Model-View-Controller (MVC)

por Rodrigo Rebouças de Almeida

Objetivo

- Separar dados ou lógica de negócios (Model) da interface do usuário (View) e do fluxo da aplicação (Control)
- A idéia é permitir que uma mesma lógica de negócios possa ser acessada e visualizada através de várias interfaces.
- Na arquitetura MVC, a lógica de negócios (chamaremos de Modelo) não sabe de quantas nem quais interfaces com o usuário estão exibindo seu estado.
- Com as diversas possibilidades de interfaces que conhecemos hoje, a MVC é uma ferramenta indispensável para desenvolvermos sistemas (Figura 1).

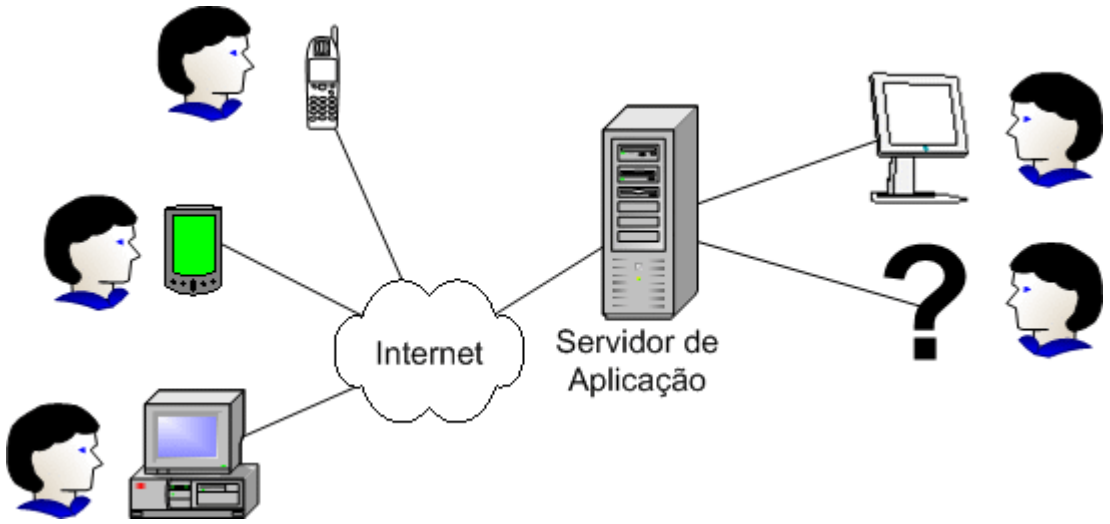

Figura 1: Aplicação sendo acessada por várias interfaces

Sobre a MVC

- Um dos primeiros padrões identificados; surgiu na comunidade de Smalltalk;
- Contexto: aplicações interativas que requerem interfaces flexíveis.

Problema:

- Interfaces com o usuário são sensíveis a mudanças:
  - O usuário está sempre querendo mudar funcionalidades e a interface das aplicações. Se o sistema não suporta estas mudanças, temos um grande problema!
- A aplicação pode ter que ser implementada em outra plataforma.
- 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
- 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?

Exemplo:

- 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.
- Veja o exemplo

Solução sem MVC:

- Uma solução simples seria a criação de uma tela de votação que armazena o estado da enquete e

incrementa os votos à medida que os botões são clicados.

