Universidade Aberta

Licenciatura em Engenharia Informática

Laboratório de Desenvolvimento de Software

Ano Letivo: 2022/2023

Relatório final da aplicação demonstradora

**Aplicação de análise e edição de imagens com recurso à framework Aforge.NET**

**Equipa:**

Isabel Carreiro, n.º 2000709

Luís Santos, n.º 1900607

Vítor Carvalho, n.º 1401892

3 de junho de 2023

**Índice**

[Introdução 1](#_Toc135677070)

[Endereço do repositório Git 1](#_Toc135677071)

[Descrição geral do projeto 1](#_Toc135677072)

[Funcionamento esquemático 2](#_Toc135677073)

[Divisão de responsabilidades pelos componentes 5](#_Toc135677074)

[Recursos a eventos para acoplamento fraco 5](#_Toc135677075)

[Emprego de exceções respeitando o estilo MVC 9](#_Toc135677076)

[Uso de interfaces para independência entre componentes 11](#_Toc135677077)

[Reflexão final 13](#_Toc135677078)

[Bibliografia 14](#_Toc135677079)

# Introdução

Na Unidade Curricular de Laboratório de Desenvolvimento de Software foi proposto a cada grupo o desenvolvimento de um projeto em linguagem C#, e que fizesse uso de uma API, entre as sugeridas pelo Professor. Ao longo do projeto foi-nos solicitada a aplicação de boas práticas de engenharia do Software, à medida que diferentes orientações nos foram apresentadas no fórum da UC. O nosso grupo, C#1, propôs-se a desenvolver uma aplicação de análise e edição de imagens com recurso à framework Aforge.NET, em que foram implementadas as seguintes funcionalidades:

* Criar uma ferramenta de estatística RGB de uma imagem (png, jpg, etc.);
* A interface da aplicação será uma linha de comandos que terá como input o comando que se pretende e o caminho para a imagem;
* O output gerado será a estatística em percentagem dos valores RGB.

# Endereço do repositório Git

Este relatório apresenta o projeto da aplicação demonstradora da API, desenvolvida por pela equipa *C#1*, composta pelos funcionários Isabel Carreiro, Luís Santos e Vítor Carvalho, no âmbito da empresa SimProgramming. O projeto está disponível no repositório git GitHub, no seguinte endereço:

[**https://github.com/lightningspirit/LDS2023**](https://github.com/lightningspirit/LDS2023)

# Descrição geral do projeto

Para implementar este projeto, procurámos aplicar princípios de engenharia de software que nos foram apresentados ao longo do semestre no fórum desta Unidade Curricular.

A nossa equipa usou a API AForge.NET, principalmente na parte de processamento de imagem, com o objetivo de criar uma ferramenta de estatística RGB de uma imagem (png, jpg, etc.).

A interface da aplicação será em linha de comandos que terá como input o comando que se pretende e o caminho para a imagem. O output gerado será a estatística em percentagem dos valores RGB.

Esta ferramenta tem como potencial mercado a integração com aplicações existentes de transformação de imagens (exemplo: geração de gradientes para lazy-load através da análise de estatística) e empresas de multimédia que necessitem de uma ferramenta deste género durante o seu processo de trabalho.

# Funcionamento esquemático

Para o desenvolvimento desta aplicação, optámos pela variante MVC proposta por Krasner & Pope, que processa no Controller os *inputs* do utilizador e não prevê comunicação da View para o Controller. O Diagrama seguinte mostra as responsabilidades de cada componente e as mensagens trocadas entre eles.

