Appendix 3

➢ CreatePrintClass File

```
package xx.xx.grading.system;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Arrays;

import java.util.Comparator;

import java.util.Set;


public class CreatePrintClassFile {


        // this class will create the HTML file for printing class records


        protected static final String printFileName = "data/print.html";
```

```
private Object[][] elements;
SchoolClass sc;
Subject subject;

protected CreatePrintClassFile(Set<Grade> gradeList,
        Set<Student> studentsOfClass, SchoolClass schoolClass, Subject sub) {

    this.sc = schoolClass;
    this.subject = sub;

    int listSize = studentsOfClass.size();
    elements = new Object[listSize][6];

    int row = 0;
    for (Student s : studentsOfClass) {
        elements[row][0] = s;
```

```
for (Grade g : gradeList) {
        if (g.getStudent().getStudentID() == s.getStudentID()
                && g.getSubject().getSubjectID() == sub.getSubjectID()) {
            int trimester = g.getTrimester().getTrimesterID();
            elements[row][trimester] = g.getMark();
        }
}

if (elements[row][1] != null && elements[row][2] != null
            && elements[row][3] != null && elements[row][4] != null) {
        int markFirstTrimester = (int) elements[row][1];
        int markSecondTrimester = (int) elements[row][2];
        int markThirdTrimester = (int) elements[row][3];
        int markFourthTrimester = (int) elements[row][4];
        double avg = (double) (markFirstTrimester + markSecondTrimester
                + markThirdTrimester + markFourthTrimester) / 4.0;
```

```
                elements[row][5] = avg;
            }
            row++;
        }
        Arrays.sort(elements, new Comparator<Object[]>() {
            @Override
            public int compare(final Object[] row1, final Object[] row2) {
                final Student s1 = (Student) row1[0];
                final Student s2 = (Student) row2[0];
                final String value1 = String.format("%s %s", s1.getSurname(),
                        s1.getName());
                final String value2 = String.format("%s %s", s2.getSurname(),
                        s2.getName());
                return value1.compareTo(value2);
            }
        });
```

```
}

protected void writeFile() {
        try (FileWriter f = new FileWriter(printFileName);
                        BufferedWriter writeFile = new BufferedWriter(f)) {

                writeFile
                                .write("<html><head><title>Grades For Class</title></head><body><h1>Student Grades</h1>");
                writeFile
                                .write("<I>Right click on the page in order to print.</I>");

                writeFile.write("<h3>Class: " + sc.toString() + "</h3>");
                writeFile.write("<h3>Subject: " + subject.toString() + "</h3>");
                writeFile
                                .write("<table border=\"4\"><tr><th>Student Name (ID)</th><th>1st Trimester</th>");
```

```
writeFile
                .write("<th>2nd Trimester</th><th>3rd Trimester</th><th>Written Grade</th><th>Average</th></tr>");


for (int row = 0; row < elements.length; row++) {
        writeFile.write("<tr>");
        writeFile.write("<th align=\"left\">"
                        + ((Student) elements[row][0]).toString() + "</th>");
        for (int i = 1; i <= 4; i++) {
                if (elements[row][i] != null)
                        writeFile.write("<th align=\"right\">"
                                        + elements[row][i].toString() + "</th>");
                else
                        writeFile.write("<th> </th>");
        }
        if (elements[row][5] != null)
                writeFile.write("<th align=\"right\">"
```

```
                                        + String.format("%.2f", (double) elements[row][5])
                                        + "</th>");
                        else
                                writeFile.write("<th>" + " " + "</th>");
                        writeFile.write("</tr>");
                }
                writeFile.write("</table></body></html>");

                writeFile.flush();

        } catch (IOException e) {
                System.out.println("Cannot write subjects file");
        }
    }
}
```

➢ CreatePrintIndividualStudentFile

```
package xx.xx.grading.system;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.Comparator;
import java.util.HashSet;
import java.util.Set;

import javax.swing.JOptionPane;

public class CreatePrintIndividualStudentFile {

        // individual student records need to be printed
```

```
// HTML file created


protected static final String printIndividualStudent = "data/print.html";
Student student;
Set<Grade> grade = new HashSet<>();
Object[][] elements;
Set<Subject> sub = new HashSet<>();
SchoolClass sc;


protected CreatePrintIndividualStudentFile(Student s,
                Set<Grade> studentGrades, SchoolClass schoolClass) {
        this.student = s;
        this.grade = studentGrades;
        this.sc = schoolClass;

        for (Grade grade : studentGrades)
```

```
            sub.add(grade.getSubject());
    if (sub.size() == 0) {
            JOptionPane.showMessageDialog(null,
                        "There are currently no subjects taken by the student");
            return;
    }
    elements = new Object[sub.size()][6];


    int row1 = 0;
    for (Subject subject : sub) {
            elements[row1][0] = subject;
            row1++;
    }


    for (Grade g : studentGrades)
            for (int row = 0; row < elements.length; row++)
```

```
                if (g.getSubject().getSubjectID() == ((Subject) elements[row][0])
                                .getSubjectID()) {
                        int trimester = g.getTrimester().getTrimesterID();
                        elements[row][trimester] = g.getMark();
                        break;
                }

for (int row = 0; row < elements.length; row++)
        if (elements[row][1] != null && elements[row][2] != null
                        && elements[row][3] != null && elements[row][4] != null) {
                int markFirstTrimester = (int) elements[row][1];
                int markSecondTrimester = (int) elements[row][2];
                int markThirdTrimester = (int) elements[row][3];
                int markFourthTrimester = (int) elements[row][4];
                double avg = (double) (markFirstTrimester + markSecondTrimester
                                + markThirdTrimester + markFourthTrimester) / 4.0;
```

```
                elements[row][5] = avg;
        }
// sorting algorithm so that the the students printed on the browser
// will appear in an alphabetical order, making it easier for the client
// to locate the student needed
Arrays.sort(elements, new Comparator<Object[]>() {
        @Override
        public int compare(final Object[] row1, final Object[] row2) {
                final Student s1 = (Student) row1[0];
                final Student s2 = (Student) row2[0];
                final String value1 = String.format("%s %s", s1.getSurname(),
                                s1.getName());
                final String value2 = String.format("%s %s", s2.getSurname(),
                                s2.getName());
                return value1.compareTo(value2);
        }
```

```
        });
}

protected void writeFile() {

        try (FileWriter f = new FileWriter(printIndividualStudent);
                        BufferedWriter writeFile = new BufferedWriter(f)) {

                writeFile
                                .write("<html><head><title>Student File</title></head><body><h1>Student File</h1>");
                writeFile
                                .write("<I>Right click on the page in order to print.</I>");

                writeFile.write("<h3>Class: " + sc.toString() + "</h3>");
                writeFile.write("<h3>Student: " + student.toString() + "</h3>");
                writeFile
```

```
                .write("<table border=\"4\"><tr><th>Subject (ID)</th><th>1st Trimester</th>");
writeFile
                .write("<th>2nd Trimester</th><th>3rd Trimester</th><th>Written Grade</th><th>Average</th></tr>");


for (int row = 0; row < elements.length; row++) {
        writeFile.write("<tr>");
        writeFile.write("<th align=\"left\">"
                        + ((Subject) elements[row][0]).toString() + "</th>");
        for (int i = 1; i <= 4; i++) {
                if (elements[row][i] != null)
                        writeFile.write("<th align=\"right\">"
                                        + elements[row][i].toString() + "</th>");
                else
                        writeFile.write("<th> </th>");
        }
        if (elements[row][5] != null)
```

```
                        writeFile.write("<th align=\"right\">"
                                + String.format("%.2f", (double) elements[row][5])
                                + "</th>");
                    else
                        writeFile.write("<th>" + " " + "</th>");
                    writeFile.write("</tr>");
                }
                writeFile.write("</table></body></html>");

                writeFile.flush();

            } catch (IOException e) {
                System.out.println("Cannot create student file");
            }

}
```

```
}
```

➢ CreatePrintStudentFile

```
package xx.xx.grading.system;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

public class CreatePrintStudentFile {

        // another printing class

        protected static final String printStudent = "data/print.html";
```

```
SchoolClass sc;
Set<Student> students = new HashSet<>();
Object[] elements;


protected CreatePrintStudentFile(Set<Student> studentsOfClass,
                SchoolClass schoolClass) {
        this.sc = schoolClass;
        this.students = studentsOfClass;
        elements = new Object[students.size()];


        int row = 0;
        for (Student s : studentsOfClass) {
                elements[row] = s;
                row++;
        }
}
```

```
protected void writeFile() {
        try (FileWriter f = new FileWriter(printStudent);
                        BufferedWriter writeFile = new BufferedWriter(f)) {


                writeFile.write("<html><head><title>Students of " + sc.toString()
                                + "</title></head><body>");
                writeFile.write("<h3>Students of " + sc.toString() + "</h3>");
                writeFile
                                .write("<table border=\"4\"><tr><th>Student Name (ID)</th></tr>");
                for (int row = 0; row < elements.length; row++) {
                        writeFile.write("<tr>");
                        writeFile.write("<th>" + elements[row].toString()
                                        + "</th></tr>");
                }
                writeFile.write("</table></body></html>");
```

```
                writeFile.flush();


            } catch (IOException e) {
                System.out.println("Cannot write subjects file");
            }
        }
}
```

