MODEL-VIEW-CONTROLER

Prof. Fellipe Aleixo (fellipe.aleixo@ifrn.edu.br)

O Que é MVC?

- Modelo de arquitetura de software
  - Separar dados ou lógica de negócios (*Model*) da interface do usuário (*View*) e do fluxo da aplicação (*Control*)

- Objetivos:
  - A ideia é permitir que uma mesma lógica de negócios possa ser acessada e visualizada através de várias interfaces
    - (1) reusabilidade do código e (2) separação dos conceitos

O Que é MVC?

- Interfaces com o usuário são sensíveis a mudanças:
  - O usuário está sempre querendo mudar funcionalidades e a interface das aplicações

- A mesma aplicação possui diferentes requisitos dependendo do usuário:
  - um digitador prefere uma interface onde tudo pode ser feito através do teclado e visualizado como texto.
  - um gerente prefere uma interface através do mouse e de menus com visualização gráfica

O Que é MVC?

- A aplicação pode ter que ser implementada em outra plataforma
- Neste contexto, se o código para a interface gráfica é muito acoplado ao código da aplicação, o desenvolvimento pode se tornar muito caro e difícil
- Outro exemplo: quantas interfaces possíveis existem para a lógica de negócio das contas correntes de um banco?

O Que é MVC?

- Tipos de componentes:
  - Modelo – consiste dos dados da aplicação e as regras de negócio
  - Visão – pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama
  - Controlador – faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão

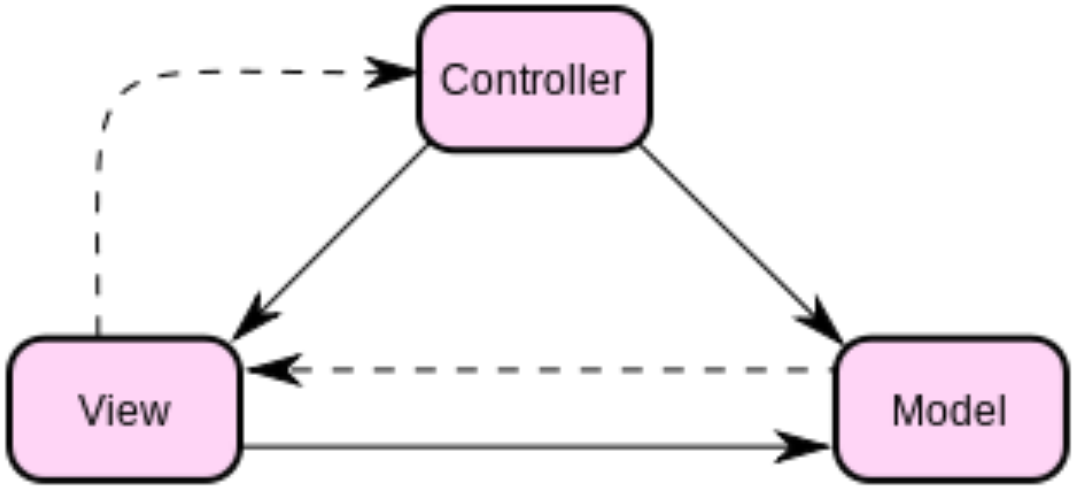
- A lógica de negócios não sabe de quantas nem quais interfaces com o usuário estão exibindo seu estado

Interação entre os Componentes

- O modelo MVC define também como deve ser a interação entre os tipos de componentes
  - Controlador → Visão: definir/alterar a apresentação do modelo
  - Controlador → Modelo: atualizar o estado do modelo
  - Modelo → Visão (e Controladores): notifica mudanças em seu estado
  - Visão → Modelo: solicita informações para gerar uma representação das mesmas

Interação entre os Componentes

- Ilustração da interação entre os componentes



Utilização em Aplicações Web

- O modelo MVC vem sendo amplamente utilizado no desenvolvimento de aplicações Web

- Adotado como padrão por vários frameworks voltados ao desenvolvimento de sistemas Web
  - As interpretações de cada framework variam, principalmente no modo que as responsabilidades são divididas entre o cliente e o servidor

Utilização em Aplicações Web

- Frameworks Web mais recentes adotam uma abordagem "*thin client*"
  - Modelo, visão e a lógica do controlador inteiros no servidor
    - Cliente envia uma requisição Web (com dados de formulários) que é tratada por um controlador
    - Controlador aciona a lógica do negócio dos elementos do modelo
    - Controlador também monta uma resposta com elementos da visão e envia de volta ao cliente

