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

**Proposed grade: 95.0 / 100**  
**Result Description**  
[[-]](javascript:void(0);)**SOURCE CODE ANALYZER REPORT**

*Error Msg: Avoid if (x != y) ..; else ..;*

*Variable Name:*

*Priority: Medium*

*Class Name: SkeletonValidator*

*Error Msg: Possible unsafe assignment to a non-final static field in a constructor.*

*Variable Name: connection*

*Priority: Medium*

*Class Name: DBConnectionManager*

[[-]](javascript:void(0);)**Grading and Feedback**

*Collections and String - 10.0 / 10.0(Success)*

*Classes and Objects and JDBC API - 15.0 / 15.0(Success)*

*Operators,Control Statements and array - 10.0 / 10.0(Success)*

*Classes and Objects and Collections - 15.0 / 15.0(Success)*

*File IO - 10.0 / 10.0(Success)*

*Utility - 10.0 / 10.0(Success)*

*DB connection - 10.0 / 10.0(Success)*

*Java Date API, Operators and Control Statements - 10.0 / 10.0(Success)*

*Comments and best practices/standards - 5.0 / 10.0(Partial)*

Test Case Failed

UNOAdmission/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/uno\_admission

9 username=root

10 password=

UNOAdmission/inputFeed.txt

1 A001,S001,2020-01-15,EEE,2020-01-25,YES,YES,Approved

2 A002,S002,2020-02-04,MECH,2020-02-12,N0,YES,Approved

3 A003,S003,2020-04-23,CSE,2020-05-27,YES,NO,Approved

4 A004,S004,2020-07-16,IT,2020-07-24,NO,NO,Approved

5 A005,S005,2020-08-10,ECE,2020-08-11,YES,YES,Approved

6 A006,S006,2020-09-01,EEE,2020-09-10,YES,NO,Pending

7 A007,S007,2020-10-19,CIVIL,2020-10-28,N0,YES,Approved

UNOAdmission/src/com/cts/unoadm/dao/StudentAdmissionDAO.java

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

2

3 *import* java.util.ArrayList;

4 *import* java.util.List;

5

6 *import* com.cts.unoadm.exception.StudentAdmissionException;

7 *import* com.cts.unoadm.vo.StudentAdmission;

8 *import* com.cts.unoadm.util.\*;

9 *import* java.sql.\*;

10

11 /\*

12 CREATE DATABASE CTSUNO

13

14 CREATE TABLE UNO\_ADMISSION(

15 ADMISSION\_ID VARCHAR(50) PRIMARY KEY,

16 STUDENT\_CODE VARCHAR(50)NOT NULL,

17 DATE\_OF\_COUNSELING DATE NOT NULL,

18 DEPARTMENT\_NAME VARCHAR(20),

19 DATE\_OF\_ADMISSION DATE NOT NULL,

20 PREFER\_COLLEGE\_HOSTEL VARCHAR(20),

21 FIRST\_GRADUATE VARCHAR(20),

22 MANAGER\_APPROVAL VARCHAR(10),

23 ADMISSION\_FEE DOUBLE,

24 TUITION\_FEE DOUBLE,

25 HOSTEL\_FEE DOUBLE,

26 TOTAL\_COLLEGE\_FEE DOUBLE,

27 FINAL\_STATUS\_OF\_ADMISSION VARCHAR(20)

28 );

29 \*/

30 *public* *class* StudentAdmissionDAO {

31

32 *public* *boolean* addStudentAdmissionDetails(List<StudentAdmission> stdAdmissions) throws StudentAdmissionException {

33 *boolean* recordsAdded = *false*;

34 Connection con = DBConnectionManager.getInstance().getConnection();

35 PreparedStatement ps = *null*;

36 *try*{

37 //inserting values of list stdAdmissions into database

38 String query = "insert into UNO\_ADMISSION(ADMISSION\_ID,STUDENT\_CODE,DATE\_OF\_COUNSELING,DEPARTMENT\_NAME,DATE\_OF\_ADMISSION,PREFER\_COLLEGE\_HOSTEL,FIRST\_GRADUATE,MANAGER\_APPROVAL,ADMISSION\_FEE,TUITION\_FEE,HOSTEL\_FEE,TOTAL\_COLLEGE\_FEE,FINAL\_STATUS\_OF\_ADMISSION) values(?,?,?,?,?,?,?,?,?,?,?,?,?)";

