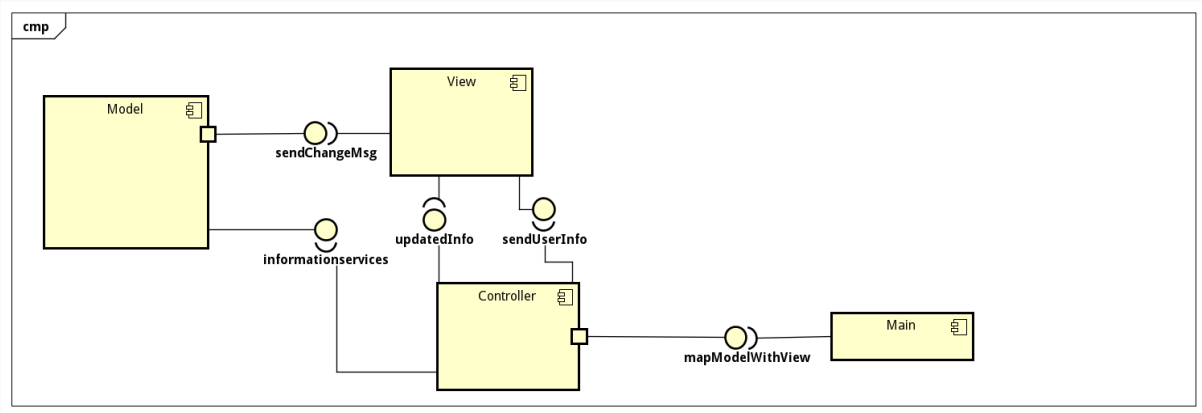
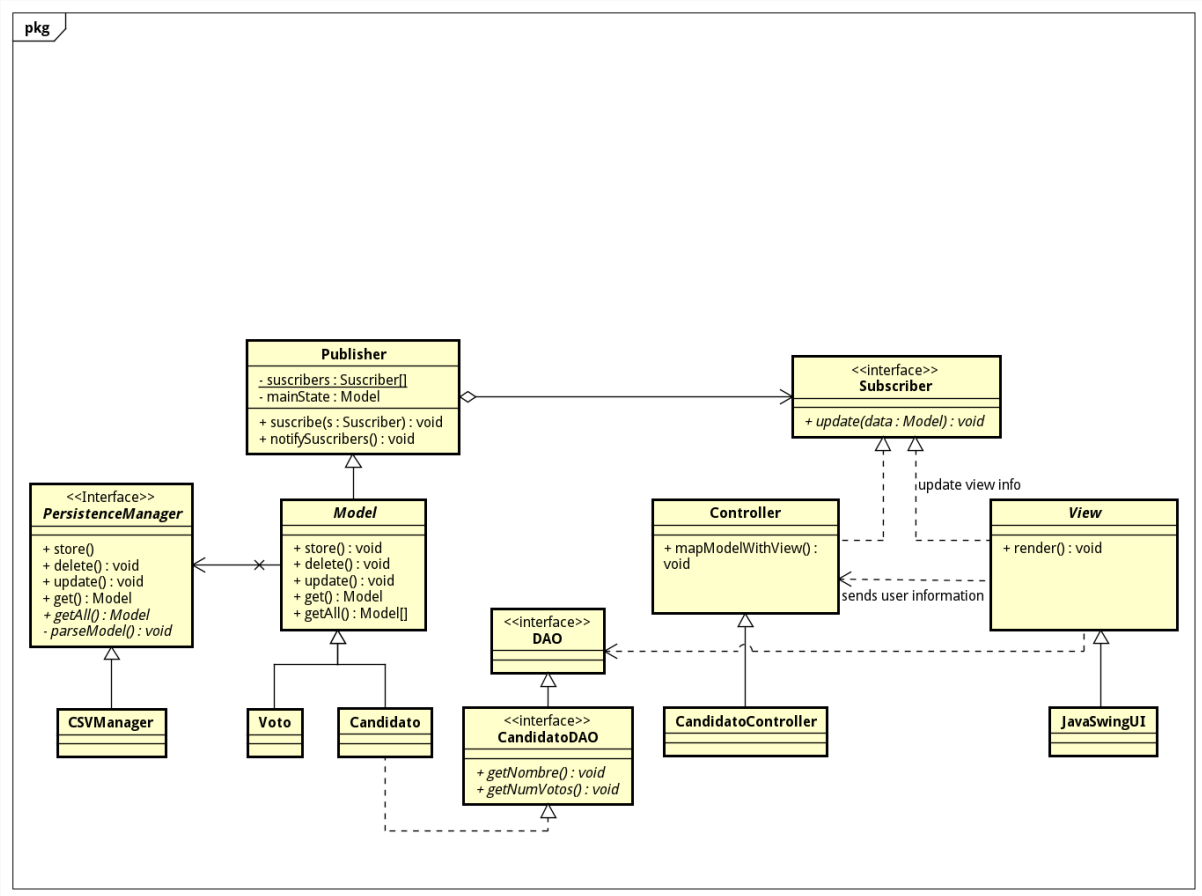
U2_T5 - Primera implementación de MVC

