Lab 7
Java Applet for Employee Management

Roberto Loja
n01100239

Kenn Baker
CENG 212

1. Source Code

1.1 Employee.java

```
/**
 * This is the superclass Employee. It contains the general information for
 * SalesMan and Technician
 */
import java.io.Serializable;

public class Employee implements Comparable<Employee>, Serializable
{
     // Auto generated serialVersionUID.
     private static final long serialVersionUID = 1885732714722332776L;

     private String name;                      // Name of employee
     private int number;                            // Employee number.

     /**
      * Constructor.
      * @param nom Name of the employee
      * @param number Employee number
      */
     public Employee(String nom, int number )
     {
          name = nom;
          this.number = number;
     }
     /**
      * Method for comparison of objects.
      * @param o Employee to compare this with, on the basis of employee name.
      * @return 1 if alphabetically higher than emp, -1 if alphabetically lower
      * than emp, 0 if equal.
      */
     @Override
     public int compareTo(Employee o)
     {
          return this.getName().compareToIgnoreCase(o.getName());
     }

     /**
      * Returns a String of format
      * "name, ID: number".
      * @return A String.
      */
     @Override
     public String toString()
     {
          return this.name + ", ID: " + this.number;
     }

     // getters and setters
     public String getName()    { return this.name;  }
     public int getNumber()  { return this.number; }
     public void setName(String name){ this.name = name; }
     public void setNumber(int number)    { this.number = number; }
}
```

1.2 Salesman.java

```
/**
 * The Salesman class extends Employee, providing a specialized toString()
 * and constructor.
 * @author Roberto Loja
 * @version November 20, 2015
 */
class Salesman extends Employee
{
     // Auto generated serialVersionUID.
     private static final long serialVersionUID = 6811555279668064 7L;
     private double salesTarget;
     private String territory;

     /**
      * Default constructor, takes no parameters, calls Employee's constructor
      * with params name = "no name", employee number = 0. Used to create
      * placeholder Salesman objects.
      */
     public Salesman()
     {
          super("no name", 0);
     }

     /**
      * Complete constructor.
      * @param nom Salesman's name.
      * @param number Salesman's employee number.
      * @param target Salesman's sales target.
      * @param territory Salesman's operating territory.
      */
     public Salesman(String nom, int number, double target, String territory)
     {
          super(nom, number);
          this.salesTarget = target;
          this.territory = territory;
     }

     @Override
     public String toString()
     {
          return super.toString() + ", salesman in " + territory + ". "
                  + "Sales target: " + salesTarget;
     }

     // Getters and setters
     public void setSalesTarget(double target)  { this.salesTarget = target; }
     public void setTerritory(String territory) { this.territory = territory; }

     public double getSalesTarget() { return this.salesTarget; }
     public String getTerritory()   { return this.territory; }
}
```

1.3 Technician.java

```java
/**
 * The Technician class extends Employee, providing a specialized toString()
 * and constructor.
 * @author Roberto Loja
 * @version November 20, 2015
 */
class Technician extends Employee
{
     // Auto generated serialVersionUID.
     private static final long serialVersionUID = 7103581497297119247L;
     private int level;
     private String department;

     /**
      * Default constructor, takes no parameters, calls Employee's constructor
      * with params name = "no name", employee number = 0. Used to create
      * placeholder Technician objects.
      */
     public Technician()
     {
          super("no name", 0);
     }

     /**
      * Full constructor.
      * @param nom Technician's name.
      * @param number Technician's employee number.
      * @param level Technician's level.
      * @param dept Technician's department.
      */
     public Technician(String nom, int number, int level, String dept)
     {
          super(nom, number);
          this.level = level;
          this.department = dept;
     }

     @Override
     public String toString()
     {
          return super.toString() + ", technician in " + department + ". "
                 + "Level: " + level;
     }

     // Getters and setters
     public void setLevel(int level)          { this.level = level; }
     public void setDepartment(String dept) { this.department = dept; }

     public int getLevel()        { return this.level; }
     public String getDepartment() { return this.department; }
}
```