Mensagens no estilo arquitetónico MVC por Krasner & Pope:

**Diagram

Description automatically generated**

Inicialmente, modelamos a nossa aplicação com a variante MVC proposta por Krasner & Pope. No entanto, antes de fazermos um novo diagrama de sequência que seguisse a variante Curry & Grace, atualizámos o diagrama de sequência anterior pois, após termos feito uma implementação inicial do código, detetámos alguns problemas com a versão inicial do diagrama, nomeadamente:

* Numa situação de erro, o programa não deve terminar abruptamente. Deve ser dada oportunidade ao utilizador para corrigir o erro, quando possível. Nesse sentido, não fazia sentido manter os códigos Exit 1, 2 e 3;
* Não faz sentido haver um construtor de Controller a receber o comando como parâmetro visto que antes do utilizador inserir o comando, já teve de haver uma instanciação do Controller para que as instruções do programa sejam impressas;
* Não faz sentido a existência da função FornecerResultados. O Controller irá chamar a função ExecutarComando que irá aplicar a operação desejada sobre a imagem cujo path foi também indicado. Da forma como tínhamos feito inicialmente, esses resultados teriam de ficar guardados algures no Model para que posteriormente quando o Controller chamasse FornecerResultados estes fossem retornados. Na nova versão, a função ExecutarComando executa a ação sobre a imagem e, se a operação for bem-sucedida, a View apresenta de imediato os resultados.

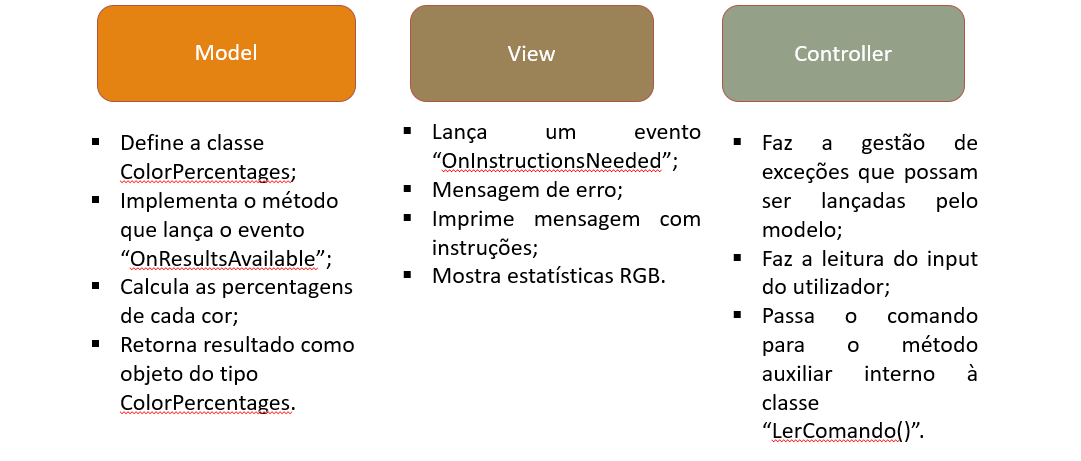
Desenhamos o Diagrama de Sequência entre as componentes usando este estilo arquitetónico, que se encontra a seguir:

Uma imagem com diagrama

Descrição gerada automaticamente

# Divisão de responsabilidades pelos componentes

Para o desenvolvimento da nossa aplicação, uma das etapas foi a divisão das responsabilidades pelos componentes do estilo arquitetónico MVC, segundo a variante que escolhemos: Krasner & Pope.