Utilização em Aplicações Web

- Bibliotecas baseadas em Javascript foram desenvolvidas para criadas para permitir que componentes MVC executem no cliente

Justificativa

- Separação entre os “dados” e a “apresentação” das aplicações
  - Alterações feitas no layout não afetam a manipulação dos dados – podem sofrer alterações

“acesso à dados” e “lógica do negócio”

+

“lógica de apresentação”

+

“controle da interação com o usuário”

Aplicação do MVC

Exemplo: sistema de enquete

Aplicação em 5 (cinco) passos

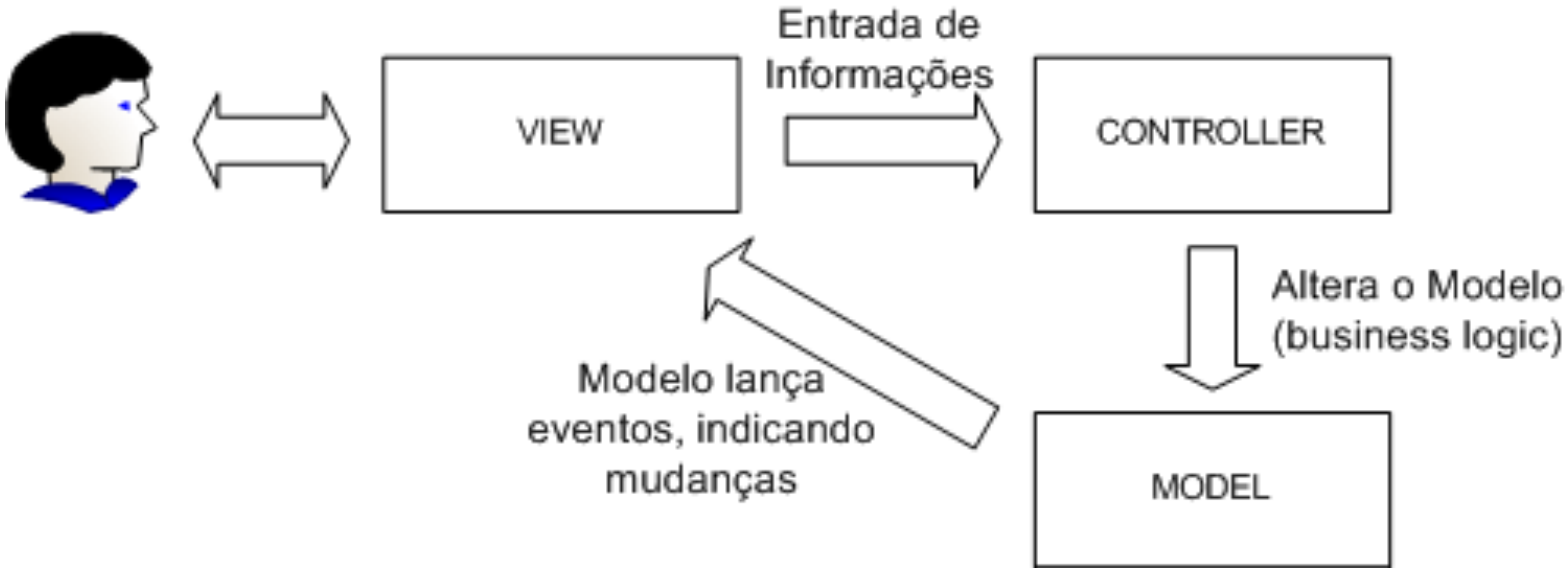
O Sistema: Enquete com Usuários

- Queremos implementar um sistema de votação, fazer uma enquete
  - A enquete deve permitir o voto dos usuários
  - Os votos são contabilizados e exibidos de duas formas:
    - Tela com votos absolutos, que mostra os totais de votos para cada opção
    - Tela com percentual de votos

Solução com MVC

- Aplicação dividida em três partes:
  - Modelo: Lógica de negócio
  - Visão: Camada de interface com o usuário
    - Nesta camada o usuário vê o estado do modelo e pode manipular a interface, para ativar a lógica do negócio
  - Controlador: Transforma eventos gerados pela interface em ações de negócio, alterando o modelo