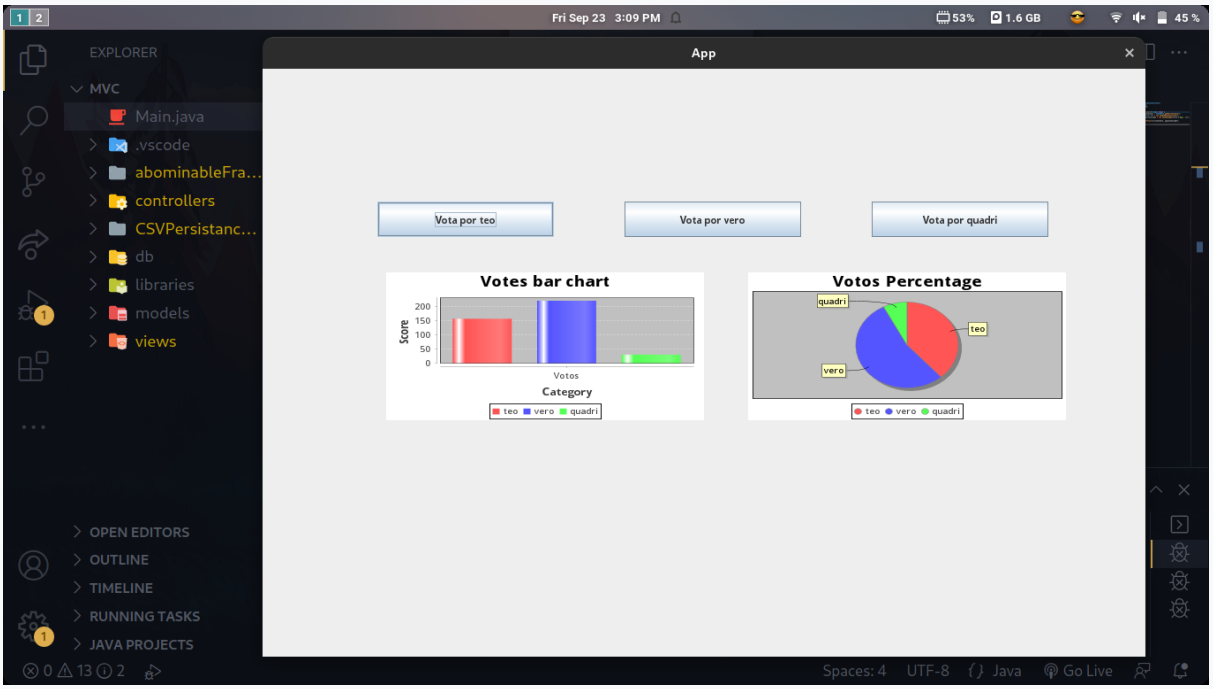
Jorge Teodoro Dawn Rodriguez
Arquitectura de Software
Prof. Juan Garcilazo