39 *for*(StudentAdmission e:stdAdmissions) {

40 ps = con.prepareStatement(query);

41 ps.setString(1,e.getAdmissionId());

42 ps.setString(2,e.getStudentCode());

43 ps.setDate(3,ApplicationUtil.convertUtilToSqlDate(e.getDateOfCounseling()));

44 ps.setString(4,e.getDepartmentName());

45 ps.setDate(5,ApplicationUtil.convertUtilToSqlDate(e.getDateOfAdmission()));

46 ps.setString(6,e.getPreferCollegeHostel());

47 ps.setString(7,e.getFirstGraduate());

48 ps.setString(8,e.getManagerApproval());

49 ps.setDouble(9,e.getAdmissionFee());

50 ps.setDouble(10,e.getTuitionFee());

51 ps.setDouble(11,e.getHostelFee());

52 ps.setDouble(12,e.getTotalCollegeFee());

53 ps.setString(13,e.getFinalStatusOfAdmission());

54 *int* i = ps.executeUpdate();

55 *if*(i>0)

56 {

57 recordsAdded = *true*;

58 }

59 *else*

60 {

61 recordsAdded = *false*;

62 }

63 }

64

65

66 }

67 *catch*(SQLException e)

68 {

69 *try*{

70 con.rollback();

71 }*catch*(Exception e1){

72 e.printStackTrace();

73 }

74 }

75 *catch*(Exception e) {

76 e.printStackTrace();

77 //throw new StudentAdmissionException("Database Value Insertion Failed", e.getCause());

78 }

79 *finally*{

80 *try*{

81 ps.close();

82 con.close();

83 }*catch*(Exception e) {

84 e.printStackTrace();

85 //throw new StudentAdmissionException("Database Value Insertion Failed", e.getCause());

86 }

87 }

88 //code here

89

90 *return* recordsAdded;

91 }

92

93 *public* List<StudentAdmission> getAllStudentAdmissionDetails() throws StudentAdmissionException {

94

95 List<StudentAdmission> stdAdmissions = *new* ArrayList<StudentAdmission>();

96

97 //code here

98 //Retrieval of all records from database

99 String query = "select \* from UNO\_ADMISSION";

100 *try*(Connection con = DBConnectionManager.getInstance().getConnection();

101 Statement st = con.createStatement();

102 ResultSet rs = st.executeQuery(query);){

103 *while*(rs.next())

104 {

105 //storing retrieved records in object

106 StudentAdmission obj = *new* StudentAdmission();

107 obj.setAdmissionId(rs.getString(1));

108 obj.setStudentCode(rs.getString(2));

109 obj.setDateOfCounseling(*new* java.util.Date(rs.getDate(3).getTime()));

110 obj.setDepartmentName(rs.getString(4));

111 obj.setDateOfAdmission(*new* java.util.Date(rs.getDate(5).getTime()));

112 obj.setPreferCollegeHostel(rs.getString(6));

113 obj.setFirstGraduate(rs.getString(7));

114 obj.setManagerApproval(rs.getString(8));

115 obj.setAdmissionFee(rs.getDouble(9));

116 obj.setTuitionFee(rs.getDouble(10));

117 obj.setHostelFee(rs.getDouble(11));

118 obj.setTotalCollegeFee(rs.getDouble(12));

119 obj.setFinalStatusOfAdmission(rs.getString(13));

120 //adding StudentAdmission object into arraylist

121 stdAdmissions.add(obj);

122 }

123

124 }*catch*(Exception e)

125 {

126 e.printStackTrace();

127 //throw new StudentAdmissionException("Database Value Retrieval Failed", e.getCause());

128 }

129 *return* stdAdmissions;

130

131 }

132 }

UNOAdmission/src/com/cts/unoadm/exception/StudentAdmissionException.java

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

2

3 *public* *class* StudentAdmissionException *extends* Exception {

4

5 *private* *static* *final* *long* serialVersionUID = -1105431869622052445L;

6

7 /\*\*

8 \* @param message

9 \* @param cause

10 \*/

11 *public* StudentAdmissionException(String message, Throwable cause) {

12 *super*(message, cause);

13 }

14 }

15

UNOAdmission/src/com/cts/unoadm/main/MainApp.java

1 *package* com.cts.unoadm.main;

2

3 *import* com.cts.unoadm.skeletonvalidator.SkeletonValidator;

4 *import* com.cts.unoadm.service.\*;

5 *import* com.cts.unoadm.util.\*;

6 *public* *final* *class* MainApp {

7 *private* MainApp(){}

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

9 //Don't delete this code

10 //Skeletonvalidaton starts

11 *new* SkeletonValidator();

12 //Skeletonvalidation ends

13

14 //Write your code here..

