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

**Proposed grade: 92.5 / 100**  
**Result Description**  
[[-]](javascript:void(0);)**Grading and Feedback**

*DB connection - 20.0 / 20.0(Success)*

*Service calls - 70.0 / 70.0(Success)*

*Comments and best practices/standards - 2.5 / 10.0(Partial)*

Test Case Failed

ConstructionCostTimeEstimate/database.properties

1 #IF NEEDED, YOU CAN MODIFY THIS PROPERTY FILE

2 #ENSURE YOU ARE NOT CHANGING THE NAME OF THE PROPERTY

3 #YOU CAN CHANGE THE VALUE OF THE PROPERTY

4 #LOAD THE DETAILS OF DRIVER CLASS, URL, USERNAME AND PASSWORD using this properties file only.

5 #Do not hard code the values

6

7 drivername=com.mysql.jdbc.Driver

8 url=jdbc:mysql://localhost:3306/ctsuniversity

9 username=root

10 password=

ConstructionCostTimeEstimate/inputfeed.txt

1 CP001,2021-11-17,Residential,Apartments,100000

2 CP002,2021-09-15,Commercial,ShoppingComplex,200000

3 CP003,2021-08-10,Residential,Villa,1000000

4 CP004,2021-07-13,Commercial,CommunityHall,100000

5 CP005,2021-08-12,Infrastructural,FlyOver,1000000

ConstructionCostTimeEstimate/src/com/cts/conctes/client/CostAndTimeEstimation.java

1 *package* com.cts.conctes.client;

2

3 *import* java.io.FileReader;

4 *import* com.cts.conctes.exception.ConstructionEstimationException;

5 *import* com.cts.conctes.service.ConstructionProjectEstimationService;

6 @SuppressWarnings("all")

7 *public* *class* CostAndTimeEstimation {

8

9 *public* *static* *void* main(String[] args) throws ConstructionEstimationException

10 {

11 ConstructionProjectEstimationService cpeService = *new* ConstructionProjectEstimationService();

12 //WRITE YOUR CODE HERE

13 *boolean* isTrue = cpeService.addConstructionProjectDetails("inputfeed.txt");

14

15 *if*(isTrue){

16 System.out.println("All are added successfully into the database");

17 }

18

19 }

20 }

21

ConstructionCostTimeEstimate/src/com/cts/conctes/dao/CostAndTimeEstDAO.java

1 *package* com.cts.conctes.dao;

2

3 *import* java.sql.Connection;

4 *import* java.sql.PreparedStatement;

5 *import* java.sql.SQLException;

6 *import* java.sql.ResultSet;

7 *import* java.util.ArrayList;

8 *import* java.util.Date;

9

10 *import* com.cts.conctes.exception.ConstructionEstimationException;

11 *import* com.cts.conctes.model.ConstructionProject;

12 *import* com.cts.conctes.util.ApplicationUtil;

13

14

15

16 *public* *class* CostAndTimeEstDAO {

17

18 *public* *static* Connection connection = *null*;

19

20 *public* *boolean* insertConstructionProject(ArrayList <ConstructionProject> constProjects) throws ConstructionEstimationException {

21 *boolean* recordsAdded = *false*;

22

23 //WRITE YOUR CODE HERE

24 *int* index=0;

25 *int* size=constProjects.size();

26 connection=DBConnectionManager.getInstance().getConnection();

27 String query="insert into constructionproject values(?,?,?,?,?,?,?); ";

28 *for*(ConstructionProject cp:constProjects){

29 *try*{

30 PreparedStatement ps=connection.prepareStatement(query);

31 ps.setString(1,cp.getProjectId());

32 ps.setDate(2,ApplicationUtil.utilToSqlDateConverter(cp.getPlannedDOStart()));

33 ps.setString(3,cp.getTypeOfProject());

34 ps.setString(4,cp.getStructure());

35 ps.setDouble(5,cp.getAreaInSqFt());

36 ps.setDouble(6,cp.getEstimatedCostInlac());

37 ps.setDouble(7,cp.getEstimatedTimeInMonths());

38

39 *int* row=ps.executeUpdate();

40 *if*(row>0){

41 index=index+1;