Diagrama de componentes y clases



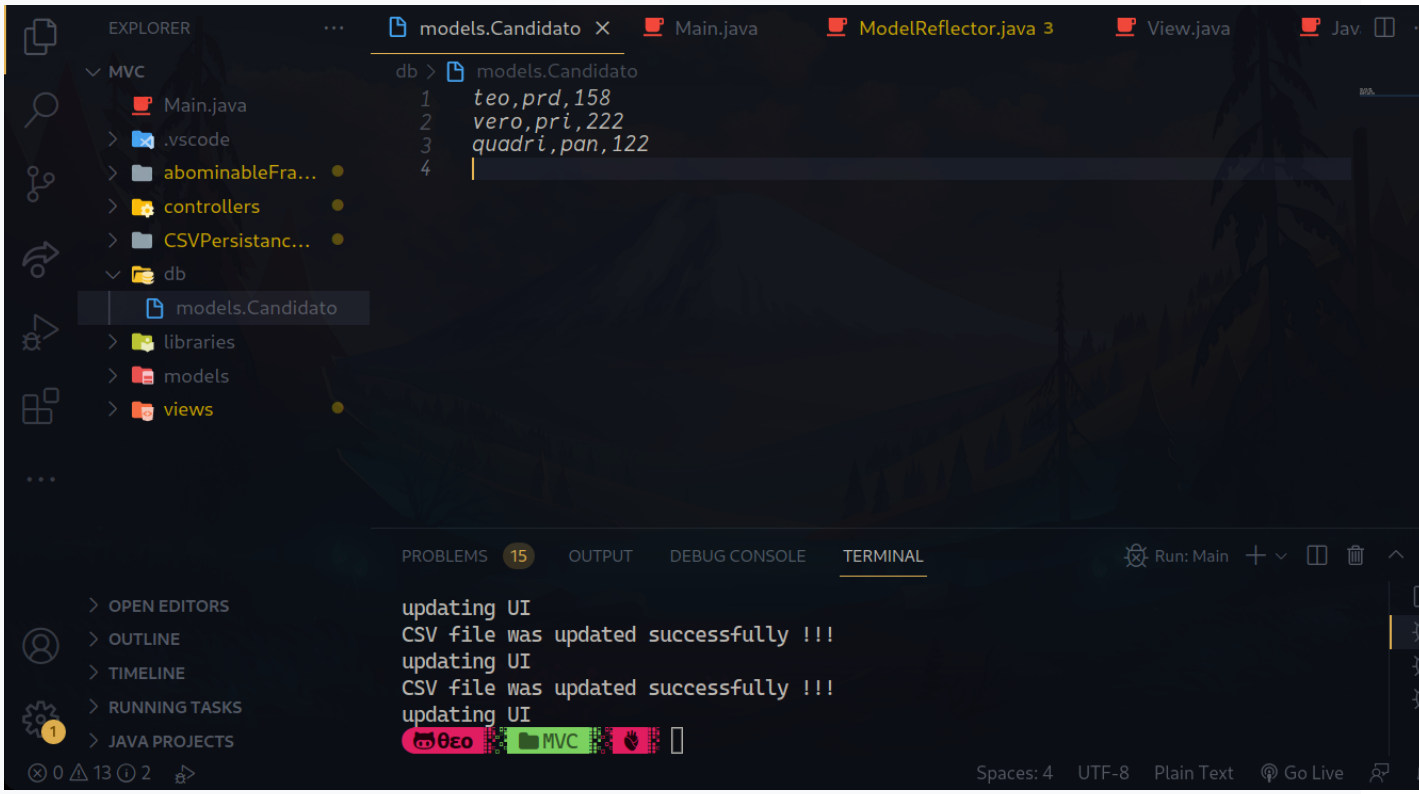


Capturas de pantalla de la ejecución.