1.4 Company.java

```
import java.util.ArrayList;
import java.util.Collections;

/**
 * Contains an ArrayList of Employee objects and the methods to access or
 * manipulate that list.
 * @author Roberto Loja
 * @version November 30 2015
 */
class Company
{
     private ArrayList<Employee> employees = new ArrayList<Employee>();

     // Class constants. Used for input validation.
     private static final String[] VALID_DEPTS = {"Operations", "Development",
"Quality Assurance"};
     private static final String[] VALID_TERRITORIES = {"North", "South", "West",
"East"};
     private static final int MAX_LEVEL = 3;

     // Calling nextEmpNum() returns a unique employee number.
     private int empNum = 0;
     private int nextEmpNum()
     {
          if (empNum == 0)
          {
               // Find highest employee number previously assigned.
               for (Employee emp : employees)
               {
                    if (emp.getNumber() > empNum)
                         empNum = emp.getNumber();
               }
          }
          return ++empNum;
     }


     /**
      * Constructor. Calls resetList().
      */
     public Company()
     {
          this.resetList();
     }

     /**
      * Resets the ArrayList employees to its initial state.
      */
     public void resetList()
     {
          // Clear the list.
          employees.clear();

          // Reset employee numbers.
          empNum = 1;

          // Default employees.
          employees.add(new Salesman("Shirley", nextEmpNum(), 500, "North"));
          employees.add(new Salesman("Jeff", nextEmpNum(), 800, "East"));
          employees.add(new Salesman("Frank", nextEmpNum(), 900, "West"));
          employees.add(new Salesman("George", nextEmpNum(), 750, "West"));
          employees.add(new Salesman("Lisa", nextEmpNum(), 750, "South"));
          employees.add(new Technician("John", nextEmpNum(), 3, "Operations"));
          employees.add(new Technician("Anne", nextEmpNum(), 3, "Development"));
          employees.add(new Technician("Troy", nextEmpNum(), 2, "Development"));
          employees.add(new Technician("Mark", nextEmpNum(), 1, "Quality 
Assurance"));
          employees.add(new Technician("Jessica", nextEmpNum(), 2, "Quality 
Assurance"));
     }

     /**
      * Sort employees by Employee number.
      */
     public void sortList()
     {
          Collections.sort(employees);
     }

     /**
      * Adds an employee to ArrayList employees. Parameter type must be "t" or
      * "s", territory must be present in VALID_TERRITORIES, department must be
      * present in VALID_DEPTS, level must be greater than 0 and less than
      * MAX_LEVEL. If any of the above conditions are not met, method returns
      * an error code. -1 means the employee type does not exist, 1 means a
      * parameter was invalid.
      * @param type The type of employee (e.g. s for Salesman).
      * @param name Employee name.
      * @param territory If type.equals("s"), this is the employee's territory.
      * @param department If type.equals("T"), this is the employee's department.
      * @param level If type.equals("T"), this is the employee's level.
      * @param salesTarget If type.equals("S"), this is the sales target.
      * @return 0 if employee is successfully added to ArrayList employees.
      */
     public int addEmployee(String type, String name, String territory,
               String department, int level, double salesTarget)
     {
          int value = 0;
          if (type.equalsIgnoreCase("t"))
          {
               // Add a technician with error checking.
               if (validateString(VALID_DEPTS, department) &&
                         (level > 0 && level <= MAX_LEVEL))
               {
                    Technician tech = new Technician(name, nextEmpNum(), level,
                              department);
```

```
                employees.add(tech);

            } else {
                // ERROR: A parameter could not be validated.
                value = 1;
            }

        } else if (type.equalsIgnoreCase("s")) {
            // Add a salesman with error checking.
            if (validateString(VALID_TERRITORIES, territory))
            {
                Salesman sm = new Salesman(name, nextEmpNum(), salesTarget,
                        territory);
                employees.add(sm);
            } else {
                // ERROR: A parameter could not be validated.
                value = 1;
            }
        } else {
            // ERROR: This type of employee doesn't exist.
            value = -1;
        }
        return value;
    }

    /**
     * Deletes specified employee from ArrayList employees.
     * @param employeeNumber The number of the employee to be deleted.
     * @return 0 if successful, -1 if specified employee could not be found.
     */
    public int deleteEmployee(int employeeNumber)
    {
        boolean found = false;
        int value = 0;
        int i = 0;

        while (!found && i < employees.size())
        {
            if (employees.get(i).getNumber() == employeeNumber)
            {
                employees.remove(i);
                found = true;
            }
            i++;
        }

        if (!found)
            value = -1;

        return value;
    }

    /**
     * Checks if a given string is present in a given set. Used to validate
     * strings passed to addEmployee().
```

```
      * @param validSet An array of permitted Strings, likely a class constant.
      * @param toValidate The String to be validated.
      * @return True if toValidate is permitted, false otherwise.
      */
     private boolean validateString(String[] validSet, String toValidate)
     {
          boolean validated = false;

          for (String item : validSet)
          {
               if (item.equals(toValidate))
                    validated = true;
          }

          return validated;
     }

     // Getter for ArrayList employees.
     public ArrayList<Employee> getEmployees() { return employees; }
     public String[] getValidDepts() { return VALID_DEPTS; }
     public String[] getValidTerritories() { return VALID_TERRITORIES; }
     public int getMaxLevel() { return MAX_LEVEL; }
}
```

