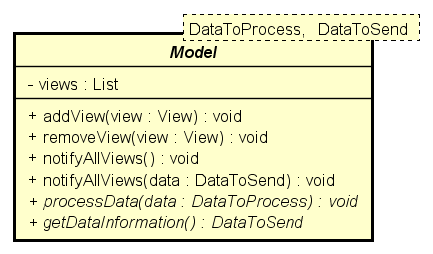
***Patrón arquitectónico MVC***

***“instructivo”***

Clase Model: Esta clase define el comportamiento de un componente modelo, generando subclases de Model podemos crear modelos específicos, conforme se necesite. Un modelo solo se debe crear para procesar datos y entregar datos. La clase model está diseñada sobre el patrón Observer pero específicamente para usarse con las vistas.



Recuerda que toda subclase de modelo debe intentar encapsular todo su contenido realizando únicos servicios que Model posee.

addView: agrega una vista para obsercar el modelo

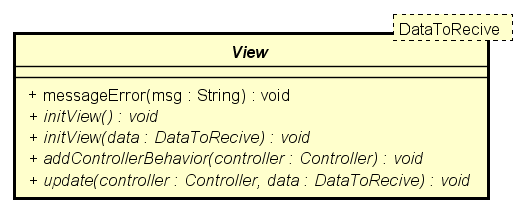
removeView: elimina una vista observadora.

notifyAllViews: Llama al método update de todas las vistas agregadas, la cual puede pasarse un dato para que las vistas interactúen con él.

processData: procesa un dato que se solicite.

getDataInformation: regresa un dato que el modelo tenga o procese.

Clase View: Esta clase define el comportamiento de un componente vista, generando subclases de View podemos generar vistas específicas. Con esta clase busca que cada elemento de la vista pueda agregar un comportamiento que algún controlador tenga.



Update será llamado cuando se requiera actualizar el contenido grafico que tenga la vista, en esta implementación se limita que el dato recibido sea el que contenga la información necesaria para visualizar en toda la vista, si se necesita pasar otro tipo más de un tipo de dato, tendrá que desarrollar otra vista y modelo para procesarlo.

initView deberá encargarse de generar y cargar datos si fuera necesarios en la vista. De igual forma debe hacer que la vista se presente en pantalla.

addControllerBehavior se encarga que a partir de un controlador podemos añadir el comportamiento de una acción de la vista.

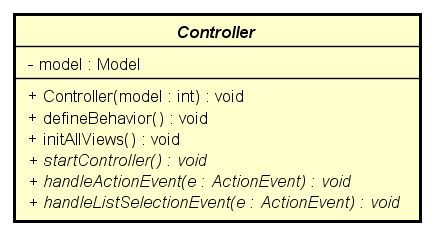
messageError produce un mensaje de error en pantalla, se puede generar mas métodos para presentar diversos tipos de mensajes en pantalla.

Clase Controller: Esta clase define el comportamiento de un componente controlador, generando subclases de Controller podemos generar controladores específicos. Si se requiere generar más respuestas a eventos, se puede agregar fácilmente en la clase controller un método para cada respectivo evento, por ejemplo:

ActionEvent

ListSelectEvent

Si se repite estos eventos en una vista, por ejemplo, tener varios botones, la mejor opción sería definir un switch y diferenciar cada botón con un respectivo comando y de esta manera el switch se comportara dependiendo del comando del evento.



defineBehavior: llama al método addControllerBehavior de todas loas vistas que contenga el modelo.

initAllViews: llama al método initView de todas las vistas que contenga el modelo.

