8th homework assignment; JAVA, Academic year 2012/2013; FER

As usual, please see the last page. I mean it! You are back? OK. Here we have two 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`.

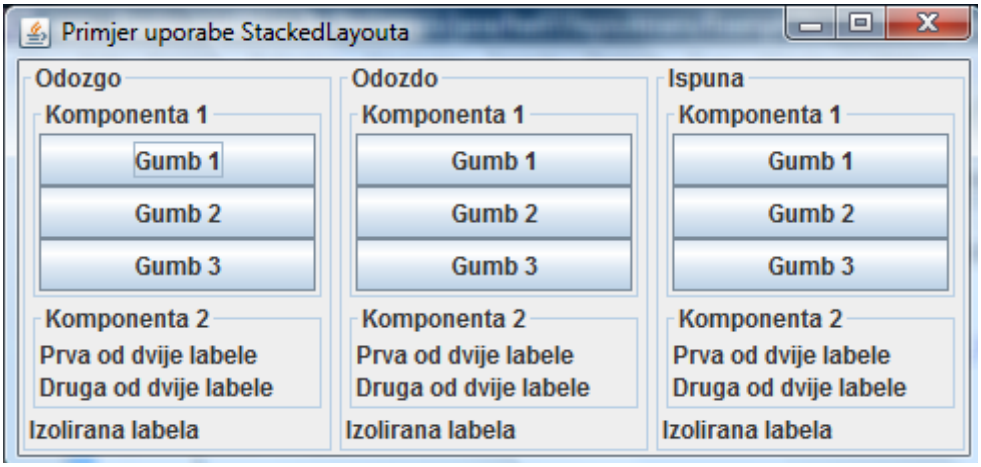
Now, your first task is to write `StackedLayout` layout manager. Place it in package `hr.fer.zemris.java.hw07.layoutmans`. It is a manager that places the components along the vertical axis; it does not respect their preferred width when doing layout but instead it always stretches the components to 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.