Cuando realizamos muchos votos al candidato quadri, las gráficas se actualizan.

El framework realiza la actualización a partir de la información que la view manda al controlador. La persistencia se maneja con CSVs.



Código fuente

```java
import controllers.CandidatoController;
import models.Candidato;
import views.JavaSwingUI;

public class Main {
    public static void main(String[] args) {
        Candidato nullCandidato = Candidato.getNullInstance();
        CandidatoController ctr = new CandidatoController();
        JavaSwingUI appJavaSwingUI = new JavaSwingUI("App", ctr);

        ctr.mapModelWithView(nullCandidato, appJavaSwingUI);

    }
}
```

Código del framework abominable MVC

```java
package abominableFramework;

import java.util.ArrayList;

public class Publisher {
```

```
    private static ArrayList<Subscriber> suscribers = new ArrayList<>();

    public void suscribe(Subscriber s) {
        this.suscribers.add(s);
    }

    public void notifySuscribers() {
        for (Subscriber subscriber : suscribers) {
            subscriber.update();
        }
    }

}
```

```
package abominableFramework;

public interface Subscriber {

    public abstract void update();

}
```

```
package abominableFramework;

public abstract interface PersistenceManager {

    public void store(Model model);

    public void delete(Model model);

    public void update(Model model);

    public Model get(String id);

    public Model[] getAll(Model model);

    public abstract void parseModel();

}
```

```
package abominableFramework;

import CSVPersistanceLayer.CSVManager;

public abstract class Model extends Publisher {

    private static final PersistenceManager persistenceManager = new
CSVManager();

    public void store() {
        persistenceManager.store(this);
        notifySuscribers();
    }
```

```
    public void delete() {
        persistenceManager.delete(this);
        notifySuscribers();
    }

    public void update() {
        persistenceManager.update(this);
        notifySuscribers();
    }

    public Model get(String id) {
        Model result = persistenceManager.get(id);
        return result;
    }

    public Model[] getAll() {
        Model[] result = persistenceManager.getAll(this);
        return result;
    }

}
```

```
package abominableFramework;

public abstract interface View extends Subscriber {

    public abstract void render();

    /**
     * @see Subscriber#update(Model)
     */
    public abstract void update();

}
```

```
package abominableFramework;

public abstract class Controller implements Subscriber {
    protected Publisher model;
    protected Subscriber view;

    public Controller() {

    }

    public void mapModelWithView(Publisher modelClass, Subscriber view) {
        this.model = modelClass;
        this.view = view;
        modelClass.suscribe(this);
        modelClass.suscribe(view);
    }

    /**
     * @see Subscriber#update(Model)
```

```
     */
    public void update() {

    }

}
```

```
package abominableFramework;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

public class ModelReflector {

    private static String capitalize(String str) {
        if (str == null || str.isEmpty()) {
            return str;
        }

        return str.substring(0, 1).toUpperCase() + str.substring(1);
    }

    private static Method getGetterMethod(Class cls, Field field) {
        Method getter = null;
        try {
            getter = cls.getDeclaredMethod("get" + capitalize(field.getName()));
        } catch (NoSuchMethodException | SecurityException e) {
            e.printStackTrace();
        }
        return getter;
    }

    private static String getModelInstanceAttribute(Method getter, Model order) {
        String result = null;
        try {
            result = String.valueOf(getter.invoke(order));
        } catch (IllegalAccessException | IllegalArgumentException |
InvocationTargetException e) {
            e.printStackTrace();
        }
        return result;
    }

    public static String[] getModelAttributes(Model model) {
        Class<?> cls = model.getClass();
        Field[] fields = cls.getDeclaredFields();
        String[] fieldNames = new String[fields.length];

        Field field;
        for (int i = 0; i < fields.length; i++) {
            field = fields[i];
            field.setAccessible(true);
```

```
            fieldNames[i] = field.getName();
        }

        return fieldNames;
    }

    public static String[] getModelInstanceAttributes(Model model) {
        Class<?> cls = model.getClass();
        Field[] fields = cls.getDeclaredFields();
        String[] attrNames = new String[fields.length];

        Field field;
        for (int i = 0; i < fields.length; i++) {
            field = fields[i];
            field.setAccessible(true);

            Method getter = getGetterMethod(cls, field);
            String attr = getModelInstanceAttribute(getter, model);

            attrNames[i] = attr;
        }

        return attrNames;
    }

    private static Class<Model> getModelTypeClass(String className) throws
ClassNotFoundException {
        Class<Model> ModelExtendedClass;
        ModelExtendedClass = (Class<Model>) Class.forName(className);
        return ModelExtendedClass;
    }

    private static Constructor<Model> getModelTypeConstructor(Class<Model>
objClass)
            throws NoSuchMethodException, SecurityException {
        Constructor<Model> constructor;
        constructor = objClass.getConstructor();
        return constructor;
    }

    private static Model getModelTypeInstance(Constructor<Model> constructor)
            throws InstantiationException, IllegalAccessException,
IllegalArgumentException, InvocationTargetException {
        Object object;
        object = constructor.newInstance();
        return (Model) object;
    }

    public static Model getModelInstance(String className)
            throws Exception {
        Model Model;

        try {
            Class<Model> objClass = getModelTypeClass(className);
```

```
            Constructor<Model> objConstructor =
getModelTypeConstructor(objClass);
            Model = getModelTypeInstance(objConstructor);
        } catch (ClassNotFoundException e) {
            throw e;
        } catch (NoSuchMethodException e) {
            throw e;
        } catch (Exception e) {
            throw e;
        }

        return Model;
    }
}
```