Solução com MVC



- Há outras formas de gerar informação para as visões
  - Exemplo: o controlador, chama a lógica do negócio, recebe resultados e os repassa para a visão apropriada
    - É papel do controlador escolher a visão mais apropriada

PASSO 1

- Isole a ”lógica do negócio" de seu sistema
  - Ex.: crie um pacote separado para armazenar as classes que representam o modelo do seu sistema
  - Os dados do modelo podem estar armazenados em um SGBD
- Atenção! As classes que compõem o modelo de negócio não podem conhecer NADA do ambiente externo! Não deve haver referencias para o mundo fora do negócio
- De volta ao nosso exemplo, vamos isolar a lógica do negócio do sistema de enquete
  - Classe enquete.model.EnqueteSimples

```
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

public class EnqueteSimples {

    private Map <String,Integer>opcoes;

    public EnqueteSimples(){
        opcoes = new HashMap<String, Integer>();
    }

    /**
     * Adiciona uma opção para ser votada na enquete
     * @param opcao nome da opção
     */
    public void addOpcao(String opcao){
        opcoes.put(opcao,new Integer(0));
    }

    /**
     * Retorna um iterador de opções disponíveis na enquete
     * @return Iterator opções disponiveis na enquete
     */
    public Set <String> getOpcoes(){
        return opcoes.keySet();
    }
```

```
    /**
     * Incrementa um voto para opção
     * @param opcao opção que receberá voto
     */
    public void votar(String opcao){
        int votoAtual = ((Integer)opcoes.get(opcao)).intValue();
        opcoes.put(opcao,new Integer(++votoAtual));
    }
    /**
     * Retorna a soma dos votos de todas as opções da enquete
     * @return int soma dos votos de todas as opções da enquete
     */
    public int getTotalVotos(){

        int total = 0;
        for(Integer votos : opcoes.values()){
            total+= votos.intValue();
        }
        return total;
    }
    /**
     * Retorna a quantidade de votos de uma opção individual
     * @param opcao opção que se quer o voto
     * @return int quantidade de votos da opção
     */
    public int getVotos(String opcao){
        return (opcoes.get(opcao)).intValue();
    }
}
```