1.5 UseCompany.java

```
/**
 * Test class for Company.java
 * @author
 * @version
 */
class UseCompany
{
     public static void main(String argv[])
     {
          System.out.println("Call constructor and display default list: ");
          Company company = new Company();

          for (Employee emp : company.getEmployees())
               System.out.println(emp);

          System.out.println();
          System.out.println("Add employee and display list again:");
          company.addEmployee("T", "NEW NAME", "", "Operations", 3, 0);

          for (Employee emp : company.getEmployees())
               System.out.println(emp);

          System.out.println();
          System.out.println("Try to add Salesman with territory=\"Canada\":");
          if (company.addEmployee("S", "BAD DATA", "Canada", "", 0, 1000) == 1)
               System.out.println("Failed.");
          else
```

```
                System.out.println("Succeeded");

            System.out.println();
            System.out.println("Try to add an employee of type \"Manager\":");
            if (company.addEmployee("Manager", "BAD DATA", "", "", 0, 1000) == -1)
                System.out.println("Failed.");
            else
                System.out.println("Succeeded");

            System.out.println();
            System.out.println("Remove employee with ID number 8, display again:");
            company.deleteEmployee(8);

            for (Employee emp : company.getEmployees())
                System.out.println(emp);

            System.out.println();
            System.out.println("Reset list and display:");
            company.resetList();

            for (Employee emp : company.getEmployees())
                System.out.println(emp);
        }
}
```

1.6 CompanyApplet.java

```
import java.awt.BorderLayout;
import java.awt.GridLayout;

import javax.swing.JApplet;
import javax.swing.JLabel;

/**
 * Main applet class.
 *
 * @author
 * @version
 */

// the GUI logic (i.e. applet display) consists of three panels - one to display
// a summary of data; one to allow the add or delete of a specific record; and
// one to contain any required control buttons. This is similar to the Brandi's
// Bagels and the GICApplet.

//
/*
 * In the top panel, the applet will display all of the Technicians and/or
 * Salesmen as were controlled in previous project. They will be in a text area
 * (JTextArea) or other suitable component.
 *
 * The middle area will provide a panel with text fields/radio buttons/combo
 * boxes/etc. to enter data to add a new Technician's or SalesMan's or the
 * Employee Number to delete an employee.
 *
 * The bottom area will provide a panel with buttons for reset to original data,
 * display, sort, add and delete. You can use a single listener and then check
 * the buttons with an if statement or create individual listeners.
 */

public class CompanyApplet extends JApplet {
     JLabel label1, label2;

     public CompanyApplet() { // Default layout manager is BorderLayout
          // super("Company Applet");

     }

     public void init() {

          setLayout(new GridLayout(3, 1));
          Company company = new Company();
          setSize(500, 500);

          TopPanel topPanel = new TopPanel(company);
          MiddlePanel middlePanel = new MiddlePanel(company);
          add(topPanel);
          add(middlePanel);

          add(new BottomPanel(company, topPanel, middlePanel));

     }
}
```

1.7 TopPanel.java

```
import java.awt.BorderLayout;

import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class TopPanel extends JPanel {

     // In the top panel, the applet will display all of the Technicians and/or
     // Salesmen as were controlled in previous project. They will be in a text
     // area (JTextArea) or other suitable component.

     private JTextArea displayInformation;
     private Company company;

     public TopPanel(Company company) {
          setLayout(new BorderLayout());
          displayInformation = new JTextArea();
          displayInformation.setEditable(false);
           JScrollPane scroll = new JScrollPane (displayInformation);

scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
```

```
		add(scroll, BorderLayout.CENTER);
		this.company = company;
		updateDisplayInformation();
	}

	public void resetInformation (){
		company.resetList();
		updateDisplayInformation();
	}

	public void updateDisplayInformation() {
		String displayString = "";
		for (Employee e : company.getEmployees()){
			displayString += e;
			displayString += "\n";
		}
		displayInformation.setText(displayString);

	}
}
```

