7th homework assignment; JAVA, Academic year 2011/2012; FER

As usual, please see the last page. I mean it! You are back? OK. Here we have three problems for you to solve.

Problem 1.

We will start with problems illustrating layout manager creation. Please read:

http://docs.oracle.com/javase/tutorial/uiswing/layout/custom.html

Be aware that each container can reserve parts of its surface for other purposes and these parts of its surface can not contain components. Fortunately, those "reserved" parts, if exists, are always bound to component top, bottom, left and right sides, and are knows as insets. For example, consider a container whose `size` is 300 x 200. Lets assume that it also has following insets defined: `left=10, top=20, right=15, bottom=5.` With these insets, area left for components has width 300-10-15=275 and height 200-20-5=175. So instead of placing components on that container from (0,0) to (299,199) you can only use (10,20) to (284,194).

Now make your self familiar with the source code of `BoxLayout` that is part of Java Standard Edition. Analyze how it uses following methods: `Container#getSize()`, `Container#getInsets()`, `Container#getComponentCount()`, `Container#getComponent(int index)`. Analyze how it calculates `minimum, preferred` and `maximum` size for layouts and how it uses `SizeRequirements` class. Observe how it invalides layout automatically each time a new component is added or removed from container. Learn the difference between `SizeRequirements.calculateTiledPositions` and `SizeRequirements.calculateAlignedPositions`.

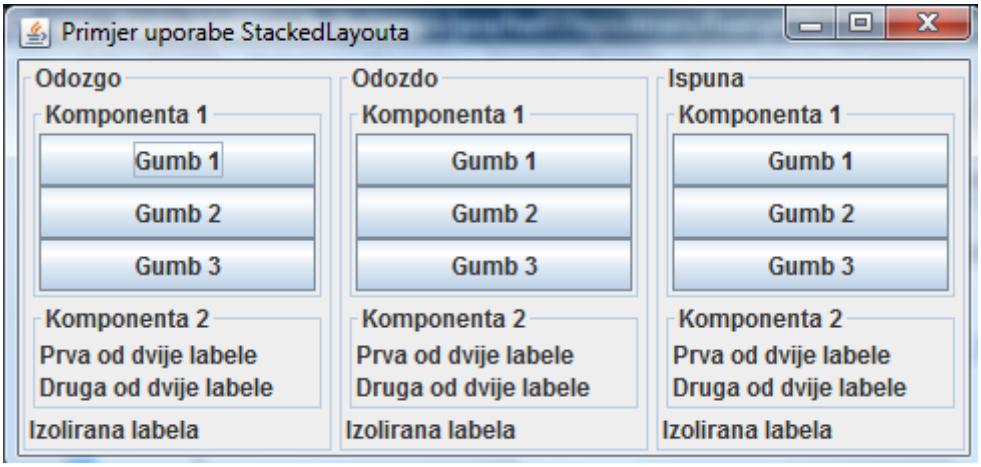
Now, your first task is to write `StackedLayout` layout manager. Place it in package `hr.fer.zemris.java.hw07.layoutmans`. It is a manager that places the components along the vertical axis; it does not respect their preferred width when doing layout but instead it always stretches the components to fill the horizontal area. However, it does respect their preferred height (with a twist :-)) You are to write internal public enum `StackedLayoutDirection` which defines following values: `FROM_TOP, FROM_BOTTOM` and `FILL`. In case of `FROM_TOP`, components are placed from top of container. In case of `FROM_BOTTOM`, components are placed in such a way that the last component is placed at the bottom of container. In case of `FILL`, components are stretched so that they fill entire container.

Here is the usage example. Copy it and run it.