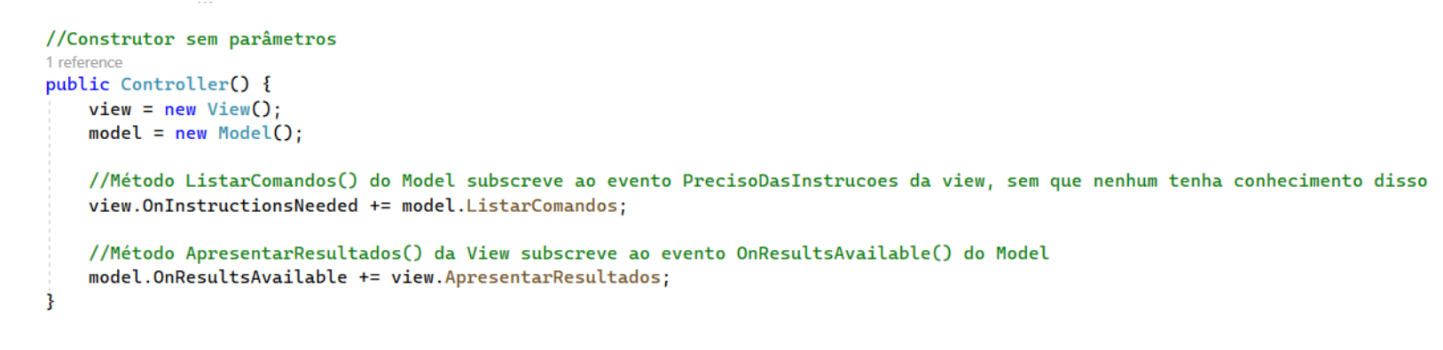


# Recursos a eventos para acoplamento fraco

Na implementação preliminar da aplicação, identificaram-se os seguintes pontos de acoplamento entre os componentes:

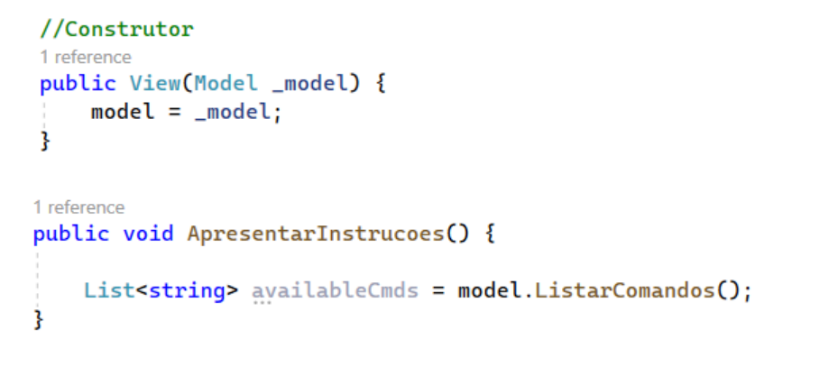
* No Controller:
  + Controller com View: view.ApresentarInstrucoes();
  + Controller com View: view.ImprimirMensagemErro(excp.Message);
  + Controller com Model: model.ValidarComando(comando, ref cmd, ref path);
  + Controller com Model: model.ExecutarComando(cmd, path).
* Na View:
  + View com Model: model.ListarComandos().
* No Model:
  + Model com View: view.ApresentarResultados().

A equipa decidiu implementar comunicação por eventos nos pontos de acoplamento View-Model (model.ListarComando()) e Model-View (view.ApresentarResultados()).

A subscrição dos eventos é feita no Controller:

**Exemplo 1 - Acoplamento View-Model, a View tem o método ApresentarInstrucoes() que chama o método ListarComando() do Model:**

* Código inicial com comunicação por invocação de métodos:

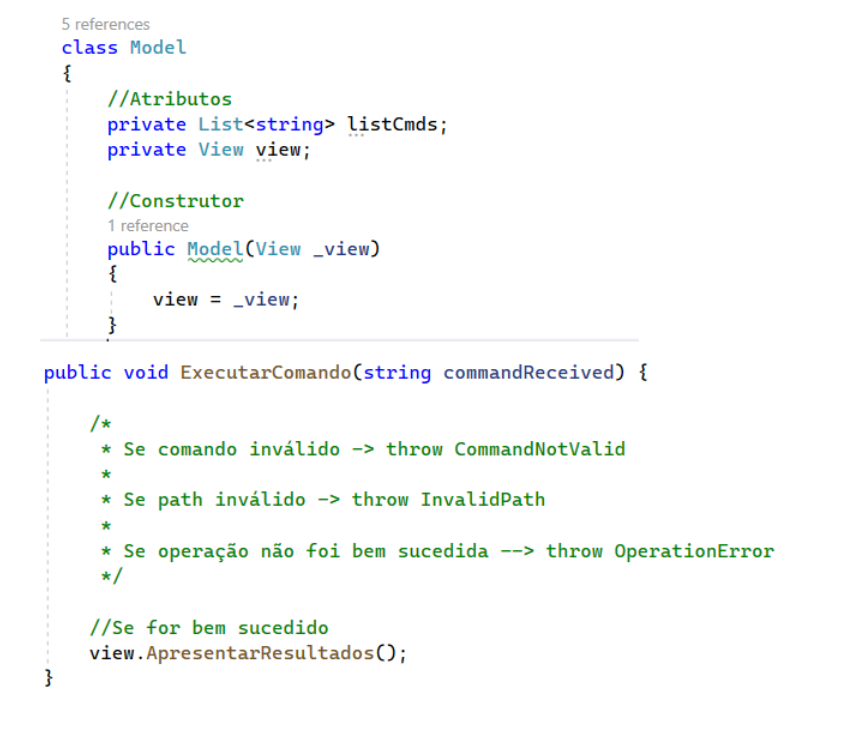
****

* Código após implementação da comunicação por eventos:

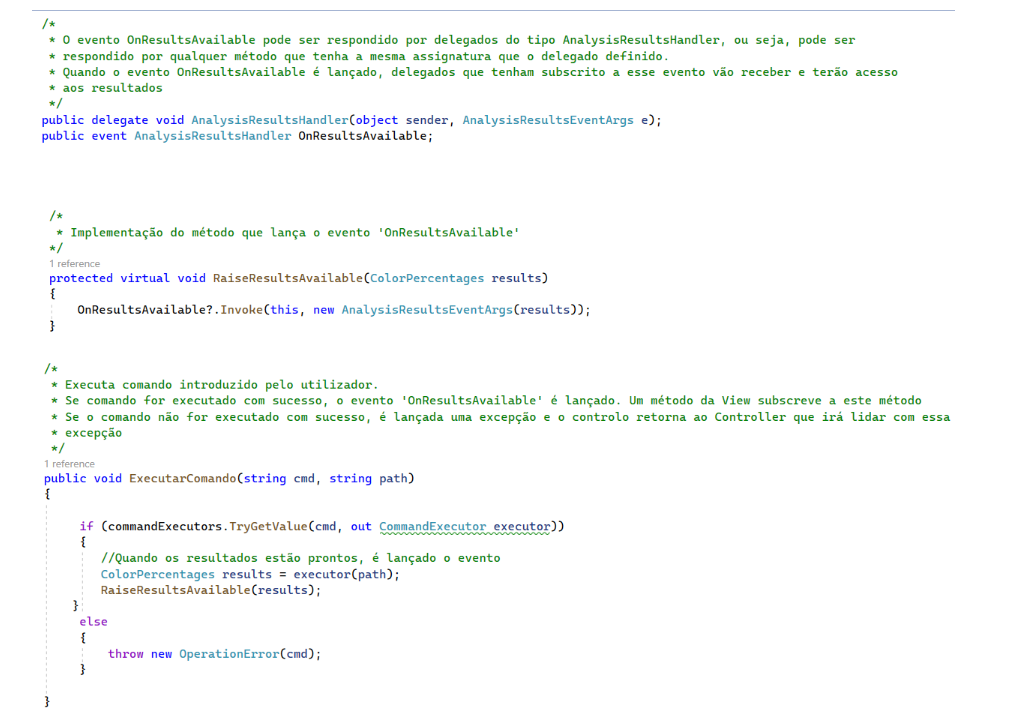


**Exemplo 2 - Acoplamento Model-View, o Model tem o método ExecutarComando() que chama o método da view ApresentarResultados():**

* Código inicial com comunicação por invocação de métodos:



* Código após implementação da comunicação por eventos:



Para além disso, o Model deixou de ter uma instância de View como Atributo. Desta forma o seu construtor segue o protótipo Model() em vez de Model(View).

