

Faculdade de Ciências e Tecnologia
Departamento de Matemática e Computação
Bacharelado em Ciência da Computação

unesp

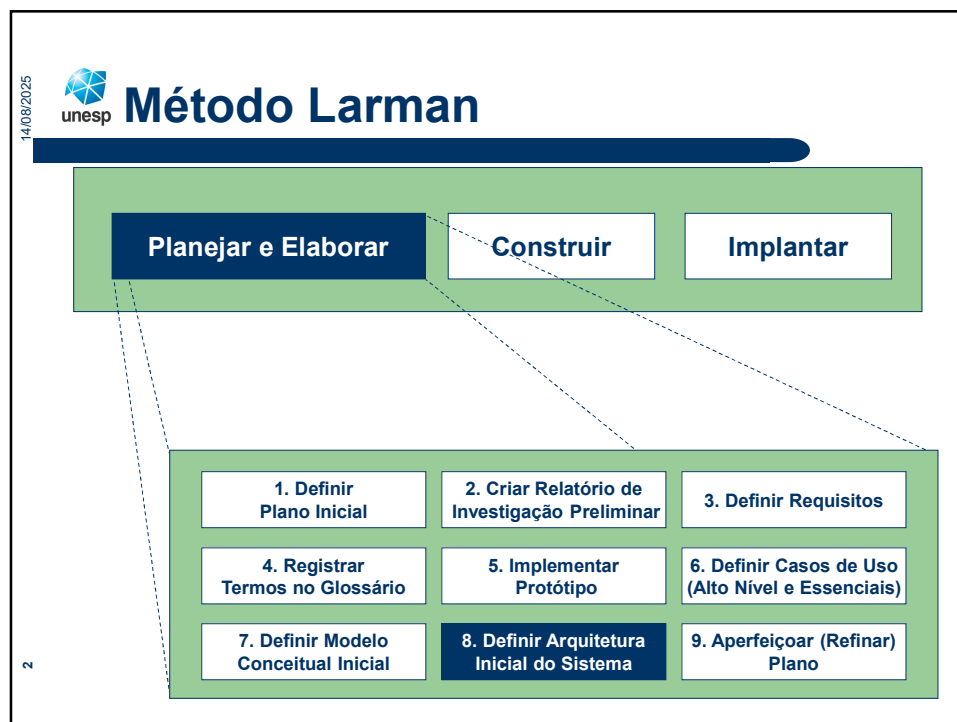
UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Engenharia de Software II

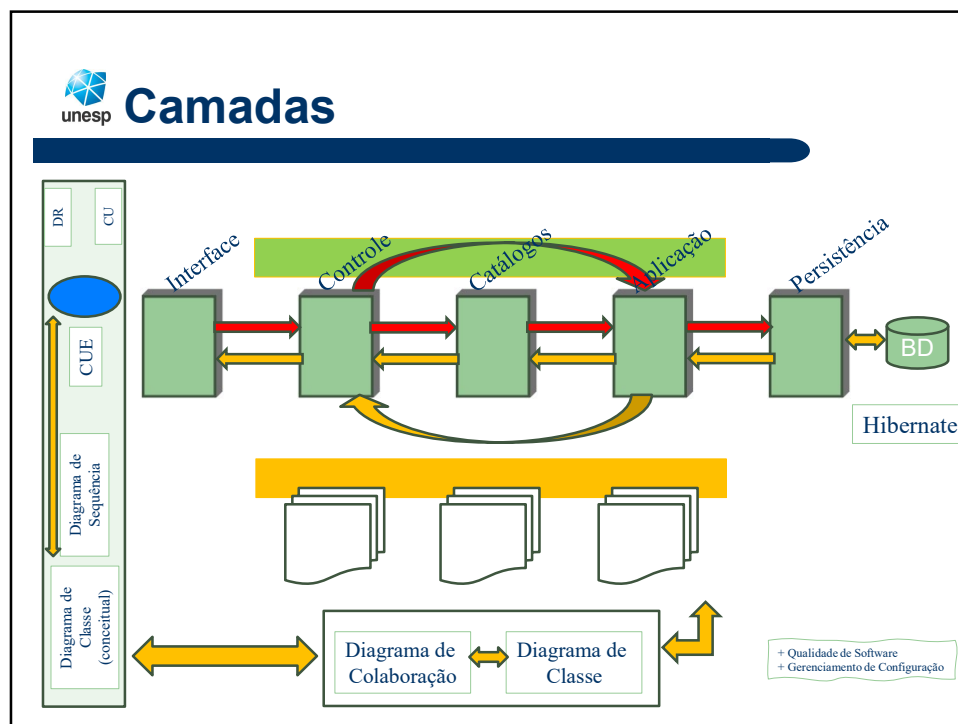
Aula 03

Prof. Dr. Rogério Eduardo Garcia
(rogerio.garcia@unesp.br)