15 *try*{

16 StudentAdmissionService service = *new* StudentAdmissionService();

17 System.out.println(service.addStudentAdmissionDetails("inputFeed.txt"));

18 System.out.println(service.searchStudentAdmission("A005"));

19 }*catch*(Exception e){e.printStackTrace();}

20 //List<StudentAdmission> studentAdmissionList = service.buildStudentAdmissionsList(ApplicationUtil.readFile("inputFeed.txt"));

21 /\*for(StudentAdmission e:studentAdmissionList)

22 {

23 System.out.println(e);

24 }\*/

25 }

26

27 }

28

UNOAdmission/src/com/cts/unoadm/service/StudentAdmissionService.java

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

2

3 *import* java.util.ArrayList;

4 *import* java.util.List;

5

6 *import* com.cts.unoadm.exception.StudentAdmissionException;

7 *import* com.cts.unoadm.vo.StudentAdmission;

8 *import* com.cts.unoadm.util.\*;

9 *import* com.cts.unoadm.dao.\*;

10 *public* *class* StudentAdmissionService {

11

12 /\*\*

13 \* @param empReimburseRecords

14 \* @return List<StudentAdmission>

15 \*/

16 *public* *static* List<StudentAdmission> buildStudentAdmissionsList(List<String> studentAdmissionRecords) {

17 List<StudentAdmission> studentAdmissionList = *new* ArrayList<StudentAdmission>();

18

19 //Code here

20 //storing each line into List of StudentAdmission objects

21 *for*(String e:studentAdmissionRecords) {

22 String res[] = e.split(",");

23 String admissionId = res[0];

24 String studentCode = res[1];

25 String dateOfCounseling = res[2];

26 String departmentName = res[3];

27 String dateOfAdmission = res[4];

28 String preferCollegeHostel = res[5];

29 String firstGraduate = res[6];

30 String managerApproval = res[7];

31 StudentAdmission obj = *new* StudentAdmission();

32 obj.setAdmissionId(admissionId);

33 obj.setStudentCode(studentCode);

34 //converting String to java.uti.Date

35 obj.setDateOfCounseling(ApplicationUtil.convertStringToDate(dateOfCounseling));

36 obj.setDepartmentName(departmentName);

37 //converting String to java.uti.Date

38 obj.setDateOfAdmission(ApplicationUtil.convertStringToDate(dateOfAdmission));

39 obj.setPreferCollegeHostel(preferCollegeHostel);

40 obj.setFirstGraduate(firstGraduate);

41 obj.setManagerApproval(managerApproval);

42 *double*[] studentAdmissionCosts = calculateTotalCollegeFee(preferCollegeHostel,firstGraduate,departmentName);

43 obj.setAdmissionFee(studentAdmissionCosts[0]);

44 obj.setTuitionFee(studentAdmissionCosts[1]);

45 obj.setHostelFee(studentAdmissionCosts[2]);

46 obj.setTotalCollegeFee(studentAdmissionCosts[3]);

47 obj.setFinalStatusOfAdmission("AdmissionSuccessfull");

48

49

50 studentAdmissionList.add(obj);

51 }

52

53 *return* studentAdmissionList;

54 }

55

56

57 *public* *boolean* addStudentAdmissionDetails(String inputFeed) throws StudentAdmissionException {

58

59 //Code here

60 List<StudentAdmission> studentAdmissionList = StudentAdmissionService.buildStudentAdmissionsList(ApplicationUtil.readFile(inputFeed));

61 StudentAdmissionDAO stdDao = *new* StudentAdmissionDAO();

62 *return* stdDao.addStudentAdmissionDetails(studentAdmissionList);

63 }

64

65 *public* *static* *double*[] calculateTotalCollegeFee(String preferCollegeHostel, String firstGraduate, String departmentName) {

66 *double*[] studentAdmissionCosts = *new* *double*[4];

67

68 //Code here..

