FeMASS
FACULDADE PROF. MIGUEL ÂNGELO DA SILVA SANTOS

Projeto de Sistemas II

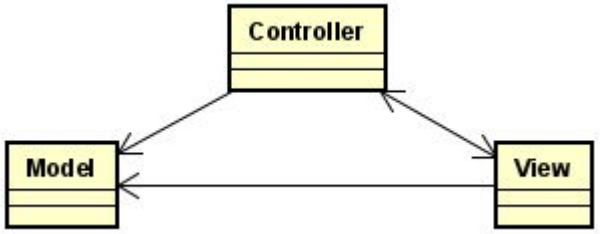
Faculdade Prof. Miguel Ângelo da Silva Santos

Material 5 - Padrão MVC

Professor: Isac Mendes Lacerda, M.Sc., PMP, CSM
e-mail: isac.curso@gmail.com

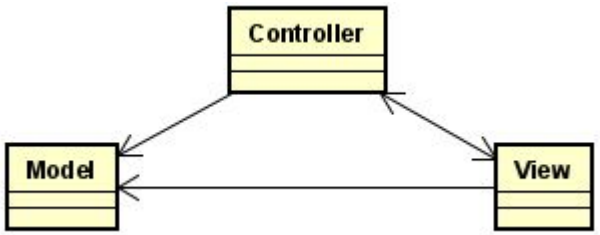
O que é e quando usar?

- ***Model-View-Controller*** é um padrão composto e também está incluído no livro do GoF.
- É **utilizado** para **implementar interfaces de usuários.**
- Diz respeito à **separação da aplicação em três** partes, interconectadas: ***Model***, ***View*** e ***Controller***.



Motivação

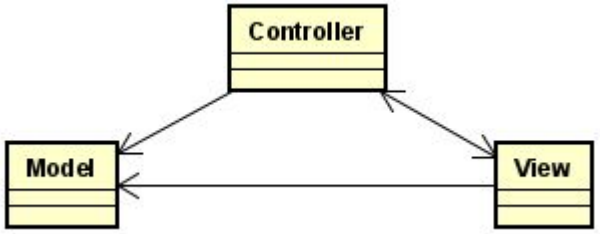
- **Melhora da manutenção** do código, através da **separação** da **informação armazenada** da **apresentada**.



Estrutura conceitual

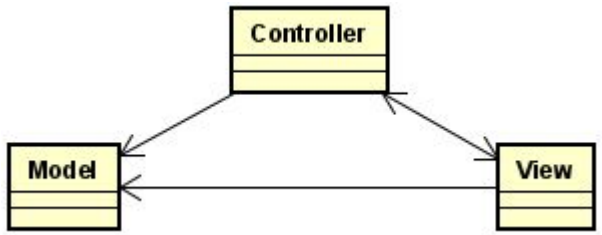
- A proposta consiste em **separar o dado** de negócio **e os seus métodos** de manipulação **da apresentação** desse mesmo dado.

- Inclui: **classe de *Model***, ***View*** e **classe *Controller*** para cada conceito importante.