42 }

43 *if*(index==size){

44 recordsAdded=*true*;

45 }

46 }

47 *catch*(SQLException e){

48 //TODO Auto-generated catch block

49 e.printStackTrace();

50 }

51 }

52

53 *return* recordsAdded;

54 }

55 *public* ArrayList<ConstructionProject>getConstructionProjectsData()throws ConstructionEstimationException{

56 ArrayList<ConstructionProject>consApplications=*new* ArrayList<ConstructionProject>();

57 connection=DBConnectionManager.getInstance().getConnection();

58 String query="select \* from constructionproject";

59

60 *try*{

61 PreparedStatement ps=connection.prepareStatement(query);

62 ResultSet rs=ps.executeQuery();

63 *while*(rs.next()){

64 String id=rs.getString(1);

65 Date d=rs.getDate(2);

66 String typeProject=rs.getString(3);

67 String structure=rs.getString(4);

68 *double* areaSqFt=rs.getDouble(5);

69 *double* costInLac=rs.getDouble(6);

70 *double* timeInMonths=rs.getDouble(7);

71 consApplications.add(*new* ConstructionProject(id,d,typeProject,structure,areaSqFt,costInLac,timeInMonths));

72 }

73 }*catch*(SQLException e){

74 //TODO Auto-generated catch block

75 e.printStackTrace();

76 }

77 *return* consApplications;

78 }

79

80 }

81

ConstructionCostTimeEstimate/src/com/cts/conctes/dao/DBConnectionManager.java

1 *package* com.cts.conctes.dao;

2 *import* java.io.FileReader;

3 *import* java.io.FileInputStream;

4 *import* java.io.FileNotFoundException;

5 *import* java.io.IOException;

6 *import* java.sql.Connection;

7 *import* java.sql.DriverManager;

8 *import* java.sql.SQLException;

9 *import* java.util.Properties;

10

11 *import* com.cts.conctes.exception.ConstructionEstimationException;

12

13

14 @SuppressWarnings("all")

15 *public* *class* DBConnectionManager {

16

17 *private* *static* Connection con = *null*;

18 *private* *static* DBConnectionManager instance;

19

20 *public* DBConnectionManager() throws ConstructionEstimationException

21 {

22

23 //WRITE YOUR CODE HERE

24 //return con;

25 }

26 *public* *static* DBConnectionManager getInstance() throws ConstructionEstimationException

27 {

28

29 //WRITE YOUR CODE HERE

30 *if*(instance==*null*){

31 instance=*new* DBConnectionManager();

32 }

33 *return* instance;

34 }

35 *public* Connection getConnection()

36 {

37 //WRITE YOUR CODE HERE

38 *try*{

39 Class.forName("com.mysql.jdbc.Driver");

40 }*catch*(ClassNotFoundException e){

41 //TODO Auto-generated catch block

42 e.printStackTrace();

43 }

44 Properties properties=*new* Properties();

45 FileReader f=*null*;

46 *try*{

47 f=*new* FileReader("database.properties");

48 }*catch*(FileNotFoundException e2){

49 //TODO Auto-generated catch block

50 e2.printStackTrace();

51 }

52 *try*{

53 properties.load(f);

54 }*catch*(IOException e1){

55 //TODO Auto-generated catch block

56 e1.printStackTrace();

57 }

58 String url=properties.getProperty("url");

59 String user=properties.getProperty("username");

60 String password=properties.getProperty("password");

61

62 *try*{

63 con=DriverManager.getConnection(url,user,password);

64 }*catch*(SQLException e){

65 //TODO Auto-generated catch block

66 e.printStackTrace();

67 }

68 *return* con;

69 }

70 }

71

ConstructionCostTimeEstimate/src/com/cts/conctes/exception/ConstructionEstimationException.java

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

2

3 *public* *class* ConstructionEstimationException *extends* Exception{

4

5 String strMsg1;

6 Throwable strMsg2;

7

8

9 *public* ConstructionEstimationException() {

10 *super*();

11 }

12

13

14

15 }

ConstructionCostTimeEstimate/src/com/cts/conctes/model/ConstructionProject.java

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

2

3 *import* java.util.Date;