➢ FirstColumnReadOnlyTableModel

```
package xx.xx.grading.system;

import javax.swing.table.DefaultTableModel;

// in some cases (i.e. when the grades are printed) one column needs to be non - editable
// in order to avoid changes in major elements
```

```
@SuppressWarnings("serial")
public class FirstColumnReadOnlyTableModel extends DefaultTableModel {

	public FirstColumnReadOnlyTableModel(Object[][] data, Object[] columnNames) {
		super(data, columnNames);
	}

	@Override
	public boolean isCellEditable(int row, int column) {

		return column != 0;
	}
}
```

- **Grade**

```
package xx.xx.grading.system;

public class Grade {

	// parameters of the object Grade

	private Student student;
	private Subject subject;
	private Trimester trimester;
	private int mark;

	public Student getStudent() {
		return student;
	}

	public void setStudent(Student student) {
```

```
        this.student = student;
}

public Subject getSubject() {
        return subject;
}

public void setSubject(Subject subject) {
        this.subject = subject;
}

public Trimester getTrimester() {
        return trimester;
}

public void setTrimester(Trimester trimester) {
```

```
        this.trimester = trimester;
}

public int getMark() {
        return mark;
}

public void setMark(int mark) {
        this.mark = mark;
}

@Override
public String toString() {
        return String.format("%d %d %d %d\n", student.getStudentID(),
                        subject.getSubjectID(), trimester.getTrimesterID(), mark);
}
```

```
        public String toPrintLine(){
                return String.format("%d~%d~%d~%d\n", student.getStudentID(),
                                subject.getSubjectID(), trimester.getTrimesterID(), mark);
        }

}
```

➢ GradeForm

```
package xx.xx.grading.system;

import java.awt.BorderLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.HeadlessException;
import java.awt.Window;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.HashSet;
import java.util.Set;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;

@SuppressWarnings("serial")
public class GradeForm extends JDialog implements ActionListener {
```

```
// when it's time to print the grades, this method is called from mainForm


private JLabel labelClass, labelTrimester, labelSubject, labelClassName,
                    labelTrimesterNumber, labelSubjectName;
private JButton buttonOK, buttonCancel;
private JTable studentGradesTable;
private Set<Grade> existingGrades = new HashSet<>();
private Set<Grade> gradeListCopy;
private Subject subCopy;
private Trimester trimesterCopy;


public GradeForm(Window owner, Set<Student> studentsOfClass,
                    Set<Grade> gradeList, SchoolClass schoolClass, Subject sub,
                    Trimester trimester) throws HeadlessException {


        super(owner, ModalityType.APPLICATION_MODAL);
```

```
this.gradeListCopy = gradeList;
this.subCopy = sub;
this.trimesterCopy = trimester;

setTitle("Grades for Class");
setSize(700, 300);
setDefaultCloseOperation(HIDE_ON_CLOSE);

JPanel generalPanel = new JPanel(new BorderLayout());
add(generalPanel);
JPanel panel = new JPanel(new GridBagLayout());
generalPanel.add(panel, BorderLayout.PAGE_START);

GridBagConstraints c1 = new GridBagConstraints();
c1.gridx = 0;
c1.gridy = 0;
```

```
c1.anchor = GridBagConstraints.WEST;
labelClass = new JLabel("Class: ");
panel.add(labelClass, c1);

GridBagConstraints c2 = new GridBagConstraints();
c2.gridx = 0;
c2.gridy = 1;
c2.anchor = GridBagConstraints.WEST;
labelSubject = new JLabel("Subject: ");
panel.add(labelSubject, c2);

GridBagConstraints c3 = new GridBagConstraints();
c3.gridx = 0;
c3.gridy = 2;
c3.anchor = GridBagConstraints.WEST;
labelTrimester = new JLabel("Trimester: ");
```

```
panel.add(labelTrimester, c3);


GridBagConstraints c4 = new GridBagConstraints();
c4.gridx = 1;
c4.gridy = 0;
c4.anchor = GridBagConstraints.WEST;
labelClassName = new JLabel(schoolClass.toString());
panel.add(labelClassName, c4);

GridBagConstraints c5 = new GridBagConstraints();
c5.gridx = 1;
c5.gridy = 1;
c5.anchor = GridBagConstraints.WEST;
labelSubjectName = new JLabel(sub.toString());
panel.add(labelSubjectName, c5);
```

```
GridBagConstraints c6 = new GridBagConstraints();
c6.gridx = 1;
c6.gridy = 2;
c6.anchor = GridBagConstraints.WEST;
labelTrimesterNumber = new JLabel(trimester.toString());
panel.add(labelTrimesterNumber, c6);


for (Grade grade : gradeList) {
        if (sub.getSubjectID() == grade.getSubject().getSubjectID()
                        && trimester.getTrimesterID() == grade.getTrimester()
                                        .getTrimesterID()) {
                for (Student s : studentsOfClass)
                        if (grade.getStudent().getStudentID() == s.getStudentID()) {
                                existingGrades.add(grade);
                        }
        }
```

```
}

int listSize = studentsOfClass.size();
String[] columnNames = { "Students", "Grades" };
Object[][] elements = new Object[listSize][columnNames.length];

int row = 0;
for (Student s : studentsOfClass) {
        elements[row][0] = s;
        for (Grade g : existingGrades) {
                if (g.getStudent().getStudentID() == s.getStudentID()) {
                        elements[row][1] = g.getMark();
                        break;
                }
        }
        row++;
```

```
}

FirstColumnReadOnlyTableModel model = new FirstColumnReadOnlyTableModel(
                    elements, columnNames);
studentGradesTable = new JTable(model);

JScrollPane scrollPane = new JScrollPane(studentGradesTable);
generalPanel.add(scrollPane, BorderLayout.CENTER);

JPanel panelButtons = new JPanel(new GridBagLayout());
generalPanel.add(panelButtons, BorderLayout.PAGE_END);

GridBagConstraints c8 = new GridBagConstraints();
c8.gridx = 0;
c8.gridy = 3;
c8.anchor = GridBagConstraints.WEST;
```

```
        buttonOK = new JButton("OK");
        panelButtons.add(buttonOK, c8);
        buttonOK.addActionListener(this);

        GridBagConstraints c9 = new GridBagConstraints();
        c9.gridx = 1;
        c9.gridy = 3;
        c9.anchor = GridBagConstraints.EAST;
        buttonCancel = new JButton("Cancel");
        panelButtons.add(buttonCancel, c9);
        buttonCancel.addActionListener(this);

        setVisible(true);
}

@Override
```

```
public void actionPerformed(ActionEvent arg) {


        if (arg.getSource().equals(buttonCancel)) {
                int answer = JOptionPane.showConfirmDialog(null,
                                "All changes will be lost. Do you want to continue?");
                if (answer != 0)
                        return;


        } else if (arg.getSource().equals(buttonOK)) {
                gradeListCopy.removeAll(existingGrades);


                for (int row = 0; row < studentGradesTable.getRowCount(); row++) {
                        Object markPlaceHolder = studentGradesTable.getValueAt(row, 1);
                        if (markPlaceHolder != null) {
                                try {
                                        int mark = Integer.parseInt(markPlaceHolder.toString());
```

```
            if (mark <= 20 && mark >= 0) {
                  Grade newGrade = new Grade();
                  newGrade.setMark(mark);
                  newGrade.setStudent((Student) studentGradesTable
                              .getValueAt(row, 0));
                  newGrade.setSubject(subCopy);
                  newGrade.setTrimester(trimesterCopy);
                  gradeListCopy.add(newGrade);
            } else
                  throw new NumberFormatException();
      } catch (NumberFormatException e) {
            JOptionPane
                        .showMessageDialog(
                                    null,
                                    ""
                                                + markPlaceHolder.toString()
```

```
                                                                    + "' was not saved. Only integer values between 0 and
20 are accepted");
                                        } catch (Exception e) {
                                                JOptionPane.showMessageDialog(null, e.toString());
                                        }
                                }
                        }
                }
                setVisible(false);
        }
}
```

