Programação Orientada a Objetos

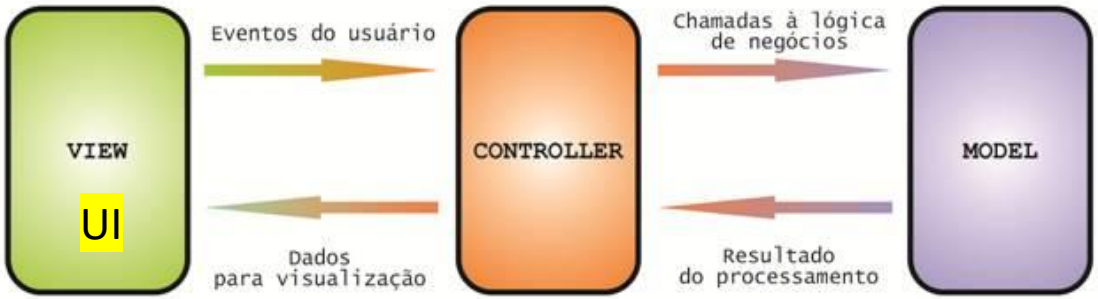
Prof. Dr. Josenalde Barbosa de Oliveira

josenalde.oliveira@ufrn.br

https://github.com/josenalde/apds

MODEL-VIEW-CONTROLLER

- Ideia de Trygve Reenskaug (Xerox PARC) em 1979 descrita em 1987 e depois 1992 por Steve Burbeck para projetos na linguagem SmartTalk-80. Linguagem com primeira versão em 1972 que introduziu na versão **80** conceitos de **POO** usados até hoje, incluindo o padrão MVC
- Por comparação, Java 1.0 surge em 1996 (Sun Microsystems), que é adquirida pela Oracle em 2008
- Ideia: 3 camadas independentes para reduzir acoplamento (dependências) e aumentar coesão (finalidade bem especificada) das classes do projeto

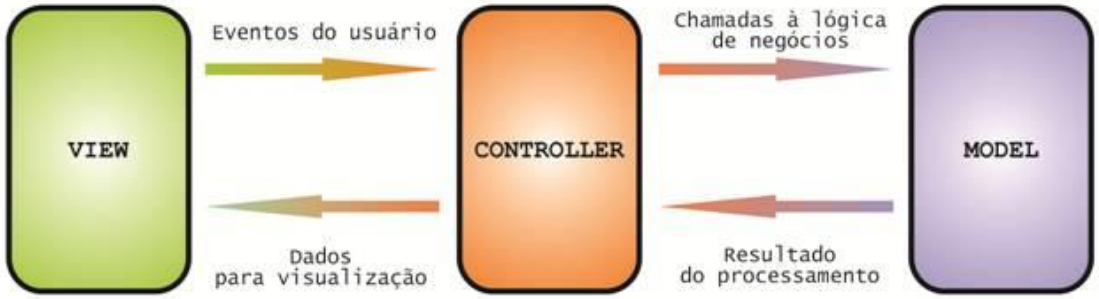


Fonte: https://www.devmedia.com.br/padrao-mvc-java-magazine/21995

MODEL-VIEW-CONTROLLER

Arquitetura de software

- No MODEL poderemos ter também o padrão **Data Access Object** (DAO) para separar classes de acesso ao Banco de Dados das classes de Regras de Negócio (RN, lógica). (Para se aprofundar ver também a API JPA)
- No MVC isolam-se regras de negócio da UI, permitindo UIs diferentes sem alterar as RN
- Pode ser aplicado em desktop, web, mobile...
- Comunicação entre UI e regras de negócio se dá pelas classes controladoras



Acoplamento: é o grau em que uma classe conhece a outra. Se o conhecimento da classe A sobre a classe B for através de sua interface, temos um baixo acoplamento, e isso é bom. Por outro lado, se a classe A depende de membros da classe B que não fazem parte da interface de B, então temos um alto acoplamento, o que é ruim. Coesão: quando temos uma classe elaborada de forma que tenha um único e bem focado propósito, dizemos que ela tem uma alta coesão, e isso é bom. Quando temos uma classe com propósitos que não pertencem apenas a ela, temos uma baixa coesão, o que é ruim.