1.8 MiddlePanel.java

```
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.NumberFormat;
import java.util.Arrays;
import java.util.Vector;

import javax.swing.ButtonGroup;
import javax.swing.JComboBox;
import javax.swing.JFormattedTextField;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;
import javax.swing.text.NumberFormatter;

public class MiddlePanel extends JPanel implements ActionListener {

	// The middle area will provide a panel with text fields/radio buttons/combo
	// boxes/etc. to enter data to add a new Technician's or SalesMan's or the
	// Employee Number to delete an employee.


	private final static String TECHNICIAN = "Technician";
	private final  static String SALESMAN = "Salesman";

	private Company company;
```

```
//

	private final JLabel nameLabel = new JLabel("NAME");
	private JTextField nameTextArea;


	private ButtonGroup  typeOfEmployeeGroup ;
	private final JLabel typeOfEmployee = new JLabel("TYPE OF EMPLOYEE");
	private JRadioButton salesmanRadioButton;
	private JRadioButton technicianRadioButton;

	private JPanel levelPanel;
	private final JLabel levelsLabel = new JLabel("LEVEL");
	private JComboBox levels;

	private JPanel saleTargetPanel;
	private final JLabel salesTargetLabel = new JLabel("SALES TARGET");
	private JTextField salesTarget;

	private JPanel terrorityPanel;
	private final JLabel terrorityLabel = new JLabel("TERRORITY");
	private JComboBox terrority;

	private JPanel departmentPanel;
	private final JLabel departmentLabel = new JLabel("DEPARTMENT");
	private JComboBox department;


	public MiddlePanel(Company company) {
		this.company = company;

		setLayout(new GridLayout(5,1));

		//intFormatter = createIntegerFormatter();


		JPanel namePanel = new JPanel();
		//namePanel.setLayout(new FlowLayout());
		nameTextArea = new JTextField(15);
		namePanel.add(nameLabel);
		namePanel.add(nameTextArea);


		JPanel typePanel = new JPanel();



		salesmanRadioButton = new JRadioButton(SALESMAN);
		salesmanRadioButton.setActionCommand(SALESMAN);
		salesmanRadioButton.addActionListener(this);
		technicianRadioButton = new JRadioButton(TECHNICIAN);
		technicianRadioButton.setActionCommand(TECHNICIAN);
		technicianRadioButton.addActionListener(this);

	    typeOfEmployeeGroup =  new ButtonGroup();
```

```
typeOfEmployeeGroup.add(salesmanRadioButton);
typeOfEmployeeGroup.add(technicianRadioButton);
typeOfEmployeeGroup.setSelected(salesmanRadioButton.getModel(), true);

typePanel.add(typeOfEmployee);
typePanel.add(salesmanRadioButton);
typePanel.add(technicianRadioButton);


JPanel firstPanel = new JPanel();


 levelPanel = new JPanel();


levels = populateComboBoxes(generateLevelList() );
levelPanel.add(levelsLabel);
levelPanel.add(levels);

 saleTargetPanel = new JPanel();
salesTarget = new JTextField();
salesTarget.setColumns(15);

saleTargetPanel.add(salesTargetLabel);
saleTargetPanel.add(salesTarget);

firstPanel.add(levelPanel);
levelPanel.setVisible(false);

firstPanel.add(saleTargetPanel);


JPanel secondPanel = new JPanel();

 terrorityPanel = new JPanel();
terrority = populateComboBoxes(company.getValidTerritories());

terrorityPanel.add(terrorityLabel);
terrorityPanel.add(terrority);

 departmentPanel = new JPanel();
department = populateComboBoxes( company.getValidDepts());
departmentPanel.add(departmentLabel);
departmentPanel.add(department);
departmentPanel.setVisible(false);

secondPanel.add(terrorityPanel);
secondPanel.add(departmentPanel);

add(namePanel);

add(typePanel);
add(firstPanel);
```

```
            add(secondPanel);
            setVisible(true);

      }

      private JComboBox<?> populateComboBoxes ( Object arrayFilled){
            Vector comboBoxItems=new Vector<Object>();


            if (arrayFilled instanceof int[]){
                  int[] intArrayFilled = (int[]) arrayFilled;
                  for (int i : intArrayFilled){

                        comboBoxItems.add(i+"");
                  }

            }
            else if (arrayFilled instanceof String[]){
                  String[] stringArrayFilled = (String[]) arrayFilled;
                  for (String s : stringArrayFilled){
                        comboBoxItems.add(s);
                  }
            }
            return new JComboBox<>(comboBoxItems);
      }

      public void sendDataFields() {
            //JOptionPane.showMessageDialog(this, );
            String getTypeofEmployee =
typeOfEmployeeGroup.getSelection().getActionCommand();

            if (getTypeofEmployee.equals(SALESMAN)){
                  try {
                        company.addEmployee("s", nameTextArea.getText(),
terrority.getSelectedItem().toString(), "", -1,
Double.parseDouble(salesTarget.getText()));

                  }catch (NumberFormatException e){
                        JOptionPane.showMessageDialog(this, "INVALID SALES ");
                  }

            }
            else if (getTypeofEmployee.equals(TECHNICIAN)){
                  company.addEmployee("t", nameTextArea.getText(), "",
department.getSelectedItem().toString(),Integer.parseInt( levels.getSelectedItem().t
oString()), 0);
            }
      }

      private int [] generateLevelList(){
            int maxlevel = company.getMaxLevel();
            int[] arrayReturn = new int[maxlevel];
            for (int i = 1 ; i < maxlevel+1;i++){
                  arrayReturn[i-1] = i;
```

```
		}
		return arrayReturn;
	}

	@Override
	public void actionPerformed(ActionEvent arg0) {
		String selected = arg0.getActionCommand() ;

		if (selected.equalsIgnoreCase(TECHNICIAN)){
			terrorityPanel.setVisible(false);
			saleTargetPanel.setVisible(false);
			salesTarget.setText("");
			departmentPanel.setVisible(true);
			levelPanel.setVisible(true);


		}
		else if(selected.equalsIgnoreCase(SALESMAN)){
			terrorityPanel.setVisible(true);
			saleTargetPanel.setVisible(true);
			departmentPanel.setVisible(false);
			levelPanel.setVisible(false);
		}

	}


}
```