➢ **GradePreparation**

```
package xx.xx.grading.system;


import java.awt.GridBagConstraints;
```

```
import java.awt.GridBagLayout;
import java.awt.HeadlessException;
import java.awt.Window;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Arrays;
import java.util.Set;
import java.util.Vector;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
```

```
@SuppressWarnings("serial")
public class GradePreparation extends JDialog implements ActionListener {

    private JLabel labelClass, labelSubject, labelTrimester;
    private JButton buttonOK, buttonCancel;
    private JComboBox<SchoolClass> comboSchoolClass;
    private JComboBox<Subject> comboSubject;
    private SchoolClass returnValueSc = null;
    private Subject returnValueSub = null;
    private Trimester returnValueTrimester = null;
    private JComboBox<Trimester> comboTrimester;
    private final boolean ignoreTrimester;
    private final boolean ignoreSubject;

    public GradePreparation(Window owner, Set<SchoolClass> schoolClassList,
                    Set<Subject> subjectList, Trimester[] trimesters)
```

```
        throws HeadlessException {

super(owner, ModalityType.APPLICATION_MODAL);

ignoreTrimester = trimesters == null;
ignoreSubject = subjectList == null;

setTitle("Please select");
setSize(700, 500);
setDefaultCloseOperation(HIDE_ON_CLOSE);

JPanel panel = new JPanel(new GridBagLayout());
add(panel);

GridBagConstraints c1 = new GridBagConstraints();
c1.gridx = 0;
```

```
c1.gridy = 0;
c1.anchor = GridBagConstraints.WEST;
labelClass = new JLabel("Class: ");
panel.add(labelClass, c1);

if (subjectList != null) {
        GridBagConstraints c2 = new GridBagConstraints();
        c2.gridx = 0;
        c2.gridy = 1;
        c2.anchor = GridBagConstraints.WEST;
        labelSubject = new JLabel("Subject: ");
        panel.add(labelSubject, c2);
}

if (trimesters != null) {
        GridBagConstraints c3 = new GridBagConstraints();
```

```
        c3.gridx = 0;
        c3.gridy = 2;
        c3.anchor = GridBagConstraints.WEST;
        labelTrimester = new JLabel("Trimester: ");
        panel.add(labelTrimester, c3);
}

Vector<SchoolClass> schoolClasses = new Vector<SchoolClass>(
                schoolClassList);
GridBagConstraints c4 = new GridBagConstraints();
c4.gridx = 1;
c4.gridy = 0;
c4.anchor = GridBagConstraints.WEST;
comboSchoolClass = new JComboBox<SchoolClass>(schoolClasses);
panel.add(comboSchoolClass, c4);
```

```
if (subjectList != null) {
        Vector<Subject> subjects = new Vector<Subject>(subjectList);
        GridBagConstraints c5 = new GridBagConstraints();
        c5.gridx = 1;
        c5.gridy = 1;
        c5.anchor = GridBagConstraints.WEST;
        comboSubject = new JComboBox<Subject>(subjects);
        panel.add(comboSubject, c5);
}

if (trimesters != null) {
        Vector<Trimester> trimester = new Vector<Trimester>(
                Arrays.asList(trimesters));
        GridBagConstraints c6 = new GridBagConstraints();
        c6.gridx = 1;
        c6.gridy = 2;
```

```
        c6.anchor = GridBagConstraints.WEST;
        comboTrimester = new JComboBox<Trimester>(trimester);
        panel.add(comboTrimester, c6);
}

GridBagConstraints c7 = new GridBagConstraints();
c7.gridx = 0;
c7.gridy = 4;
c7.anchor = GridBagConstraints.WEST;
buttonOK = new JButton("OK");
panel.add(buttonOK, c7);
buttonOK.addActionListener(this);

GridBagConstraints c8 = new GridBagConstraints();
c8.gridx = 1;
c8.gridy = 4;
```

```
        c8.anchor = GridBagConstraints.EAST;
        buttonCancel = new JButton("Cancel");
        panel.add(buttonCancel, c8);
        buttonCancel.addActionListener(this);

        setVisible(true);
}

@Override
public void actionPerformed(ActionEvent arg) {
        if (arg.getSource().equals(buttonCancel)) {
                int answer = JOptionPane.showConfirmDialog(null,
                                "Your selections will be lost. Do you want to continue?");
                if (answer == 0) {
                        returnValueSc = null;
                        returnValueSub = null;
```

```
                returnValueTrimester = null;
        } else
                return;


} else if (arg.getSource().equals(buttonOK)) {
        SchoolClass schoolClass = comboSchoolClass
                        .getItemAt(comboSchoolClass.getSelectedIndex());
        if (ignoreSubject == false) {
                Subject subject = comboSubject.getItemAt(comboSubject
                                .getSelectedIndex());
                returnValueSub = subject;
        }
        if (ignoreTrimester == false) {
                Trimester trimester = comboTrimester.getItemAt(comboTrimester
                                .getSelectedIndex());
                returnValueTrimester = trimester;
```

```
                }

                returnValueSc = schoolClass;

        }

        setVisible(false);
}

public Subject getReturnValueSub() {
        return returnValueSub;
}

public SchoolClass getReturnValueSc() {
        return returnValueSc;
}
```

```
        public Trimester getReturnValueTrimester() {
                return returnValueTrimester;
        }
}
```

