

ENGENHARIA DE SOFTWARE

Ciclo de Vida do Desenvolvimento de Software

Exemplo: Estacionamento

Professor Giuliano Bertoti

Atividade de Requisitos

- Coletar os requisitos funcionais e não-funcionais do sistema.
- (O que são requisitos? São as necessidades do cliente!)
- Funcionais: são tarefas ou ações do sistema
- Não-funcionais: são qualidades do sistema

Atividade de Requisitos

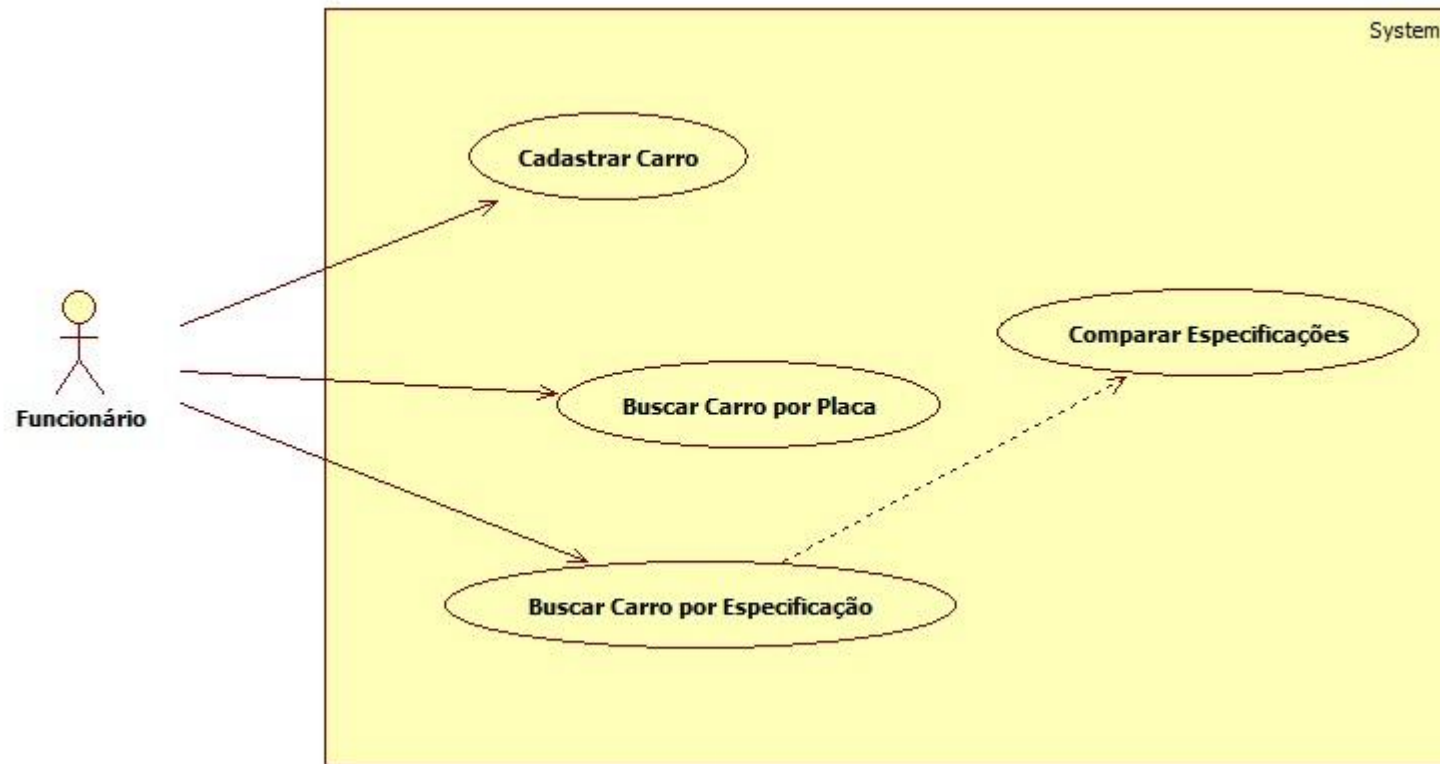
- Funcionais (story cards):
 - O funcionário do estacionamento (ator) pode cadastrar a entrada de carros inserindo placa, marca, modelo e cor do carro;
 - O funcionário do estacionamento pode buscar um conjunto de carros passando para o sistema marca, modelo e cor.
 - O funcionário do estacionamento pode buscar 1 carro passando para o sistema sua placa.
 - Etc...