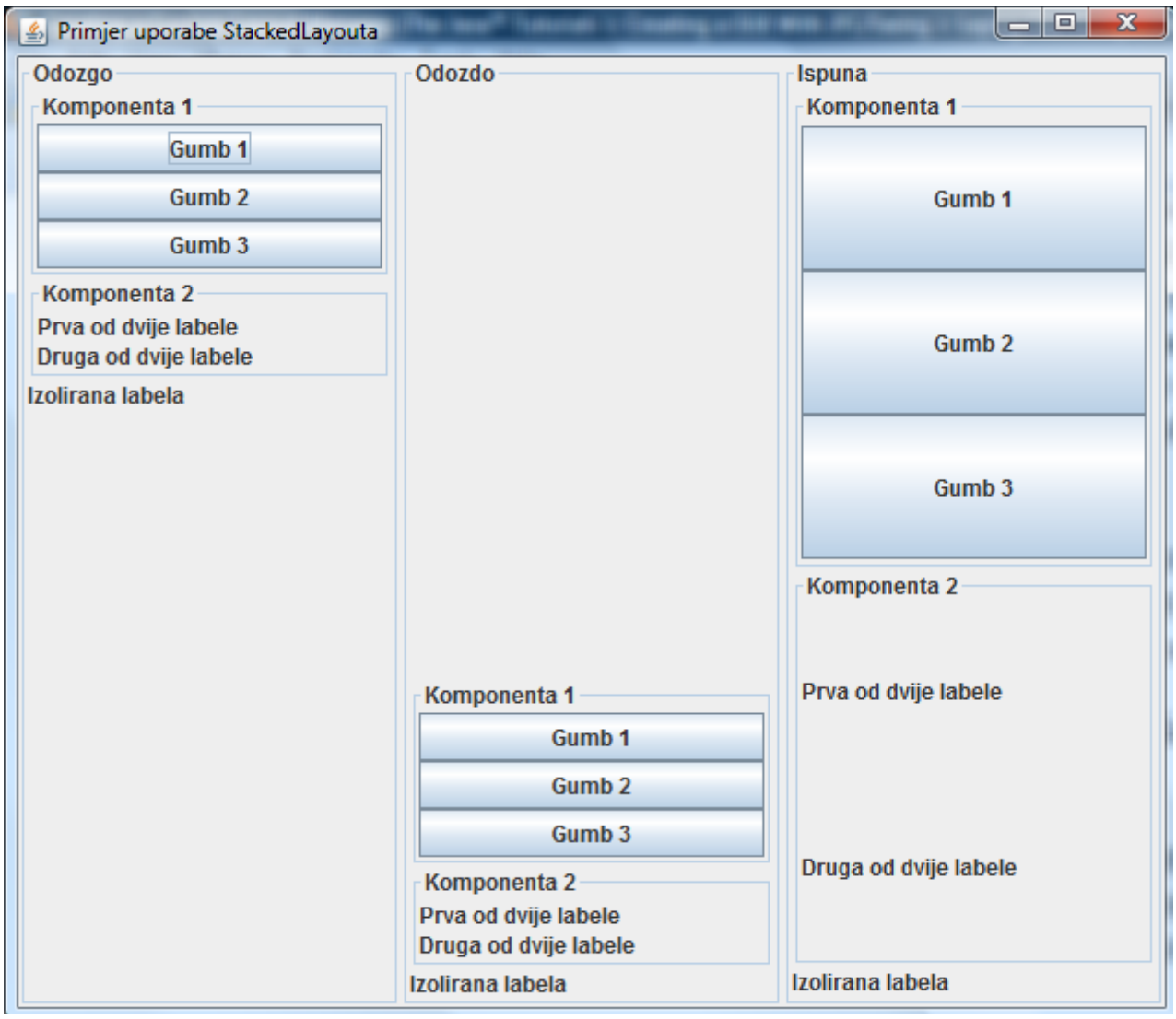
```java
package hr.fer.zemris.java.hw07.layoutmans;

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.

Your task is to create a simple text file editor called `JNotepad++`. The name of this editor must be shown in window's title. `JNotepad++` must allow user to work with multiple documents at the same time. For this, you must use `JTabbedPane` component:

http://docs.oracle.com/javase/7/docs/api/javax/swing/JTabbedPane.html
http://docs.oracle.com/javase/tutorial/uiswing/components/tabbedpane.html

For this problem use the package `hr.fer.zemris.java.hw07.jnotepadpp` and any subpackages you need. Your application must be startable by method `main` located in class `JnotepadPP` in package `hr.fer.zemris.java.hw07.jnotepadpp`.

For text editing use `JTextArea` component. For each open document you will create a new instance of `JTextArea` for it; this component will be then (indirectly) added to `JTabbedPane`. I say indirectly because you must wrap it into `JScrollPane` and you may add this `JScrollPane` into `JPanel` (or other containers) which will eventually be added into `JTabbedPane`.

Your application must provide following functionality to user:

- creating a new blank document,
- opening existing document,
- saving document,
- saving-as document (warn user if file already exists),
- cut/copy/paste text,
- statistical info,
- exiting application.

All of those actions must be available from:

- menus (organize them as you see fit),
- dockable toolbar,
- keyboard shortcuts.

Please see:
http://docs.oracle.com/javase/7/docs/api/javax/swing/JToolBar.html
http://docs.oracle.com/javase/tutorial/uiswing/components/toolbar.html

For open/save file selection use standard Java build-in dialogs: `JFileChooser`. See:
http://docs.oracle.com/javase/7/docs/api/javax/swing/JFileChooser.html
http://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html

If user attempts to close program, you must check if there are any modified but unsaved text documents. If there are, ask user for each document if he wants to save the changes, discard the changes or abort the closing action. Simplest way to implement this is to set default closing operation to be `DO_NOTHING_ON_CLOSE` and then to register in your window constructor your implementation of `WindowListener` so that you can be informed when user attempts to close the program (use method `windowClosing` of interface `WindowListener`). Actually, read about `WindowListener` interface:

http://docs.oracle.com/javase/7/docs/api/java/awt/event/WindowListener.html

and about `WindowAdapter` class:

http://docs.oracle.com/javase/7/docs/api/java/awt/event/WindowAdapter.html

Using method `addWindowListener` add an instance of an anonimous class that derives from `WindowAdapter` and not directly from `WindowListener` (do you understand why this is convenient?). In you implementation of `windowClosing` method call a method that will do the required checking. If everything is OK, this method should end with a call to `dispose()` method which will close the window and eventually the program. If user decides to abort closing, you must skip the call to the `dispose()` method. When user calls the "exit" action from menu, you should simply call again your method that will check the status of all documents and that will allow user to abort the closing.

For communication with user, please use `JOptionPane` and its methods `showMessageDialog` and `showConfirmDialog`. See:

http://docs.oracle.com/javase/7/docs/api/javax/swing/JOptionPane.html
http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html

When user requests statistical info on document, you should calculate:

- a number of characters found in document (everything counts),
- a number of non-blank characters found in document (you don't count spaces, enters and tabs),
- a number of lines that the document contains.

Calculate this and show an informational message to user having text similar to: "Your document has X characters, Y non-blank characters and Z lines.".

When opening and saving the files, always use UTF-8 code page.

At all times, the path of currently selected document must be visible in window's title. To find out when the tab has changes, add appropriate listener to `JTabbedPane` component. Be careful to add this only once and not for each opened document. Here is a helpful example.

```
tabbedPane.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            System.out.println("Tab: " + tabbedPane.getSelectedIndex());
        }
    });
```

If user is currently editing C:\example.txt, the expected window title is:

```
C:\example.txt - JNotepad++
```

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.

Equip the project with appropriate `build.xml`. You must support two run targets. Target `run1` must start the solution of the first problem: a `JFrame` must open that will demonstrate the behavior of the developed `LayoutManager`.

The target `run2` must start `JNotepad++`.

You are not required to create any unit tests in this homework, so you can remove appropriate task from the `build.xml` if its existence starts to fail the builds due to the fact that no tests exists.