Código del CSVPersistance

```
package CSVPersistanceLayer;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Reader;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.List;

import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVPrinter;
import org.apache.commons.csv.CSVRecord;

import abominableFramework.Model;
import abominableFramework.ModelReflector;
import abominableFramework.PersistenceManager;
import models.Candidato;

public class CSVManager implements PersistenceManager {
    private String root = "db/";

    private boolean fileAlredyCreated(String fileName) {
        File f = new File(fileName);
        return (f.exists() && !f.isDirectory());
    }

    private void createFileForEntity(String className) throws IOException {
        String fileName = root + className;
        File entityFile = new File(fileName);
        entityFile.createNewFile();
```

```
}

private void createFileIfNotCreated(String fileName) {
    String filePath = root + fileName;
    if (!fileAlredyCreated(filePath)) {
        try {
            createFileForEntity(fileName);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

@Override
public void delete(Model model) {
}

@Override
public Model get(String id) {
    return null;
}

// TODO Make real use of reflection and not depend on the Candidato class
@Override
public Model[] getAll(Model model) {
    ArrayList<Candidato> info = new ArrayList<>();
    Reader in;
    Iterable<CSVRecord> records;

    try {
        in = new FileReader(root + model.getClass().getName());
        records = CSVFormat.EXCEL.parse(in);
        for (CSVRecord record : records) {
            String name = record.get(0);
            String partidoPolitico = record.get(1);
            Long numeroVotos = Long.valueOf(record.get(2));
            Candidato candidato = new Candidato(name, partidoPolitico);
            candidato.setNumero_de_votos(numeroVotos);

            info.add(candidato);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    Candidato[] candidatosArr = new Candidato[info.size()];
    candidatosArr = info.toArray(candidatosArr);

    return candidatosArr;
}

@Override
public void parseModel() {
```

```
    }

    @Override
    public void store(Model model) {
        createFileIfNotCreated(model.getClass().getName());
        String filePath = root + model.getClass().getName();

        String[] info = ModelReflector.getModelInstanceAttributes(model);

        try (BufferedWriter writer = Files.newBufferedWriter(
                Paths.get(filePath),
                StandardOpenOption.APPEND,
                StandardOpenOption.CREATE);
                CSVPrinter csvPrinter = new CSVPrinter(writer,
CSVFormat.DEFAULT)) {
            csvPrinter.printRecord(info);
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    private static String[] toArray(CSVRecord rec) {
        String[] arr = new String[rec.size()];
        int i = 0;
        for (String str : rec) {
            arr[i++] = str;
        }
        return arr;
    }

    private static void print(CSVPrinter printer, String[] s) throws Exception {
        for (String val : s) {
            printer.print(val != null ? String.valueOf(val) : "");
        }
        printer.println();

    }

    // TODO
    @Override
    public void update(Model model) {
        File f = new File(root + model.getClass().getName());
        Class cls = model.getClass();

        try (CSVParser parser = new CSVParser(new FileReader(f),
CSVFormat.DEFAULT)) {
            List<CSVRecord> list = new ArrayList();
            try {
                list = parser.getRecords();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
```

```
            String edited = f.getAbsolutePath();

            String[] attributes =
ModelReflector.getModelInstanceAttributes(model);

            f.delete();
            try (CSVPrinter printer = new CSVPrinter(new FileWriter(edited),

CSVFormat.DEFAULT.withRecordSeparator(System.getProperty("line.separator")))) {
                for (CSVRecord record : list) {
                    String[] s = toArray(record);

                    if (s[0].equalsIgnoreCase(attributes[0])) {
                        for (int i = 0; i < s.length; i++) {
                            s[i] = attributes[i];
                        }
                    }

                    print(printer, s);
                }
                parser.close();
                printer.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("CSV file was updated successfully !!!");
    }

}
```