Atividade de Requisitos

- Você pode também especificar os requisitos funcionais utilizando-se da UML (mais formal):
 - Diagrama de Casos de Uso
 - Diagrama de Sequência

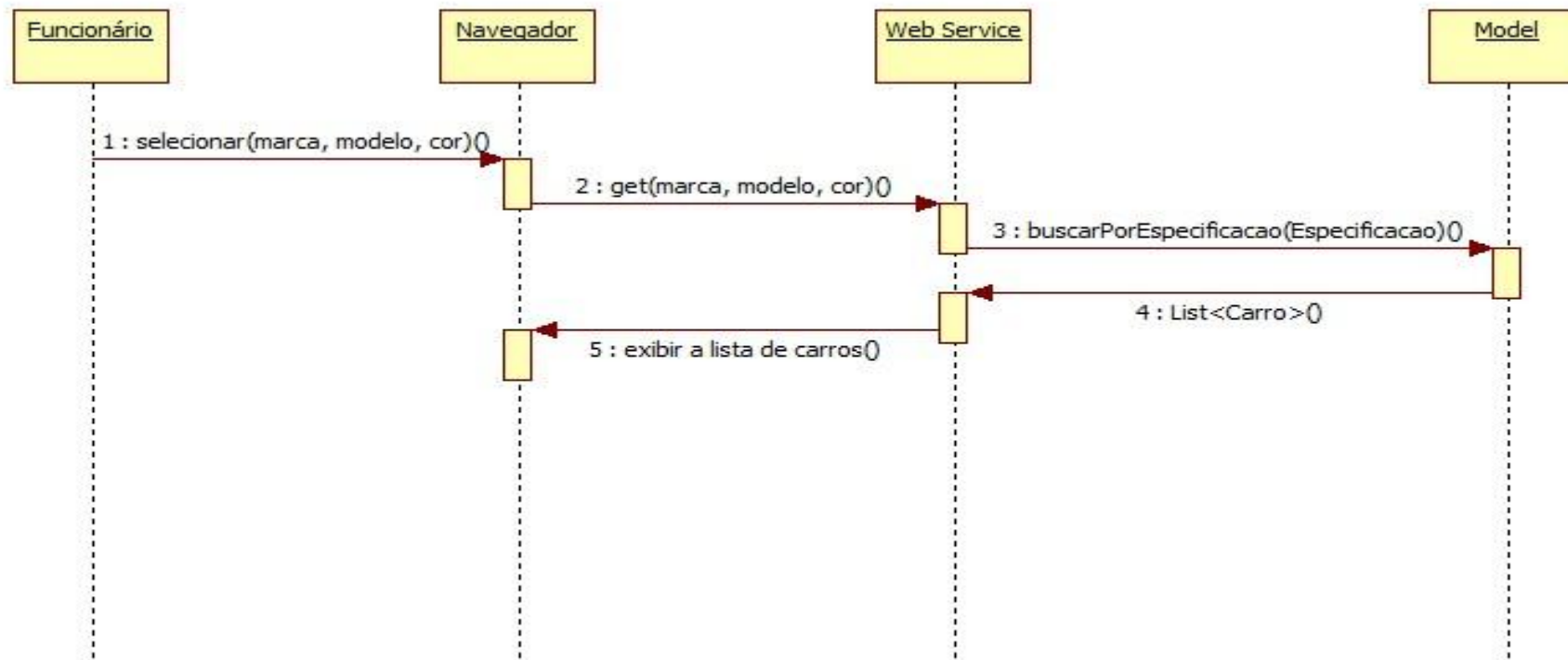
Atividade de Requisitos

- Casos de Uso



Atividade de Requisitos

- Sequência



Atividade de Requisitos

- Não-funcionais:
 - Separação de Interesses: definir uma arquitetura de aplicação com modelo MVC (separando assim lógica de negócios, comportamentos e interação com usuário).
 - Portabilidade: executar o sistema em diferentes plataformas (ex. web, nativa iOS, nativa Android e etc). Importante: a Separação de Interesses do item anterior me ajuda nisso, porque como minha lógica de negócios está encapsulada em um módulo do sistema, se torna possível reutilizá-las em diferentes plataformas.
 - Usabilidade: criar uma interface de simples acesso e uso às funções do sistema.
 - Etc.. (Desempenho é desejável também)