- Ao incrementar uma opção, as telas que exibem os resultados seriam atualizadas.
- Observe a solução a seguir:
- arquivo TelaVotacao.java:

```
public class TelaVotacao extends Frame implements ActionList

	private TelaResultado telaResult;
	private TelaResultadoPercentual telaResultPerc;

	private Map opcoes;
	private List listaOpcoes; // para obter as opções em o

	public TelaVotacao() {
		super("Tela de Votação - Enquete");

		telaResult = new TelaResultado(this);
		telaResult.setLocation(120, 5); // posicao na te

		telaResultPerc = new TelaResultadoPercentual(thi
		telaResultPerc.setLocation(250, 5); // posicao n

		listaOpcoes = new Vector();
		this.opcoes = new HashMap();

		this.adicionaOpcao("Opção 1");
		this.adicionaOpcao("Opção 2");
		this.adicionaOpcao("Opção 3");
		this.adicionaOpcao("Opção 4");
		criarBotoes();

		telaResult.inicializar(listaOpcoes.iterator());
		telaResultPerc.inicializar(listaOpcoes.iterator(

		this.setSize(100, 120);
		this.setLayout(new GridLayout(0, 1));

		this.addWindowListener(new WindowAdapter() {
			public void windowClosing(WindowEvent e) {
				e.getWindow().hide();
				System.exit(0);
			}
		});
		this.show();
		telaResult.show();
```

```
        telaResultPerc.show();
    }

    /**
     * Adiciona uma opcao à enquete
     **/
    private void adicionaOpcao(String opcao) {
        listaOpcoes.add(opcao);
        this.opcoes.put(opcao, new Integer(0));
    }

    /**
     * Cria os botoes da tela de votos
     **/
    public void criarBotoes() {
        Iterator it = listaOpcoes.iterator();
        String opcao;
        Button botao;
        while (it.hasNext()) {
            opcao = (String) it.next();
            botao = new Button(opcao);
            botao.setActionCommand(opcao);
            botao.addActionListener(this);
            this.add(botao);
        }
    }

    /**
     * Método executado ao clicar nos botões
     */
    public void actionPerformed(ActionEvent event) {
        String opcao = event.getActionCommand();
        this.votar(opcao); // incrementando o voto

        // Atualizando a tela de resultados absolutos
        telaResult.novoVoto(opcao, getVotos(opcao));

        // Atualizando a tela de resultados percentuais
        telaResultPerc.novoVoto(opcao, getVotos(opcao),
    }

    /**
     * Incrementa o voto da opção entrada
     */
    public void votar(String opcao) {
        int votoAtual = ((Integer) this.opcoes.get(opcao
        this.opcoes.put(opcao, new Integer(++votoAtual))
    }
```

```
	/**
	 * Retorna a soma dos votos de todas as opções da enqu
	 * @return int soma dos votos de todas as opções da en
	 */
	public int getTotalVotos() {
		Iterator votos = this.opcoes.values().iterator()
		int total = 0;
		while (votos.hasNext()) {
			total += ((Integer) votos.next()).intValue
		}
		return total;
	}

	/**
	 * Retorna a quantidade de votos de uma opção individu
	 */
	public int getVotos(String opcao) {
		return ((Integer) this.opcoes.get(opcao)).intVal
	}
}
```

Arquivo TelaResultado.java:

```
public class TelaResultado extends Window{

	private Map labels = new HashMap();

	public TelaResultado(Frame parent){
		super(parent);
		this.setSize(110,120);
		this.setLayout(new GridLayout(0,2)); // Grid com

		this.add(new Label("Votos"));
		this.add(new Label());
	}

	/**
	 * Recebe a lista de opcoes inicial
	 */
	public void inicializar(Iterator opcoes) {
		String opcao;

		Label label;
		Label votos;
		while(opcoes.hasNext()){
			opcao = (String)opcoes.next();
```

```
                if(!labels.containsKey(opcao)){
                      label = new Label(opcao+" - ");
                      votos = new Label(""+0);
                      labels.put(opcao,votos);
                      this.add(label);
                      this.add(votos);
                }
          }
    }

    /**
     * Atualiza a quantidade de votos para uma determinada
     */
    public void novoVoto(String opcao, int nvotos) {
          Label votos;
          votos = (Label)labels.get(opcao);
          votos.setText(""+nvotos);
    }

}
```

- A classe TelaResultadoPercentual é semelhante à TelaResultado, a diferença é o cálculo da percentagem.

A solução acima funciona, mas temos muitos problemas!

- E se, a partir de agora, o cliente começa a acessar a Internet e tem a idéia de colocar a enquete para os usuários da rede?
- E pior, o cliente comprou um celular com suporte a WAP e sacou que seria interessante ter a enquete também em WAP!
- Os problemas aumentarão à medida que nosso cliente comprar um celular com suporte a J2ME, um PALM, um relógio que acessa a Internet... Temos um problema.
- Qual o problema da solução acima?
  - A classe TelaVotacao representa a interface de votação e ao mesmo tempo armazena o estado da enquete!
  - Não conseguimos extrair o "business logic" da enquete pois ele está misturado com a interface!

- Premissa básica para uma boa solução:
  - "A LÓGICA DE NEGÓCIO NÃO DEVE CONHECER NADA SOBRE AS TELAS QUE EXIBEM SEU ESTADO!"
- Onde está a lógica de negócio, acima?
  - Onde está a parte gráfica?
  - Há acoplamento? (Um pode existir sem o outro?)

Solução com a MVC:

- A aplicação é dividida em três partes (Figura 2):
  - Modelo (MODEL): Lógica de negócio;
  - Visão (VIEW): Camada de interface com o usuário. Nesta camada o usuário vê o estado do modelo e pode manipular a interface, para ativar a lógica doenegócio;
  - Controlador (CONTROLLER): Transforma eventos gerados pela interface em ações de negócio, alterando o modelo.