Controllers

```
package controllers;

import abominableFramework.Controller;
import abominableFramework.Publisher;
import abominableFramework.Subscriber;
import models.Candidato;
import models.CandidatoDao;

public class CandidatoController extends Controller {
    public void increaseVotosCanditado(String canditadoName) {
        Candidato candidato = Candidato.getNullInstance();
        candidato = candidato.findCandidatoByName(canditadoName);
        candidato.setNumero_de_votos(candidato.getNumero_de_votos() + 1);
        candidato.update();
    }

    public CandidatoDao[] getAllCandidatos() {
        Candidato candidato = Candidato.getNullInstance();
```

```
        CandidatoDao[] candidatos = (CandidatoDao[]) candidato.getAll();
        return candidatos;
    }
}
```

Models

```
package models;

import abominableFramework.Model;

public class Candidato extends Model implements CandidatoDao {
    private String nombre;
    private String partido_politico;
    private long numero_de_votos;

    public Candidato(String nombre, String partido_politico) {
        this.nombre = nombre;
        this.partido_politico = partido_politico;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getPartido_politico() {
        return partido_politico;
    }

    public void setPartido_politico(String partido_politico) {
        this.partido_politico = partido_politico;
    }

    public long getNumero_de_votos() {
        return numero_de_votos;
    }

    public void setNumero_de_votos(long numero_de_votos) {
        this.numero_de_votos = numero_de_votos;
    }

    public static Candidato getNullInstance() {
        return new Candidato("", "");
    }

    public Candidato findCandidatoByName(String name) {
        Candidato candidato;
        for (Model model : getAll()) {
            candidato = (Candidato) model;
```

```
            if (candidato.getNombre().equals(name)) {
                return candidato;
            }
        }
        return null;
    }

    @Override
    public long getNumVotos() {
        return numero_de_votos;
    }

}
```

DAO

```
package models;

public interface CandidatoDao {
    String getNombre();

    long getNumVotos();
}
```