69 *if*("YES".equals(preferCollegeHostel))

70 {

71 studentAdmissionCosts[2]=75000;

72 }

73 *else*{

74 studentAdmissionCosts[2]=0;

75 }

76 *if*("EEE".equals(departmentName)) {

77

78 studentAdmissionCosts[0]=30000;

79 studentAdmissionCosts[1]=45000;

80 }

81 *else* *if*("ECE".equals(departmentName)) {

82

83 studentAdmissionCosts[0]=30000;

84 studentAdmissionCosts[1]=50000;

85 }

86 *else* *if*("CSE".equals(departmentName)) {

87 studentAdmissionCosts[0]=30000;

88 studentAdmissionCosts[1]=45000;

89

90 }

91 *else* *if*("MECH".equals(departmentName)) {

92 studentAdmissionCosts[0]=30000;

93 studentAdmissionCosts[1]=55000;

94

95 }

96 *else* *if*("CIVIL".equals(departmentName)) {

97 studentAdmissionCosts[0]=30000;

98 studentAdmissionCosts[1]=50000;

99

100 }

101 *else* *if*("IT".equals(departmentName)) {

102 studentAdmissionCosts[0]=30000;

103 studentAdmissionCosts[1]=45000;

104

105 }

106 //for first graduate discount is there

107 *if*("YES".equals(firstGraduate)) {

108 studentAdmissionCosts[3]=studentAdmissionCosts[0]+studentAdmissionCosts[1]+studentAdmissionCosts[2]-20000;

109 }

110 *else*{

111 studentAdmissionCosts[3]=studentAdmissionCosts[0]+studentAdmissionCosts[1]+studentAdmissionCosts[2];

112 }

113 *return* studentAdmissionCosts;

114 }

115

116 *public* *boolean* searchStudentAdmission(String admissionId) throws StudentAdmissionException {

117 *boolean* status = *false*;

118

119 //Code here..

120 StudentAdmissionDAO stdDao = *new* StudentAdmissionDAO();

121 List<StudentAdmission> stdAdmissions = stdDao.getAllStudentAdmissionDetails();

122 *for*(StudentAdmission e:stdAdmissions) {

123 *if*(e.getAdmissionId().equals(admissionId)) {

124 status = *true*;

125 System.out.println(e);

126 *break*;

127 }

128 }

129 *return* status;

130 }

131 }

132

UNOAdmission/src/com/cts/unoadm/skeletonvalidator/SkeletonValidator.java

1 *package* com.cts.unoadm.skeletonvalidator;

2

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

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

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

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

7

8 /\*\*

9 \* @author t-aarti3

10 \* This class is used to verify if the Code Skeleton is intact and not

11 \* modified by participants thereby ensuring smooth auto evaluation

12 \* \*/

13

14 *public* *class* SkeletonValidator {

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

16 *public* SkeletonValidator() {

17 validateClassName("com.cts.unoadm.util.DBConnectionManager");

18 validateClassName("com.cts.unoadm.util.ApplicationUtil");

19 validateClassName("com.cts.unoadm.service.StudentAdmissionService");

20 validateClassName("com.cts.unoadm.dao.StudentAdmissionDAO");

21 validateClassName("com.cts.unoadm.vo.StudentAdmission");

22 validateClassName("com.cts.unoadm.exception.StudentAdmissionException");

23

24

25 validateMethodSignature(

26 "addStudentAdmissionDetails:boolean,getAllStudentAdmissionDetails:List",

27 "com.cts.unoadm.dao.StudentAdmissionDAO");

28 validateMethodSignature(

29 "buildStudentAdmissionsList:List,addStudentAdmissionDetails:boolean,calculateTotalCollegeFee:double[],searchStudentAdmission:boolean",

30 "com.cts.unoadm.service.StudentAdmissionService");

31 validateMethodSignature(

32 "readFile:List,convertUtilToSqlDate:Date,convertStringToDate:Date,checkIfValidAdmission:boolean",

33 "com.cts.unoadm.util.ApplicationUtil");

34 validateMethodSignature(

35 "getConnection:Connection,getInstance:DBConnectionManager",

36 "com.cts.unoadm.util.DBConnectionManager");

37

38

39

40 }

41

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

43

44 *boolean* iscorrect = *false*;

45 *try* {

46 Class.forName(className);

47 iscorrect = *true*;