Atividade de Projeto

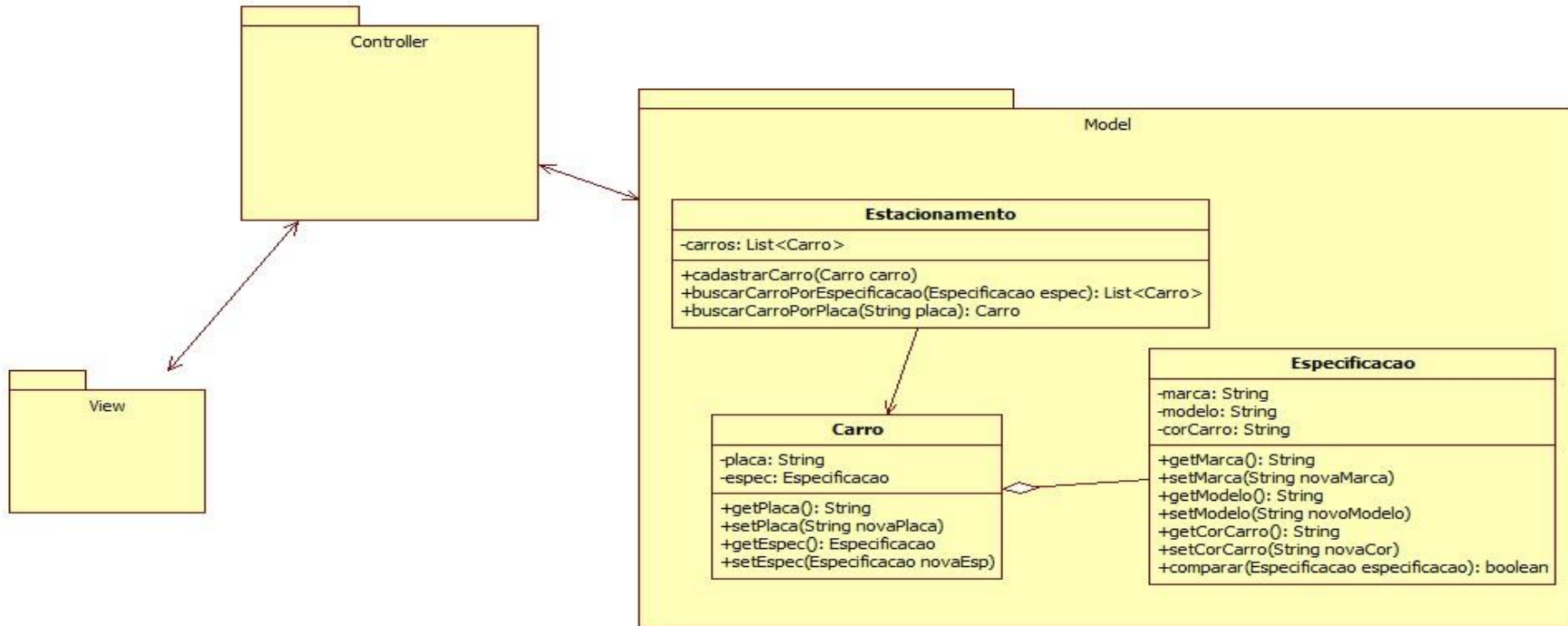
- Transformar os requisitos (tanto funcionais quanto não-funcionais) em algo que possa ser implementado.

Atividade de Projeto

- Projeto da Arquitetura do Sistema
 - O Projeto de Arquitetura me ajuda a obter os requisitos não-funcionais “Separação de Interesses” e “Portabilidade”.
 - Será utilizado o Diagrama de Classes da UML (Unified Modeling Language)
 - Lembre-se: Classes são nomeadas como substantivos com todas as primeiras letras maiúsculas, atributos como substantivos com a primeira minúscula e métodos como verbos com a primeira minúscula (e as demais maiúsculas)
 - Note: os métodos das Classes são os meus requisitos funcionais.