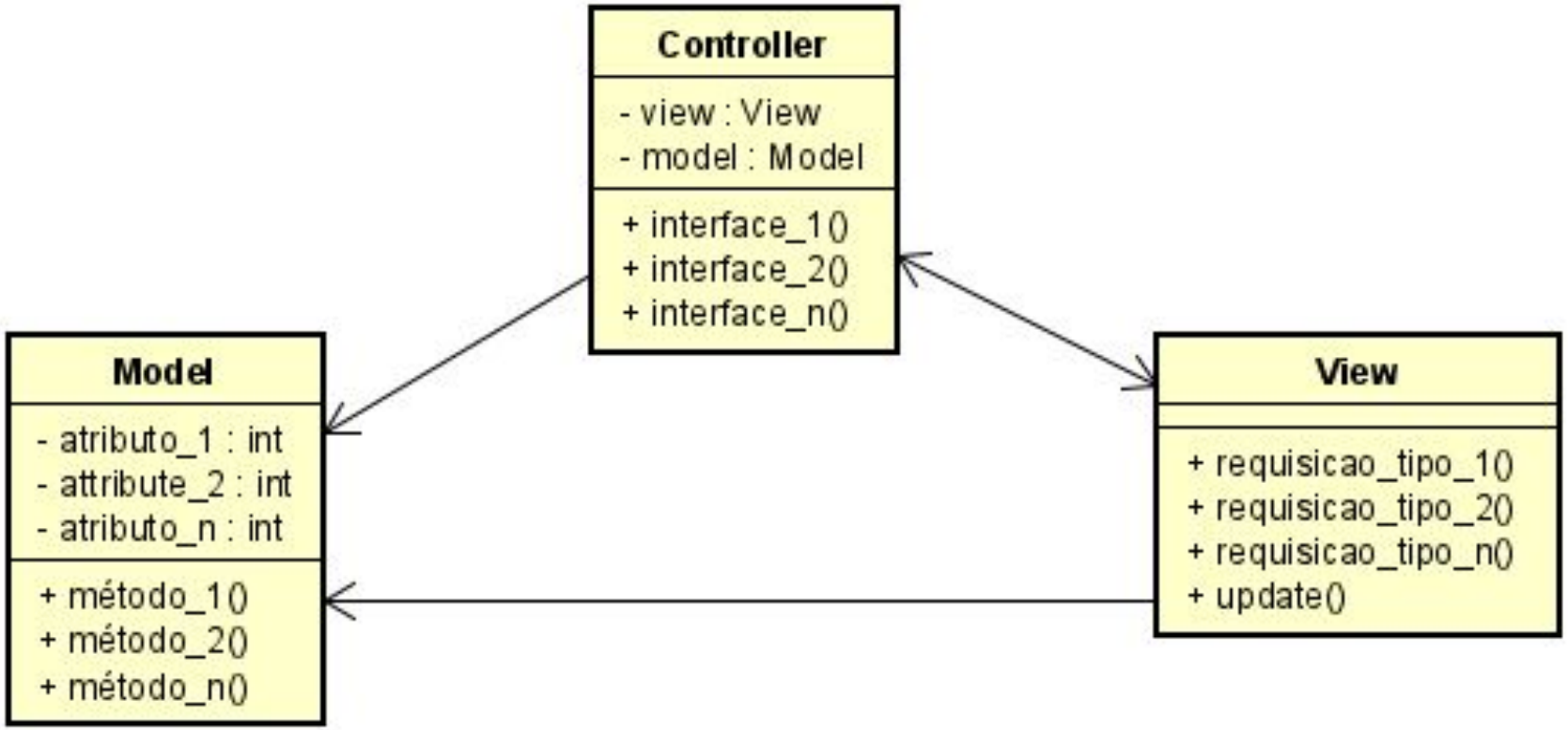


Estrutura conceitual

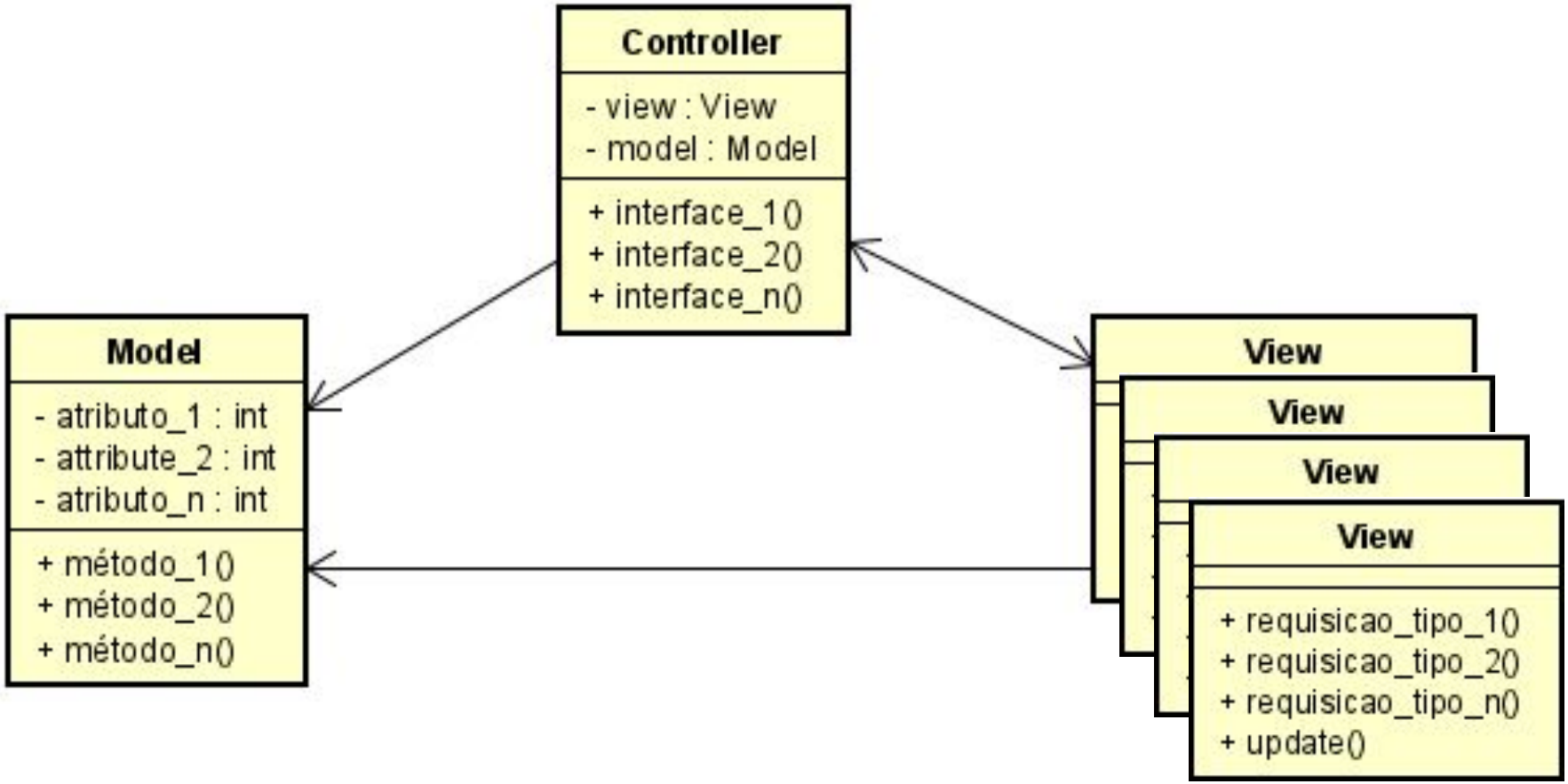
- ***Model*** : usada para **definir** todas as **operações** que ocorrem nos **dados**.
- ***View*** : é uma **representação da interface** do usuário. Ela terá métodos que ajudam a construir interfaces *web* ou GUI (*Guide User Interface*). **Não deve ter lógica**, mas **apenas exibição** dos **dados**.
- ***Controller*** : usada para **receber requisições**. **Encaminhar requisições** para outros.



Estrutura conceitual (genérica)



Estrutura conceitual (genérica)



Estrutura conceitual (*model*)

```
 6    package model;
 7
 8    public class Model {
 9        private int a1;
10        private int a2;
11        private int an;
12
13 +      public int get_a1() {...3 lines }
16 +      public int get_a2() {...3 lines }
19 +      public int get_an() {...3 lines }
22 +      public void set_a1(int a1) {...3 lines }
25    }
```

Estrutura conceitual (*controller*)