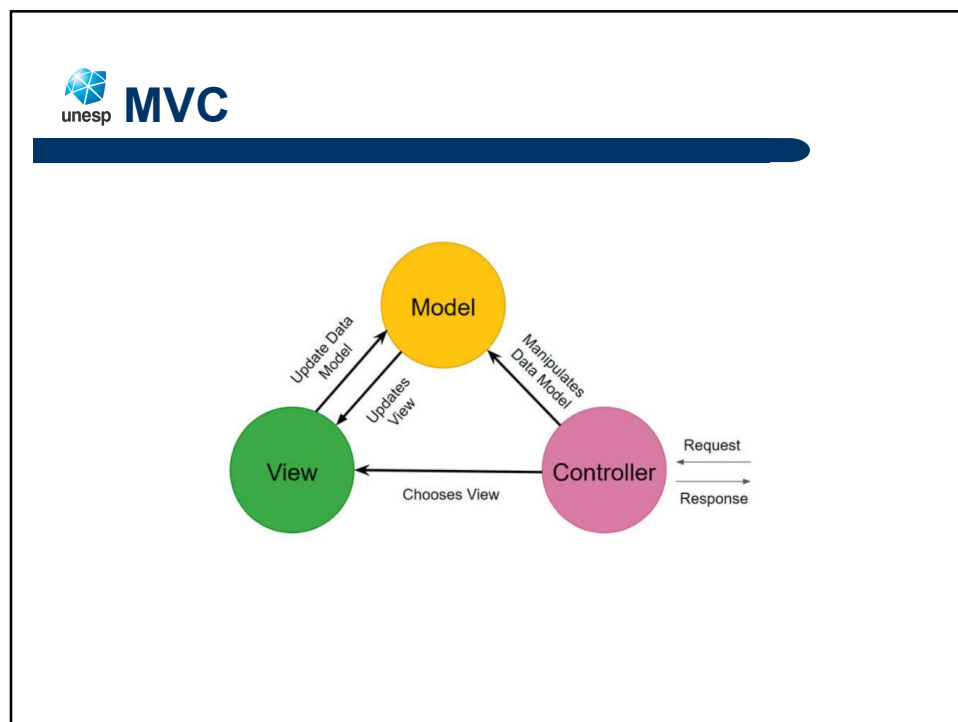
1



2




3



4

14/08/2025

 **MVC**


Prof. Dr. Rogério Eduardo Garcia

5

- O padrão Observer em Java é um padrão de projeto comportamental que permite que um objeto notifique outros objetos sobre alterações em seu estado. Ele estabelece uma relação um-para-muitos, onde um objeto (o sujeito/observable) mantém uma lista de dependentes (observadores) e notifica-os automaticamente quando ocorre alguma mudança.

5

14/08/2025

 **Padrão Observer**


Prof. Dr. Rogério Eduardo Garcia

6

- Subject/Observable:
 - O objeto que é observado e notifica os observadores sobre as mudanças em seu estado.
- Observer:
 - O objeto que se inscreve para receber notificações do Subject e reage às mudanças em seu estado.

6

14/08/2025

 **Funcionamento**


Prof. Dr. Rogério Eduardo Garcia

- **1. Inscrição:**
 - Os observadores se inscrevem no Subject para receber notificações.
- **2. Notificação:**
 - Quando o estado do Subject muda, ele notifica todos os observadores inscritos.
- **3. Atualização:**
 - Os observadores recebem a notificação e atualizam seu estado de acordo com as mudanças no Subject.
- **4. Remoção:**
 - Os observadores podem se desinscrever do Subject para parar de receber notificações.

7

7

14/08/2025

 **Vantagens**

Prof. Dr. Rogério Eduardo Garcia

- **Acoplamento flexível:**
 - O Subject e os Observadores ficam fracamente acoplados, permitindo que eles evoluam independentemente.
- **Facilidade de adição e remoção de Observadores:**
 - Novos observadores podem ser adicionados e removidos sem modificar o Subject.
- **Reutilização de código:**
 - O padrão Observer pode ser reutilizado em diferentes partes do código.

8

8

14/08/2025

unesp

Desvantagens

Prof. Dr. Rogério Eduardo Garcia

- **Vazamentos de memória:**
 - Se os observadores não forem removidos corretamente, podem ocorrer vazamentos de memória.
- **Sobrecarga**
 - Se houver muitos observadores, a notificação pode introduzir sobrecarga de desempenho.
- **Ordem de atualização:**
 - A ordem em que os observadores são atualizados pode ser imprevisível, o que pode causar problemas em alguns casos.

9

9

14/08/2025

unesp

Observer - Modelo

Prof. Dr. Rogério Eduardo Garcia

```
classDiagram
    class Subject {
        +Attach(Observer)
        +Detach(Observer)
        +Notify()
    }
    class ConcreteSubject {
        +GetState()
        +SetState()
        +subjectState
    }
    class Observer {
        +Update()
    }
    class ConcreteObserver {
        +Update()
        +observerState
    }
    Subject <|-- ConcreteSubject
    Observer <|-- ConcreteObserver
    Subject --> "many" Observer : observers
    Subject ..> Observer : for all o in observers { o->Update() }
    ConcreteSubject ..> Subject : return subjectState
    ConcreteObserver ..> ConcreteSubject : observerState = subject->GetState()
    ConcreteObserver ..> ConcreteObserver : observerState
```

The diagram illustrates the Observer design pattern. It features four classes: **Subject**, **ConcreteSubject**, **Observer**, and **ConcreteObserver**. **Subject** is an abstract class with methods `Attach(Observer)`, `Detach(Observer)`, and `Notify()`. **ConcreteSubject** inherits from **Subject** and implements `GetState()`, `SetState()`, and maintains a `subjectState` attribute. **Observer** is an abstract class with the `Update()` method. **ConcreteObserver** inherits from **Observer** and implements `Update()`, which calls `subject->GetState()` to update its `observerState`. The **Subject** class maintains a collection of **Observer** objects (labeled `observers`). The `Notify()` method iterates over this collection and calls `Update()` on each observer. The `ConcreteSubject` class has a `subjectState` attribute and a `GetState()` method that returns it. The `ConcreteObserver` class has an `observerState` attribute that is updated by calling `subject->GetState()` on the `subject` attribute.

10

10