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

49

50 } *catch* (ClassNotFoundException e) {

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

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

53

54 } *catch* (Exception e) {

55 LOG.log(Level.SEVERE,

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

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

58 }

59 *return* iscorrect;

60 }

61

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

63 Class cls = *null*;

64 *try* {

65

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

67 *boolean* errorFlag = *false*;

68 String[] methodSignature;

69 String methodName = *null*;

70 String returnType = *null*;

71

72 *for* (String singleMethod : actualmethods) {

73 *boolean* foundMethod = *false*;

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

75

76 methodName = methodSignature[0];

77 returnType = methodSignature[1];

78 cls = Class.forName(className);

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

80 *for* (Method findMethod : methods) {

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

82 foundMethod = *true*;

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

84 errorFlag = *true*;

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

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

87

88 } *else* {

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

90 }

91

92 }

93 }

94 *if* (!foundMethod) {

95 errorFlag = *true*;

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

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

98 }

99

100 }

101 *if* (!errorFlag) {

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

103 }

104

105 } *catch* (Exception e) {

106 LOG.log(Level.SEVERE,

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

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

109 }

110 }

111

112 }

113

UNOAdmission/src/com/cts/unoadm/util/ApplicationUtil.java

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

2

3

4 *import* java.util.\*;

5 *import* java.io.\*;

6 *import* java.text.\*;

7

8 *import* com.cts.unoadm.exception.StudentAdmissionException;

9

10 *public* *final* *class* ApplicationUtil {

11

12 /\*\*

13 \* @param fileName

14 \* @return List<String>

15 \* @throws StudentAdmissionException

16 \*/

17

18 *private* ApplicationUtil(){}

19 *public* *static* List<String> readFile(String fileName) throws StudentAdmissionException {

20 List<String> studentAdmissionList = *new* ArrayList<String>();

21 //Code here..

22 FileReader fr = *null*;

23 BufferedReader br = *null*;

24 *try*{

25 fr=*new* FileReader(fileName);

26 br = *new* BufferedReader(fr);

27 String line = *null*;

28 *while*((line=br.readLine())!=*null*)

29 {

30 String []res = line.split(",");

31 String managerApproval = res[7];

32 Date dtOfCounseling = convertStringToDate(res[2]);

33 Date dtOfAdmission = convertStringToDate(res[4]);

34 *if*(checkIfValidAdmission(dtOfCounseling,dtOfAdmission,managerApproval))

35 {

36 studentAdmissionList.add(line);

37 }

38

39 }

40 }*catch*(Exception e){e.printStackTrace();}

41

42 *return* studentAdmissionList;

43 }

44

45 /\*\*

46 \* @param util

47 \* Date

48 \* @return sql Date

49 \*/

50 *public* *static* java.sql.Date convertUtilToSqlDate(java.util.Date uDate) {

51

52 java.sql.Date sDate = *new* java.sql.Date(uDate.getTime());

53

54 //Code here..

55

56

57 *return* sDate;

58 }

59

60 /\*\*

61 \* @param inDate

62 \* @return Date

63 \*/

64 *public* *static* Date convertStringToDate(String inDate) {

65

66 //Code here..

67 *try*{

68 SimpleDateFormat format = *new* SimpleDateFormat("yyyy-MM-dd",Locale.ENGLISH);

69 *return* format.parse(inDate);

70 }*catch*(Exception e){e.printStackTrace();

71 *return* *null*;

72 }

73 }

74

75 *public* *static* *boolean* checkIfValidAdmission(Date dtOfCounseling, Date dtOfAdmission, String manager) {

76 *boolean* admissionValidity = *false*;

77

78 //Code here..

79 *if*("Approved".equals(manager) && ((dtOfAdmission.getTime() - dtOfCounseling.getTime())/(1000\*60\*60\*24))%365<= 10)

80 {

81 admissionValidity = *true*;

82 }

83

84 *return* admissionValidity;

85 }

86 }

87

UNOAdmission/src/com/cts/unoadm/util/DBConnectionManager.java

1 /\*\*

2 \* Don't change this code

3 \*/

4 *package* com.cts.unoadm.util;

5 *import* java.io.FileInputStream;

6 *import* java.io.FileNotFoundException;

7 *import* java.io.IOException;

8 *import* java.sql.Connection;

9 *import* java.sql.DriverManager;

10 *import* java.sql.SQLException;