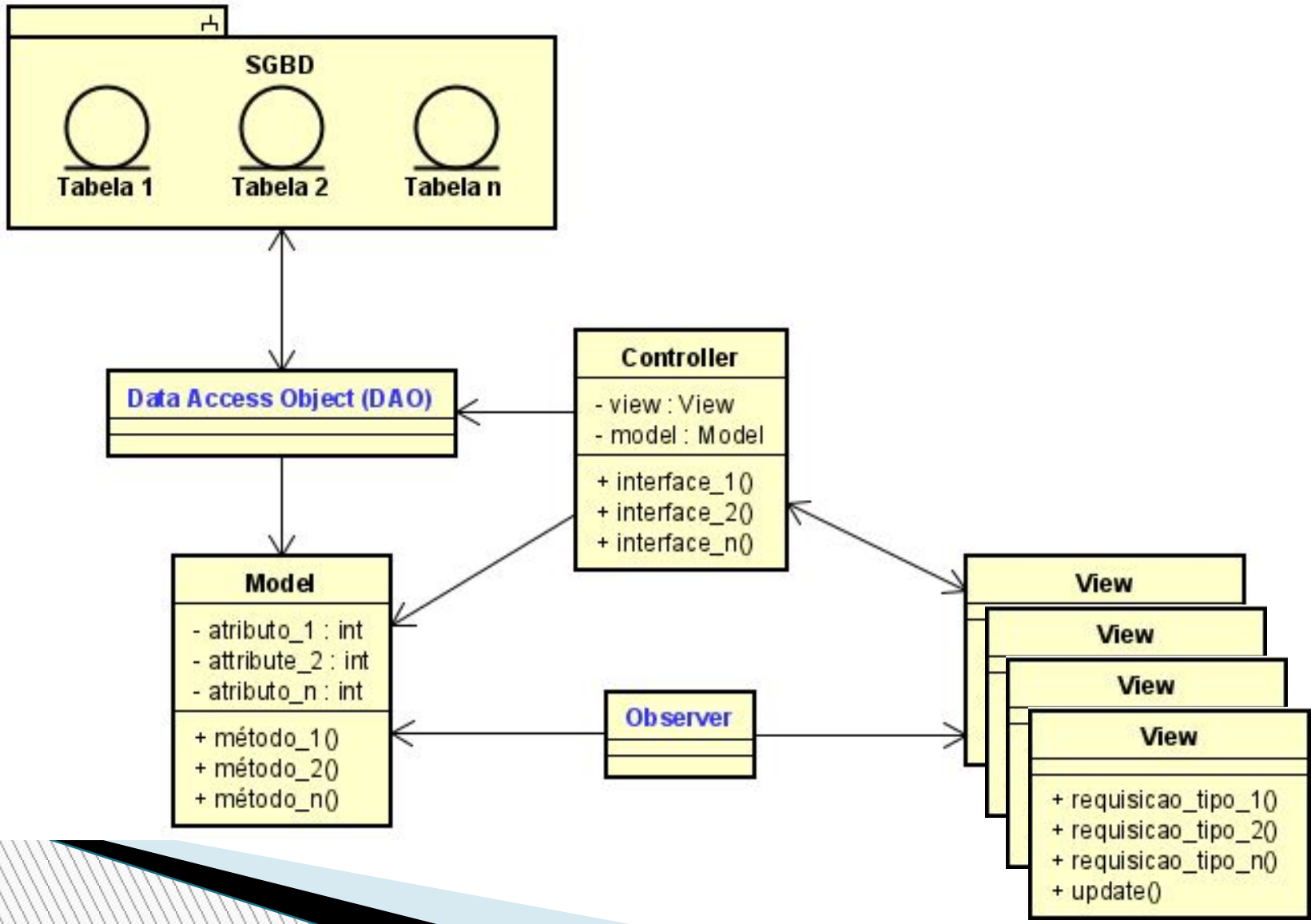
```
6   package controller;
7
8   import model.Model;
9   import view.View;
10  import java.util.*;
11
12  public class Controller {
13      public ArrayList<View> view = new ArrayList<>();
14      public Model model;
15
16      public Controller() {...4 lines }
20      public int interface_get_a1() {...3 lines }
23      public int interface_get_a2() {...3 lines }
26      public int interface_get_an() {...3 lines }
29      public void interface_set_a1(int v) {...3 lines }
32      public void nova_view() {...3 lines }
35
36  }
```