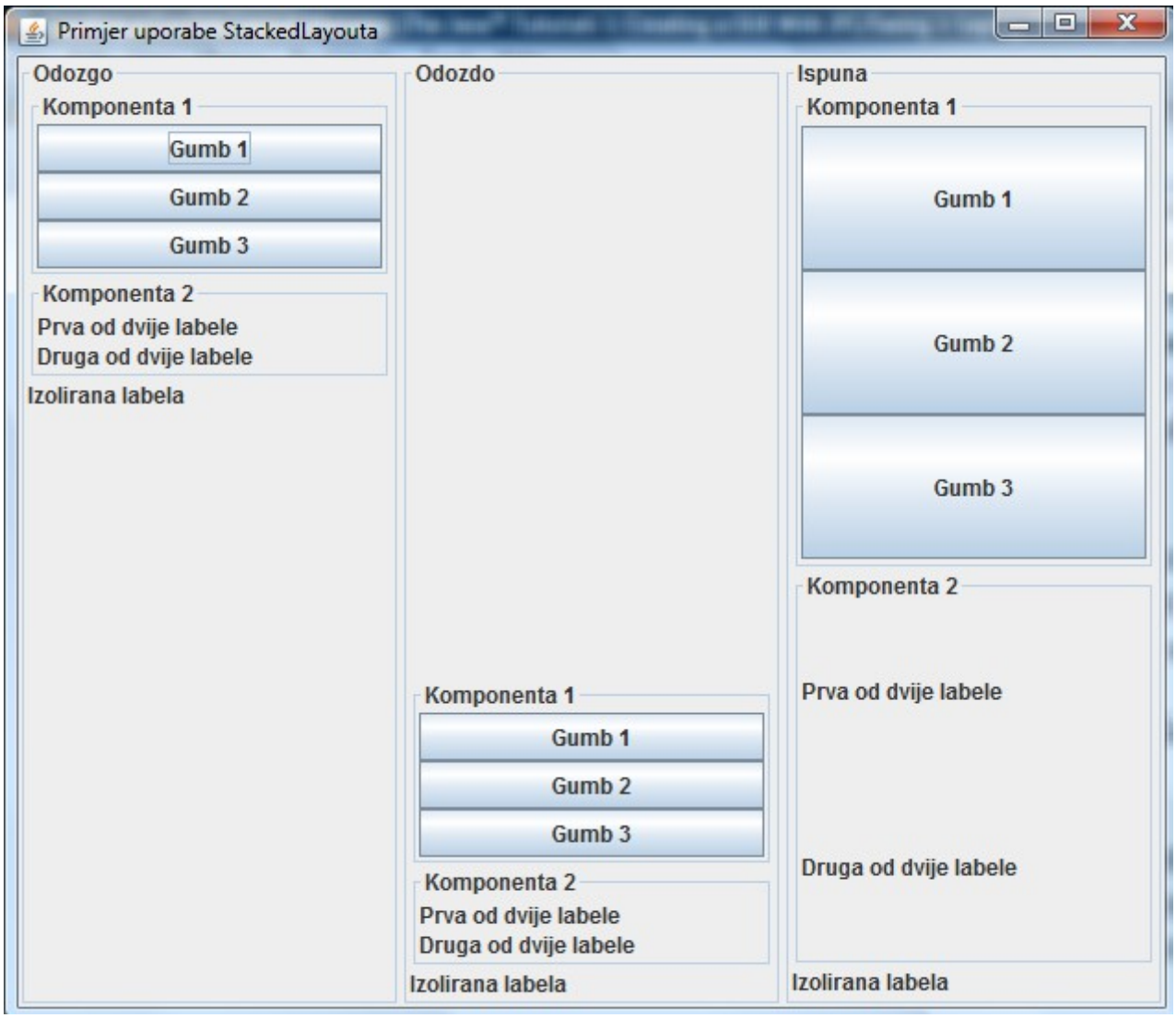
```java
package hr.fer.zemris.java.hw07.layoutmans;

import hr.fer.zemris.java.hw07.layoutmans.Stackedlayout.StackedLayoutDirection;

import java.awt.GridLayout;
import java.awt.LayoutManager2;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

public class ExampleStackedFrame extends JFrame {

	private static final long serialVersionUID = 8818691790593467664L;

	public ExampleStackedFrame() {
		setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
		setTitle("Primjer uporabe StackedLayouta");
		initGUI();
		pack();
	}

	private void initGUI() {
		this.getContentPane().setLayout(new GridLayout(1,3));
		this.getContentPane().add(makePanel(
			"Odozgo", new Stackedlayout(StackedLayoutDirection.FROM_TOP)));
		this.getContentPane().add(makePanel(
			"Odozdo", new Stackedlayout(StackedLayoutDirection.FROM_BOTTOM)));
		this.getContentPane().add(makePanel(
			"Ispuna", new Stackedlayout(StackedLayoutDirection.FILL)));
	}

	private JComponent makePanel(String tekst, LayoutManager2 manager) {
		JPanel panel = new JPanel(manager);
		panel.setBorder(BorderFactory.createTitledBorder(tekst));

		JPanel p1 = new JPanel(new GridLayout(3,1));
		p1.setBorder(BorderFactory.createTitledBorder("Komponenta 1"));
		p1.add(new JButton("Gumb 1"));
		p1.add(new JButton("Gumb 2"));
		p1.add(new JButton("Gumb 3"));

		JPanel p2 = new JPanel(new GridLayout(2,1));
		p2.setBorder(BorderFactory.createTitledBorder("Komponenta 2"));
		p2.add(new JLabel("Prva od dvije labele"));
		p2.add(new JLabel("Druga od dvije labele"));

		panel.add(p1);
		panel.add(p2);
		panel.add(new JLabel("Izolirana labela"));

		return panel;
	}
```

```
	public static void main(String[] args) {
		SwingUtilities.invokeLater(new Runnable() {
			@Override
			public void run() {
				new ExampleStackedFrame().setVisible(true);
			}
		});
	}
}
```

If you did it correctly, you should get the frame as illustrated on following picture.