➢ MainForm

```
package xx.parisi.grading.system;

// random access files xD

import java.awt.HeadlessException;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
```

```
import java.net.URI;
import java.net.URISyntaxException;
import java.util.HashSet;
import java.util.Set;

import javax.swing.JDialog;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;

@SuppressWarnings("serial")
public class MainForm extends JDialog implements ActionListener {

        private JMenuItem menuStudentsNew, menuStudentsUpdate, menuStudentsDelete,
                        menuStudentsPrint, menuStudentsIndividualPrint;
```

```
private JMenuItem menuSubjectsNew, menuSubjectsUpdate, menuSubjectsDelete;
private JMenuItem menuClassesNew, menuClassesUpdate, menuClassesDelete;
private JMenuItem menuGradesForClass, menuGradesPrint;
private JMenuItem menuHelpAbout;
private Program program;

public MainForm(Program program) throws HeadlessException {

        super(null, ModalityType.APPLICATION_MODAL);

        this.program = program;

        setTitle("Grading Application");
        setSize(700, 700);
        setDefaultCloseOperation(HIDE_ON_CLOSE);
```

```
JMenuBar bar = new JMenuBar();
setJMenuBar(bar);

JMenu menuStudents = new JMenu("Students");
bar.add(menuStudents);

JMenu menuSubjects = new JMenu("Subjects");
bar.add(menuSubjects);

JMenu menuClasses = new JMenu("Classes");
bar.add(menuClasses);

JMenu menuGrades = new JMenu("Grades");
bar.add(menuGrades);

JMenu menuHelp = new JMenu("Help");
```

```
bar.add(menuHelp);


menuStudentsNew = new JMenuItem("New");
menuStudents.add(menuStudentsNew);
menuStudentsNew.addActionListener(this);
menuStudentsUpdate = new JMenuItem("Update");
menuStudents.add(menuStudentsUpdate);
menuStudentsUpdate.addActionListener(this);
menuStudentsDelete = new JMenuItem("Delete");
menuStudents.add(menuStudentsDelete);
menuStudentsDelete.addActionListener(this);
menuStudentsPrint = new JMenuItem("Print");
menuStudents.add(menuStudentsPrint);
menuStudentsPrint.addActionListener(this);
menuStudentsIndividualPrint = new JMenuItem("Print Student File");
menuStudents.add(menuStudentsIndividualPrint);
```

```
menuStudentsIndividualPrint.addActionListener(this);


menuSubjectsNew = new JMenuItem("New");
menuSubjects.add(menuSubjectsNew);
menuSubjectsNew.addActionListener(this);
menuSubjectsUpdate = new JMenuItem("Update");
menuSubjects.add(menuSubjectsUpdate);
menuSubjectsUpdate.addActionListener(this);
menuSubjectsDelete = new JMenuItem("Delete");
menuSubjects.add(menuSubjectsDelete);
menuSubjectsDelete.addActionListener(this);

menuClassesNew = new JMenuItem("New");
menuClasses.add(menuClassesNew);
menuClassesNew.addActionListener(this);
menuClassesUpdate = new JMenuItem("Update");
```

```
menuClasses.add(menuClassesUpdate);
menuClassesUpdate.addActionListener(this);
menuClassesDelete = new JMenuItem("Delete");
menuClasses.add(menuClassesDelete);
menuClassesDelete.addActionListener(this);

menuGradesForClass = new JMenuItem("For class");
menuGrades.add(menuGradesForClass);
menuGradesForClass.addActionListener(this);
menuGradesPrint = new JMenuItem("Print");
menuGrades.add(menuGradesPrint);
menuGradesPrint.addActionListener(this);

menuHelpAbout = new JMenuItem("About");
menuHelp.add(menuHelpAbout);
menuHelpAbout.addActionListener(this);
```

```
        setVisible(true);

}

@Override
public void actionPerformed(ActionEvent arg) {
        if (arg.getSource().equals(menuHelpAbout))
                JOptionPane.showMessageDialog(null, "xx", "About",
                                JOptionPane.INFORMATION_MESSAGE);

        else if (arg.getSource().equals(menuSubjectsNew)) {
                Subject temp = new Subject();
                temp.setSubjectID(program.getNextSubjectID());
                SubjectForm sf = new SubjectForm(this, temp);
                Subject justSaved = sf.getReturnValue();
```

```
            sf.dispose();


            if (justSaved == null)
                  return;
            program.subjectList.add(justSaved);
}

else if (arg.getSource().equals(menuSubjectsUpdate)) {
            if (program.subjectList.size() > 0) {
                  SubjectTableForm updateForm = new SubjectTableForm(this,
                              program.subjectList);
                  Subject returnValue = updateForm.getReturnValue();
                  if (returnValue == null)
                        return;
                  updateForm.dispose();
```

```
                SubjectForm subForm = new SubjectForm(this, returnValue);
                Subject returnValue2 = subForm.getReturnValue();
                subForm.dispose();
                program.updateSubject(returnValue2);
        } else
                JOptionPane.showMessageDialog(null, "There are no files saved",
                                "Information", JOptionPane.INFORMATION_MESSAGE);
}

else if (arg.getSource().equals(menuSubjectsDelete)) {
        if (program.subjectList.size() > 0) {
                SubjectTableForm deleteForm = new SubjectTableForm(this,
                                program.subjectList);
                Subject returnValue = deleteForm.getReturnValue();
                deleteForm.dispose();
                if (returnValue == null)
```

```
                    return;
                program.deleteSubject(returnValue);
        } else
                JOptionPane.showMessageDialog(null, "There are no files saved",
                        "Information", JOptionPane.INFORMATION_MESSAGE);
}

else if (arg.getSource().equals(menuClassesNew)) {
        SchoolClass temp = new SchoolClass();
        temp.setSchoolClassID(program.getNextSchoolClassID());
        SchoolClassForm scf = new SchoolClassForm(this, temp);
        SchoolClass justSaved = scf.getReturnValue();
        scf.dispose();

        if (justSaved == null)
                return;
```

```
            program.schoolClassList.add(justSaved);
    }

    else if (arg.getSource().equals(menuClassesUpdate)) {
            if (program.schoolClassList.size() > 0) {
                    SchoolClassTableForm updateForm = new SchoolClassTableForm(
                                    this, program.schoolClassList);
                    SchoolClass returnValue = updateForm.getReturnValue();
                    if (returnValue == null)
                            return;
                    updateForm.dispose();

                    SchoolClassForm scForm = new SchoolClassForm(this, returnValue);
                    SchoolClass returnValue2 = scForm.getReturnValue();
                    scForm.dispose();
                    program.updateSchoolClass(returnValue2);
```

```
        } else
                JOptionPane.showMessageDialog(null, "There are no files saved",
                        "Information", JOptionPane.INFORMATION_MESSAGE);
}

else if (arg.getSource().equals(menuClassesDelete)) {
        if (program.schoolClassList.size() > 0) {
                SchoolClassTableForm deleteForm = new SchoolClassTableForm(
                        this, program.schoolClassList);
                SchoolClass returnValue = deleteForm.getReturnValue();
                deleteForm.dispose();
                if (returnValue == null)
                        return;
                program.deleteSchoolClass(returnValue);
        } else
                JOptionPane.showMessageDialog(null, "There are no files saved",
```

```
                                "Information", JOptionPane.INFORMATION_MESSAGE);
    }

    else if (arg.getSource().equals(menuStudentsNew)) {
        if (program.schoolClassList.size() > 0) {
            Student temp = new Student();
            temp.setStudentID(program.getNextStudentID());
            StudentForm stf = new StudentForm(this, temp,
                                program.schoolClassList);
            Student justSaved = stf.getReturnValue();
            stf.dispose();

            if (justSaved == null)
                return;
            program.studentList.add(justSaved);
        } else
```

```
                JOptionPane.showMessageDialog(null,
                        "You need to create at least one class first",
                        "Information", JOptionPane.INFORMATION_MESSAGE);
}

else if (arg.getSource().equals(menuStudentsUpdate)) {
        if (program.studentList.size() > 0) {
                StudentTableForm updateForm = new StudentTableForm(this,
                        program.studentList);
                Student returnValue = updateForm.getReturnValue();
                if (returnValue == null)
                        return;
                updateForm.dispose();

                StudentForm stForm = new StudentForm(this, returnValue,
                        program.schoolClassList);
```

```
                Student returnValue2 = stForm.getReturnValue();
                stForm.dispose();
                program.updateStudent(returnValue2);
        } else
                JOptionPane.showMessageDialog(null, "There are no files saved",
                                "Information", JOptionPane.INFORMATION_MESSAGE);
}

else if (arg.getSource().equals(menuStudentsDelete)) {
        if (program.studentList.size() > 0) {
                StudentTableForm deleteForm = new StudentTableForm(this,
                                program.studentList);
                Student returnValue = deleteForm.getReturnValue();
                deleteForm.dispose();
                if (returnValue == null)
                        return;
```

```
                program.deleteStudent(returnValue);
        } else
                JOptionPane.showMessageDialog(null, "There are no files saved",
                                "Information", JOptionPane.INFORMATION_MESSAGE);
}

else if (arg.getSource().equals(menuGradesForClass)) {
        if (program.schoolClassList.size() > 0
                        && program.subjectList.size() > 0) {
                GradePreparation selectParameters = new GradePreparation(this,
                                program.schoolClassList, program.subjectList,
                                program.trimesters);
                SchoolClass returnValueSc = selectParameters.getReturnValueSc();
                Subject returnValueSub = selectParameters.getReturnValueSub();
                Trimester returnValueTrimester = selectParameters
                                .getReturnValueTrimester();
```

```
                selectParameters.dispose();

                Set<Student> studentsOfClass = createSetStudentsOfClass(returnValueSc);

                GradeForm gradeForm = new GradeForm(this, studentsOfClass,
                                program.gradeList, returnValueSc, returnValueSub,
                                returnValueTrimester);
                gradeForm.dispose();
        } else
                JOptionPane
                                .showMessageDialog(
                                                null,
                                                "You need to create at least one class and one subject",
                                                "Information", JOptionPane.INFORMATION_MESSAGE);
}
```

```
else if (arg.getSource().equals(menuGradesPrint)) {
        if (program.schoolClassList.size() > 0
                && program.subjectList.size() > 0) {
            GradePreparation selectParameters = new GradePreparation(this,
                    program.schoolClassList, program.subjectList, null);
            SchoolClass returnValueSc = selectParameters.getReturnValueSc();
            Subject returnValueSub = selectParameters.getReturnValueSub();
            selectParameters.dispose();

            Set<Student> studentsOfClass = createSetStudentsOfClass(returnValueSc);

            CreatePrintClassFile createFile = new CreatePrintClassFile(
                    program.gradeList, studentsOfClass, returnValueSc,
                    returnValueSub);
            createFile.writeFile();
            try {
```

```
                File fil = new File(CreatePrintClassFile.printFileName);
                java.awt.Desktop.getDesktop().browse(
                                new URI("file:///"
                                                + fil.getAbsolutePath().replace('\\', '/')
                                                                .replace(" ", "%20")));
        } catch (IOException e) {
                e.printStackTrace();
        } catch (URISyntaxException e) {
                e.printStackTrace();
        }
} else
        JOptionPane.showMessageDialog(null, "There are no files saved",
                        "Information", JOptionPane.INFORMATION_MESSAGE);
// important to mention that this application was only tested on
// windows
// and using the browser Chrome
```

```
            // the absolute paths are ONLY for Windows
            // if used on a different

} else if (arg.getSource().equals(menuStudentsPrint)) {

            if (program.schoolClassList.size() > 0) {
                        GradePreparation selectParameters = new GradePreparation(this,
                                    program.schoolClassList, null, null);
                        SchoolClass returnValueSc = selectParameters.getReturnValueSc();
                        selectParameters.dispose();

                        Set<Student> studentsOfClass = createSetStudentsOfClass(returnValueSc);

                        CreatePrintStudentFile createFile = new CreatePrintStudentFile(
                                    studentsOfClass, returnValueSc);
```

```
                createFile.writeFile();
                try {
                        File fil = new File(CreatePrintStudentFile.printStudent);
                        java.awt.Desktop.getDesktop().browse(
                                        new URI("file:///"
                                                        + fil.getAbsolutePath().replace('\\', '/')
                                                                        .replace(" ", "%20")));
                } catch (IOException e) {
                        e.printStackTrace();
                } catch (URISyntaxException e) {
                        e.printStackTrace();
                }
        } else
                JOptionPane.showMessageDialog(null, "There are no files saved",
                                "Information", JOptionPane.INFORMATION_MESSAGE);
}
```

```
else if (arg.getSource().equals(menuStudentsIndividualPrint)) {

        if (program.schoolClassList.size() > 0) {
                GradePreparation selectParameters = new GradePreparation(this,
                                program.schoolClassList, null, null);
                SchoolClass returnValueSc = selectParameters.getReturnValueSc();
                selectParameters.dispose();

                Set<Student> studentsOfClass = createSetStudentsOfClass(returnValueSc);

                StudentTableForm students = new StudentTableForm(this,
                                studentsOfClass);
                Student s = students.getReturnValue();
                students.dispose();
```

```
Set<Grade> studentGrades = new HashSet<>();


for (Grade g : program.gradeList)
        if (g.getStudent().getStudentID() == s.getStudentID())
                studentGrades.add(g);


CreatePrintIndividualStudentFile createFile = new CreatePrintIndividualStudentFile(
                s, studentGrades, returnValueSc);
createFile.writeFile();
try {
        File fil = new File(
                        CreatePrintIndividualStudentFile.printIndividualStudent);
        java.awt.Desktop.getDesktop().browse(
                        new URI("file:///"
                                        + fil.getAbsolutePath().replace('\\', '/')
                                                        .replace(" ", "%20")));
```

```
                    } catch (IOException e) {
                        e.printStackTrace();
                    } catch (URISyntaxException e) {
                        e.printStackTrace();
                    }
                } else
                    JOptionPane.showMessageDialog(null, "There are no files saved",
                            "Information", JOptionPane.INFORMATION_MESSAGE);
        }
}

protected Set<Student> createSetStudentsOfClass(SchoolClass sc) {
        Set<Student> studentsOfClass = new HashSet<>();
        for (Student s : program.studentList)
            if (s.getSchoolClass().getSchoolClassID() == sc.getSchoolClassID())
                studentsOfClass.add(s);
```

```
            return studentsOfClass;
        }
}
```

