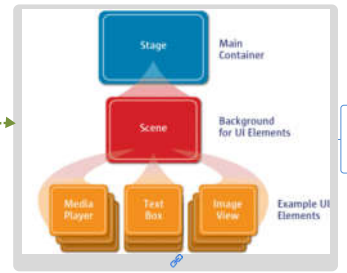

É como uma peça de teatro : `Stage` é o contêiner principal que geralmente é `Window` com borda e o típico botão minimizar, maximizar e fechar. Dentro do `Stage` você adiciona um `Scene` que pode, é claro, ser trocado por outro `Scene`. Dentro dos `Scene` nós JavaFX reais `AnchorPane`, como `TextBox`, etc. são adicionados.

Exemplo

Camdas MVC (JavaFX)

MVC

VIEW
- VIEW-MAIN (*.java)
  - DockletListener
  - Stage
  - Scene
  - Tela
- FXML (*.fxml)
  - Scene Builder
- .CSS
- UI
- Subtópico 6
- Telas
- Front End

CONTROLLER
- Pega os dados da VIEW e passa para o DTO
- Pega o DTO e passa para a View

MODEL
- DAO ou DAL
  - Data Access Object
  - Data Access Layer
  - SQL
    - Transações Begin, Commit e Rollback
  - CRUD
  - JDBC
    - Camada de Conexão com o banco
    - Controla as Transações SQL
- BLL
  - Business Logic Layer
  - Regras de negocio
  - Validações
  - Tratamento de dados
- DTO
  - Data Transfer Object
  - Modelos
  - Camada de Domínio
  - Entidades

Transação SQL

```
CREATE TABLE ValueTable (id int);
BEGIN TRANSACTION;              -- aqui começa a transação
        INSERT INTO ValueTable VALUES(1);
        INSERT INTO ValueTable VALUES(2);
COMMIT;                         -- aqui termina e "grava" tudo
```

Exemplo Geral

DAL

```
public class AlunoBanco
{
    public void InserirAluno(Aluno aluno){
        ...
        "INSERT INTO ALUNO (aluno)"
        ....
    }
}

public class Aluno
{
    public string ra {get;set;}
    public string nome {get;set;}
}
```

BLL

```
public void InserirAluno(Aluno aluno){
    DAL.InserirAluno(aluno);
}
```

Apresentação

```
protected void Salvar_Click(object sender, EventArgs e)
{
    var aluno = new Aluno();
    aluno.ra = ra.Text;
    aluno.nome = nome.Text;
    BLL.InserirAluno(aluno);
}
```

Vantagens do Modelo MVC

1 - Como o modelo MVC gerencia múltiplos views usando o mesmo modelo é fácil manter, testar e atualizar sistemas compostos;
2 - É muito simples adicionar novos clientes apenas incluindo seus views e controles;
3 - Torna a aplicação escalável;
4 - É possível ter desenvolvimento em paralelo para o modelo, visualizador e controle pois são independentes;
5 - Facilita o reuso do código;
6 - Melhor nível de sustentabilidade, pois facilita a manutenção da aplicação;
7 - Melhor performance, graças a separação em camadas;
8 - Fácil transformação da interface, sem que haja necessidade de modificar a camada de negócio;
9 - Melhor desempenho e produtividade, graças a estrutura de pacotes modulares;
10 - A arquitetura modular permite aos desenvolvedores e designers desenvolverem em paralelo;
11 - Partes da aplicação podem ser alteradas sem a necessidade de alterar outras.

Autor: andre.lucyo2@gmail.com