Views

```
package views;

import javax.swing.JButton;
import javax.swing.JFrame;

import org.jfree.chart.ChartPanel;

import abominableFramework.View;
import controllers.CandidatoController;
import models.CandidatoDao;

public class JavaSwingUI extends JFrame implements View {
    private CandidatoController candidatoController;
    private ChartPanel barChartPanel;
    private ChartPanel pieChartPanel;

    public JavaSwingUI(String title, CandidatoController candidatoCtr) {
        super(title);

        this.candidatoController = candidatoCtr;

        createCharts();
        this.setVisible(true);

    }
```

```
    public void sendNewVoteToCandidate(String candidate) {
        candidatoController.increaseVotosCanditado(candidate);
    }

    public void createCharts() {

        CandidatoDao[] candidatos = candidatoController.getAllCandidatos();
        for (int i = 0; i < candidatos.length; i++) {
            CandidatoDao candidato = candidatos[i];

            JButton b = new JButton("Vota por " + candidato.getNombre());
            b.setBounds(130 + 280 * i, 150, 200, 40);// x axis, y axis, width,
height

            b.addActionListener(e ->
sendNewVoteToCandidate(candidato.getNombre()));

            this.add(b);
        }

        this.setSize(1000, 700);// width and height
        this.setLayout(null);// using no layout managers

        BarChart chart = new BarChart("Votes bar chart", candidatoController);
        barChartPanel = chart.getChartPanel();

        PieChart demo = new PieChart(candidatoController);
        pieChartPanel = demo.getChartPanel();

        barChartPanel.setBounds(140, 230, 360, 167);
        pieChartPanel.setBounds(550, 230, 360, 167);

        this.add(barChartPanel);
        this.add(pieChartPanel);
    }

    @Override
    public void render() {

    }

    @Override
    public void update() {
        System.out.println("updating UI");

        this.remove(barChartPanel);
        this.remove(pieChartPanel);
        this.revalidate();

        createCharts();
        this.revalidate();

        this.repaint();
    }
```

```
}
```

```
package views;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.CategoryDataset;
import org.jfree.data.category.DefaultCategoryDataset;

import controllers.CandidatoController;
import models.CandidatoDao;

public class BarChart {
   private ChartPanel chartPanel;

   private CandidatoController candidatoController;

   public BarChart(String chartTitle, CandidatoController candidatoCtr) {
       this.candidatoController = candidatoCtr;
       JFreeChart barChart = ChartFactory.createBarChart(
               chartTitle,
               "Category",
               "Score",
               createDataset(),
               PlotOrientation.VERTICAL,
               true, true, false);

       chartPanel = new ChartPanel(barChart);
       chartPanel.setPreferredSize(new java.awt.Dimension(560, 367));
   }

   public CategoryDataset createDataset() {

       final String votos = "Votos";

       final DefaultCategoryDataset dataset = new DefaultCategoryDataset();

       CandidatoDao[] candidatos = candidatoController.getAllCandidatos();
       for (CandidatoDao candidato : candidatos) {
           dataset.addValue(candidato.getNumVotos(), candidato.getNombre(),
votos);
       }

       return dataset;
   }

   public ChartPanel getChartPanel() {
       return chartPanel;
   }
}
```

```
package views;
```

```java
import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;
import org.jfree.data.general.PieDataset;

import controllers.CandidatoController;
import models.CandidatoDao;

public class PieChart {
   private ChartPanel chartPanel;
   private CandidatoController candidatoController;

   public PieChart(CandidatoController candidatoCtr) {
       this.candidatoController = candidatoCtr;

       chartPanel = createDemoPanel();
   }

   public ChartPanel getChartPanel() {
       return chartPanel;
   }

   public PieDataset createDataset() {
       DefaultPieDataset dataset = new DefaultPieDataset();

       CandidatoDao[] candidatos = candidatoController.getAllCandidatos();
       for (CandidatoDao candidato : candidatos) {
           dataset.setValue(candidato.getNombre(), new
Double(candidato.getNumVotos()));
       }

       return dataset;
   }

   private JFreeChart createChart(PieDataset dataset) {
       JFreeChart chart = ChartFactory.createPieChart(
               "Votos Percentage", // chart title
               dataset, // data
               true, // include legend
               true,
               false);

       return chart;
   }

   public ChartPanel createDemoPanel() {
       JFreeChart chart = createChart(createDataset());
       return new ChartPanel(chart);
   }
```

```
}
```