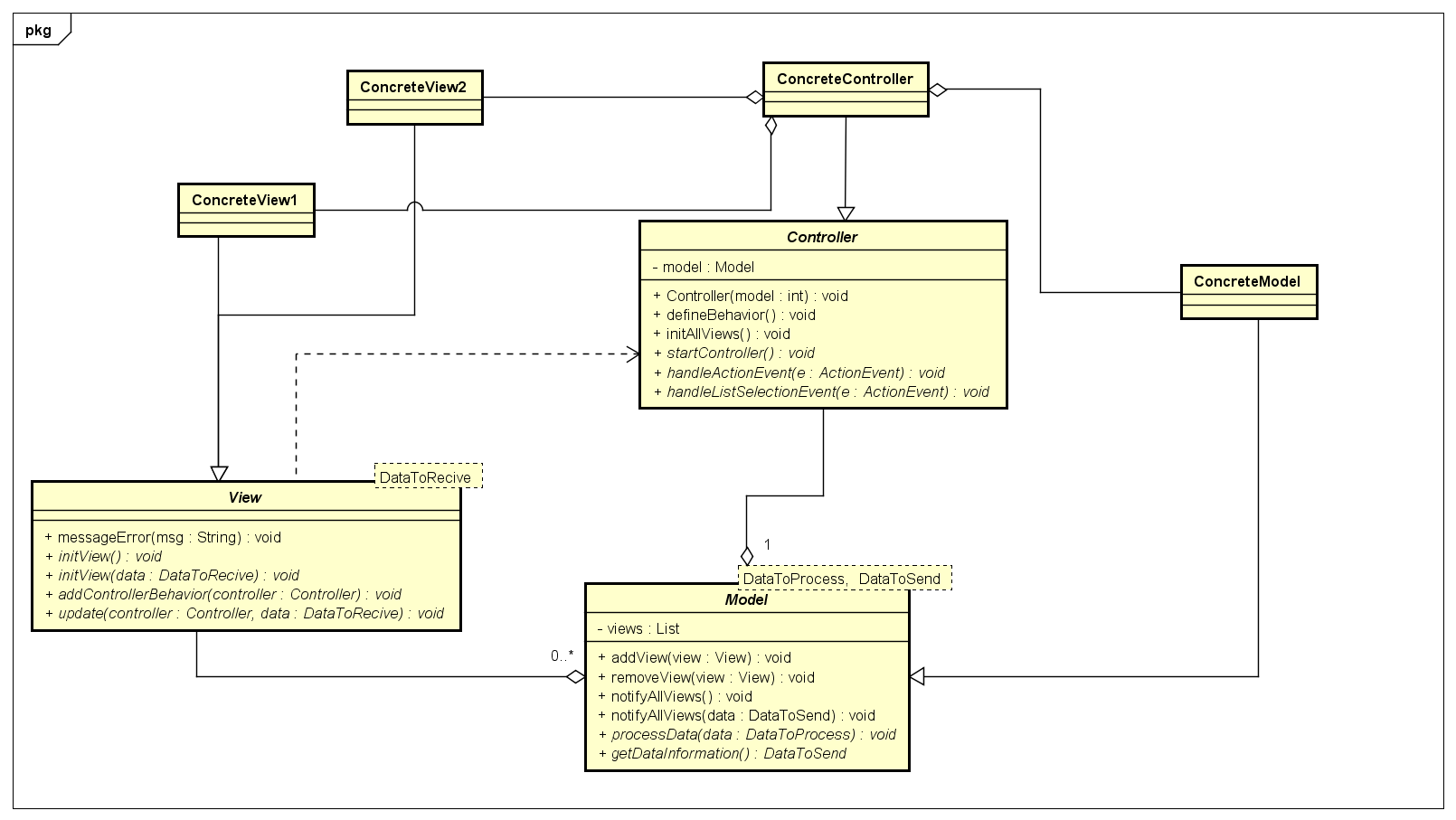
startController: función por definirse que deberá iniciar todo el programa.

handleActionEvent: función por definirse, deberá definir como reaciona el modelo ante una interacción del usuario a las vistas.

Definir un comportamiento general para un controlador con las vistas limita mucho la creatividad que se puede lograr en una interfaz muy llamativa, por lo que esta clase contendrá a estas vistas específicas que se vayan creando para no tener que limitar esta creatividad que los usuarios puedan definir.