Estrutura conceitual (*view*)

```
 7  import controller.Controller;
 8
 9  public class View {
10      static int count = 0;
11      public Controller controller;
12
13      public View (Controller controller) {...4 lines }
17      public void req_get_a1() {...3 lines }
20      public void req_get_a2() {...3 lines }
23      public void req_get_an() {...3 lines }
26      public void req_set_a1(int v) {...3 lines }
29      public void update() {...6 lines }
35
36  }
```

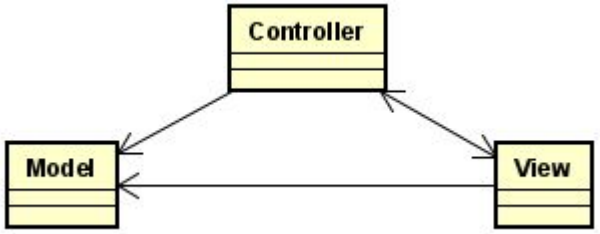
Demonstração no Netbeans...

MVC (composto com outros padrões)



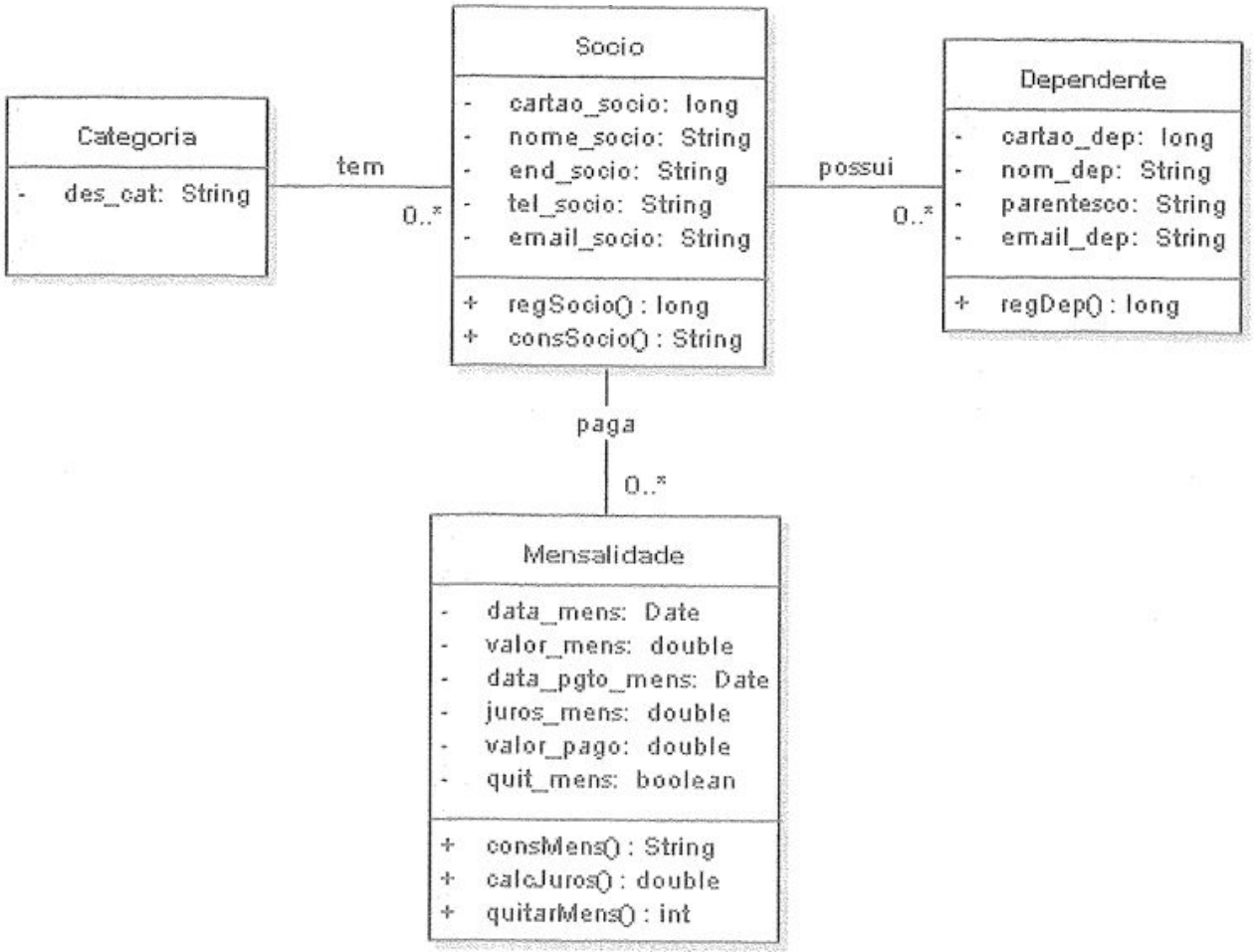
Vantagens e desvantagens

- A separação, **simplifica a manutenção**.
- Permite **alterações independentes** no ***frontend*** com poucas mudanças ou nenhuma no ***backend***.
- Favorece a **divisão de trabalho**.
- Os **modelos (*models*)** podem ser **alterados sem impactar as visões** (*views*).
- Os **controladores** (*controllers*) também podem ser **alterados sem impactar as visões e modelos**.
- Aumento do número de classes.



Exercício 1

Baseado nas classes de *model* abaixo, de um sistema hipotético de um clube, inclua e implemente as classes de *controller* e *view.* Com isso, instancie objetos de cada um dos conceitos, com o padrão MVC implementado completamente. **Nos três próximos slides seguem descrições para cada uma das classes.**



Exercício 1 (descrição de classes)

1. `Categoria`

   Essa classe representa as possíveis categorias de sócios estabelecidas pelo clube. Seu único atributo é a descrição da categoria.

2. `Socio`

   Essa classe armazena as informações referentes aos sócios do clube. Os atributos dessa classe são autoexplicativos. A classe tem dois métodos, um para registrar um sócio e outro para consultar um sócio específico.