# Emprego de exceções respeitando o estilo MVC

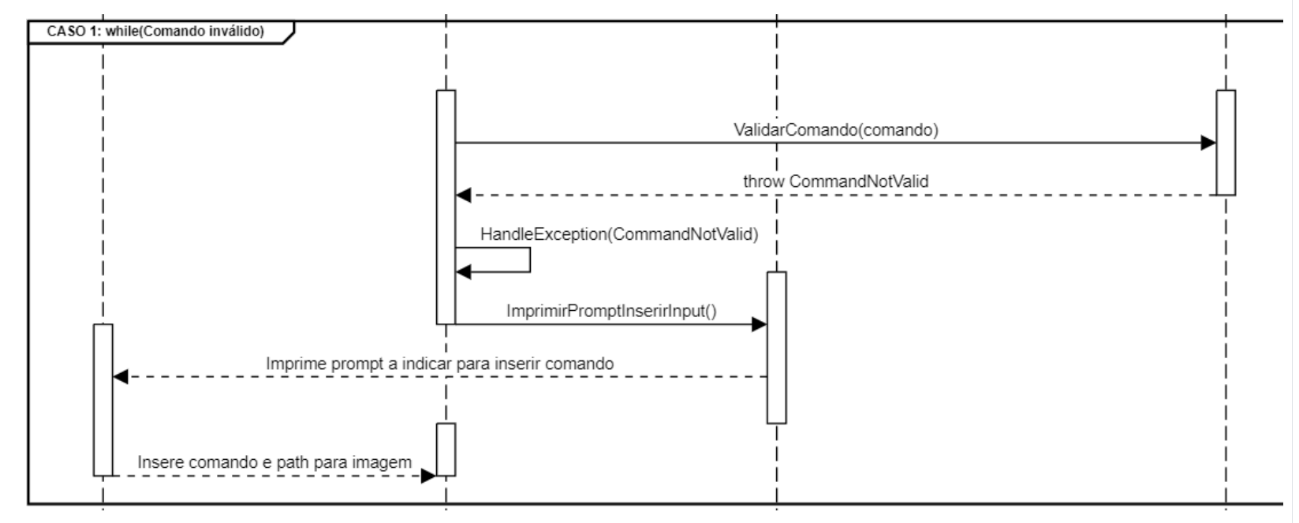
O diagrama inicial que a equipa tinha submetido no e-fólio A já contemplava exceções. No tópico 3 a equipa fez algumas alterações neste sentido que vieram provar estar de acordo com o que foi pedido neste tópico, por exemplo, inicialmente tínhamos previsto que quando o utilizador inseria um comando inválido a execução do programa terminava de imediato, sem dar a oportunidade de corrigir. Tal como referido nos Diaporamas 5-6, uma boa gestão de erros deve dar a oportunidade ao utilizador de corrigir o input errado.

Definimos quatro tipo de exceções que podem ocorrer:

* CommandNotValid;
* EmptyCommandArguments;
* InvalidPath;
* OperationError - nesta última, visto que a exceção ocorreu devido a um erro interno ao sistema, não damos a oportunidade ao utilizador de voltar a inserir um novo comando.