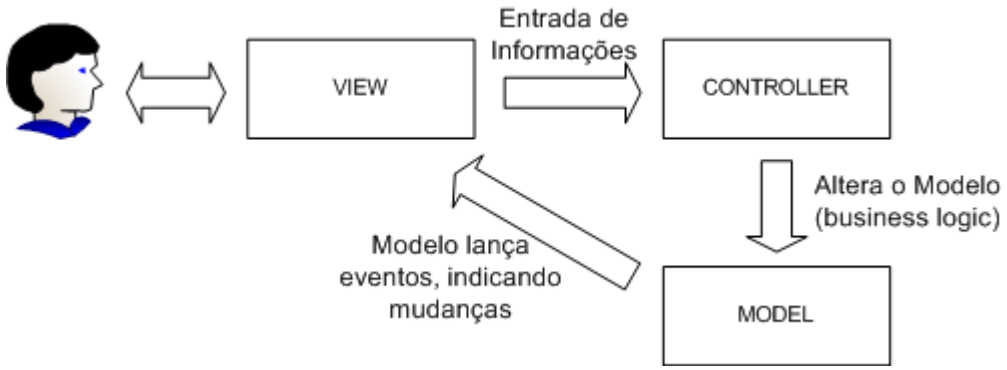

Figura 2: Fluxo de eventos e informações em uma arquitetura MVC

MVC - Como implementar (5 passos)

PASSO 1:

- Isole o "Business Logic" de seu sistema. Para pequenas aplicações, crie um pacote separado para armazenar as classes que representam o modelo do seu sistema.
- A camada "Model" pode estar armazenada em um SGBD, pode ser uma aplicação J2EE, ou até um simples pacote isolado contendo as classes de

negócio.

- 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 em nosso sisteminha de enquete!
- Classe `enquete.model.EnqueteSimples`:

```
public class EnqueteSimples {

	private Map opcoes;


	public EnqueteSimples(){
		opcoes = new HashMap();
	}

	/**
	 * 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 enquet
	 * @return Iterator opções disponiveis na enquete
	 */
	public Iterator getOpcoes(){
		return opcoes.keySet().iterator();
	}

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

	/**
	 * Retorna a soma dos votos de todas as opções da enqu
	 * @return int soma dos votos de todas as opções da en
```

```
         */
        public int getTotalVotos(){
              Iterator votos = opcoes.values().iterator();
              int total = 0;
              while(votos.hasNext()){
                    total+= ((Integer)votos.next()).intValue()
              }
              return total;
        }

        /**
         * Retorna a quantidade de votos de uma opção individu
         * @param opcao opção que se quer o voto
         * @return int quantidade de votos da opção
         */
        public int getVotos(String opcao){
              return ((Integer)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*"!
- Veja a figura 3:

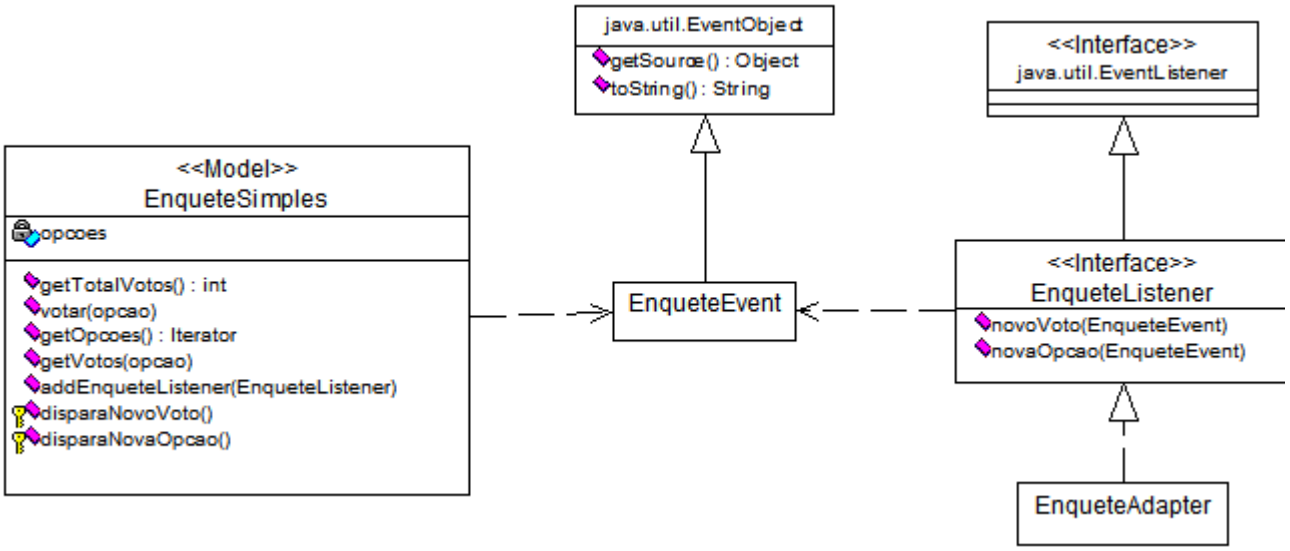

Figura 3: Modelo implementando padrão Observer

- Vejamos a nova classe `enquete.model.EnqueteSimples`:

```
public class EnqueteSimples {

	private Map opcoes;
	private List enqueteListeners = new LinkedList();

	public EnqueteSimples(){ (...) }

	/**
	 * 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);
	}

	public Iterator getOpcoes(){ (...)  }

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

	public int getTotalVotos(){
```

```
            (...)
      }

      public int getVotos(String opcao){ (...) }


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

      /**
       * Informa aos objetos interessados nos eventos lançad
       * pela Enquete que um novo voto foi contabilizado.
       */
      private synchronized void disparaNovoVoto(String opcao
            Iterator listeners = this.enqueteListeners.itera
            while(listeners.hasNext()){
                  ((EnqueteListener)listeners.next()).novoVo
            }
      }

      /**
       * Informa aos objetos interessados nos enventos lança
       * pela Enquete que uma nova opção foi adicionada.
       */
      private synchronized void disparaNovaOpcao(String opca
            Iterator listeners = this.enqueteListeners.itera
            while(listeners.hasNext()){
                  ((EnqueteListener)listeners.next()).novaOp
            }
      }

}
```

- Classe `enquete.model.EnqueteEvent`:

```
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 Enquet
	 * 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(op
	}

	/**
	 * Retorna o total de votos da enquete
	 * @return int
	 */
	public int getTotalVotos() {
		return ((EnqueteSimples)this.source).getTotalVot
	}

}
```

- Interface enquete.model.EnqueteListener:

```
public interface EnqueteListener extends EventListener {


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

```
	/**
	 * Invocado quando uma nova opção é adicionada à Enque
	 * @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:

```
public class TelaResultado extends Window implements Enquete

	private Map labels = new HashMap();

	public TelaResultado(Frame parent){
		super(parent);
		this.setSize(110,120);
		this.setLayout(new GridLayout(0,2)); // Grid com

		this.add(new Label("Votos"));
		this.add(new Label());

	}

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

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

}


public class TelaResultadoPercentual extends Window implemen

      private Map labels = new HashMap();


      public TelaResultadoPercentual(Frame parent){
            super(parent);
            this.setSize(180,120);
            this.setLayout(new GridLayout(0,2)); // Grid com

            this.add(new Label("Percentual"));
            this.add(new Label());
      }

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

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

}

public class TelaVotacao extends Frame implements EnqueteLis


	private Collection botoes = new Vector();

	private ActionListener controller;

	public TelaVotacao(ActionListener controller){
		super("Tela de Votação - Enquete");
		this.setSize(100,120);
		this.setLayout(new GridLayout(0,1)); // Grid com

		this.controller = controller;
	    this.addWindowListener(new WindowAdapter() {
	        public void windowClosing(WindowEvent e) {
	            e.getWindow().hide();
	            System.exit(0);
	        }
	    });
	}

	/**
	 * @see enquete.model.EnqueteListener#novaOpcao(Enquet
	 */
	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(Enquete
	 */
	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.
- No nosso 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`:

```
public class TelaVotacaoCtrl implements ActionListener{

	private EnqueteSimples enquete;

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

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

- Perceba que esta classe é fortemente ligada ao modelo.
- Algumas documentações sobre o MVC indicam que a classe controladora possui referências tanto da tela quanto do modelo. Essa abordagem "amarra" o

controlador à interface! Não queremos isso! (veja observações abaixo)

PASSO 5:

- Junte os pedaços de sua 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(enque

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

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

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

            // Adicionando as interfaces interessadas na mud
            // 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();
        }

}
```

Conseqüências:

- Alguns problemas que o MVC pode causar:
  - Se tivermos muitas visões e o modelo for atualizado com muita frequência, a performance do sistema pode ser abalada.
  - Se o padrão não for implementado com cuidado, podemos ter casos como o envio de atualizações para visões que estão minimizadas ou fora do campo de visão do usuário.
  - Ineficiência: uma visão pode ter que fazer inúmeras chamadas ao modelo, dependendo de sua interface.

Clique aqui para fazer o download do código fonte desta aula

MVC programa