If you try to resize it, this should be the result:



Please note that your `StackedLayout` should not accept container in constructor and it should not remember it in any way. However, you are free to assume (and please document that assumption) that instances of your layout manager will not be shared among multiple containers.

You are not allowed to derive your layout manager from any existing managers (either by extending it or by encapsulating it); you must write you manager simply by extending `LayoutManager2` and adding appropriate code. You can, however, use classes that are prepared to help the autors that create layout managers (for example: `SizeRequirements`).

Problem 2.

Make your self familiar with the source code of `BorderLayout` that is part of Java Standard Edition. Observe the difference between BoxLayout and BorderLayout: while BoxLayout can handle arbitrary number of components, BorderLayout expected to get its components from a method:

```
void addLayoutComponent(Component comp, Object constraints);
```

where constraints is come predefined constraint and based on that constraint it remembers the component. See how the layout process is now performed.

Your job it to write `SimpleLayout` layout manager (put it in package `hr.fer.zemris.java.hw07.layoutmans`) that defines internal public `enum SimpleLayoutPlacement` with values *`LEFT`* and *`CENTER`*. The component added with *`LEFT`* constraint should always be placed on the left or right side depending on components orientation (please check documentation of `Component#getComponentOrientation()` and of class `ComponentOrientation`); if component is "LEFT_TO_RIGHT", it should be placed along left side of container; otherwise along right side of container. Its preferred width should be respected; its height should be such that it covers entire vertical available space (minus insets). Component added with *`CENTER`* constraint should fill entire remaining space. Please note that user does not have to add both components (or even a single one).

In same package add class `ExampleSimpleLayoutFrame` that will illustrate behavior of your layout manager. The content of the frame should be similar to the content of frame `ExampleStackedFrame`: in first panel you should add a component with *`LEFT`* constraint; in middle panel you should add a component with *`CENTER`* constraint and it right panel you should add two components, one with *`LEFT`* and one with *`CENTER`* constraint.

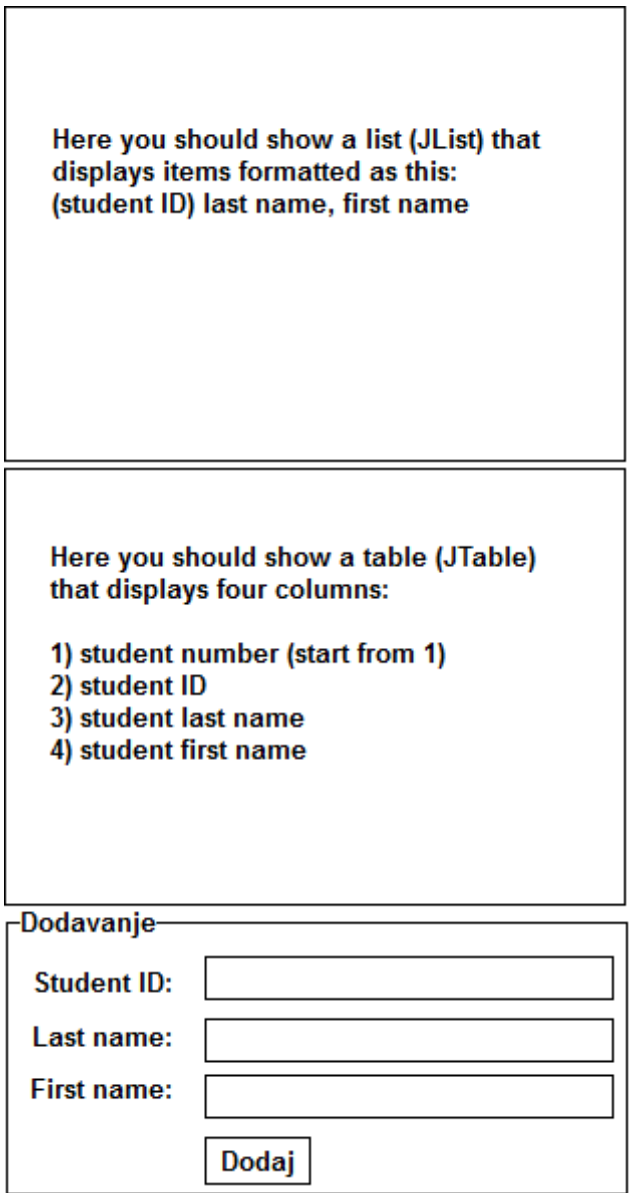
You are not allowed to derive your layout manager from any existing managers (either by extending it or by encapsulating it); you must write you manager simply by extending `LayoutManager2` and adding appropriate code. You can, however, use classes that are prepared to help the autors that create layout managers (for example: `SizeRequirements`).