Fonte: https://www.devmedia.com.br/padrao-mvc-java-magazine/21995

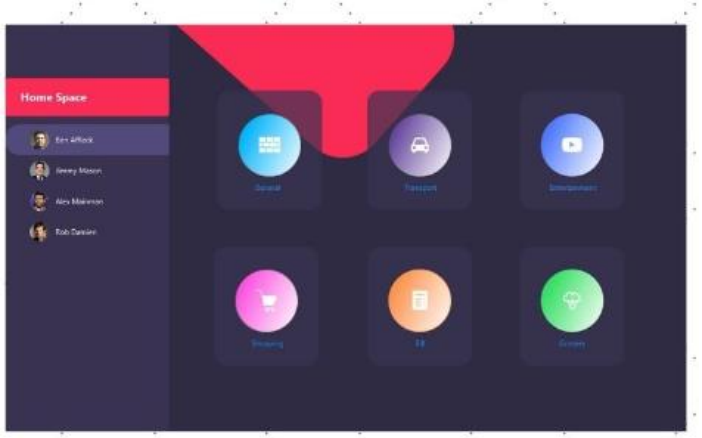
- Model possui a lógica da aplicação: RNs, persistência/crud e classes de entidades

MODEL-VIEW-CONTROLLER

Arquitetura de software

- Na VIEW (camada de visualização/apresentação) há a interação com o usuário (entrada e saída)
- Em Java/Desktop (swing, awt, swt, javafx (fxml) etc.)
- Em Web (html, jsp, jsf, js, css, bootstrap, thymeleaf etc.)

- No CONTROLLER, as requisições do usuário são direcionadas pelo CONTROLLER para o MODEL. Se o MODEL gera respostas, envia para o CONTROLLER e este para a VIEW