➢ NonEditableTableModel

```
package xx.xx.grading.system;

import javax.swing.table.DefaultTableModel;

@SuppressWarnings("serial")
public class NonEditableTableModel extends DefaultTableModel {

        public NonEditableTableModel(Object[][] data, Object[] columnNames) {
            super(data, columnNames);
        }
```

```
        @Override
        public boolean isCellEditable(int row, int column) {

                return false;
        }
}
```

➢ **Program**

```
package xx.xx.grading.system;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
```

```
import javax.swing.JOptionPane;

public class Program {

    // data files are created the first time the program is run. They will be
    // saved ad updated whenever needed
    private final String subjectFileName = "data/subjects.txt";
    private final String schoolClassFileName = "data/schoolClasses.txt";
    private final String studentFileName = "data/students.txt";
    private final String gradeFileName = "data/grades.txt";

    private static final int YES = 0;

    protected Set<Subject> subjectList = new HashSet<Subject>();
    protected Set<SchoolClass> schoolClassList = new HashSet<SchoolClass>();
```

```
protected Set<Student> studentList = new HashSet<Student>();
protected Set<Grade> gradeList = new HashSet<Grade>();
protected Trimester[] trimesters = { new Trimester(1, "First"),
                new Trimester(2, "Second"), new Trimester(3, "Third"),
                new Trimester(4, "Written Grade") };

public static void main(String[] args) {

        Program program = new Program(); // the program will start the first
        // time by creating empty lists for the user to fill in when
        // exiting the program, the lists will be saved as files that will
        // be re-opened when using the program again
        program.loadLists();

        MainForm mainform = new MainForm(program);
        mainform.dispose();
```

```
        program.saveLists();
}

// the following methods are used from the main form and they do exactly
// what their name says
// iteratiors are used - faster and more efficient
protected void updateSubject(Subject subjectToUpdate) {
        for (Subject s : subjectList)
                if (s.getSubjectID() == subjectToUpdate.getSubjectID()) {
                        s.setName(subjectToUpdate.getName());
                        break;
                }
}

protected void updateSchoolClass(SchoolClass schoolClassToUpdate) {
```

```
        for (SchoolClass sc : schoolClassList)
                if (sc.getSchoolClassID() == schoolClassToUpdate.getSchoolClassID()) {
                        sc.setName(schoolClassToUpdate.getName());
                        break;
                }
}

protected void updateStudent(Student studentToUpdate) {
        for (Student student : studentList)
                if (student.getStudentID() == studentToUpdate.getStudentID()) {
                        student.setName(studentToUpdate.getName());
                        break;
                }
}

protected void deleteStudent(Student studentToDelete) {
```

```
Set<Grade> gradesToDelete = new HashSet<Grade>();
for (Grade grade : gradeList)
        if (grade.getStudent().getStudentID() == studentToDelete
                        .getStudentID())
                gradesToDelete.add(grade);


int answer;
if (gradesToDelete.size() > 0) {
        answer = JOptionPane.showConfirmDialog(
                        null,
                        "The student you want to delete has "
                                        + gradesToDelete.size()
                                        + " grades in his file. Do you want to proceed?");
        if (answer != YES)
                return;
```

```
        // security questions are needed, since the user may not know if
        // existing objects
        // have more parameters linked to them
        // if they do everything needs to be deleted

        gradeList.removeAll(gradesToDelete);
}

else
        answer = JOptionPane.showConfirmDialog(null,
                        "Are you sure you want to delete this student?");

if (answer == YES) {
        for (Student student : studentList)
                if (student.getStudentID() == studentToDelete.getStudentID()) {
                        studentList.remove(student);
```

```
                                break;
                            }
                    }

}

protected void deleteSubject(Subject subjectToDelete) {

        Set<Grade> gradesToDelete = new HashSet<Grade>();
        for (Grade grade : gradeList)
                if (grade.getSubject().getSubjectID() == subjectToDelete
                                .getSubjectID())
                        gradesToDelete.add(grade);

        int answer;
        if (gradesToDelete.size() > 0) {
```

```
        answer = JOptionPane.showConfirmDialog(
                        null,
                        "The subject you want to delete contains "
                                + gradesToDelete.size()
                                + " grades. Do you want to proceed?");

        if (answer != YES)
                return;

        gradeList.removeAll(gradesToDelete);
} else
        answer = JOptionPane.showConfirmDialog(null,
                        "Are you sure you want to delete this subject?");

if (answer == YES) {
        for (Subject s : subjectList)
```

```
                    if (s.getSubjectID() == subjectToDelete.getSubjectID()) {
                        subjectList.remove(s);
                        break;
                    }
            }
    }

    protected void deleteSchoolClass(SchoolClass schoolClassToDelete) {

            Set<Student> studentToDelete = new HashSet<Student>();
            for (Student s : studentList)
                if (s.getSchoolClass().getSchoolClassID() == schoolClassToDelete
                        .getSchoolClassID()) {
                    studentToDelete.add(s);
                }
```

```
int answer;
if (studentToDelete.size() > 0) {
        answer = JOptionPane.showConfirmDialog(null,
                        "The schoolClass you want to delete contains "
                                        + studentToDelete.size()
                                        + " students. Do you want to proceed?");
        if (answer != YES)
                return;

        // deletes all students in the class that is going to be deleted
        studentList.removeAll(studentToDelete);
} else
        answer = JOptionPane.showConfirmDialog(null,
                        "Are you sure you want to delete this class?");

if (answer == YES) {
```

```
                for (SchoolClass sc : schoolClassList)
                        if (sc.getSchoolClassID() == schoolClassToDelete
                                        .getSchoolClassID()) {
                                schoolClassList.remove(sc);
                                break;
                        }
        }
}

// IDs are generated: they have to be unique and instead of generating
// random values
// and chacking if they already exist, their numbers will be the previous ID
// + 1
protected int getNextStudentID() {
        int maxStudentID = 0;
        for (Student student : studentList)
```

```
                if (maxStudentID < student.getStudentID())
                        maxStudentID = student.getStudentID();
        return maxStudentID + 1;
}

protected int getNextSubjectID() {
        int maxSubjectID = 0;
        for (Subject sub : subjectList)
                if (maxSubjectID < sub.getSubjectID())
                        maxSubjectID = sub.getSubjectID();
        return maxSubjectID + 1;
}

protected int getNextSchoolClassID() {
        int maxSchoolClassID = 0;
        for (SchoolClass sc : schoolClassList)
```

```
                if (maxSchoolClassID < sc.getSchoolClassID())
                        maxSchoolClassID = sc.getSchoolClassID();
        return maxSchoolClassID + 1;
}

private void loadLists() {

        buildSubjectList();
        buildSchoolClassList();
        buildStudentList();
        buildGradeList();
}

// reading data from the files created in the beginning
private void buildGradeList() {
```

```
        try (FileReader f = new FileReader(gradeFileName);
                        BufferedReader data = new BufferedReader(f)) {
                String line = data.readLine();
                while (line != null) {
                        Grade grade = convertStringToGrade(line);
                        gradeList.add(grade);
                        line = data.readLine();
                }

        } catch (IOException e) {
                System.out.println("Cannot read grades file");
        }

}

private Grade convertStringToGrade(String line) {
```

```
String delims = "[~]";
String[] elements = line.split(delims);
Grade grade = new Grade();


int studentID = Integer.parseInt(elements[0]);
for (Student y : studentList)
        if (y.getStudentID() == studentID) {
                grade.setStudent(y);
                break;
        }
int subjectID = Integer.parseInt(elements[1]);
for (Subject y : subjectList)
        if (y.getSubjectID() == subjectID) {
                grade.setSubject(y);
                break;
        }
```

```
            int trimesterID = Integer.parseInt(elements[2]);
            grade.setTrimester(trimesters[trimesterID - 1]);
            grade.setMark(Integer.parseInt(elements[3]));

            return grade;
    }

    private void buildStudentList() {

            try (FileReader f = new FileReader(studentFileName);
                            BufferedReader data = new BufferedReader(f)) {
                    String line = data.readLine();
                    while (line != null) {
                            Student student = convertStringToStudent(line);
                            studentList.add(student);
```

```
                line = data.readLine();
            }

        } catch (IOException e) {
            System.out.println("Cannot read students file");
        }

}

private Student convertStringToStudent(String line) {
        String delims = "[~]";
        String[] elements = line.split(delims);
        Student student = new Student();
        student.setStudentID(Integer.parseInt(elements[3]));
        student.setName(elements[0]);
        student.setSurname(elements[1]);
```

```
        int classID = Integer.parseInt(elements[2]);
        for (SchoolClass y : schoolClassList)
            if (y.getSchoolClassID() == classID) { // the parsed ID to be the
                                                                    // same with the
                                                                    // schoolClassID
                student.setSchoolClass(y);
                break;
            }

        return student;
}

private void buildSchoolClassList() {

        try (FileReader f = new FileReader(schoolClassFileName);
```

```
                BufferedReader data = new BufferedReader(f)) {
            String line = data.readLine();
            while (line != null) {
                SchoolClass schoolClass = convertStringToSchoolClass(line);
                schoolClassList.add(schoolClass);
                line = data.readLine();
            }

        } catch (IOException e) {
            System.out.println("Cannot read schoolClasses file");
        }

}

// converting methods are for reading data from files in the correct order
// and recognising what is what
```

```
private SchoolClass convertStringToSchoolClass(String line) {
        String delims = "[~]";
        String[] elements = line.split(delims);
        SchoolClass schoolClass = new SchoolClass();
        schoolClass.setSchoolClassID(Integer.parseInt(elements[1]));
        schoolClass.setName(elements[0]);
        return schoolClass;
}

private void buildSubjectList() {

        try (FileReader f = new FileReader(subjectFileName);
                        BufferedReader data = new BufferedReader(f)) {
                String line = data.readLine();
                while (line != null) {
                        Subject subject = convertStringToSubject(line);
```

```
                subjectList.add(subject);
                line = data.readLine();
            }

        } catch (IOException e) {
            System.out.println("Cannot read subjects file");
        }

}

private Subject convertStringToSubject(String line) {
        String delims = "[~]";
        String[] elements = line.split(delims);
        Subject subject = new Subject();
        subject.setSubjectID(Integer.parseInt(elements[1]));
        subject.setName(elements[0]);
```

```
        return subject;
}

private void saveLists() {

        writeSubjectList();
        writeSchoolClassList();
        writeStudentList();
        writeGradeList();

}

// changes are saved
private void writeGradeList() {

        try (FileWriter f = new FileWriter(gradeFileName);
```

```
                    BufferedWriter data = new BufferedWriter(f)) {


                for (Grade eachOne : gradeList)
                        data.write(eachOne.toPrintLine());
                data.flush();
                data.close();


        } catch (IOException e) {
                System.out.println("Cannot write grades file");
        }

}

private void writeStudentList() {

        try (FileWriter f = new FileWriter(studentFileName);
```

```
                    BufferedWriter data = new BufferedWriter(f)) {


                for (Student eachOne : studentList)
                        data.write(eachOne.toPrintedLine());
                data.flush();
                data.close();


        } catch (IOException e) {
                System.out.println("Cannot write students file");
        }

}

private void writeSchoolClassList() {

        try (FileWriter f = new FileWriter(schoolClassFileName);
```

```
                    BufferedWriter data = new BufferedWriter(f)) {

                for (SchoolClass eachOne : schoolClassList)
                    data.write(eachOne.toPrintLine());
                data.flush();
                data.close();

        } catch (IOException e) {
                System.out.println("Cannot write schoolClasses file");
        }

}

private void writeSubjectList() {

        try (FileWriter f = new FileWriter(subjectFileName);
```

```
                    BufferedWriter data = new BufferedWriter(f)) {


                for (Subject eachOne : subjectList)
                    data.write(eachOne.toPrintLine());
                data.flush();
                data.close();


            } catch (IOException e) {
                System.out.println("Cannot write subjects file");
            }


        }


}
```

- **RowClickMouseAdapter**

```
package xx.xx.grading.system;
```

```
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.JTable;

public class RowClickMouseAdapter extends MouseAdapter {

	private JTable table;
	private int row;

	public RowClickMouseAdapter(JTable table) {
		this.table = table;
	}

	public void mouseClicked(MouseEvent e) {
```

```
                row = table.getSelectedRow();
        }

        public int getRow() {
                return row;
        }
}
```

➢ SchoolClass

```
package xx.xx.grading.system;

public class SchoolClass {

        private String name;
        private int schoolClassID;
```

```
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public int getSchoolClassID() {
        return schoolClassID;
}
public void setSchoolClassID(int schoolClassID) {
        this.schoolClassID = schoolClassID;
}

@Override
```

```
        public String toString() {
                return String.format("%s (%d)\n", name, schoolClassID);
        }

        public String toPrintLine(){
                return String.format("%s~%d\n", name, schoolClassID);
        }

}
```

➢ SchoolClassForm

```
package xx.xx.grading.system;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.HeadlessException;
```

```
import java.awt.Window;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

@SuppressWarnings("serial")
public class SchoolClassForm extends JDialog implements ActionListener {

        private JLabel labelId, labelName, labelIDNumber;
        private JTextField textName;
```

```
private JButton buttonSave, buttonCancel;
private SchoolClass returnValue = null;
private Integer scID;

public SchoolClassForm(Window owner, SchoolClass sc)
        throws HeadlessException {

    super(owner, ModalityType.APPLICATION_MODAL);

    scID = sc.getSchoolClassID();

    if (sc.getName() == null)
        setTitle("New School Class");
    else
        setTitle("Update School Class");
    setSize(700, 300);
```

```
setDefaultCloseOperation(HIDE_ON_CLOSE);


JPanel panel = new JPanel(new GridBagLayout());
add(panel);


GridBagConstraints c1 = new GridBagConstraints();
c1.gridx = 0;
c1.gridy = 0;
c1.anchor = GridBagConstraints.WEST;
labelId = new JLabel("School class ID: ");
panel.add(labelId, c1);


GridBagConstraints c2 = new GridBagConstraints();
c2.gridx = 0;
c2.gridy = 1;
c2.anchor = GridBagConstraints.WEST;
```

```
labelName = new JLabel("School class name: ");
panel.add(labelName, c2);

GridBagConstraints c3 = new GridBagConstraints();
c3.gridx = 1;
c3.gridy = 0;
c3.anchor = GridBagConstraints.WEST;
labelIDNumber = new JLabel(scID.toString());
panel.add(labelIDNumber, c3);

GridBagConstraints c4 = new GridBagConstraints();
c4.gridx = 1;
c4.gridy = 1;
c4.anchor = GridBagConstraints.WEST;
textName = new JTextField(40);
textName.setText(sc.getName());
```

```
panel.add(textName, c4);


GridBagConstraints c5 = new GridBagConstraints();
c5.gridx = 1;
c5.gridy = 3;
c5.anchor = GridBagConstraints.WEST;
buttonSave = new JButton("Save");
panel.add(buttonSave, c5);
buttonSave.addActionListener(this);


GridBagConstraints c6 = new GridBagConstraints();
c6.gridx = 1;
c6.gridy = 3;
c6.anchor = GridBagConstraints.EAST;
buttonCancel = new JButton("Cancel");
panel.add(buttonCancel, c6);
```

```
        buttonCancel.addActionListener(this);


        setVisible(true);
}

@Override
public void actionPerformed(ActionEvent arg) {
        if (arg.getSource().equals(buttonCancel)) {
                int answer = JOptionPane.showConfirmDialog(null,
                                "All changes will be lost. Do you want to continue?");
                if (answer == 0)
                        returnValue = null;
                else
                        return;

        } else if (arg.getSource().equals(buttonSave)) {
```

```
                String name = textName.getText();
                if (name == null || name.equals("")) {
                        JOptionPane.showMessageDialog(this, "Name is mandatory",
                                        "Caution", JOptionPane.ERROR_MESSAGE);
                        return;
                }

                returnValue = new SchoolClass();
                returnValue.setSchoolClassID(scID);
                returnValue.setName(name);
        }
        setVisible(false);

}

public SchoolClass getReturnValue() {
```

```
            return returnValue;
        }
}
```

