid

59 @Test(expected=InvalidInsurancePolicyException.*class*)

60 *public* *void* test13ValidatePolicyTypeWhenInvalid() throws InvalidInsurancePolicyException {

61 // Fill the code

62 policy=*new* PolicyManagement();

63 policy.validatePolicyType("Insurance");

64 }

65

66 // Test viewPolicyByInsurancePolicyId method for a valid Policy Id

67 @Test

68 *public* *void* test14ViewPolicyForValidInsurancePolicyId() throws InvalidInsurancePolicyException {

69 // Fill the code

70 policy=*new* PolicyManagement();

71 InsurancePolicy inspolicy=policy.viewPolicyByInsurancePolicyId(policyList, "100");

72 }

73

74 // Test viewPolicyByInsurancePolicyId method for an invalid Policy Id

75 @Test(expected=InvalidInsurancePolicyException.*class*)

76 *public* *void* test15ViewPolicyForInvalidInsurancePolicyId() throws InvalidInsurancePolicyException{

77 // Fill the code

78 policy=*new* PolicyManagement();

79 InsurancePolicy inspolicy=policy.viewPolicyByInsurancePolicyId(policyList, "1000");

80 }

81

82 // Test the viewPolicyByExpiryDate method

83 @Test

84 *public* *void* test16ViewPolicyByExpiryDate() throws ParseException,InvalidInsurancePolicyException {

85 // Fill the code

86 policy=*new* PolicyManagement();

87 List<InsurancePolicy>list=policy.viewPolicyByExpiryDate(policyList,s.parse("2021-23-03"));

88 }

89

90 // Test the viewPolicyByExpiryDate method for an Empty List

91 @Test(expected=InvalidInsurancePolicyException.*class*)

92 *public* *void* test17ViewPolicyByExpiryDateForEmptyList() throws ParseException,InvalidInsurancePolicyException{

93 // Fill the code

94 policy=*new* PolicyManagement();

95 List<InsurancePolicy>emptyList=*new* ArrayList<>();

96 List<InsurancePolicy>list=policy.viewPolicyByExpiryDate(emptyList,s.parse("2021-23-03"));

97 }

98

99 // Test the calculatePremiumAmountByExpiryDate method

100 @Test

101 *public* *void* test18CalculatePremiumAmountByExpiryDate() throws ParseException,InvalidInsurancePolicyException{

102 // Fill the code

103 policy=*new* PolicyManagement();

104 assertEquals(800.0,policy.calculatePremiumAmountByExpiryDate(policyList,s.parse("2021-23-06")),0);

105 }

106

107 // Test the calculatePremiumAmountByExpiryDate method for an Empty List

108 @Test(expected=InvalidInsurancePolicyException.*class*)

109 *public* *void* test19CalculatePremiumAmountByExpiryDateForEmptyList() throws ParseException,InvalidInsurancePolicyException{

110 // Fill the code

111 policy=*new* PolicyManagement();

112 List<InsurancePolicy>emptyList=*new* ArrayList<>();

113 *double* amount=policy.calculatePremiumAmountByExpiryDate(emptyList,s.parse("2021-23-06"));

114 }

115

116 // Test the countOfPolicyHoldersBasedOnThePolicyType method

117 @Test

118 *public* *void* test20CountOfPolicyHoldersBasedOnThePolicyType()throws InvalidInsurancePolicyException {

119 // Fill the code

120 policy=*new* PolicyManagement();

121 policy.countOfPolicyHoldersBasedOnPolicyType(policyList);

122 }

123

124 // Test the countOfPolicyHoldersBasedOnThePolicyType method for an Empty List

125 @Test(expected=InvalidInsurancePolicyException.*class*)

126 *public* *void* test21CountOfPolicyHoldersBasedOnThePolicyTypeForEmptyList()throws InvalidInsurancePolicyException {

127 // Fill the code

128 policy=*new* PolicyManagement();

129 List<InsurancePolicy>emptyList=*new* ArrayList<>();

130 policy.countOfPolicyHoldersBasedOnPolicyType(emptyList);

131 }

132

133 // Test the countOfPolicyHoldersWithAdditonalBonus method

134 @Test

135 *public* *void* test22CountOfPolicyHoldersWithAdditonalBonus() throws ParseException, InvalidInsurancePolicyException {

136 // Fill the code

137 policy=*new* PolicyManagement();

138 assertEquals(2,policy.countOfPolicyHoldersWithAdditonalBonus(policyList,s.parse("2021-01-05")));

139 }

140

141 // Test the countOfPolicyHoldersWithAdditonalBonus method for an empty List

142 @Test(expected=InvalidInsurancePolicyException.*class*)

143 *public* *void* test23CountOfPolicyHoldersWithAdditonalBonusForEmptyList() throws ParseException ,InvalidInsurancePolicyException{

144 // Fill the code

145 policy=*new* PolicyManagement();

146 List<InsurancePolicy>emptyList=*new* ArrayList<>();

147 policy.countOfPolicyHoldersWithAdditonalBonus(emptyList,s.parse("2021-01-05"));

148 }

149

150 }

151

## Grade

Reviewed on Tuesday, 25 May 2021, 6:24 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