4

5 *public* *class* ConstructionProject {

6

7 String projectId;

8 Date plannedDOStart;

9 String typeOfProject;

10 String structure;

11 *double* areaInSqFt;

12 *double* estimatedCostInlac;

13 *double* estimatedTimeInMonths;

14

15

16 *public* ConstructionProject() {

17 *super*();

18 }

19

20 *public* ConstructionProject(String projectId, Date plannedDOStart, String typeOfProject, String structure,

21 *double* areaInSqFt, *double* estimatedCostInlac, *double* estimatedTimeInMonths) {

22 *super*();

23 *this*.projectId = projectId;

24 *this*.plannedDOStart = plannedDOStart;

25 *this*.typeOfProject = typeOfProject;

26 *this*.structure = structure;

27 *this*.areaInSqFt = areaInSqFt;

28 *this*.estimatedCostInlac = estimatedCostInlac;

29 *this*.estimatedTimeInMonths = estimatedTimeInMonths;

30 }

31

32 *public* String getProjectId() {

33 *return* projectId;

34 }

35

36 *public* *void* setProjectId(String projectId) {

37 *this*.projectId = projectId;

38 }

39

40 *public* Date getPlannedDOStart() {

41 *return* plannedDOStart;

42 }

43

44 *public* *void* setPlannedDOStart(Date plannedDOStart) {

45 *this*.plannedDOStart = plannedDOStart;

46 }

47

48 *public* String getTypeOfProject() {

49 *return* typeOfProject;

50 }

51

52 *public* *void* setTypeOfProject(String typeOfProject) {

53 *this*.typeOfProject = typeOfProject;

54 }

55

56 *public* String getStructure() {

57 *return* structure;

58 }

59

60 *public* *void* setStructure(String structure) {

61 *this*.structure = structure;

62 }

63

64 *public* *double* getAreaInSqFt() {

65 *return* areaInSqFt;

66 }

67

68 *public* *void* setAreaInSqFt(*double* areaInSqFt) {

69 *this*.areaInSqFt = areaInSqFt;

70 }

71

72 *public* *double* getEstimatedCostInlac() {

73 *return* estimatedCostInlac;

74 }

75

76 *public* *void* setEstimatedCostInlac(*double* estimatedCostInlac) {

77 *this*.estimatedCostInlac = estimatedCostInlac;

78 }

79

80 *public* *double* getEstimatedTimeInMonths() {

81 *return* estimatedTimeInMonths;

82 }

83

84 *public* *void* setEstimatedTimeInMonths(*double* estimatedTimeInMonths) {

85 *this*.estimatedTimeInMonths = estimatedTimeInMonths;

86 }

87

88 @Override

89 *public* String toString() {

90 *return* "ConstructionProject [projectId=" + projectId + ", plannedDOStart=" + plannedDOStart + ", typeOfProject="

91 + typeOfProject + ", structure=" + structure + ", areaInSqFt=" + areaInSqFt + ", estimatedCostInlac="

92 + estimatedCostInlac + ", estimatedTimeInMonths=" + estimatedTimeInMonths + "]";

93 }

94

95 }

96

ConstructionCostTimeEstimate/src/com/cts/conctes/service/ConstructionProjectEstimationService.java

1 *package* com.cts.conctes.service;

2

3 *import* java.util.ArrayList;

4 *import* java.util.Date;

5 *import* java.util.List;

6

7 *import* com.cts.conctes.dao.CostAndTimeEstDAO;

8 *import* com.cts.conctes.exception.ConstructionEstimationException;

9 *import* com.cts.conctes.model.ConstructionProject;

10 *import* com.cts.conctes.util.ApplicationUtil;

11

12

13 *public* *class* ConstructionProjectEstimationService {

14

15 *public* *static* ArrayList <ConstructionProject> buildConstructionProjectList(List <String> consProjectRecords) {

16

17

18

19 *final* String COMMADELIMITER = ",";