- **SchoolClassTableForm**

```
package xx.xx.grading.system;

import java.awt.HeadlessException;
import java.awt.Window;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.Set;

import javax.swing.JDialog;
```

```
import javax.swing.JScrollPane;
import javax.swing.JTable;

@SuppressWarnings("serial")
public class SchoolClassTableForm extends JDialog implements ActionListener,
        MouseListener {

    private JTable schoolClassTable;
    private SchoolClass returnValue = null;

    public SchoolClassTableForm(Window owner, Set<SchoolClass> schoolClassList)
            throws HeadlessException {

        super(owner, ModalityType.APPLICATION_MODAL);

        int listSize = schoolClassList.size();
```

```
String[] columnNames = { "ID", "Name" };
Object[][] elements = new Object[listSize][columnNames.length];


int row = 0;
for (SchoolClass sc : schoolClassList) {
        elements[row][0] = sc.getSchoolClassID();
        elements[row][1] = sc.getName();
        row++;
}

NonEditableTableModel model = new NonEditableTableModel(elements,
                columnNames);
schoolClassTable = new JTable(model);
schoolClassTable.addMouseListener(this);
```

```
        JScrollPane scrollPane = new JScrollPane(schoolClassTable);


        setSize(200, 200);
        setDefaultCloseOperation(HIDE_ON_CLOSE);
        add(scrollPane);
        setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {


}

@Override
public void mouseClicked(MouseEvent e) {
```

```
        if (e.getClickCount() < 2)
                return;
        int selectedRow = schoolClassTable.getSelectedRow();
        returnValue = new SchoolClass();
        int id = (int) schoolClassTable.getValueAt(selectedRow, 0);
        returnValue.setSchoolClassID(id);
        String name = (String)schoolClassTable.getValueAt(selectedRow, 1);
        returnValue.setName(name);
        setVisible(false);
}

@Override
public void mouseEntered(MouseEvent e) {
        // TODO Auto-generated method stub

}
```

```
@Override
public void mouseExited(MouseEvent e) {
        // TODO Auto-generated method stub

}

@Override
public void mousePressed(MouseEvent e) {
        // TODO Auto-generated method stub

}

@Override
public void mouseReleased(MouseEvent e) {
        // TODO Auto-generated method stub
```

```
        }

        public SchoolClass getReturnValue() {
                return returnValue;
        }

}
```

- **Student**

```
package xx.xx.grading.system;

public class Student {

        private String name;
        private String surname;
```

```
private SchoolClass schoolClass;
private int studentID;

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

public String getSurname() {
        return surname;
}

public void setSurname(String surname) {
```

```
        this.surname = surname;
}

public SchoolClass getSchoolClass() {
        return schoolClass;
}

public void setSchoolClass(SchoolClass schoolClass) {
        this.schoolClass = schoolClass;
}

public int getStudentID() {
        return studentID;
}

public void setStudentID(int studentID) {
```

```
            this.studentID = studentID;
        }

        @Override
        public String toString() {
            return String.format("%s, %s (%d)", surname, name, studentID);
        }

        public String toPrintedLine() {
            return String.format("%s~%s~%d~%d\n", name, surname,
                    schoolClass.getSchoolClassID(), studentID);
        }
}
```

- **StudentForm**

```
package xx.xx.grading.system;
```

```
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.HeadlessException;
import java.awt.Window;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Set;
import java.util.Vector;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
```

```
import javax.swing.JTextField;


@SuppressWarnings("serial")
public class StudentForm extends JDialog implements ActionListener {

        private JLabel labelId, labelName, labelIDNumber, labelSurname, labelschoolClassID;
        private JTextField textName, textSurname;
        private JButton buttonSave, buttonCancel;
        private Student returnValue = null;
        private Integer studentID;
        private JComboBox<SchoolClass> comboSchoolClass;

        public StudentForm(Window owner, Student student, Set<SchoolClass> schoolClassList) thr

                super(owner, ModalityType.APPLICATION_MODAL);
```

```
studentID = student.getStudentID();


if (student.getName() == null)
        setTitle("New student");
else
        setTitle("Update student");
setSize(700, 300);
setDefaultCloseOperation(HIDE_ON_CLOSE);

JPanel panel = new JPanel(new GridBagLayout());
add(panel);

GridBagConstraints c1 = new GridBagConstraints();
c1.gridx = 0;
c1.gridy = 0;
c1.anchor = GridBagConstraints.WEST;
```

```
labelId = new JLabel("ID: ");
panel.add(labelId, c1);

GridBagConstraints c2 = new GridBagConstraints();
c2.gridx = 0;
c2.gridy = 1;
c2.anchor = GridBagConstraints.WEST;
labelName = new JLabel("Name: ");
panel.add(labelName, c2);

GridBagConstraints c3 = new GridBagConstraints();
c3.gridx = 0;
c3.gridy = 2;
c3.anchor = GridBagConstraints.WEST;
labelSurname = new JLabel("Surname: ");
panel.add(labelSurname, c3);
```

```
GridBagConstraints c4 = new GridBagConstraints();
c4.gridx = 0;
c4.gridy = 3;
c4.anchor = GridBagConstraints.WEST;
labelschoolClassID = new JLabel("Class: ");
panel.add(labelschoolClassID, c4);

GridBagConstraints c5 = new GridBagConstraints();
c5.gridx = 1;
c5.gridy = 0;
c5.anchor = GridBagConstraints.WEST;
studentID = student.getStudentID();
labelIDNumber = new JLabel(studentID.toString());
panel.add(labelIDNumber, c5);
```

```
GridBagConstraints c6 = new GridBagConstraints();
c6.gridx = 1;
c6.gridy = 1;
c6.anchor = GridBagConstraints.WEST;
textName = new JTextField(40);
textName.setText(student.getName());
panel.add(textName, c6);

GridBagConstraints c7 = new GridBagConstraints();
c7.gridx = 1;
c7.gridy = 2;
c7.anchor = GridBagConstraints.WEST;
textSurname = new JTextField(40);
textSurname.setText(student.getSurname());
panel.add(textSurname, c7);
```

```
Vector<SchoolClass> schoolClasses = new Vector<SchoolClass>(schoolClassList);
GridBagConstraints c8 = new GridBagConstraints();
c8.gridx = 1;
c8.gridy = 3;
c8.anchor = GridBagConstraints.WEST;
comboSchoolClass = new JComboBox<SchoolClass>(schoolClasses);
panel.add(comboSchoolClass, c8);

GridBagConstraints c9 = new GridBagConstraints();
c9.gridx = 1;
c9.gridy = 4;
c9.anchor = GridBagConstraints.WEST;
buttonSave = new JButton("Save");
panel.add(buttonSave, c9);
buttonSave.addActionListener(this);
```

```
        GridBagConstraints c10 = new GridBagConstraints();
        c10.gridx = 1;
        c10.gridy = 4;
        c10.anchor = GridBagConstraints.EAST;
        buttonCancel = new JButton("Cancel");
        panel.add(buttonCancel, c10);
        buttonCancel.addActionListener(this);

        setVisible(true);
}

@Override
public void actionPerformed(ActionEvent arg) {
        if (arg.getSource().equals(buttonCancel)) {
                int answer = JOptionPane.showConfirmDialog(null,
                                "All changes will be lost. Do you want to continue?");
```

```
            if (answer == 0)
                  returnValue = null;
            else
                  return;

} else if (arg.getSource().equals(buttonSave)) {
            String name = textName.getText();
            String surname = textSurname.getText();
            if (name == null || name.equals("") || surname == null || surname.equals("")) {
                  JOptionPane.showMessageDialog(this, "Name and surname are mandatory",
                              "Caution", JOptionPane.ERROR_MESSAGE);
                  return;
            }
            SchoolClass schoolClass = comboSchoolClass.getItemAt(comboSchoolClass.getSelectedIndex());

            returnValue = new Student();
```

```
                    returnValue.setStudentID(studentID);
                    returnValue.setName(name);
                    returnValue.setSurname(surname);
                    returnValue.setSchoolClass(schoolClass); // TODO correct this according to combo box
                }
                setVisible(false);

            }

            public Student getReturnValue() {
                return returnValue;
            }
        }
```