Atividade de Projeto

- Projeto da Arquitetura do Sistema (Diagrama de Classes UML)

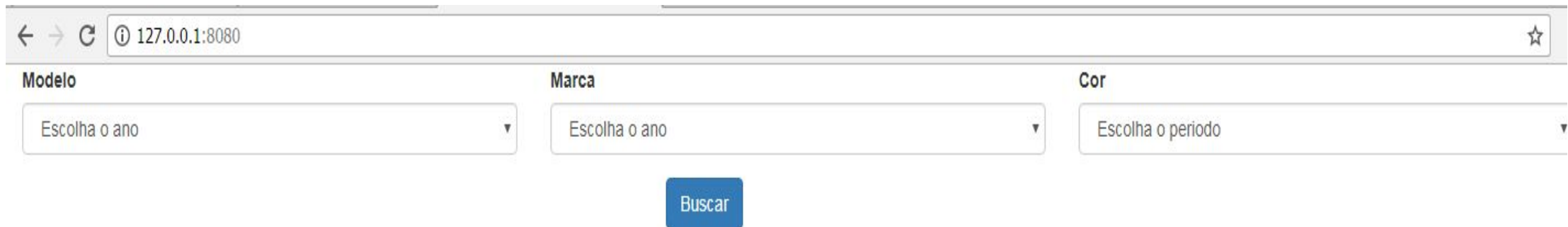


Atividade de Projeto

- Projeto de Interação com o Usuário
 - O Projeto de Interação com o Usuário me ajuda a obter o requisito não-funcional de Usabilidade.
 - Veja Usability Guidelines (princípios norteadores)
 - <https://www.nngroup.com/articles/ten-usability-heuristics/>

Atividade de Projeto

- Projeto de Interação com o Usuário
 - Estética de Design Minimalista (8): “cada unidade extra de informação compete [na cognição do usuário] com unidades relevantes”.
 - Na busca por especificação do carro, por exemplo, oferecer para o funcionário apenas as informações de fato relevantes: marca, modelo e cor do carro.



A screenshot of a web interface for searching car specifications. At the top is a browser address bar showing the URL '127.0.0.1:8080'. Below it are three dropdown menus labeled 'Modelo', 'Marca', and 'Cor'. The 'Modelo' and 'Marca' dropdowns have the placeholder text 'Escolha o ano', while the 'Cor' dropdown has 'Escolha o periodo'. A blue 'Buscar' button is positioned below the 'Marca' dropdown.

← → ↻ 127.0.0.1:8080 ☆

Modelo Marca Cor

Escolha o ano Escolha o ano Escolha o periodo

Buscar

Atividade de Projeto

- Projeto de Interação com o Usuário
 - Reconhecimento ao invés de lembrança (6): “minimize o carregamento de memória do usuário...”.
 - No nosso caso, por exemplo, oferecer uma lista de marcas ao invés de requisitar que o funcionário se lembre disso.

Marca

Escolha o ano ▼

Escolha o ano

vw

fiat

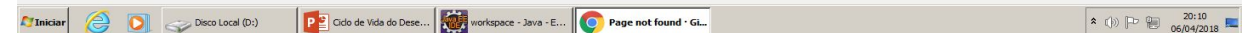
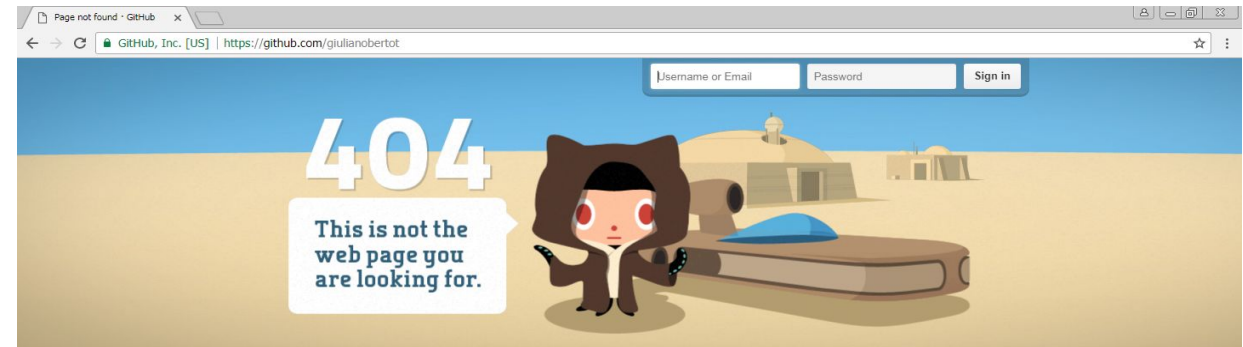
ferrari