20 ArrayList <ConstructionProject> consProjectRecordList = *new* ArrayList<ConstructionProject>();

21

22 //WRITE YOUR CODE HERE

23 *for*(String s:consProjectRecords){

24 String[] s1=s.split(COMMADELIMITER);

25 String id=s1[0];

26 Date d=ApplicationUtil.stringToDateConverter(s1[1]);

27 String typeproject=s1[2];

28 String structure=s1[3];

29 *double* areaInSqFt=Double.parseDouble(s1[4]);

30 *double* costs[]=estimateTimeAndCostForConstruction(typeproject,structure,areaInSqFt);

31 *double* estimatedCostInLac=costs[0];

32 *double* estimatedTimeInMon=costs[1];

33

34 ConstructionProject p=*new* ConstructionProject(id,d,typeproject,structure,areaInSqFt,estimatedCostInLac,estimatedTimeInMon);

35 consProjectRecordList.add(p);

36 }

37

38 *return* consProjectRecordList;

39 }

40

41 *public* *boolean* addConstructionProjectDetails(String inputFeed) throws ConstructionEstimationException {

42

43 //WRITE YOUR CODE HERE

44 ArrayList<ConstructionProject>p=buildConstructionProjectList(ApplicationUtil.readFile(inputFeed));

45 CostAndTimeEstDAO obj=*new* CostAndTimeEstDAO();

46 *if*(obj.insertConstructionProject(p)){

47 ArrayList<ConstructionProject> p1=obj.getConstructionProjectsData();

48 *for*(ConstructionProject cp:p1){

49 System.out.println(cp);

50 }

51 *return* *true*;

52 }

53 *return* *false*;

54 }

55

56 *public* *static* *double*[] estimateTimeAndCostForConstruction(String projectType,String structure,*double* areaInSqFt)

57 {

58

59 *double* costEstimateInRs=0.0,timeEstimateInMonths=0.0;

60 *double* costs[] = {costEstimateInRs,timeEstimateInMonths};

61 /\*

62 \* The Cost Estimate and

63 \*

64 Based on the type of the Project & the Structure , according to the required

65 area of Construction, the cost & time have to be calculated based on the base

66 data available in the table provided in the use case document:

67 For eg. If the Project Type is �Commercial� and the structure

68 is �Shopping Complex� the cost incurred for the construction of

69 per sq. ft is Rs.2600 and the time taken for the construction of

70 the 1000 sq ft of the same project is 0.23 Months,

71 calculation has to be performed on the similar basis

72 i.e Pro rata basis depending upon the type and the area of construction.

73

74 \*/

75

76 //WRITE YOUR CODE HERE

77 *if*(projectType.equals("Commercial")){

78 *if*(structure.equals("Shopping Complex")){

79 costs[0]=2600\*areaInSqFt;

80 costs[1]=0.23\*areaInSqFt/1000;

81 }

82 *else* *if*(structure.equals("ResApartments")){

83 costs[0]=2750\*areaInSqFt;

84 costs[1]=0.24\*areaInSqFt/1000;

85 }

86 *else*{

87 costs[0]=2600\*areaInSqFt;

88 costs[1]=0.2\*areaInSqFt/1000;

89 }

90 }

91

92 *else* *if*(projectType.equals("Infrastructural")){

93 *if*(structure.equals("Bridge")){

94 costs[0]=10000\*areaInSqFt;

95 costs[1]=0.25\*areaInSqFt/1000;

96 }

97 *else* *if*(structure.equals("FlyOver")){

98 costs[0]=14000\*areaInSqFt;

99 costs[1]=0.22\*areaInSqFt/1000;

100 }

101 *else*{

102 costs[0]=8000\*areaInSqFt;

103 costs[1]=0.25\*areaInSqFt/1000;

104 }

105 }

106 *else*{

107 *if*(structure.equals("House")){

108 costs[0]=2250\*areaInSqFt;

109 costs[1]=0.26\*areaInSqFt/1000;

110 }

111 *else* *if*(structure.equals("Apartments")){

112 costs[0]=2500\*areaInSqFt;

113 costs[1]=0.24\*areaInSqFt/1000;

114 }

115 *else*{

116 costs[0]=2750\*areaInSqFt;

117 costs[1]=0.23\*areaInSqFt/1000;

118 }

119 }

120 *return* costs;