- **StudentTableForm**

```
package xx.xx.grading.system;
```

```
import java.awt.HeadlessException;
import java.awt.Window;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.Set;

import javax.swing.JDialog;
import javax.swing.JScrollPane;
import javax.swing.JTable;

@SuppressWarnings("serial")
public class StudentTableForm extends JDialog implements ActionListener,
                    MouseListener {
```

```
private JTable studentTable;
private Student returnValue = null;

public StudentTableForm(Window owner, Set<Student> studentList)
                throws HeadlessException {

        super(owner, ModalityType.APPLICATION_MODAL);

        int listSize = studentList.size();

        String[] columnNames = { "ID", "Name", "Surname", "Class" };
        Object[][] elements = new Object[listSize][columnNames.length];

        int row = 0;
        for (Student student : studentList) {
```

```
            elements[row][0] = student.getStudentID();
            elements[row][1] = student.getName();
            elements[row][2] = student.getSurname();
            elements[row][3] = student.getSchoolClass();
            row++;
    }

    NonEditableTableModel model = new NonEditableTableModel(elements,
                        columnNames);
    studentTable = new JTable(model);
    studentTable.addMouseListener(this);

    JScrollPane scrollPane = new JScrollPane(studentTable);

    setSize(200, 200);
    setDefaultCloseOperation(HIDE_ON_CLOSE);
```

```
        add(scrollPane);
        setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {

}

@Override
public void mouseClicked(MouseEvent e) {

        if (e.getClickCount() < 2)
                return;
        int selectedRow = studentTable.getSelectedRow();
        returnValue = new Student();
```

```
            int id = (int) studentTable.getValueAt(selectedRow, 0);
            returnValue.setStudentID(id);
            String name = (String) studentTable.getValueAt(selectedRow, 1);
            returnValue.setName(name);
            String surname = (String) studentTable.getValueAt(selectedRow, 2);
            returnValue.setSurname(surname);
            SchoolClass schoolClass = (SchoolClass)studentTable.getValueAt(selectedRow, 3);
            returnValue.setSchoolClass(schoolClass);
            setVisible(false);
    }

    public Student getReturnValue() {
            return returnValue;
    }

    @Override
```

```
public void mouseEntered(MouseEvent e) {


}


@Override
public void mouseExited(MouseEvent e) {
}


@Override
public void mousePressed(MouseEvent e) {
}


@Override
public void mouseReleased(MouseEvent e) {
}
```

```
}
```

➢ Subject

```
package xx.xx.grading.system;

public class Subject {

        private String name;
        private int subjectID;

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
```

```
}

public int getSubjectID() {
        return subjectID;
}

public void setSubjectID(int subjectID) {
        this.subjectID = subjectID;
}

@Override
public String toString() {
        return String.format("%s (%d)\n", name, subjectID);
}

public String toPrintLine(){
```

```
            return String.format("%s~%d\n", name, subjectID);
        }
}
```

➢ SubjectForm

```
package xx.xx.grading.system;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.HeadlessException;
import java.awt.Window;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JDialog;
```

```
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

@SuppressWarnings("serial")
public class SubjectForm extends JDialog implements ActionListener {

        private JLabel labelId, labelName, labelIDNumber;
        private JTextField textName;
        private JButton buttonSave, buttonCancel;
        private Subject returnValue = null;
        private Integer subjectID;

        public SubjectForm(Window owner, Subject subject) throws HeadlessException {
```

```
super(owner, ModalityType.APPLICATION_MODAL);


subjectID = subject.getSubjectID();


if (subject.getName() == null) // name = null means new subject
        setTitle("New subject"); // form title
else
        setTitle("Update subject");
setSize(700, 300); // size in pixels
setDefaultCloseOperation(HIDE_ON_CLOSE); // what happens when clicking
                                                                    // on the x button


// Create and populate the panel.
JPanel panel = new JPanel(new GridBagLayout()); // divides form in rows
                                                                        // and columns
add(panel);
```

```
GridBagConstraints c1 = new GridBagConstraints();
c1.gridx = 0;
c1.gridy = 0;
c1.anchor = GridBagConstraints.WEST; // alignment
labelId = new JLabel("ID: "); // JLabel does not allow the user to write
                                                    // in a box, it just creates a label
panel.add(labelId, c1);

GridBagConstraints c2 = new GridBagConstraints();
c2.gridx = 0;
c2.gridy = 1;
c2.anchor = GridBagConstraints.WEST;
labelName = new JLabel("Name: ");
panel.add(labelName, c2);
```

```
GridBagConstraints c3 = new GridBagConstraints();
c3.gridx = 1;
c3.gridy = 0;
c3.anchor = GridBagConstraints.WEST;
labelIDNumber = new JLabel(subjectID.toString());
panel.add(labelIDNumber, c3);

GridBagConstraints c4 = new GridBagConstraints();
c4.gridx = 1;
c4.gridy = 1;
c4.anchor = GridBagConstraints.WEST;
textName = new JTextField(40);
textName.setText(subject.getName()); // passes the subject name if there
                                                                    // is one, else the box will be
                                                                    // empty (null value)
panel.add(textName, c4);
```

```
GridBagConstraints c5 = new GridBagConstraints();
c5.gridx = 1;
c5.gridy = 3;
c5.anchor = GridBagConstraints.WEST;
buttonSave = new JButton("Save");
panel.add(buttonSave, c5);
buttonSave.addActionListener(this);

GridBagConstraints c6 = new GridBagConstraints();
c6.gridx = 1;
c6.gridy = 3;
c6.anchor = GridBagConstraints.EAST;
buttonCancel = new JButton("Cancel");
panel.add(buttonCancel, c6);
buttonCancel.addActionListener(this);
```

```
        setVisible(true);
}

@Override
public void actionPerformed(ActionEvent arg) { // for when an event is
                                                                    // created!
        if (arg.getSource().equals(buttonCancel)) {
                int answer = JOptionPane.showConfirmDialog(null,
                                "All changes will be lost. Do you want to continue?");
                if (answer == 0)
                        returnValue = null;
                else
                        return;

        } else if (arg.getSource().equals(buttonSave)) {
```

```
                String name = textName.getText();
                if (name == null || name.equals("")) {
                        JOptionPane.showMessageDialog(this, "Name is mandatory",
                                        "Caution", JOptionPane.ERROR_MESSAGE);
                        return;
                }

                returnValue = new Subject();
                returnValue.setSubjectID(subjectID);
                returnValue.setName(name);
        }
        setVisible(false);

}

public Subject getReturnValue() {
```

```
            return returnValue;
        }
}
```

➢ SubjectTableForm

```
package xx.xx.grading.system;

import java.awt.HeadlessException;
import java.awt.Window;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.Set;

import javax.swing.JDialog;
```

```
import javax.swing.JScrollPane;
import javax.swing.JTable;


@SuppressWarnings("serial")
public class SubjectTableForm extends JDialog implements ActionListener,
                MouseListener {


        private JTable subjectTable;
        private Subject returnValue = null;


        public SubjectTableForm(Window owner, Set<Subject> subjectList)
                        throws HeadlessException {


                super(owner, ModalityType.APPLICATION_MODAL);


                int listSize = subjectList.size();
```

```
String[] columnNames = { "ID", "Name" };
Object[][] elements = new Object[listSize][columnNames.length];


int row = 0;
for (Subject sub : subjectList) {
        elements[row][0] = sub.getSubjectID();
        elements[row][1] = sub.getName();
        row++;
}

NonEditableTableModel model = new NonEditableTableModel(elements,
                columnNames);
subjectTable = new JTable(model);
subjectTable.addMouseListener(this);
```

```
        JScrollPane scrollPane = new JScrollPane(subjectTable);


        setSize(200, 200);
        setDefaultCloseOperation(HIDE_ON_CLOSE);
        add(scrollPane);
        setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {


}

@Override
public void mouseClicked(MouseEvent e) {
```

```
        if (e.getClickCount() < 2)
                return;
        int selectedRow = subjectTable.getSelectedRow();
        returnValue = new Subject();
        int id = (int) subjectTable.getValueAt(selectedRow, 0);
        returnValue.setSubjectID(id);
        String name = (String)subjectTable.getValueAt(selectedRow, 1);
        returnValue.setName(name);
        setVisible(false);
}

@Override
public void mouseEntered(MouseEvent e) {


}
```

```
@Override
public void mouseExited(MouseEvent e) {

}

@Override
public void mousePressed(MouseEvent e) {

}

@Override
public void mouseReleased(MouseEvent e) {

}

public Subject getReturnValue() {
```

```
            return returnValue;
        }

}
```

➢ Trimester

```
package xx.xx.grading.system;

public class Trimester {

        private String name;
        private int trimesterID;

        public Trimester(int trimesterID, String name) {

            this.name = name;
```

```
            this.trimesterID = trimesterID;
        }

        public String getName() {
            return name;
        }

        public int getTrimesterID() {
            return trimesterID;
        }

        @Override
        public String toString() {
            return String.format("%d~%s\n", trimesterID, name);
        }
}
```