1.9 BottomPanel.java

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

import javax.swing.JButton;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

//The bottom area will provide a panel with buttons for reset to original data,
//display, sort, add and delete. You can use a single listener and then check the
buttons
//with an if statement or create individual listeners.

public class BottomPanel extends JPanel {

	private JButton resetButton;
	private JButton sortButton;
	private JButton addButton;
	private JButton deleteButton;
	Company company;
	TopPanel topPanel;
	MiddlePanel middlePanel;
```

```
public BottomPanel(Company company, TopPanel topPanel,
          MiddlePanel middlePanel) {

     setLayout(new FlowLayout());
     resetButton = new ResetButton();
     sortButton = new SortButton();
     addButton = new AddButton();
     deleteButton = new DeleteButton();

     this.company = company;
     this.topPanel = topPanel;
     this.middlePanel = middlePanel;

     add(resetButton);
     add(sortButton);
     add(addButton);
     add(deleteButton);

}

class ResetButton extends JButton implements ActionListener {

     public ResetButton() {
          super("Reset");
          addActionListener(this);
     }

     @Override
     public void actionPerformed(ActionEvent arg0) {
          topPanel.resetInformation();
     }

}

class SortButton extends JButton implements ActionListener {

     public SortButton() {
          super("Sort");
          addActionListener(this);
     }

     @Override
     public void actionPerformed(ActionEvent arg0) {

          // JOptionPane.showMessageDialog(this, "Sort Showed");
          company.sortList();
          topPanel.updateDisplayInformation();
     }

}

class AddButton extends JButton implements ActionListener {

     public AddButton() {
          super("Add");
          addActionListener(this);
```

```
		}

		@Override
		public void actionPerformed(ActionEvent arg0) {
			middlePanel.sendDataFields();
			topPanel.updateDisplayInformation();
			// JOptionPane.showMessageDialog(this, "Add Showed");

		}

	}

	class DeleteButton extends JButton implements ActionListener {

		public DeleteButton() {
			super("Delete");
			addActionListener(this);
		}

		@Override
		public void actionPerformed(ActionEvent arg0) {
			int input = -9999;
			do {
				try {
					input = Integer
							.parseInt((String) JOptionPane
									.showInputDialog(
											this,
											"Enter only Employee
number you wish to delete",
											"",

	JOptionPane.INFORMATION_MESSAGE,
											null, null, ""));
				} catch (NumberFormatException e) {
					JOptionPane.showMessageDialog(this,
							"INVALID EMPLOYEE NUMBERS");
				}
			} while (input == -9999);

			int found = company.deleteEmployee(input);

			if (found == 0) {
				topPanel.updateDisplayInformation();
			}
			else {
				JOptionPane.showMessageDialog(this,
						"EMPLOYEE NUMBER " + input + " NOT FOUND");
			}
		}
	}
}
```

2. Bugs and Needed Improvements

Currently, the cancel button on the Delete Employee dialogue box does not work. It reacts to any input (including no input) by displaying an error message. Further, we would have liked to include the ability to save and load the employees array to an object file.

3. Web location

The applet can be accessed via the following webpage:

http://munro.humber.ca/~n01100239/website/Applet.html