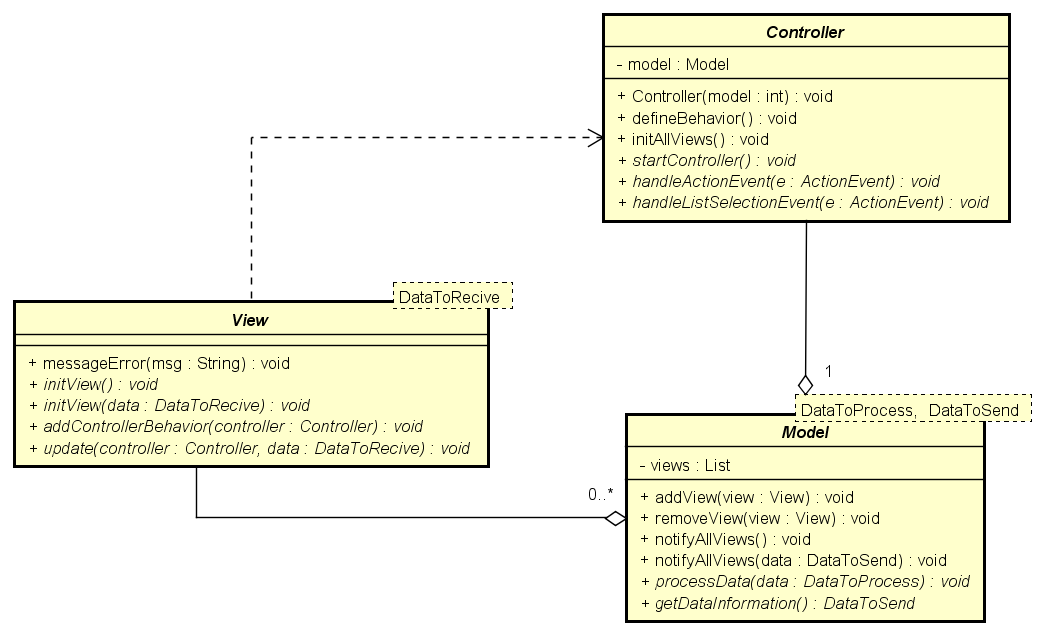
Se busca que el controlador pueda contener vistas y un modelo en particular.

En particular se pretende llegar a una solución como esta:



Las clases concretas herederán las clases abstractas model, view y controller teniendo en cuante el uso de cada clase. De igual manera las clases concretas dependeran del tipo de dato que recive y envia el modelo.

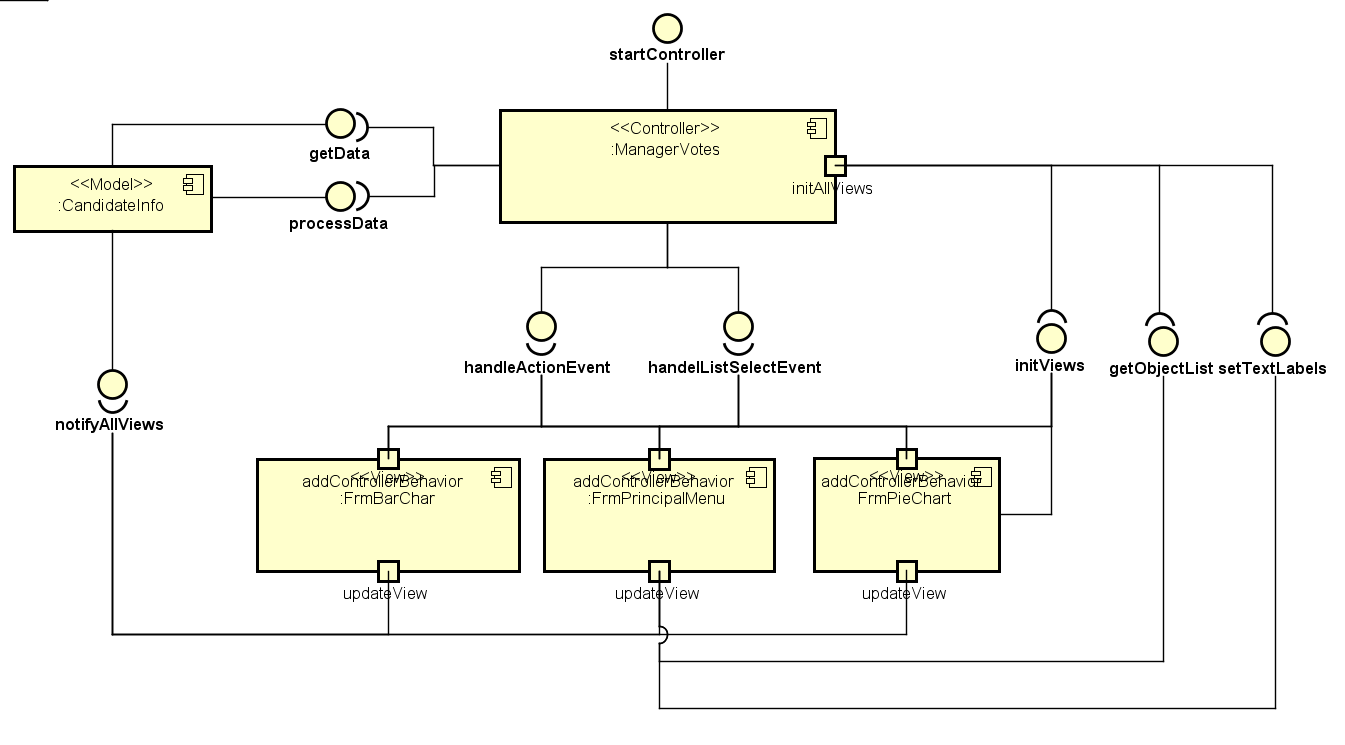
Un diagrama de clases general:



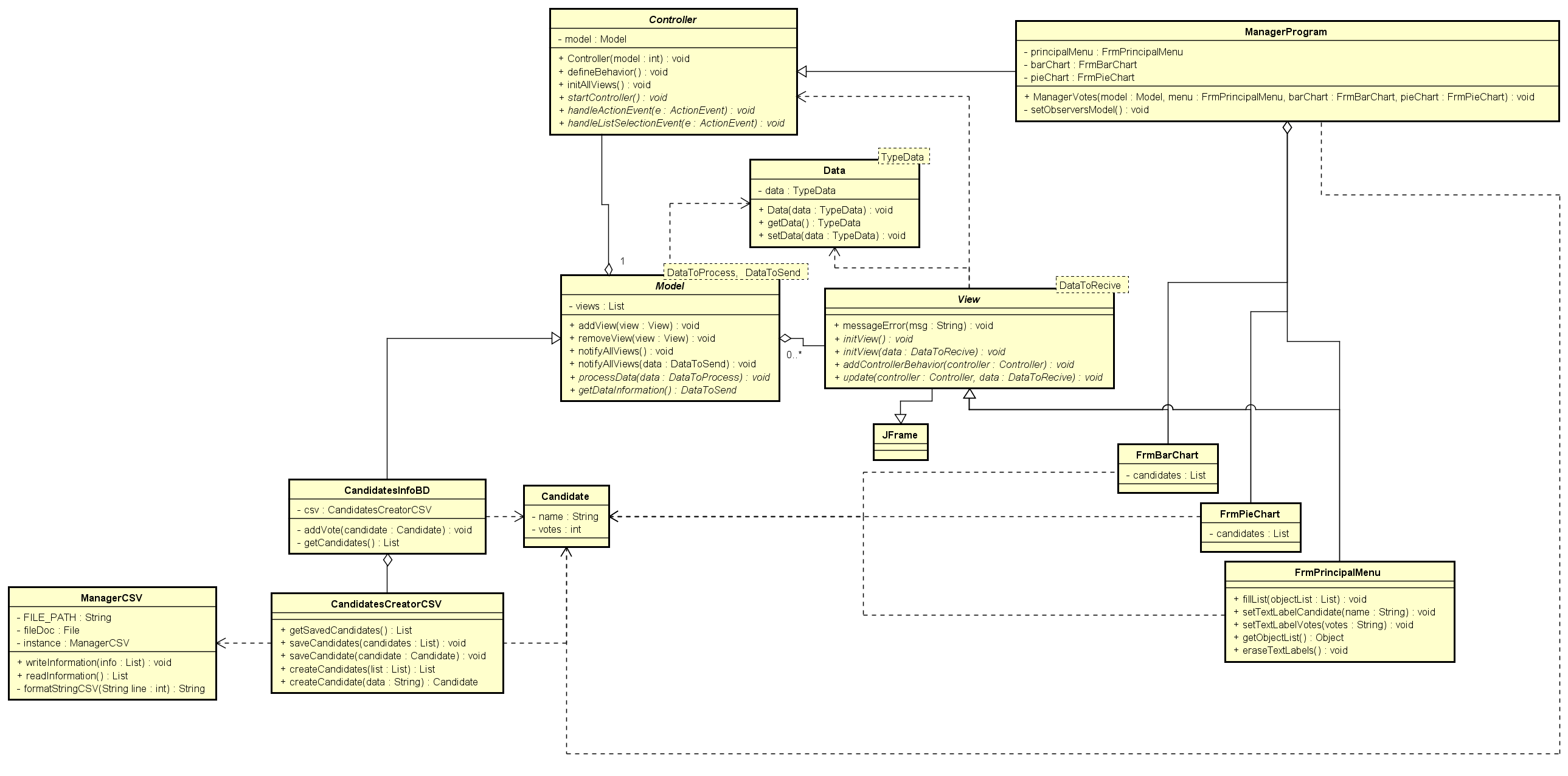
**Ejemplo de implementación**

**“Sistema de votos”**

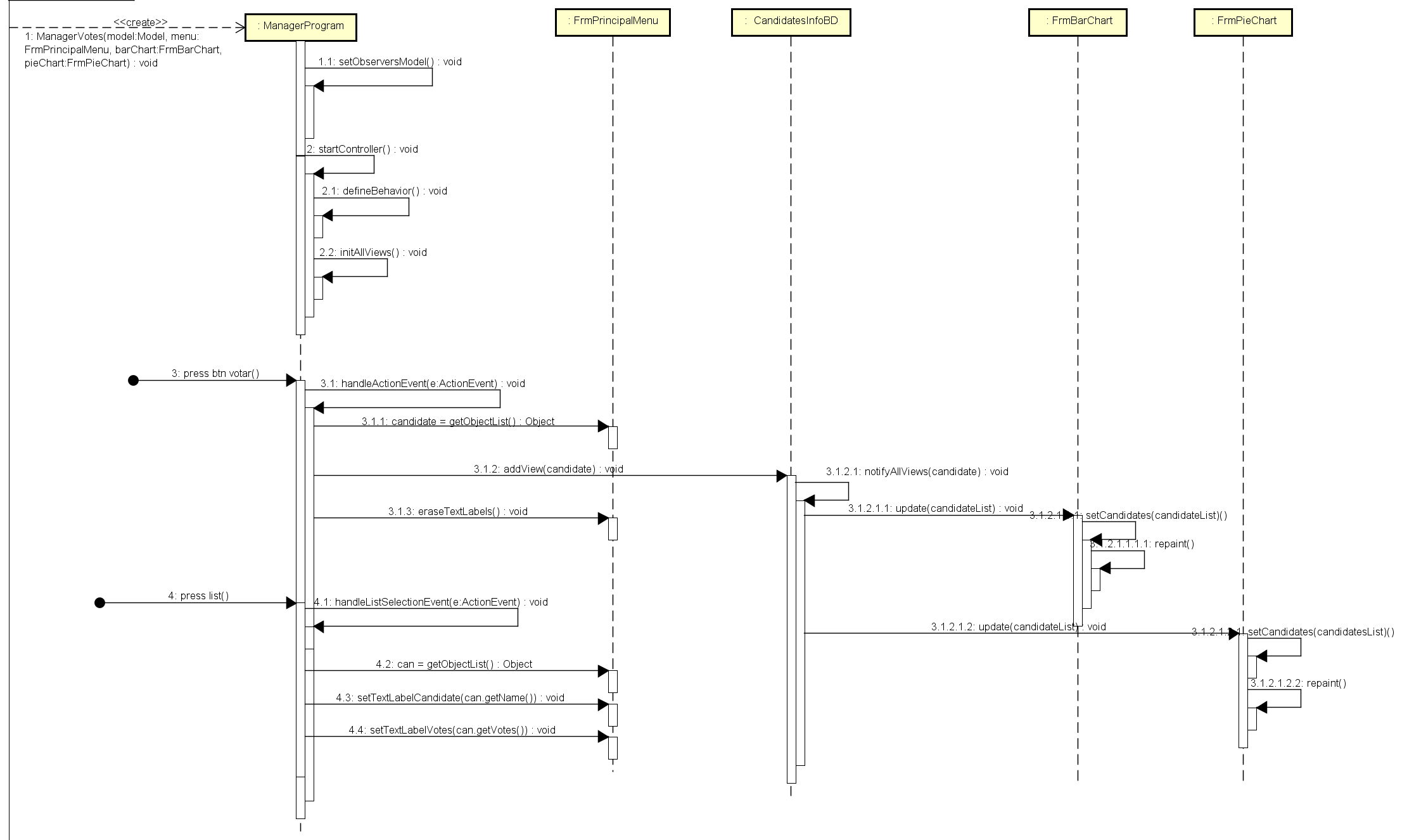
***Diagrama de componentes***

******

***Diagrama de clases***



***Diagrama de secuencia***



***Código fuente***

**Controller**

public abstract class Controller implements EventListener {

private Model model;

public Controller(Model model) {

this.model = model;

}

public void defineBehavior() {

getModel().getAllViews().forEach(new Consumer<View>() {

public void accept(View view) {

view.addControllerBehavior(Controller.this);

}

});

}

public void initAllViews() {

getModel().getAllViews().forEach(new Consumer<View>() {

@Override

public void accept(View view) {

view.initView(Controller.this.getModel().getDataInformation());

}

});

}

public abstract void startController();

public abstract void handleActionEvent(ActionEvent e);

public abstract void handleListSelectionEvent(ListSelectionEvent e);

public Model getModel() {

return model;

}

public void setModel(Model model) {

this.model = model;