121

122 }

123

124

125 }

126

ConstructionCostTimeEstimate/src/com/cts/conctes/skeleton/SkeletonValidator.java

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

2

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

4 *import* java.util.ArrayList;

5 *import* java.util.List;

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

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

8

9 *import* com.cts.conctes.model.ConstructionProject;

10

11

12

13 /\*\*

14 \* @author 222805

15 \*

16 \* This class is used to verify if the Code Skeleton is intact and not modified by participants thereby ensuring smooth auto evaluation

17 \*

18 \*/

19 *public* *class* SkeletonValidator {

20

21 *public* SkeletonValidator() {

22 validateClassName("com.cts.conctes.model.ConstructionProject");

23 validateClassName("com.cts.conctes.dao.CostAndTimeEstDAO");

24 validateClassName("com.cts.conctes.dao.DBConnectionManager");

25 validateClassName("com.cts.conctes.exception.ConstructionEstimationException");

26 validateClassName("com.cts.conctes.service.ConstructionProjectEstimationService");

27 validateClassName("com.cts.conctes.util.ApplicationUtil");

28

29 validateMethodSignature("insertConstructionProject:boolean","com.cts.conctes.dao.CostAndTimeEstDAO");

30 validateMethodSignature("getInstance:DBConnectionManager","com.cts.conctes.dao.DBConnectionManager");

31 validateMethodSignature("getConnection:Connection","com.cts.conctes.dao.DBConnectionManager");

32 validateMethodSignature("buildConstructionProjectList:ArrayList,addConstructionProjectDetails:boolean,estimateTimeAndCostForConstruction:double[]","com.cts.conctes.service.ConstructionProjectEstimationService");

33

34

35 }

36

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

38

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

40

41 *boolean* iscorrect = *false*;

42 *try* {

43 Class.forName(className);

44 iscorrect = *true*;

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

46

47 } *catch* (ClassNotFoundException e) {

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

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

50

51 } *catch* (Exception e) {

52 LOG.log(Level.SEVERE,

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

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

55 }

56 *return* iscorrect;

57

58 }

59

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

61 Class cls = *null*;

62 *try* {

63

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

65 *boolean* errorFlag = *false*;

66 String[] methodSignature;

67 String methodName = *null*;

68 String returnType = *null*;

69

70 *for* (String singleMethod : actualmethods) {

71 *boolean* foundMethod = *false*;

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

73

74 methodName = methodSignature[0];

75 returnType = methodSignature[1];

76 cls = Class.forName(className);

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

78 *for* (Method findMethod : methods) {

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

80 foundMethod = *true*;

81 *if* (!(findMethod.getReturnType().getSimpleName().equals(returnType))) {

82 errorFlag = *true*;

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

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

85

86 } *else* {

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

88 }

89

90 }

91 }

92 *if* (!foundMethod) {

93 errorFlag = *true*;

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

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

96 }

97

98 }

99 *if* (!errorFlag) {

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

101 }

102

103 } *catch* (Exception e) {

104 LOG.log(Level.SEVERE,

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

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

107 }

108 }

109

110 }

111

ConstructionCostTimeEstimate/src/com/cts/conctes/util/ApplicationUtil.java

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

2 *import* java.io.FileReader;

3 *import* java.io.BufferedReader;

4 *import* java.io.FileInputStream;

5 *import* java.io.FileNotFoundException;

6 *import* java.io.IOException;

7 *import* java.io.InputStreamReader;

8 *import* java.text.ParseException;

9 *import* java.text.SimpleDateFormat;

10 *import* java.util.ArrayList;

11 *import* java.util.Date;

12 *import* java.util.List;

13 *import* java.util.StringTokenizer;

14

15 *import* com.cts.conctes.exception.ConstructionEstimationException;

16

17

18 @SuppressWarnings("all")

19 *public* *class* ApplicationUtil {

20

21 *public* *static* List<String> readFile(String inputfeed) throws ConstructionEstimationException {

22 List<String> constructionProjects = *new* ArrayList<String>();

23

24 //WRITE YOUR CODE HERE

25 BufferedReader br=*null*;

26 *try*{

27 br=*new* BufferedReader(*new* FileReader(inputfeed));