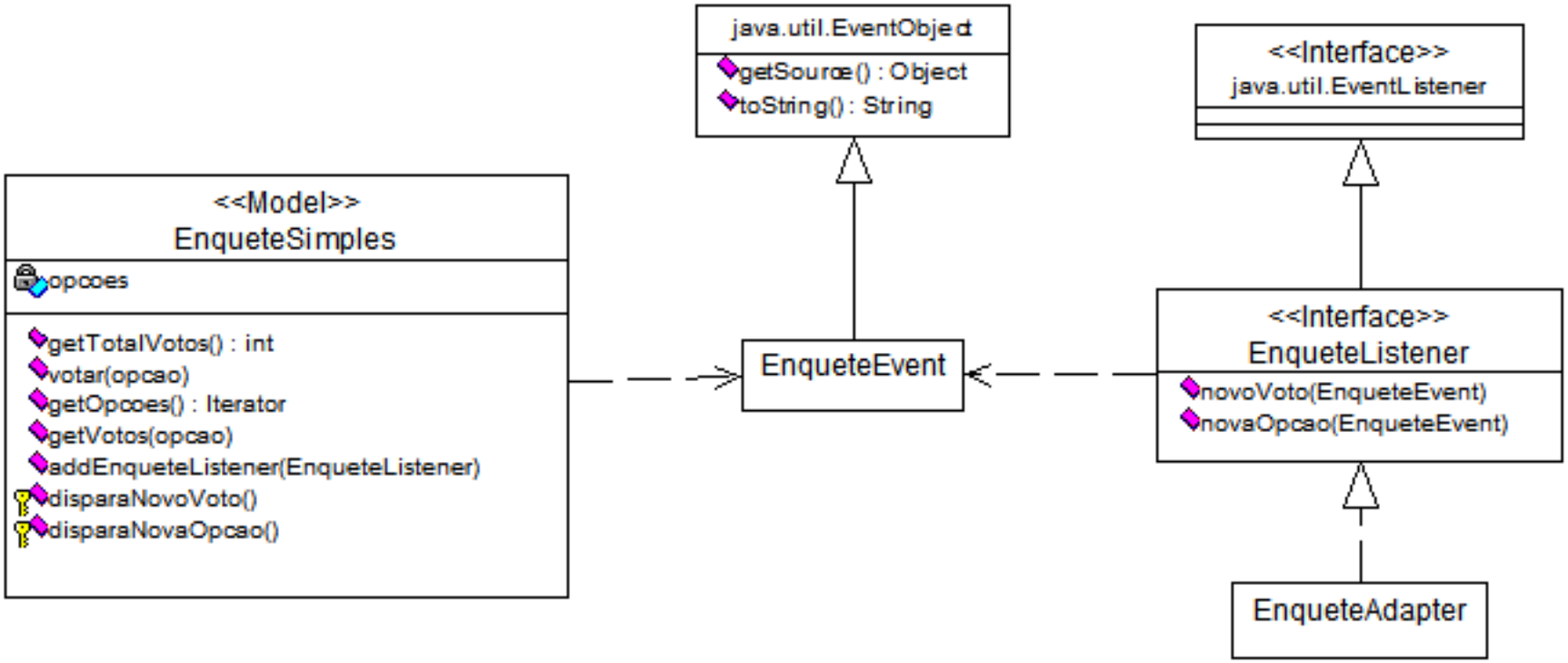
PASSO 2

- As classes que compõem o modelo de negócio não devem conhecer nada sobre as camadas de interface que exibem suas informações

- Como fazer com que o modelo informe mudanças em seu estado para as interfaces, sem conhecê-las?

- Aplicaremos então o padrão Observer! O nosso modelo de negócio será o gerador de eventos para as interfaces, as quais serão "listeners"

PASSO 2


java.util.EventObject
getSource() : Object
toString() : String
<<Interface>>
java.util.EventListener
<<Model>>
EnqueteSimples
opcoes
getTotalVotos() : int
votar(opcao)
getOpcoes() : Iterator
getVotos(opcao)
addEnqueteListener(EnqueteListener)
disparaNovoVoto()
disparaNovaOpcao()
EnqueteEvent
<<Interface>>
EnqueteListener
novoVoto(EnqueteEvent)
novaOpcao(EnqueteEvent)
EnqueteAdapter

```
import java.util.List;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.Set;

public class EnqueteSimples {

    private Map <String,Integer>opcoes;
    private List <EnqueteListener>enqueteListeners = new LinkedList();


    public EnqueteSimples(){
        opcoes = new HashMap<String, Integer>();
    }

    /**
     * Adiciona uma opção para ser votada na enquete
     * @param opcao nome da opção
     */
    public void addOpcao(String opcao){
        opcoes.put(opcao,new Integer(0));
        this.disparaNovaOpcao(opcao);
    }

    /**
     * Retorna um iterador de opções disponíveis na enquete
     * @return Iterator opções disponiveis na enquete
     */
    public Set <String> getOpcoes(){
        return opcoes.keySet();
    }
```

```
    /**
     * Incrementa um voto para opção
     * @param opcao opção que receberá voto
     */
    public void votar(String opcao){
        int votoAtual = (opcoes.get(opcao)).intValue();
        opcoes.put(opcao,new Integer(++votoAtual));
        this.disparaNovoVoto(opcao);
    }

    /**
     * Retorna a soma dos votos de todas as opções da enquete
     * @return int soma dos votos de todas as opções da enquete
     */
    public int getTotalVotos(){

        int total = 0;
        for(Integer votos : opcoes.values()){
            total+= votos.intValue();
        }
        return total;
    }

    /**
     * Retorna a quantidade de votos de uma opção individual
     * @param opcao opção que se quer o voto
     * @return int quantidade de votos da opção
     */
    public int getVotos(String opcao){
        return (opcoes.get(opcao)).intValue();
    }
```

```
    /**
     * Adiciona um EnqueteListener, um objeto interessado em
     * receber eventos lançados pela Enquete
     * @see EnqueteListener
     * @param listener objeto interessado em receber eventos
     */
    public synchronized void addEnqueteListener(EnqueteListener listener){
        if(enqueteListeners.contains(listener)){ return; }
        this.enqueteListeners.add(listener);
    }

    /**
     * Informa aos objetos interessados nos eventos lançados
     * pela Enquete que um novo voto foi contabilizado.
     */
    private synchronized void disparaNovoVoto(String opcao){
        for(EnqueteListener listeners : this.enqueteListeners){
            listeners.novoVoto(new EnqueteEvent(this,opcao));
        }
    }

    /**
     * Informa aos objetos interessados nos enventos lançados
     * pela Enquete que uma nova opção foi adicionada.
     */
    private synchronized void disparaNovaOpcao(String opcao){

        for(EnqueteListener listeners : this.enqueteListeners){
            listeners.novaOpcao(new EnqueteEvent(this,opcao));
        }
    }

}
```