}

}

**ManagerProgram(Controlador)**

public class ManagerProgram extends Controller {

private FrmBarChart barChart;

private FrmPieChart pieChart;

private PrincipalMenu principalMenu;

public ManagerProgram(Model model, FrmBarChart bc, FrmPieChart pc, PrincipalMenu menu) {

super(model);

barChart = bc;

pieChart = pc;

principalMenu = menu;

setObserversModel();

}

@Override

public void startController() {

defineBehavior();

initAllViews();

}

private void setObserversModel() {

getModel().addView(pieChart);

getModel().addView(principalMenu);

getModel().addView(barChart);

}

@Override

public void handleActionEvent(ActionEvent e) {

Candidate candidate = (Candidate) principalMenu.getObjectList();

getModel().processData(candidate);

principalMenu.eraseTextLabels();

}

@Override

public void handleListSelectionEvent(ListSelectionEvent e) {

Candidate candidato = (Candidate) principalMenu.getObjectList();

if (candidato != null) {

principalMenu.setTextLabelCandidate(candidato.getName());

principalMenu.setTextLabelVotes(candidato.getVotes() + "");

}

}

}

**Model**

public abstract class Model<DataToProcess, DataToSend> {

private List<View> viewsList = new ArrayList<>();

public void addView(View view) {

viewsList.add(view);

}

public void removeView(View view) {

viewsList.remove(view);

}

public List<View> getAllViews() {

return viewsList;

}

public abstract DataToSend getDataInformation();

public void processData() {

processData(null);

}

public abstract void processData(DataToProcess data);

public void notifyAllViews() {

notifyAllViews(null);

}

public void notifyAllViews(DataToSend data) {

viewsList.forEach((View view) -> view.updateView(this, data));

}

}

**Candidate(parte del modelo)**

public class Candidate {

private String name;

private int votes;

public Candidate(String name, int votes) {

this.name = name;

this.votes = votes;

}

public Candidate(String name) {

this.name = name;

votes = 0;

}

public void addVote() {

votes++;

}

public void setName(String name) {

this.name = name;

}

public void setVotes(int votes) {

this.votes = votes;

}

public String getName() {

return name;

}

public int getVotes() {

return votes;

}

@Override

public String toString() {

return name + " " + votes;

}

@Override

public boolean equals(Object obj) {

if (obj == null) {

return false;

}

if (getClass() != obj.getClass()) {

return false;

}

final Candidate other = (Candidate) obj;

if (!Objects.equals(this.name, other.name)) {

return false;

}

if (this.votes != other.votes) {

return false;

}

return true;

}

}

**CandidatesCreatorCSV (parte del modelo)**

public class CandidatesCreatorCSV {

public List<Candidate> getSavedCandidates() throws FileNotFoundException, DatasNoSupportedException {

ManagerCSV csv = ManagerCSV.getInstance();

return createCandidates(csv.readInformation());

}

public void saveCandidates(List<Candidate> listCandidates) throws IOException {

ManagerCSV csv = ManagerCSV.getInstance();

csv.writeInformation(listCandidates);

}

public void saveCandidate(Candidate candidate) throws FileNotFoundException, IOException, DatasNoSupportedException {

List<Candidate> listCandidates = getSavedCandidates();

listCandidates.add(candidate);

saveCandidates(listCandidates);

}

public List<Candidate> createCandidates(List<String> list) throws DatasNoSupportedException {

List<Candidate> listCandidates = new ArrayList<>();

for (String e : list) {

Candidate cand = createCandidate(e);

listCandidates.add(cand);

}

return listCandidates;

}

public Candidate createCandidate(String data) throws DatasNoSupportedException {

String[] tokens = data.split(",");

if (tokens.length > 2) {

throw new DatasNoSupportedException();

}

return new Candidate(tokens[0], Integer.parseInt(tokens[1]));

}

}

**ManagerCSV (parte del modelo)**

public class ManagerCSV {

public static final String FILE\_PATH = "prueba.txt";

public static final String DELIMETER\_CSV = ",";

public static final String DELIMETER\_DELETE = " ";

private File fileDoc;

private static ManagerCSV instance;

private ManagerCSV() {

fileDoc = new File(FILE\_PATH);

if (!fileDoc.exists()) {

try {

fileDoc.createNewFile();

} catch (IOException ex) {

Logger.getLogger(ManagerCSV.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

public static synchronized ManagerCSV getInstance() {

if (instance == null) {

instance = new ManagerCSV();

}

return instance;

}

public void writeInformation(List<?> list) throws IOException {

FileWriter fw = new FileWriter(fileDoc);