28 }*catch*(FileNotFoundException e1){

29 //TODO Auto-generated catch block

30 e1.printStackTrace();

31 }

32 String s=*null*;

33 *try*{

34 s=br.readLine();

35 *while*(s!=*null*){

36 String s1[]=s.split(",");

37 //if(checkIfCurrentFinYearProject(stringToDateConverter(s1[1])))

38 constructionProjects.add(s);

39 s=br.readLine();

40 }

41 }*catch*(IOException e){

42 //TODO Auto-generated catch block

43 e.printStackTrace();

44 }

45 *return* constructionProjects;

46 }

47 *public* *static* java.sql.Date utilToSqlDateConverter(java.util.Date utDate) {

48 java.sql.Date sqlDate;

49

50 //WRITE YOUR CODE HERE

51 String s=*new* SimpleDateFormat("yyyy-MM-dd").format(utDate);

52 sqlDate=java.sql.Date.valueOf(s);

53 *return* sqlDate;

54 }

55

56 *public* *static* java.util.Date stringToDateConverter(String stringDate) {

57 Date strDate = *null*;

58

59 //WRITE YOUR CODE HERE

60 SimpleDateFormat sdf=*new* SimpleDateFormat("yyyy-MM-dd");

61 *try*{

62 strDate=sdf.parse(stringDate);

63 }*catch*(ParseException e){

64 //TODO Auto-generated catch block

65 e.printStackTrace();

66 }

67

68 *return* strDate;

69 }

70 *public* *static* *boolean* checkIfCurrentFinYearProject(Date dos)

71 {

72 *boolean* flag = *false*;

73 *int* givenYear,givenMonth;

74 givenYear = (dos.getYear()+1900);

75 givenMonth = dos.getMonth();

76 Date curDate = *new* Date();

77 *int* curYear,curMonth;

78 curYear = (curDate.getYear()+1900);

79 curMonth = curDate.getMonth();

80 *if*( curYear == givenYear)

81 {

82 *if*(((curMonth >=0)&&(curMonth <= 2)) && ((givenMonth >=0)&&(givenMonth <= 2)))

83 {

84 flag = *true*;

85 }

86 *else* *if*(((curMonth >=3)&&(curMonth <= 11)) && ((givenMonth >=3)&&(givenMonth <= 11)))

87 {

88 flag = *true*;

89 }

90 *else*

91 {

92 flag = *false*;

93 }

94 }

95 *else* *if*(curYear > givenYear)

96 {

97 *int* dif = curYear - givenYear;

98 *if*(dif == 1)

99 {

100 *if*(((curMonth >=0)&&(curMonth <= 2)) && ((givenMonth >=3)&&(givenMonth <= 11)))

101 {

102 flag = *true*;

103 }

104 *else* *if*(((curMonth >=3)&&(curMonth <= 11)) && ((givenMonth >=3)&&(givenMonth <= 11)))

105 {

106 flag = *false*;

107 }

108 *else*

109 {

110 flag = *false*;

111 }

112 }

113 *else*

114 {

115 flag = *false*;

116 }

117 }

118 *else* *if*(curYear < givenYear)

119 {

120 *int* dif = givenYear-curYear;

121 *if*(dif == 1)

122 {

123 *if*(((curMonth >=3)&&(curMonth <= 11)) && ((givenMonth >=0)&&(givenMonth <= 2)))

124 {

125 flag = *true*;

126 }

127 *else* *if*(((curMonth >=3)&&(curMonth <= 11)) && ((givenMonth >=3)&&(givenMonth <= 11)))

128 {

129 flag = *false*;

130 }

131 *else*

132 {

133 flag = *false*;

134 }

135 }

136 *else*

137 {

138 flag = *false*;

139 }

140 }

141 *else*

142 {

143 flag = *false*;

144 }

145

146 *return* flag;

147

148 }

149

150

151 }

152

Grade

Reviewed on Wednesday, 26 May 2021, 10:30 PM by Automatic grade  
**Grade** 92.5 / 100  
**Assessment report**  
[[+]](javascript:void(0);)**Grading and Feedback**