11 *import* java.util.Properties;

12

13 *import* com.cts.unoadm.exception.StudentAdmissionException;

14

15

16 *public* *final* *class* DBConnectionManager {

17

18 *public* *static* *final* String PROPERTY\_FILE = "database.properties";

19 *public* *static* *final* String DRIVER = "drivername";

20 *public* *static* *final* String URL = "url";

21 *public* *static* *final* String USER\_NAME = "username";

22 *public* *static* *final* String PASSWORD = "password";

23

24 *private* *static* Connection connection = *null*;

25 *private* *static* Properties props = *null*;

26 *private* *static* DBConnectionManager instance = *null*;

27 /\*\*

28 \* @throws StudentAdmissionException

29 \*/

30 *private* DBConnectionManager() throws StudentAdmissionException {

31 loadProperties();

32 *try* {

33 Class.forName(props.getProperty(DRIVER));

34 connection = DriverManager.getConnection(props.getProperty(URL), props.getProperty(USER\_NAME),

35 props.getProperty(PASSWORD));

36 } *catch* (ClassNotFoundException ex) {

37 ex.printStackTrace();

38 //throw new StudentAdmissionException("Could not find Driver class ", ex.getCause());

39 } *catch* (SQLException e) {

40 e.printStackTrace();

41 //throw new StudentAdmissionException("Database Connection Creation Failed", e.getCause());

42 }

43 *catch*(Exception e)

44 {

45 e.printStackTrace();

46 //throw new StudentAdmissionException("Database Connection Creation Failed", e.getCause());

47 }

48 }

49

50 /\*\*

51 \* @return Connection

52 \*/

53 *public* Connection getConnection() {

54 *return* connection;

55 }

56

57 /\*\*

58 \* @return DBConnectionManager

59 \* @throws StudentAdmissionException

60 \*/

61 *public* *static* DBConnectionManager getInstance() throws StudentAdmissionException {

62

63 // Code here

64

65 instance = *new* DBConnectionManager();

66

67 *return* instance;

68 }

69

70 /\*\*

71 \* @throws StudentAdmissionException

72 \*/

73 *private* *void* loadProperties() throws StudentAdmissionException {

74 FileInputStream inputStream = *null*;

75 *try* {

76 inputStream = *new* FileInputStream(PROPERTY\_FILE);

77 props = *new* Properties();

78 props.load(inputStream);

79 } *catch* (FileNotFoundException e) {

80 e.printStackTrace();

81 //throw new StudentAdmissionException("Database Property File Not Found", e.getCause());

82 } *catch* (IOException e) {

83 e.printStackTrace();

84 //throw new StudentAdmissionException("Exception during property file I/O", e.getCause());

85 } *finally* {

86 *if* (inputStream != *null*) {

87 *try* {

88 inputStream.close();

89 } *catch* (IOException e) {

90 e.printStackTrace();

91 //throw new StudentAdmissionException("Exception during property file I/O", e.getCause());

92 }

93 }

94 }

95 }

96 }

97

98

UNOAdmission/src/com/cts/unoadm/vo/StudentAdmission.java

1 /\*

2 \* Don't change this code

3 \*/

4 *package* com.cts.unoadm.vo;

5

6 *import* java.util.Date;

7

8 *public* *class* StudentAdmission {

9 String admissionId;

10 String studentCode;

11 Date dateOfCounseling;

12 String departmentName;

13 Date dateOfAdmission;

14 String preferCollegeHostel;

15 String firstGraduate;

16 String managerApproval;

17 *double* admissionFee;

18 *double* tuitionFee;

19 *double* hostelFee;

20 *double* totalCollegeFee;

21 String finalStatusOfAdmission;

22

23 *public* StudentAdmission() {

24 *super*();

25 }

26

27 *public* StudentAdmission(String admissionId, String studentCode, Date dateOfCounseling, String departmentName,

28 Date dateOfAdmission, String preferCollegeHostel, String firstGraduate, String managerApproval,

29 *double* admissionFee, *double* tuitionFee, *double* hostelFee, *double* totalCollegeFee,

30 String finalStatusOfAdmission) {

31 *super*();

32 *this*.admissionId = admissionId;

33 *this*.studentCode = studentCode;

34 *this*.dateOfCounseling = dateOfCounseling;

35 *this*.departmentName = departmentName;