Problem 3.

Prepare a text file `students.txt` and place it in your eclipse project. It should contain one student record per row, where each record contains three entries separated by TAB character: student ID (i.e. JMBAG), last name and first name.

Make a program `StudentBrowser` (place it in package `hr.fer.zemris.java.hw07.students`) that will read the content of this file and that will allow user to modify it, either by adding entries or by removing it. After each change new content of file should be stored on disk.

The GUI part should look as illustrated on following picture.



This is only an illustration pained in Paint. Make your GUI mimic this as closely as possible. Here is what you should do.

1. Read about `ListModel`; read about `AbstractListModel` – it is an implementation of interface `ListModel` that handles the registration of listeners and provides firing methods for you – all you have to do it to implement the interesting methods (`getSize()`, `getElementAt()`); you are encouraged to use this class in your homework.
   http://docs.oracle.com/javase/7/docs/api/javax/swing/ListModel.html
   http://docs.oracle.com/javase/7/docs/api/javax/swing/AbstractListModel.html

2. Read about `TableModel`; read about `AbstractTableModel` – it is an implementation of interface `TableModel` that handles the registration of listeners and provides firing methods for you – all you have to do it to implement the interesting methods (`getRowCount()`, `getElementAt()`, etc); you are encouraged to use this class in your homework.
   http://docs.oracle.com/javase/7/docs/api/javax/swing/table/TableModel.html
   http://docs.oracle.com/javase/7/docs/api/javax/swing/table/AbstractTableModel.html

Since you have multiple different view of your data, you are required to create a class `Student` (that models a single student) and an interface `StudentDatabase` that models the collection of students (put them in

package `hr.fer.zemris.java.hw07.students`).

`StudentDatabase` is an ordered collection of students. It provides methods:

```
// number of students
int size();
// get index-th student
Student getStudent(int index);
// find position of given student
int indexOf(Student student);
// remove index-th student
void remove(int index);
// add if not already present
boolean addStudent(String studentID, String lastName, String firstName);
```

Write an implementation class `FileStudentDatabase` for interface `StudentDatabase` (in the same package). In constructor, it should accept Path to file with student records. The constructor should read all records from that file. Handle exceptions as appropriate (for example, use `JoptionPane.showMessage` to display error messages). All methods that modify data should automatically flush the modifications back to file!

Now, since you have two models that depend on this database (your `ListModel` and your `TableModel`), you will utilize Observer design pattern to interconnect them. Make your `StudentDatabase` interface model a Subject and implement your `ListModel` and `TableModel` as observers of that Subject. In order to do that:

- define interface `StudentDBModificationListener` with methods:

  ```
  void studentsAdded(StudentDatabase db, int fromIndex, int toIndex);
  void studentsRemoved(StudentDatabase db, int fromIndex, int toIndex);
  void databaseChanged(StudentDatabase db);
  ```

  Methods `studentsAdded` and `studentsRemoved` should be called when a simple modification occurred (for example, a single range of students was affected); otherwise, `databaseChanged` should be called to inform the observers that a some complex change has occurred.

- Add to `StudentDatabase` class methods for registration and deregistration of `StudentDBModificationListener-s` and make sure that you implement all methods in such a way that for every modification they do alert registered listeners what has happened.

- Create your implementation of `ListModel` and `TableModel` (you can base them on `AbstractListModel` and `AbstractTableModel`) with constructors that accept a reference to `StudentDatabase.` Your implementations should register themselves as listeners on provided `StudentDatabase.` When they are notified that something has changed in `StudentDatabase`, they should notify their registered listeners with appropriate notification (those listeners will be instances of `JList` and `JTable` classes or some objects connected with them since they are the final components that visualize the student data).

Add `KeyListeners` to your `JList` and `JTable` objects. When a "`Del`" key is pressed (either when `JList` has focus or `JTable`), a selected item should be deleted from `StudentDatabase.`

Have fun!

Please note. You can consult with your peers and exchange ideas about this homework *before* you start actual coding. Once you open you IDE and start coding, consultations with others (except with me) will be regarded as cheating. You can not use any of preexisting code or libraries for this homework (whether it is yours old code or someones else). Document your code!

In order to solve this homework, create a blank Eclipse Java Project and write your code inside. Once you are done, export project as a ZIP archive and upload this archive on Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted.