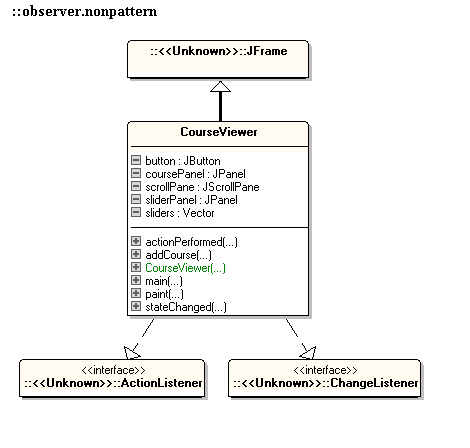
1. Proyecto 1 patrón observer

Diagrama de clases



En el primer proyecto se modificó la clase CourseViewer para que muestre un grafica de pastel, el siguiente código es de la única clase modificada CourseViewer.

1.1 Clase CourseViewer

**package** observer.nonpattern;

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** java.awt.GridBagConstraints;

**import** java.awt.GridBagLayout;

**import** java.awt.GridLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.util.Vector;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JScrollPane;

**import** javax.swing.JSlider;

**import** javax.swing.border.TitledBorder;

**import** javax.swing.event.ChangeEvent;

**import** javax.swing.event.ChangeListener;

**import** observer.CourseRecord;

**import** observer.LayoutConstants;

/\*\*

\* Presents a barchart view of a set of courses and their marks. No pattern is

\* used in this program.

\*/

@SuppressWarnings("serial")

**public** **class** CourseViewer **extends** JFrame **implements** ActionListener,

ChangeListener {

/\*\*

\* Create a CourseViewer object

\*/

**public** CourseViewer() {

**this**.setTitle("Observer Pattern -- Non Pattern Version");

**this**.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

sliders = **new** Vector<JSlider>();

sliderPanel = **new** JPanel();

sliderPanel.setLayout(**new** GridBagLayout());

sliderPanel.setBackground(Color.***white***);

**this**.getContentPane().setLayout(**new** GridBagLayout());

coursePanel = **new** JPanel();

coursePanel.setBorder(**new** TitledBorder("Courses"));

coursePanel.setLayout(**new** GridLayout(0, 1));

GridBagConstraints constraints = **new** GridBagConstraints();

constraints.fill = GridBagConstraints.***BOTH***;

scrollPane = **new** JScrollPane(coursePanel,

JScrollPane.***VERTICAL\_SCROLLBAR\_ALWAYS***,

JScrollPane.***HORIZONTAL\_SCROLLBAR\_AS\_NEEDED***);

// Layout code

constraints.weightx = 0.0;

constraints.weighty = 1.0;

constraints.gridx = 0;

constraints.gridy = 0;

sliderPanel.add(scrollPane, constraints);

button = **new** JButton("New Course");

button.addActionListener(**this**);

// Layout code

constraints.weightx = 0.0;

constraints.weighty = 0.0;

constraints.gridy = 1;

sliderPanel.add(button, constraints);

// Layout code

constraints.weightx = 0.0;

constraints.weighty = 0.0;

constraints.gridx = 0;

constraints.gridy = 0;

**this**.getContentPane().add(sliderPanel, constraints);

// Layout code

constraints.weightx = 1.0;

constraints.weighty = 1.0;

constraints.gridx = 1;

constraints.gridy = 0;

// the bar chart will be drawn over this panel

**this**.getContentPane().add(**new** JPanel(), constraints);

**this**.setVisible(**true**);

}

/\*\*

\* Add a new Course object

\*

\* **@param** courseRecord

\* the CourseRecord to be added

\*/

**public** **void** addCourse(CourseRecord courseRecord) {

**boolean** alreadyExists = **false**;

**for** (**int** i = 0; i < sliders.size(); i++) {

JSlider slider = sliders.elementAt(i);

**if** (slider.getName().equals(courseRecord.getName())) {

alreadyExists = **true**;

JOptionPane

.*showMessageDialog*(

**null**,

"Warning: Attempt to add new course with an already existing name",

"alert", JOptionPane.***ERROR\_MESSAGE***);

i = sliders.size(); // exit the loop

}

}

**if** (!alreadyExists) { // add the slider

JSlider slider = **new** JSlider();

slider.setBackground(Color.***white***);

slider.setName(courseRecord.getName());

slider.setValue(50);

slider.setMinimum(0);

slider.setMaximum(100);

slider.setMajorTickSpacing(25);

slider.setMinorTickSpacing(5);

slider.setPaintTicks(**true**);

slider.setPaintLabels(**true**);

slider.setBorder(**new** TitledBorder(courseRecord.getName()));

slider.addChangeListener(**this**);

coursePanel.add(slider);

coursePanel.revalidate();

sliders.addElement(slider);

**this**.setSize(LayoutConstants.***xOffset*** + 50 + **this**.sliders.size()

\* (LayoutConstants.***barWidth*** + LayoutConstants.***barSpacing***),

(sliders.size() + 1) \* 150 + **this**.button.getHeight());

**this**.sliderPanel.revalidate();

**this**.coursePanel.revalidate();

**this**.repaint();

**this**.setVisible(**true**);

}

}

**public** **void** paint(Graphics g) {

**super**.paint(g);

LayoutConstants.*paintBarChartOutline*(g, sliders.size());

**for** (**int** i = 0; i < sliders.size(); i++) {

JSlider record = sliders.elementAt(i);

g.setColor(LayoutConstants.***courseColours***[i]);

g.fillRect(

LayoutConstants.***xOffset*** + (i + 1)

\* LayoutConstants.***barSpacing*** + i

\* LayoutConstants.***barWidth***, LayoutConstants.***yOffset***

+ LayoutConstants.***graphHeight***

- LayoutConstants.***barHeight*** + 2

\* (LayoutConstants.***maxValue*** - record.getValue()),

LayoutConstants.***barWidth***, 2 \* record.getValue());

g.setColor(Color.***red***);

g.drawString(record.getName(),

LayoutConstants.***xOffset*** + (i + 1)

\* LayoutConstants.***barSpacing*** + i

\* LayoutConstants.***barWidth***, LayoutConstants.***yOffset***

+ LayoutConstants.***graphHeight*** + 20);

}

paintPieChart(g);

}

**public** **void** paintPieChart(Graphics graphics){

**int** radius = 100;

**double** startAngle = 0.0;

**for** (**int** i = 0; i < sliders.size(); i++) {

JSlider record = sliders.elementAt(i);

Integer value = record.getValue();

**double** ratio = ( (**double**) value / getTotalStudents()) \* 360.0;

graphics.setColor(LayoutConstants.***courseColours***[i]);

graphics.fillArc(LayoutConstants.***xOffset***, LayoutConstants.***yOffset*** + 300, 2 \* radius, 2 \* radius, (**int**) startAngle, (**int**) ratio);

startAngle += ratio;

}

}

**public** Integer getTotalStudents(){

Integer total = 0;

**for**( JSlider slider : **this**.sliders){

total += slider.getValue();

}

**return** total;

}

/\*\*

\* Manages the creation of a new course. Called when "New Course" button is pressed.

\*

\* **@param** arg0

\* not used

\*/

**public** **void** actionPerformed(ActionEvent arg0) {

String input = JOptionPane

.*showInputDialog*("Please enter the new course name:");

**if** (input != **null**)

**this**.addCourse(**new** CourseRecord(input, 50));

}

/\*\*

\* Handles the changing of the marks for a course (changing of a JSlider)

\*

\* **@param** arg0

\* not used

\*/

**public** **void** stateChanged(ChangeEvent arg0) {

**this**.repaint();

}

/\*\*

\* Sets up an initial set of three courses

\*

\* **@param** args

\* not used

\*/

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

CourseViewer viewer = **new** CourseViewer();

viewer.addCourse(**new** CourseRecord("Physics", 50));

viewer.addCourse(**new** CourseRecord("Chemistry", 50));

viewer.addCourse(**new** CourseRecord("Biology", 50));

}

// Frame contents

**private** JPanel sliderPanel;

**private** JPanel coursePanel;

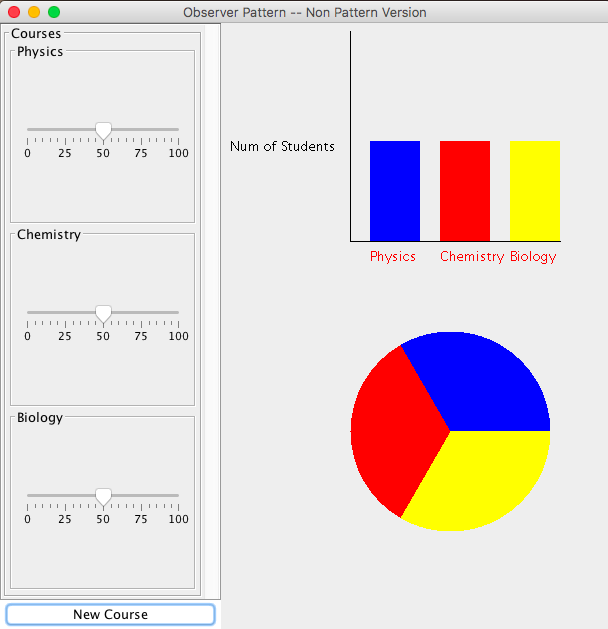
**private** Vector<JSlider> sliders;

**private** JScrollPane scrollPane;

**private** JButton button;

}

* 1. Imágenes



1. Proyecto 2 patrón observer

Este proyecto se tomó a partir de las modificaciones realizadas al proyecto 1 del patrón observer, En este proyecto se utilizo el código en el paquete observer.pattern el cual tiene implementado el patrón observer con un enfoque pull, se creó la clase PieChartObserver la cual implementa la interface Observer la cual esta al pendiente de las modificaciones realizadas en el objeto observable CourseData, además de eso se modificó la clase CourseController para utilizar la clase observer PieChartObserver en lugar de BarChartObserver, las siguientes fueron las clases modificadas para este proyecto.

2.1 Clase CourseController

**package** observer.pattern;

**import** java.awt.Color;

**import** java.awt.GridBagConstraints;

**import** java.awt.GridBagLayout;

**import** java.awt.GridLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.util.Vector;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JScrollPane;

**import** javax.swing.JSlider;

**import** javax.swing.border.TitledBorder;

**import** javax.swing.event.ChangeEvent;

**import** javax.swing.event.ChangeListener;

**import** observer.CourseRecord;

/\*\*

\* Presents a given view of a set of courses and their marks. Uses the Observer

\* pattern to be notified when a new course has been added.

\*/

@SuppressWarnings("serial")

**public** **class** CourseController **extends** JPanel **implements** Observer, ChangeListener, ActionListener {

/\*\*

\* Constructs a CourseController object

\*

\* **@param** courses

\* a set of courses and their marks

\*/

**public** CourseController(CourseData courses) {

**this**.courseData = courses;

**this**.sliders = **new** Vector<JSlider>();

**this**.setLayout(**new** GridBagLayout());

**this**.setBackground(Color.***white***);

coursePanel = **new** JPanel();

coursePanel.setBorder(**new** TitledBorder("Courses"));

coursePanel.setLayout(**new** GridLayout(0, 1));

GridBagConstraints constraints = **new** GridBagConstraints();

constraints.fill = GridBagConstraints.***BOTH***;

courses.attach(**this**);

Vector<CourseRecord> state = courses.getUpdate();

**for** (**int** i = 0; i < state.size(); i++)

**this**.addCourse(state.elementAt(i));

JScrollPane scrollPane = **new** JScrollPane(coursePanel,

JScrollPane.***VERTICAL\_SCROLLBAR\_ALWAYS***,

JScrollPane.***HORIZONTAL\_SCROLLBAR\_AS\_NEEDED***);

JButton button = **new** JButton("New Courses");

button.addActionListener(**this**);

constraints.weightx = 0.5;

constraints.weighty = 1.0;

constraints.gridx = 0;

constraints.gridy = 0;

**this**.add(scrollPane, constraints);

constraints.weightx = 0.5;

constraints.weighty = 0;

constraints.gridy = 1;

**this**.add(button, constraints);

}

/\*\*

\* Add a new course

\*

\* **@param** record

\* the new course record to be added

\*/

**public** **void** addCourse(CourseRecord record) {

JSlider slider = **new** JSlider();

slider.setBackground(Color.***white***);

slider.setName(record.getName());

slider.setValue(record.getNumOfStudents());

slider.setMinimum(0);

slider.setMaximum(100);

slider.setMajorTickSpacing(25);

slider.setMinorTickSpacing(5);

slider.setPaintTicks(**true**);

slider.setPaintLabels(**true**);

slider.setBorder(**new** TitledBorder(record.getName()));

slider.addChangeListener(**this**);

coursePanel.add(slider);

coursePanel.revalidate();

sliders.addElement(slider);

}

/\*\*

\* Informs this CourseController that a new course has been added

\*

\* **@param** o

\* the CourseData subject that has changed

\*/

**public** **void** update(Observable o) {

CourseData courses = (CourseData) o;

Vector<CourseRecord> newCourses = courses.getUpdate();

**for** (**int** i = sliders.size(); i < newCourses.size(); i++) {

**this**.addCourse((CourseRecord) newCourses.elementAt(i));

}

}

/\*\*

\* Manages the creation of a new course. Called when the "New Course" button is pressed.

\*

\* **@param** arg0

\* not used

\*/

**public** **void** actionPerformed(ActionEvent arg0) {

String input = JOptionPane.*showInputDialog*("Please enter new course name:");

**if** (input != **null**){

courseData.addCourseRecord(**new** CourseRecord(input, 50));

// leave it up notify/update mechanism to invoke this.addCourse

}

}

/\*\*

\* Handles the changing of the marks for a course (changing of a JSlider)

\*

\* **@param** arg0

\* the JSlider that has changed

\*/

**public** **void** stateChanged(ChangeEvent arg0) {

JSlider slider = (JSlider) arg0.getSource();

courseData.changeCourseRecord(slider.getName(), slider.getValue());

}

/\*\*

\* Sets up an initial set of three courses

\*

\* **@param** args

\* not used

\*/

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

CourseData data = **new** CourseData();

data.addCourseRecord(**new** CourseRecord("Physics", 50));

data.addCourseRecord(**new** CourseRecord("Chemistry", 50));

data.addCourseRecord(**new** CourseRecord("Biology", 50));

CourseController controller = **new** CourseController(data);

PieChartObserver bar = **new** PieChartObserver(data);

JScrollPane scrollPane = **new** JScrollPane(bar,

JScrollPane.***VERTICAL\_SCROLLBAR\_AS\_NEEDED***,

JScrollPane.***HORIZONTAL\_SCROLLBAR\_ALWAYS***);

JFrame frame = **new** JFrame("Observer Pattern");

frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

frame.getContentPane().setLayout(**new** GridBagLayout());

frame.setResizable(**true**);

GridBagConstraints constraints = **new** GridBagConstraints();

constraints.fill = GridBagConstraints.***BOTH***;

constraints.weightx = 0.25;

constraints.weighty = 1.0;

constraints.gridx = 0;

constraints.gridy = 0;

frame.getContentPane().add(controller, constraints);

constraints.weightx = 0.5;

constraints.weighty = 1.0;

constraints.gridx = 1;

constraints.gridy = 0;

frame.getContentPane().add(scrollPane, constraints);

frame.pack();

frame.setVisible(**true**);

}

**private** CourseData courseData;

**private** Vector<JSlider> sliders;

**private** JPanel coursePanel;

}

* 1. Clase PieChartObserver

**package** observer.pattern;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.Graphics;

**import** java.util.Vector;

**import** javax.swing.JPanel;

**import** javax.swing.JSlider;

**import** observer.CourseRecord;

**import** observer.LayoutConstants;

/\*\*

\* This class represents a bar chart view of a vector of data. Uses the Observer

\* pattern.

\*/

@SuppressWarnings("serial")

**public** **class** PieChartObserver **extends** JPanel **implements** Observer {

/\*\*

\* Creates a BarChartObserver object

\*

\* **@param** data

\* a CourseData object to observe

\*/

**public** PieChartObserver(CourseData data) {

data.attach(**this**);

**this**.courseData = data.getUpdate();

**this**.setPreferredSize(**new** Dimension(2 \* LayoutConstants.***xOffset***

+ (LayoutConstants.***barSpacing*** + LayoutConstants.***barWidth***)

\* **this**.courseData.size(), LayoutConstants.***graphHeight*** + 100

\* LayoutConstants.***yOffset***));

**this**.setBackground(Color.***white***);

}

/\*\*

\* Paint method

\*

\* **@param** g

\* a Graphics object on which to paint

\*/

**public** **void** paint(Graphics g) {

**super**.paint(g);

paintPieChart(g);

}

**public** **void** paintPieChart(Graphics graphics){

**int** radius = 100;

**double** startAngle = 0.0;

**for**(**int** i = 0; i< **this**.courseData.size(); i++){

CourseRecord curse = **this**.courseData.get(i);

Integer value = curse.getNumOfStudents();

**double** ratio = ( (**double**) value / getTotalStudents()) \* 360.0;

graphics.setColor(LayoutConstants.***courseColours***[i]);

graphics.fillArc(LayoutConstants.***xOffset***, LayoutConstants.***yOffset*** + 300, 2 \* radius, 2 \* radius, (**int**) startAngle, (**int**) ratio);

startAngle += ratio;

}

}

**public** Integer getTotalStudents(){

Integer total = 0;

**for**( CourseRecord curseData : **this**.courseData){

total += curseData.getNumOfStudents();

}

**return** total;

}

/\*\*

\* Informs this observer that the observed CourseData object has changed

\*

\* **@param** o

\* the observed CourseData object that has changed

\*/

**public** **void** update(Observable o) {

CourseData data = (CourseData) o;

**this**.courseData = data.getUpdate();

**this**.setPreferredSize(**new** Dimension(2 \* LayoutConstants.***xOffset***

+ (LayoutConstants.***barSpacing*** + LayoutConstants.***barWidth***)

\* **this**.courseData.size(), LayoutConstants.***graphHeight*** + 2

\* LayoutConstants.***yOffset***));

**this**.revalidate();

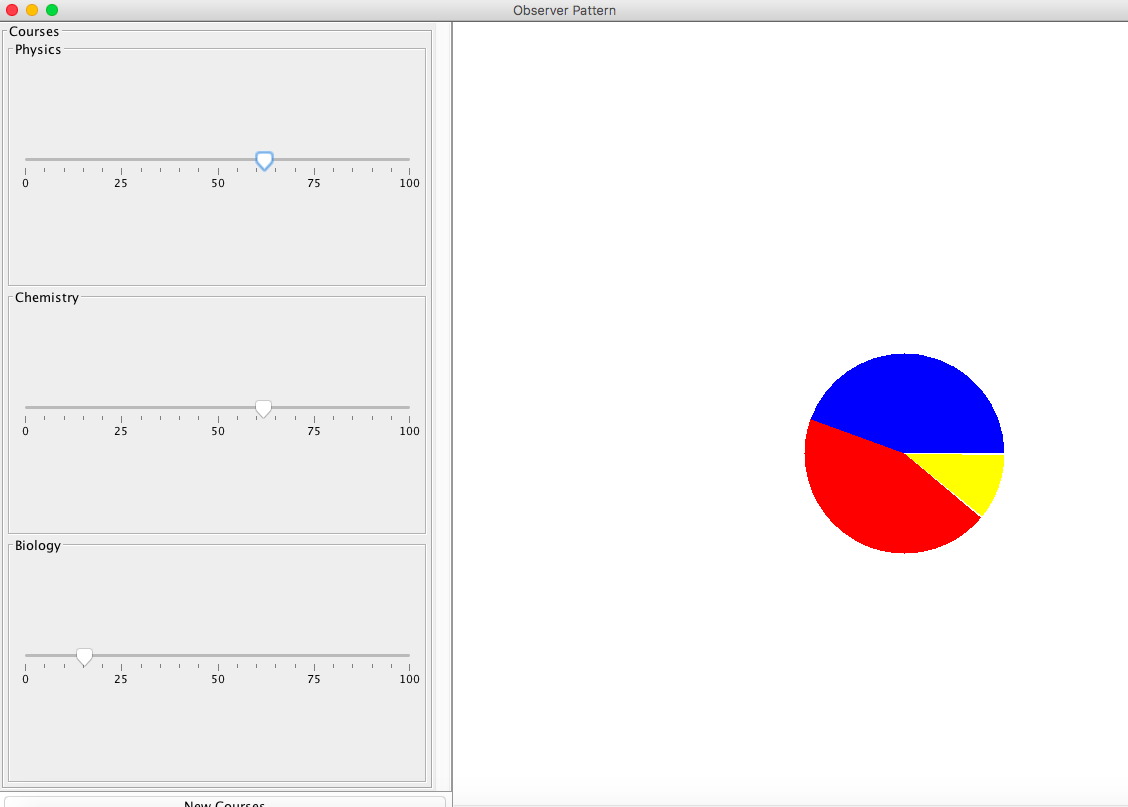
**this**.repaint();

}

**private** Vector<CourseRecord> courseData;

}

* 1. Imágenes



1. Proyecto 3 patrón observer

En este proyecto se modificó para que utilizara el patrón de diseño observer de manera pull, se implementara de forma push, se modificó la interface Observer para que las clases concretas que extienden de Observer implementaran el método update que recibiera una Vector de CourseRecord, las modificaciones que se hicieron son para que en lugar de que el observer al ser notificado de nuevos cambios en CourseData hiciera “Pull” de los datos, se le envié los datos, en este caso el Vector de CourseRecord, el siguiente es el código de las clases modificadas.

3.1 Clase CourseController

**package** observer.pattern;

**import** java.awt.Color;

**import** java.awt.GridBagConstraints;

**import** java.awt.GridBagLayout;

**import** java.awt.GridLayout;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.util.Vector;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**import** javax.swing.JScrollPane;

**import** javax.swing.JSlider;

**import** javax.swing.border.TitledBorder;

**import** javax.swing.event.ChangeEvent;

**import** javax.swing.event.ChangeListener;

**import** observer.CourseRecord;

/\*\*

\* Presents a given view of a set of courses and their marks. Uses the Observer

\* pattern to be notified when a new course has been added.

\*/

@SuppressWarnings("serial")

**public** **class** CourseController **extends** JPanel **implements** Observer, ChangeListener, ActionListener {

/\*\*

\* Constructs a CourseController object

\*

\* **@param** courses

\* a set of courses and their marks

\*/

**public** CourseController(CourseData courses) {

**this**.courseData = courses;

**this**.sliders = **new** Vector<JSlider>();

**this**.setLayout(**new** GridBagLayout());

**this**.setBackground(Color.***white***);

coursePanel = **new** JPanel();

coursePanel.setBorder(**new** TitledBorder("Courses"));

coursePanel.setLayout(**new** GridLayout(0, 1));

GridBagConstraints constraints = **new** GridBagConstraints();

constraints.fill = GridBagConstraints.***BOTH***;

courses.attach(**this**);

Vector<CourseRecord> state = courses.getUpdate();

**for** (**int** i = 0; i < state.size(); i++)

**this**.addCourse(state.elementAt(i));

JScrollPane scrollPane = **new** JScrollPane(coursePanel,

JScrollPane.***VERTICAL\_SCROLLBAR\_ALWAYS***,

JScrollPane.***HORIZONTAL\_SCROLLBAR\_AS\_NEEDED***);

JButton button = **new** JButton("New Courses");

button.addActionListener(**this**);

constraints.weightx = 0.5;

constraints.weighty = 1.0;

constraints.gridx = 0;

constraints.gridy = 0;

**this**.add(scrollPane, constraints);

constraints.weightx = 0.5;

constraints.weighty = 0;

constraints.gridy = 1;

**this**.add(button, constraints);

}

/\*\*

\* Add a new course

\*

\* **@param** record

\* the new course record to be added

\*/

**public** **void** addCourse(CourseRecord record) {

JSlider slider = **new** JSlider();

slider.setBackground(Color.***white***);

slider.setName(record.getName());

slider.setValue(record.getNumOfStudents());

slider.setMinimum(0);

slider.setMaximum(100);

slider.setMajorTickSpacing(25);

slider.setMinorTickSpacing(5);

slider.setPaintTicks(**true**);

slider.setPaintLabels(**true**);

slider.setBorder(**new** TitledBorder(record.getName()));

slider.addChangeListener(**this**);

coursePanel.add(slider);

coursePanel.revalidate();

sliders.addElement(slider);

}

/\*\*

\* Informs this CourseController that a new course has been added

\*

\* **@param** o

\* the CourseData subject that has changed

\*/

**public** **void** update(Observable o) {

CourseData courses = (CourseData) o;

Vector<CourseRecord> newCourses = courses.getUpdate();

**for** (**int** i = sliders.size(); i < newCourses.size(); i++) {

**this**.addCourse((CourseRecord) newCourses.elementAt(i));

}

}

/\*\*

\* Manages the creation of a new course. Called when the "New Course" button is pressed.

\*

\* **@param** arg0

\* not used

\*/

**public** **void** actionPerformed(ActionEvent arg0) {

String input = JOptionPane.*showInputDialog*("Please enter new course name:");

**if** (input != **null**){

courseData.addCourseRecord(**new** CourseRecord(input, 50));

// leave it up notify/update mechanism to invoke this.addCourse

}

}

/\*\*

\* Handles the changing of the marks for a course (changing of a JSlider)

\*

\* **@param** arg0

\* the JSlider that has changed

\*/

**public** **void** stateChanged(ChangeEvent arg0) {

JSlider slider = (JSlider) arg0.getSource();

courseData.changeCourseRecord(slider.getName(), slider.getValue());

}

/\*\*

\* Sets up an initial set of three courses

\*

\* **@param** args

\* not used

\*/

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

CourseData data = **new** CourseData();

data.addCourseRecord(**new** CourseRecord("Physics", 50));

data.addCourseRecord(**new** CourseRecord("Chemistry", 50));

data.addCourseRecord(**new** CourseRecord("Biology", 50));

CourseController controller = **new** CourseController(data);

PieChartObserver bar = **new** PieChartObserver(data);

JScrollPane scrollPane = **new** JScrollPane(bar,

JScrollPane.***VERTICAL\_SCROLLBAR\_AS\_NEEDED***,

JScrollPane.***HORIZONTAL\_SCROLLBAR\_ALWAYS***);

JFrame frame = **new** JFrame("Observer Pattern");

frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

frame.getContentPane().setLayout(**new** GridBagLayout());

frame.setResizable(**true**);

GridBagConstraints constraints = **new** GridBagConstraints();

constraints.fill = GridBagConstraints.***BOTH***;

constraints.weightx = 0.25;

constraints.weighty = 1.0;

constraints.gridx = 0;

constraints.gridy = 0;

frame.getContentPane().add(controller, constraints);

constraints.weightx = 0.5;

constraints.weighty = 1.0;

constraints.gridx = 1;

constraints.gridy = 0;

frame.getContentPane().add(scrollPane, constraints);

frame.pack();

frame.setVisible(**true**);

}

**private** CourseData courseData;

**private** Vector<JSlider> sliders;

**private** JPanel coursePanel;

**public** **void** update(Vector<CourseRecord> courses) {

**for** (**int** i = sliders.size(); i < courses.size(); i++) {

**this**.addCourse(courses.elementAt(i));

}

}

}

3.2 Clase CourseData

**package** observer.pattern;

**import** java.util.Vector;

**import** javax.swing.JOptionPane;

**import** observer.CourseRecord;

/\*\*

\* Represents a vector of CourseRecords.

\*/

**public** **class** CourseData **extends** Observable {

/\*\*

\* Constructs a CourseData object

\*/

**public** CourseData() {

**this**.courseData = **new** Vector<CourseRecord>();

}

/\*\*

\* Add a new CourseRecord object

\*

\* **@param** courseRecord

\* the CourseRecord to be added

\*/

**public** **void** addCourseRecord(CourseRecord courseRecord) {

**boolean** alreadyExists = **false**;

**for** (**int** i = 0; i < courseData.size(); i++) {

CourseRecord record = courseData.elementAt(i);

**if** (record.getName().equals(courseRecord.getName())) {

alreadyExists = **true**;

JOptionPane

.*showMessageDialog*(

**null**,

"Warning: Attempt to add new course with an already existing name",

"alert", JOptionPane.***ERROR\_MESSAGE***);

i = courseData.size(); // exit the loop

}

}

**if** (!alreadyExists)

**this**.courseData.addElement(courseRecord);

//this.notifyObservers();

**this**.push(courseData);

}

/\*\*

\* Update an existing CourseRecord object

\*

\* **@param** subjectName

\* the name CourseRecord to be updated

\* **@param** numOfStudents

\* the new number of students for this course

\*/

**public** **void** changeCourseRecord(String subjectName, **int** numOfStudents) {

**for** (**int** i = 0; i < courseData.size(); i++) {

CourseRecord record = courseData.elementAt(i);

**if** (record.getName().equals(subjectName)) {

record.setNumOfStudents(numOfStudents);

i = courseData.size();

}

}

//this.notifyObservers();

**this**.push(courseData);

}

/\*\*

\* Return a copy of the vector of course data. Used by Observers to pull

\* data.

\*

\* **@return** vector of course data

\*/

**public** Vector<CourseRecord> getUpdate() {

**return** (Vector<CourseRecord>) courseData.clone();

}

**private** Vector<CourseRecord> courseData;

}

3.3 Clase Observable

**package** observer.pattern;

**import** java.util.Vector;

**import** observer.CourseRecord;

/\*\*

\* An abstract class for all Observable subjects

\*/

**public** **abstract** **class** Observable {

/\*\*

\* Constructs an Observable object

\*/

**public** Observable() {

**this**.observers = **new** Vector<Observer>();

}

/\*\*

\* Attach to this Subject

\*

\* **@param** o

\* the Observer that wishes to attach

\*/

**public** **void** attach(Observer o) {

**this**.observers.addElement(o);

}

/\*\*

\* Detach from this Subject

\*

\* **@param** o

\* the Observer that wishes to detach

\*/

**public** **void** detach(Observer o) {

**for** (**int** i = 0; i < observers.size(); i++) {

**if** (observers.elementAt(i).equals(o))

observers.removeElementAt(i);

}

}

/\*\*

\* Notify all Observers that Subject has changed

\*/

/\*public void notifyObservers() {

for (int i = 0; i < observers.size(); i++) {

Observer observer = observers.elementAt(i);

observer.update(this);

}

}\*/

**public** **void** push(Vector<CourseRecord> courses){

**for** (**int** i = 0; i < observers.size(); i++) {

Observer observer = observers.elementAt(i);

observer.update(courses);

}

}

/\*\*

\* Pull updated data from this Subject

\*

\* **@return** the updated data from the Subject

\*/

**public** **abstract** Object getUpdate();

**protected** Vector<Observer> observers;

}

3.4 Interface Observer

**package** observer.pattern;

**import** java.util.Vector;

**import** observer.CourseRecord;

/\*\*

\* An interface for all Observers

\*/

**public** **interface** Observer {

/\*\*

\* Informs this observer that an observed subject has changed

\*

\* **@param** o

\* the observed subject that has changed

\*/

//public void update(Observable o);

**public** **void** update(Vector<CourseRecord> o);

}

3.5 PieChartObserver

**package** observer.pattern;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.Graphics;

**import** java.util.Vector;

**import** javax.swing.JPanel;

**import** observer.CourseRecord;

**import** observer.LayoutConstants;

/\*\*

\* This class represents a bar chart view of a vector of data. Uses the Observer

\* pattern.

\*/

@SuppressWarnings("serial")

**public** **class** PieChartObserver **extends** JPanel **implements** Observer {

/\*\*

\* Creates a BarChartObserver object

\*

\* **@param** data

\* a CourseData object to observe

\*/

**public** PieChartObserver(CourseData data) {

data.attach(**this**);

**this**.courseData = data.getUpdate();

**this**.setPreferredSize(**new** Dimension(2 \* LayoutConstants.***xOffset***

+ (LayoutConstants.***barSpacing*** + LayoutConstants.***barWidth***)

\* **this**.courseData.size(), LayoutConstants.***graphHeight*** + 100

\* LayoutConstants.***yOffset***));

**this**.setBackground(Color.***white***);

}

/\*\*

\* Paint method

\*

\* **@param** g

\* a Graphics object on which to paint

\*/

**public** **void** paint(Graphics g) {

**super**.paint(g);

paintPieChart(g);

}

**public** **void** paintPieChart(Graphics graphics){

**int** radius = 100;

**double** startAngle = 0.0;

**for**(**int** i = 0; i< **this**.courseData.size(); i++){

CourseRecord curse = **this**.courseData.get(i);

Integer value = curse.getNumOfStudents();

**double** ratio = ( (**double**) value / getTotalStudents()) \* 360.0;

graphics.setColor(LayoutConstants.***courseColours***[i]);

graphics.fillArc(LayoutConstants.***xOffset***, LayoutConstants.***yOffset*** + 300, 2 \* radius, 2 \* radius, (**int**) startAngle, (**int**) ratio);

startAngle += ratio;

}

}

**public** Integer getTotalStudents(){

Integer total = 0;

**for**( CourseRecord curseData : **this**.courseData){

total += curseData.getNumOfStudents();

}

**return** total;

}

/\*\*

\* Informs this observer that the observed CourseData object has changed

\*

\* **@param** o

\* the observed CourseData object that has changed

\*/

**public** **void** update(Observable o) {

CourseData data = (CourseData) o;

**this**.courseData = data.getUpdate();

config();

**this**.revalidate();

**this**.repaint();

}

**private** Vector<CourseRecord> courseData;

**public** **void** config(){

**this**.setPreferredSize(**new** Dimension(2 \* LayoutConstants.***xOffset***

+ (LayoutConstants.***barSpacing*** + LayoutConstants.***barWidth***)

\* **this**.courseData.size(), LayoutConstants.***graphHeight*** + 2

\* LayoutConstants.***yOffset***));

}

@Override

**public** **void** update(Vector<CourseRecord> courseRecords) {

**this**.courseData = courseRecords;

config();

**this**.revalidate();

**this**.repaint();

}

}

3.6 Imágenes