36 *this*.dateOfAdmission = dateOfAdmission;

37 *this*.preferCollegeHostel = preferCollegeHostel;

38 *this*.firstGraduate = firstGraduate;

39 *this*.managerApproval = managerApproval;

40 *this*.admissionFee = admissionFee;

41 *this*.tuitionFee = tuitionFee;

42 *this*.hostelFee = hostelFee;

43 *this*.totalCollegeFee = totalCollegeFee;

44 *this*.finalStatusOfAdmission = finalStatusOfAdmission;

45 }

46

47 *public* String getAdmissionId() {

48 *return* admissionId;

49 }

50

51 *public* *void* setAdmissionId(String admissionId) {

52 *this*.admissionId = admissionId;

53 }

54

55 *public* String getStudentCode() {

56 *return* studentCode;

57 }

58

59 *public* *void* setStudentCode(String studentCode) {

60 *this*.studentCode = studentCode;

61 }

62

63 *public* Date getDateOfCounseling() {

64 *return* dateOfCounseling;

65 }

66

67 *public* *void* setDateOfCounseling(Date dateOfCounseling) {

68 *this*.dateOfCounseling = dateOfCounseling;

69 }

70

71 *public* String getDepartmentName() {

72 *return* departmentName;

73 }

74

75 *public* *void* setDepartmentName(String departmentName) {

76 *this*.departmentName = departmentName;

77 }

78

79 *public* Date getDateOfAdmission() {

80 *return* dateOfAdmission;

81 }

82

83 *public* *void* setDateOfAdmission(Date dateOfAdmission) {

84 *this*.dateOfAdmission = dateOfAdmission;

85 }

86

87 *public* String getPreferCollegeHostel() {

88 *return* preferCollegeHostel;

89 }

90

91 *public* *void* setPreferCollegeHostel(String preferCollegeHostel) {

92 *this*.preferCollegeHostel = preferCollegeHostel;

93 }

94

95 *public* String getFirstGraduate() {

96 *return* firstGraduate;

97 }

98

99 *public* *void* setFirstGraduate(String firstGraduate) {

100 *this*.firstGraduate = firstGraduate;

101 }

102

103 *public* String getManagerApproval() {

104 *return* managerApproval;

105 }

106

107 *public* *void* setManagerApproval(String managerApproval) {

108 *this*.managerApproval = managerApproval;

109 }

110

111 *public* *double* getAdmissionFee() {

112 *return* admissionFee;

113 }

114

115 *public* *void* setAdmissionFee(*double* admissionFee) {

116 *this*.admissionFee = admissionFee;

117 }

118

119 *public* *double* getTuitionFee() {

120 *return* tuitionFee;

121 }

122

123 *public* *void* setTuitionFee(*double* tuitionFee) {

124 *this*.tuitionFee = tuitionFee;

125 }

126

127 *public* *double* getHostelFee() {

128 *return* hostelFee;

129 }

130

131 *public* *void* setHostelFee(*double* hostelFee) {

132 *this*.hostelFee = hostelFee;

133 }

134

135 *public* *double* getTotalCollegeFee() {

136 *return* totalCollegeFee;

137 }

138

139 *public* *void* setTotalCollegeFee(*double* totalCollegeFee) {

140 *this*.totalCollegeFee = totalCollegeFee;

141 }

142

143 *public* String getFinalStatusOfAdmission() {

144 *return* finalStatusOfAdmission;

145 }

146

147 *public* *void* setFinalStatusOfAdmission(String finalStatusOfAdmission) {

148 *this*.finalStatusOfAdmission = finalStatusOfAdmission;

149 }

150

151 @Override

152 *public* String toString() {

153 *return* "Student Admission Details: [admissionId=" + admissionId + ", studentCode=" + studentCode + ", dateOfCounseling="

154 + dateOfCounseling + ", departmentName=" + departmentName + ", dateOfAdmission=" + dateOfAdmission + ", preferCollegeHostel="

155 + preferCollegeHostel + ", firstGraduate=" + firstGraduate + ", managerApproval=" + managerApproval

156 + ", admissionFee=" + admissionFee + ", tuitionFee=" + tuitionFee + ", hostelFee=" + hostelFee + ", totalCollegeFee=" + totalCollegeFee

157 + ", finalStatusOfAdmission=" + finalStatusOfAdmission + "]";

158 }

159

160 }

161