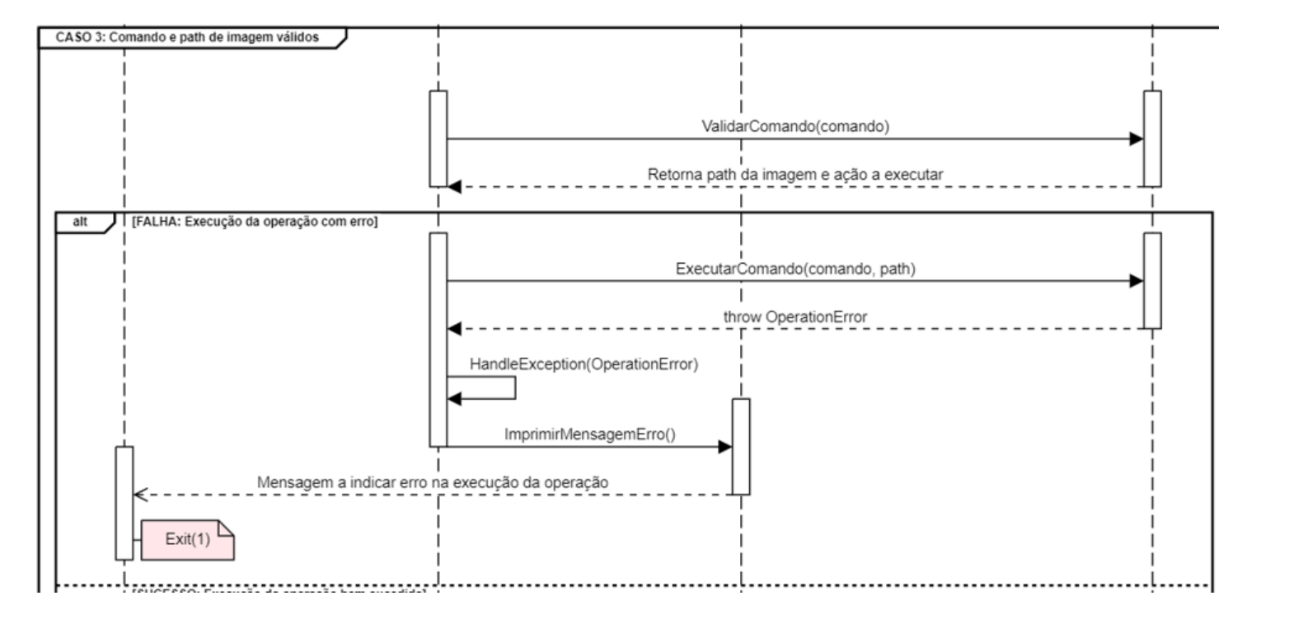


Por exemplo, se analisarmos o fragmento do diagrama de código que modela o que ocorre quando o comando inserido não é válido:

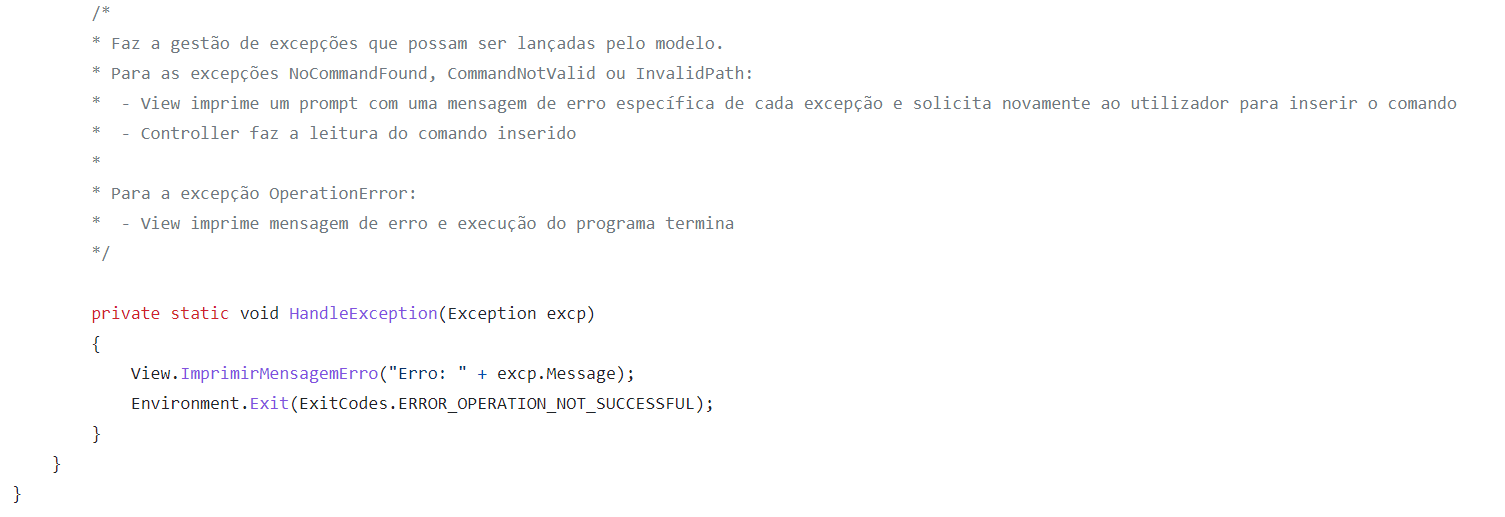


O utilizador insere um comando que o Controller passa ao método ValidarComando() do Model. Caso esse comando seja inválido, o Model lança uma exceção CommandNotValid, que será lidada pelo Controller. A forma como o Controller lida com isso é chamando um método da View que imprime uma mensagem a indicar a ocorrência de erro e solicita ao utilizador para voltar a tentar.

Em relação à situação em que há um erro interno da aplicação:



O utilizador insere um comando válido, sendo que esse comando e path para imagem validados são passados como argumento ao método ExecutarComando() do Model pelo Controller. No entanto, durante a execução há um erro e o Model lança uma exceção OperationError() que será lidada pelo Controller. A forma que o Controller tem para lidar com essa exceção é chamar um método da View que imprime mensagem de erro e faz saída do programa com um código de erro.