Classe enquete.model.EnqueteEvent

```
import java.util.EventObject;

public class EnqueteEvent extends EventObject {
    private String opcao = null;
    private int votos = 0;

    public EnqueteEvent(EnqueteSimples source){
        super(source);
    }
    public EnqueteEvent(EnqueteSimples source,String opcao){
        this(source);
        this.opcao = opcao;
    }
    /**
     * Retorna a opção associada ao evento gerado. A opção pode ser uma nova opção
     * adicionada à EnqueteSimples ou a opção escolhida para adicionar um novo voto.
     * @return String opção
     */
    public String getOpcao() { return opcao;}
    /**
     * Retorna o numero de votos da opcao
     * @return int votos
     */
    public int getVotos() { return ((EnqueteSimples)this.source).getVotos(opcao);}
    /**
     * Retorna o total de votos da enquete
     * @return int
     */
    public int getTotalVotos() { return ((EnqueteSimples)this.source).getTotalVotos();}
}
```

Interface enquete.model.EnqueteListener

```
import java.util.EventListener;

public interface EnqueteListener extends EventListener {


    /**
     * Invocado quando um novo voto é contabilizado na Enquete.
     * @param event Evento gerado pela Enquete.
     */
    public void novoVoto(EnqueteEvent event);

    /**
     * Invocado quando uma nova opção é adicionada à Enquete.
     * @param event Evento gerado pela Enquete.
     */
    public void novaOpcao(EnqueteEvent event);
}
```

PASSO 3

- Fazer com que as telas interessadas em exibir o estado atual do modelo implementem o EnqueteListener

- Veja a seguir as classes enquete.view.TelaResultado, enquete.view.TelaResultadoPercentual, enquete.view.TelaVotacao

```
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.Window;
import java.util.HashMap;
import java.util.Map;

import java.awt.Frame;


public class TelaResultado extends Window implements EnqueteListener{

    private Map <String, Label>labels = new HashMap();

    public TelaResultado(Frame parent){
        super(parent);
        this.setSize(110,120);
        this.setLayout(new GridLayout(0,2));  // Grid com qualquer numero
                                              // de linhas e uma coluna
        this.add(new Label("Votos"));
        this.add(new Label());

    }
```

```
    /**
     * @see enquete.model.EnqueteListener#novaOpcao(EnqueteEvent)
     */
    public void novaOpcao(EnqueteEvent event) {
        String opcao = event.getOpcao();

        Label label;
        Label votos;
        if(!labels.containsKey(opcao)){
            label = new Label(opcao+" - ");
            votos = new Label(""+event.getVotos());
            labels.put(opcao,votos);
            this.add(label);
            this.add(votos);
        }
    }

    /**
     * @see enquete.model.EnqueteListener#novoVoto(EnqueteEvent)
     */
    public void novoVoto(EnqueteEvent event) {
        String opcao = event.getOpcao();

        Label votos;
        votos = labels.get(opcao);
        votos.setText(""+event.getVotos());
    }

}
```

```
import java.awt.GridLayout;
import java.awt.Label;
import java.awt.Window;
import java.util.HashMap;
import java.util.Map;

import java.awt.Frame;

public class TelaResultadoPercentual extends Window implements EnqueteListener{

    private Map <String,Label>labels = new HashMap();


    public TelaResultadoPercentual(Frame parent){
        super(parent);
        this.setSize(180,120);
        this.setLayout(new GridLayout(0,2));  // Grid com qualquer numero
                                              // de linhas e uma coluna
        this.add(new Label("Percentual"));
        this.add(new Label());
    }
```

```
    /**
     * @see enquete.model.EnqueteListener#novaOpcao(EnqueteEvent)
     */
    public void novaOpcao(EnqueteEvent event) {
        String opcao = event.getOpcao();

        Label label;
        Label votos;
        if(!labels.containsKey(opcao)){
            label = new Label(opcao+" - ");
            votos = new Label(""+event.getVotos()+" %");
            labels.put(opcao,votos);
            this.add(label);
            this.add(votos);
        }
    }

    /**
     * @see enquete.model.EnqueteListener#novoVoto(EnqueteEvent)
     */
    public void novoVoto(EnqueteEvent event) {
        String opcao = event.getOpcao();

        Label votos;
        votos = labels.get(opcao);
        votos.setText(""+(event.getVotos()*100/event.getTotalVotos())+" %");
    }

}
```

```
import java.awt.Button;
import java.awt.GridLayout;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.ArrayList;
import java.util.Collection;
import java.awt.Frame;

public class TelaVotacao extends Frame implements EnqueteListener{

    private Collection <String>botoes = new ArrayList();
    private ActionListener controller;

    public TelaVotacao(ActionListener controller){
        super("Tela de Votação - Enquete");
        this.setSize(100,120);
        this.setLayout(new GridLayout(0,1)); // Grid com qualquer numero
                                             // de linhas e uma coluna
        this.controller = controller;
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                e.getWindow().hide();
                System.exit(0);
            }
        });
    }
```

```
    /**
     * @see enquete.model.EnqueteListener#novaOpcao(EnqueteEvent)
     */
    public void novaOpcao(EnqueteEvent event) {
        String opcao = event.getOpcao();
        Button botao;

        if(!botoes.contains(opcao)){
            botoes.add(opcao);
            botao = new Button(opcao);
            botao.setActionCommand(opcao);
            botao.addActionListener(controller);
            this.add(botao);
        }
    }

    /**
     * @see enquete.model.EnqueteListener#novoVoto(EnqueteEvent)
     */
    public void novoVoto(EnqueteEvent event) {
        // Nothing to do
    }

}
```