   O método `regSocio` retorna um `long` que representa o número do cartão do sócio, e o método `consSocio` retorna uma `String` contendo os dados de um determinado cliente. Observe que um sócio pertence a uma categoria, mas uma categoria pode estar associada a muitos sócios.

Exercício 1 (descrição de classes)

3. Dependente

Essa classe armazena as informações referentes aos possíveis dependentes de um sócio. Os atributos da classe são autoexplicativos. O único método contido pela classe permite gerar uma nova instância da mesma. Pode-se perceber que um dependente está relacionado a um único sócio, mas um sócio pode não ter nenhum dependente ou pode ter vários.

Exercício 1 (descrição de classes)

4. `Mensalidade`

A classe em questão representa as mensalidades que devem ser pagas por cada sócio. Seus atributos são data da mensalidade e data em que a mensalidade foi efetivamente paga, do tipo `Date`, valor da mensalidade, juros da mensalidade e o valor efetivamente pago do tipo `double`, e um atributo denominado `quit_mens` do tipo `boolean`, que determina se a mensalidade foi quitada ou não. Já os métodos da classe são:

Método	Descrição
`consMens`	É disparado para consultar cada mensalidade ainda não paga de um determinado sócio. Retorna uma `String` com os dados da mensalidade.
`calcJuros`	Calcula os juros de uma mensalidade, no caso de esta estar atrasada. Retorna um `double` contendo o valor atual, após a aplicação de juros, da mensalidade.
`quitarMens`	Permite a quitação de uma mensalidade, retornando verdadeiro, se a operação foi concluída com sucesso, ou falso, se ocorreu algum problema quando se tentou quitar a mensalidade.