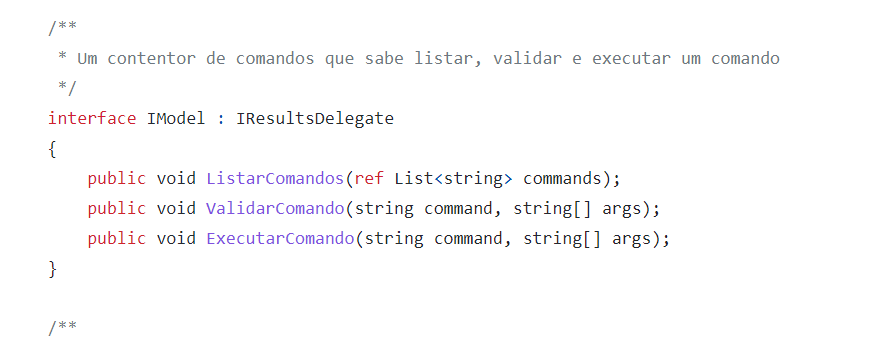


# Uso de interfaces para independência entre componentes

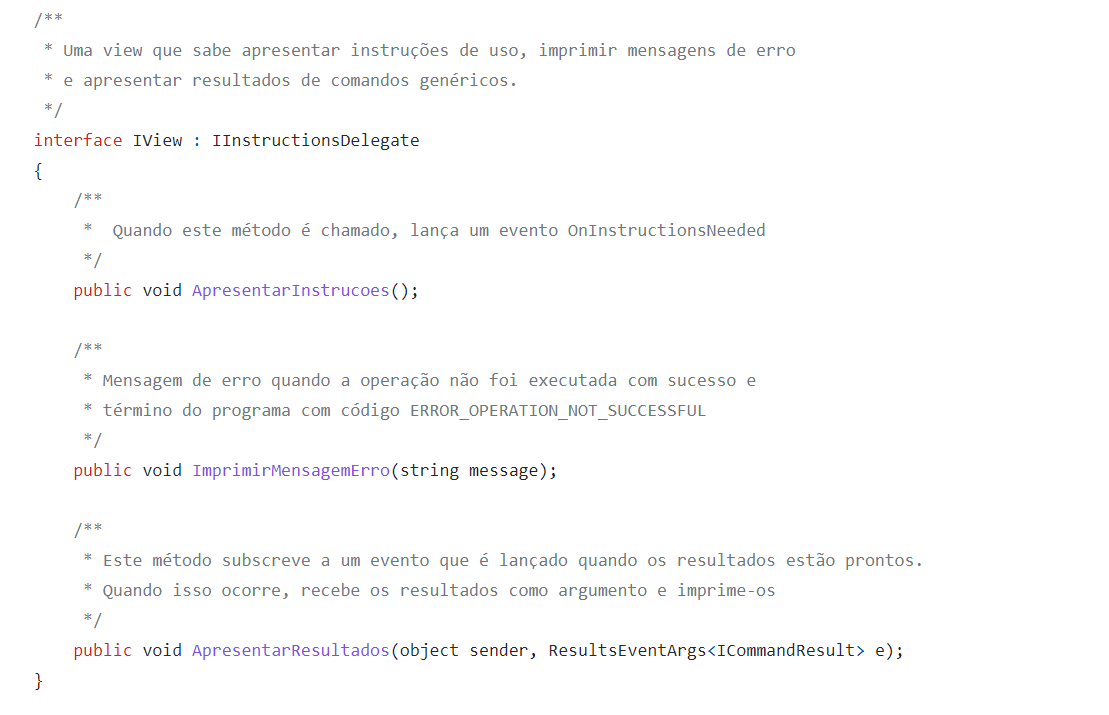
A criação de interfaces iModel e iView permitem que a aplicação use diferentes tipos de aplicação do Model e View, caso seja necessário. As interfaces servem de documentação e contrato entre os vários componentes.

Na nossa API foram identificadas cinco dependências entre componentes.

**Definição da interface iModel:**



**Definição da interface iView:**



# 

# Reflexão final

Fazendo um balanço ao trabalho realizado, nomeadamente, às aprendizagens desenvolvidas e às dificuldades sentidas, consideramos que conseguimos cumprir com as funcionalidades a que nos propusemos no início do projeto, o que já por si é um motivo de satisfação e de objetivo cumprido. No entanto, temos também que reconhecer que existem aspetos que poderiam ser melhorados e que infelizmente não o conseguimos fazer dentro do prazo estabelecido para a entrega final do projeto. Poderíamos ter efetuado um comando para tornar a imagem “turva” e criado um intervalo de luminosidade para definir as cores, por exemplo, red (R: 100-255; G: 0-100; B: 0-100), green (R: 0-100; G: 100-255; B: 0-100) e blue (R: 0-100; G: 0-100; B: 100-255).

Por último gostaríamos de referir que o desenvolvimento deste projeto nos trouxe grandes desafios e a descoberta do mundo das API’s. Consideramos que as aprendizagens que realizamos sobre o modelo MVC foram bastante significativas, assim como sobre a API com que trabalhamos. Contudo, não podemos deixar de referir que, o desenvolvimento do trabalho tornou-se complexo porque na nossa equipa existiam elementos que não tinham experiência de trabalho com API’s.

Consideramos que colocar em prática os princípios de engenharia de software abordados nesta UC e em simultâneo lidar com uma API, nunca utilizada por alguns elementos da equipa, foi uma tarefa muito desafiante. O processo teria sido mais acessível se primeiro tivéssemos um contacto orientado com a API e depois então trabalhássemos os princípios de engenharia de Software.

Em suma, a entreajuda foi muito relevante para o projeto, uma vez que possibilitou enriquecer o trabalho desenvolvido ao longo do semestre.

# 

# Bibliografia

* Guerreiro, Sérgio (2015). Introdução à Engenharia de Software. FCA: Lisboa;
* Material fornecido pelos docentes no espaço da unidade curricular.