PASSO 4

- Implemente o controlador, ou seja, a classe que receberá os eventos da interface e transformará estes eventos em ações no modelo

- Nesse exemplo, o controlador é uma classe simples que atende aos eventos executados pelos botões da classe TelaVotacao e incrementa os votos no modelo

- Veja:
  - Classe enquete.controller.TelaVotacaoCtrl

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TelaVotacaoCtrl implements ActionListener{

    private EnqueteSimples enquete;

    public TelaVotacaoCtrl(EnqueteSimples enquete){
        this.enquete = enquete;
    }

    /**
     * Evento lançado pelo clique nos botoes da TelaVotacao
     * @see java.awt.event.ActionListener#actionPerformed(ActionEvent)
     */
    public void actionPerformed(ActionEvent event) {
        enquete.votar(event.getActionCommand());
    }
}
```

PASSO 5

- Junte os pedaços da aplicação
- Isso pode ser feito via programação de uma classe ou através de um deployment através de XML, por exemplo
- Veja a classe enquete.Enquete

```
public class Enquete{
    public static void main(String[] args) {
        // Modelo
        EnqueteSimples enquete= new EnqueteSimples();

        // Controlador da Interface "TelaVotacao"
        TelaVotacaoCtrl ctrl = new TelaVotacaoCtrl(enquete);

        // Interface que altera o estado do modelo
        TelaVotacao votacao = new TelaVotacao(ctrl);
        votacao.setLocation(5,5);

        // Interface que exibe o resultado absoluto da votacao
        TelaResultado resultado = new TelaResultado(votacao);
        resultado.setLocation(120,5);

        // Interface que exibe o resultado percentual da votacao
        TelaResultadoPercentual resultadoPerc =
                                new TelaResultadoPercentual(votacao);
        resultadoPerc.setLocation(250,5);

        // Adicionando as interfaces interessadas na mudança do estado do modelo
        enquete.addEnqueteListener(votacao);
        enquete.addEnqueteListener(resultado);
        enquete.addEnqueteListener(resultadoPerc);

        // Povoando o modelo
        enquete.addOpcao("Opção 1");
        enquete.addOpcao("Opção 2");
        enquete.addOpcao("Opção 3");
        enquete.addOpcao("Opção 4");

        // Exibindo as interfaces
        votacao.show();
        resultado.show();
        resultadoPerc.show();
    }
}
```

MVC no Play Framework

http://www.playframework.com/documentation/1.0/main