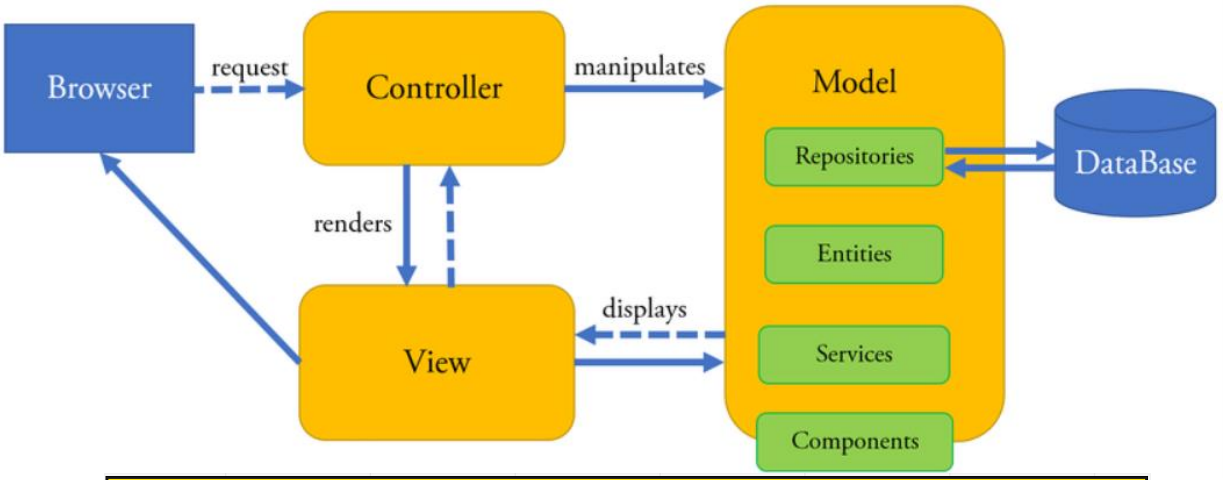


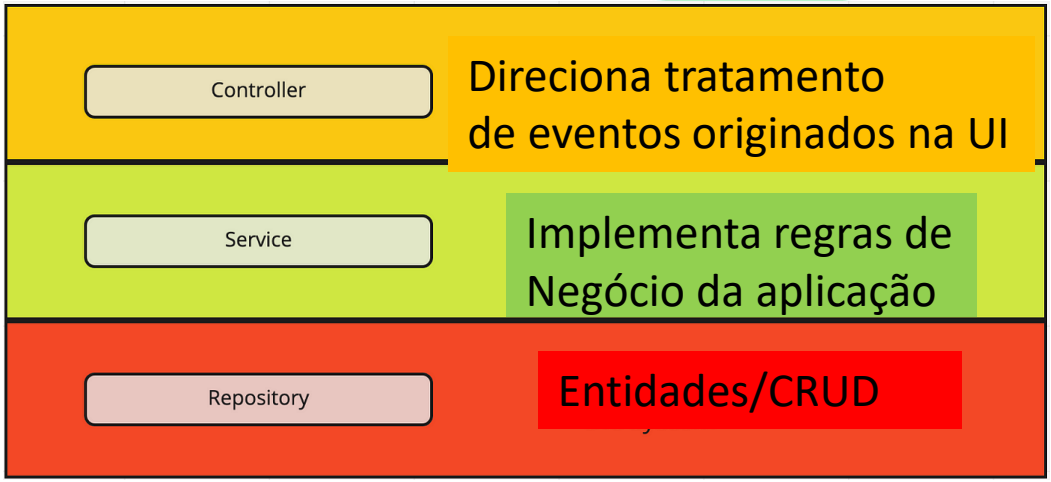
- Com MVC, a manutenção e atualização se torna mais fácil; os testes podem ser isolados por camada (salvo testes de integração); requer do desenvolvedor maior atenção na etapa de modelagem e posterior implementação
- No MODEL, a comunicação com a base de dados pode utilizar a API Java Database Connectivity (JDBC)

MODEL-VIEW-CONTROLLER

Arquitetura de software

- É uma boa prática criar ao menos mais três pacotes internos ao pacote MODEL
- Entidades: representa uma tabela do BD, suas referências e dependências. Em Java, uma classe JavaBean (POJO: Plain Old Java Objects) tem um construtor, atributos privados, getters e setters
- No pacote DAO colocamos classes relacionadas ao CRUD e acesso ao BD, para cada entidade. Com API também é chamada REPOSITORY.
- No pacote SERVICE, são classes com as regras de negócio. Acoplamento entre a classe DAO e a classe de entidade.

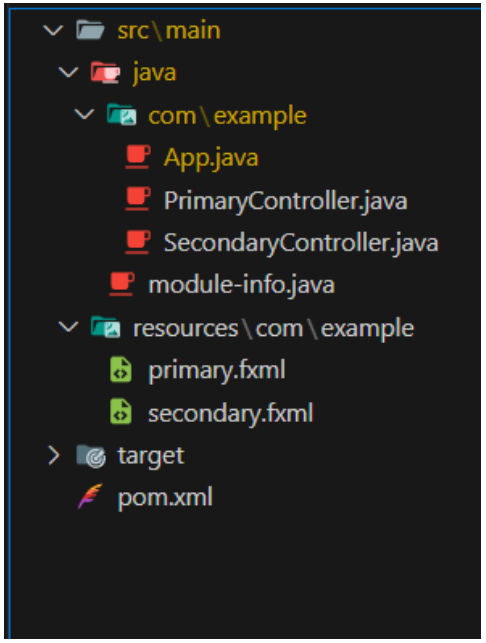




MODEL-VIEW-CONTROLLER

Exemplo Introdutório
com JavaFX no VSCODE

- Ctrl + Shift + P: Create Java Project -> JavaFX (com Maven) – adiciona .jars das dependências
  - Group ID (com.example) – br.ufrn.eaj.tads.poo (este será o nome do package principal)
  - Artfact ID (demo) – nome do projeto – test
  - Version 1.0 (pode variar)
  - *No src\main*
    - *\java*
      - *Group ID*
    - *\resources (telas do Scene Builder) – equivale a \view*
    - *\target (binários gerados)*
    - *module-info.java, pom.xml*



MODEL-VIEW-CONTROLLER

Exemplo Introdutório
com JavaFX no VSCODE

- *No src\main*
  - *\java*
    - *tecinfo\poo*
      - *controller*
      - *model*
        - *dao (repository)*
        - *entity*
        - *service*
      - *App.java*
  - *\resources (telas do Scene Builder) – equivale a \view*
  - *\target (binários gerados)*
  - *module-info.java, pom.xml*