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

String csvString = formatStringCSV(list.get(i).toString());

fw.write(csvString + "\n");

}

fw.close();

}

public List<String> readInformation() throws FileNotFoundException {

Scanner fr = new Scanner(fileDoc);

List<String> informationFile = new ArrayList<>();

while (fr.hasNext()) {

informationFile.add(fr.nextLine());

}

return informationFile;

}

public static String formatStringCSV(String lineString) {

return lineString.replaceAll(DELIMETER\_DELETE, DELIMETER\_CSV);

}

}

**CandidateInfoBD (modelo concreto)**

public class CandidatesInfoBD extends Model<Candidate, List<Candidate>> {

private CandidatesCreatorCSV csv = new CandidatesCreatorCSV();

private void addVote(Candidate candidate) {

List<Candidate> candidates = null;

try {

candidates = csv.getSavedCandidates();

int indexCand = candidates.indexOf(candidate);

if (indexCand != -1) {

Candidate cand = candidates.get(indexCand);

cand.addVote();

candidates.set(indexCand, cand);

}

csv.saveCandidates(candidates);

notifyAllViews(getDataInformation());

} catch (FileNotFoundException ex) {

} catch (DatasNoSupportedException ex) {

} catch (IOException ex) {

}

}

private List<Candidate> getCandidates() throws FileNotFoundException, DatasNoSupportedException {

return csv.getSavedCandidates();

}

@Override

public List<Candidate> getDataInformation() {

try {

return getCandidates();

} catch (FileNotFoundException ex) {

} catch (DatasNoSupportedException ex) {

}

return null;

}

@Override

public void processData(Candidate data) {

addVote(data);

}

}

**View**

public abstract class View<DataToRecive> extends JFrame {

private Controller controller;

public void initView() {

initView(null);

}

public abstract void initView(DataToRecive data);

public abstract void updateView(Model model, DataToRecive data);

public abstract void addControllerBehavior(Controller controller);

public void showMessageError(String cause) {

JOptionPane.showMessageDialog(this, cause);

}

public void setController(Controller controller) {

this.controller = controller;

}

public Controller getController() {

return controller;

}

}

**PrincipalMenu (vista concreta)**

public class PrincipalMenu extends View<List<Candidate>> {

public PrincipalMenu() {

initComponents();

}

public JButton getBtnVotar() {

return btnVotar;

}

public JLabel getLabelCandidato() {

return labelCandidato;

}

public JLabel getLabelVotos() {

return labelVotos;

}

public JList getListCandidatos() {

return listCandidatos;

}

public JPanel getPanelFoto() {

return panelFoto;

}

public void setBtnVotar(JButton btnVotar) {

this.btnVotar = btnVotar;

}

public void setLabelCandidato(JLabel labelCandidato) {

this.labelCandidato = labelCandidato;

}

public void setLabelVotos(JLabel labelVotos) {

this.labelVotos = labelVotos;

}

public void setListCandidatos(JList listCandidatos) {

this.listCandidatos = listCandidatos;

}

public void setPanelFoto(JPanel panelFoto) {

this.panelFoto = panelFoto;

}

// Variables declaration - do not modify

private javax.swing.JButton btnVotar;

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JLabel labelCandidato;

private javax.swing.JLabel labelVotos;

private javax.swing.JList listCandidatos;

private javax.swing.JPanel panelFoto;

// End of variables declaration

public void fillList(List<?> objectList) {

DefaultListModel model = new DefaultListModel();

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

model.add(i, objectList.get(i));

}

listCandidatos.setModel(model);

}

public Object getObjectList() {

return listCandidatos.getSelectedValue();

}

public void setTextLabelCandidate(String name) {

labelCandidato.setText("Nombre: " + name);

}

public void setTextLabelVotes(String votes) {

labelVotos.setText("votos: " + votes);

}

public void eraseTextLabels() {

labelVotos.setText("Votos: ");

labelCandidato.setText("Nombre: ");

}

@Override

public void addControllerBehavior(Controller controller) {

getBtnVotar().addActionListener(controller::handleActionEvent);

getListCandidatos().addListSelectionListener(controller::handleListSelectionEvent);

}

@Override

public void initView(List<Candidate> data) {

fillList((List<?>) data);

setLocationRelativeTo(null);

this.setVisible(true);

}

@Override

public void updateView(Model model, List<Candidate> data) {

fillList((List<?>) data);

}

}

**FrmPieChart (vista concreta)**

public class FrmPieChart extends View<List<Candidate>> {

private int TOTAL\_VOTOS;

List<Candidate> candidatos = new ArrayList<>();

public FrmPieChart() {

this.setBackground(Color.WHITE);

this.setSize(new Dimension(400, 400));

setLocation(50, 100);

TOTAL\_VOTOS = 0;

}

public FrmPieChart(List<Candidate> candidatos) {

this.setBackground(Color.WHITE);

this.setSize(new Dimension(400, 400));

setLocation(50, 100);

this.candidatos = candidatos;

TOTAL\_VOTOS = 0;

}

public void setCandidatos(List<Candidate> candidatos) {

this.candidatos = candidatos;

}

private int getTotalVotos() {

int total = 0;

for (Candidate candidato : candidatos) {

total += candidato.getVotes();

}

return total;

}

@Override

public void paint(Graphics g) {

super.paint(g);

int total\_Candidatos = candidatos.size();

int[] grados\_Candidato = new int[total\_Candidatos];

TOTAL\_VOTOS = getTotalVotos();

for (int i = 0; i < total\_Candidatos; i++) {

if (TOTAL\_VOTOS > 0) {

grados\_Candidato[i] = candidatos.get(i).getVotes() \* 360 / TOTAL\_VOTOS;

}

}

int width = this.getWidth() / 4 - 30;

int height = this.getHeight() / 4 - 35;

g.setColor(new Color(255, 0, 0));

int grados = 0;

for (int i = 0; i < total\_Candidatos; i++) {

grados += grados\_Candidato[i];

g.setColor(new Color(200, (i \* 50) % 255, (i \* 100) % 255));

g.fillArc(width, height, 300, 300, grados - grados\_Candidato[i], grados\_Candidato[i]);

g.setColor(Color.BLACK);

g.drawString("Candidato " + (i + 1) + " " + candidatos.get(i).getName() + " "

+ " Votos: " + candidatos.get(i).getVotes(), 20, 45 + (i \* 25));

}

}

@Override

public void addControllerBehavior(Controller controller) {

}

@Override

public void initView(List<Candidate> data) {

if (data != null) {

setCandidatos(data);

this.setVisible(true);

}

}

@Override

public void updateView(Model model, List<Candidate> data) {

setCandidatos(data);

repaint();

}

}

**FrmBarChart (vista concreta)**

public class FrmBarChart extends View<List<Candidate>> {

private List<Candidate> candidatos = new ArrayList<>();

public FrmBarChart() {

this.setBackground(Color.WHITE);

this.setSize(new Dimension(400, 400));

setLocation(910, 100);

}

public FrmBarChart(List<Candidate> candidatos) {

this.setBackground(Color.WHITE);

this.candidatos = candidatos;

this.setSize(new Dimension(400, 400));

setLocation(910, 100);

}

public List<Candidate> getCandidatos() {

return candidatos;

}

public void setCandidatos(List<Candidate> candidatos) {

this.candidatos = candidatos;

}

private int encontrarMayorDato() {

int mayor = 0;

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

if (mayor < candidatos.get(i).getVotes()) {

mayor = candidatos.get(i).getVotes();

}

}

return mayor;

}

@Override

public void paint(Graphics g) {

super.paint(g); //To change body of generated methods, choose Tools | Templates.

int mayor = encontrarMayorDato();

int numeroCandidatos = candidatos.size();

int[] largo\_votos = new int[numeroCandidatos];

for (int i = 0; i < numeroCandidatos; i++) {

if (mayor > 0) {

largo\_votos[i] = candidatos.get(i).getVotes() \* 350 / mayor;

}

}

for (int i = 0; i < numeroCandidatos; i++) {

g.setColor(new Color(200, (i \* 50) % 255, (i \* 100) % 255));

g.drawString("Candidato " + (i + 1) + " " + candidatos.get(i).getName() + " "

+ candidatos.get(i).getVotes(), 10, 50 + (i \* 55));

g.fillRect(40, 60 + (i \* 55), largo\_votos[i], 30);

}

}

@Override

public void addControllerBehavior(Controller controller) {

}

@Override

public void initView(List<Candidate> data) {

setCandidatos(data);

this.setVisible(true);

}

@Override

public void updateView(Model model, List<Candidate> data) {

setCandidatos(data);

repaint();

}

}

**Main**

public class NewMain {

public static void main(String[] args) throws IOException, DatasNoSupportedException {

Model model = new CandidatesInfoBD();

Controller mv = new ManagerProgram(model, new FrmBarChart(), new FrmPieChart(), new PrincipalMenu());

mv.startController();

}

***Ejecución***