Atividade de Projeto

- Projeto de Interação com o Usuário
 - Prevenção de Erros (5): “design que previna o usuário a entrar, por exemplo, com valores errados”.
 - Ex. Ao lado do campo de busca por placa, mostrar um exemplo de como o valor da placa deve ser preenchido: AAA-1111 ou AAA1111 (você pode, além disso, colocar máscaras nos campos para prevenir entradas indesejadas)

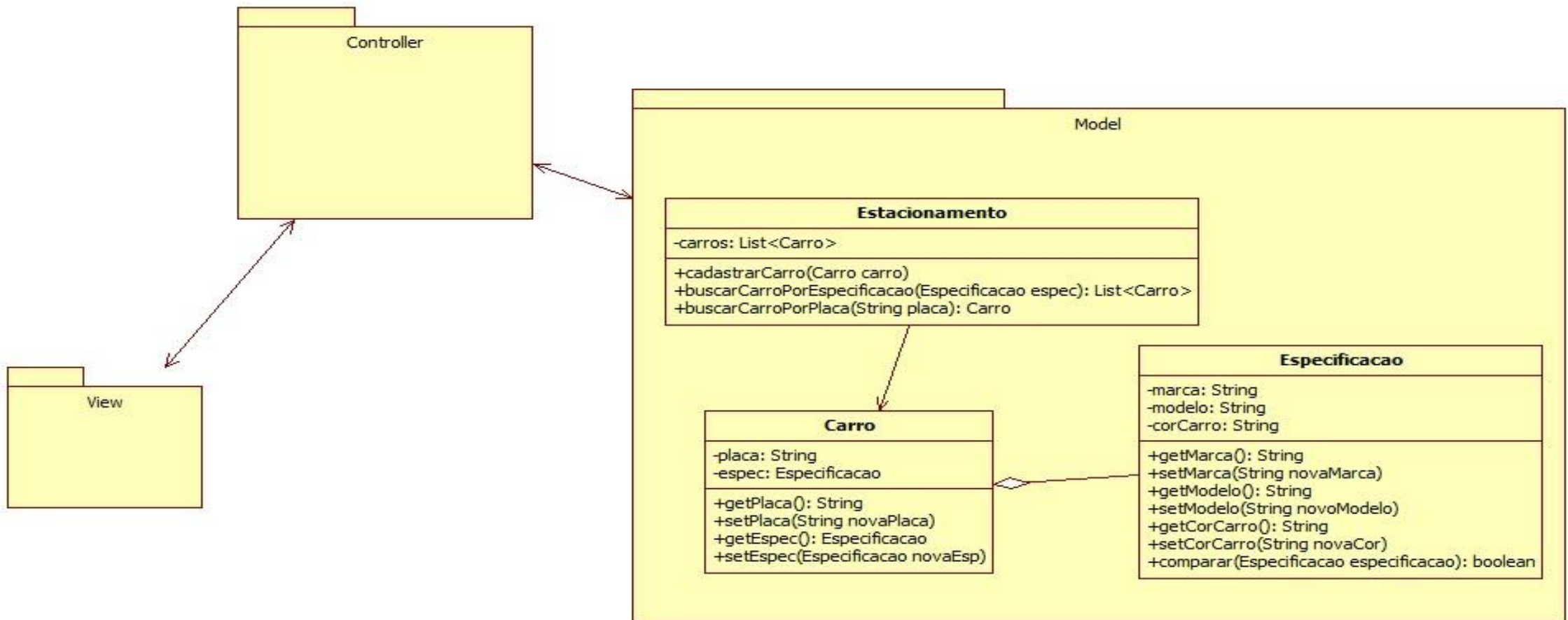
Atividade de Projeto

- Projeto de Interação com o Usuário
 - “Ajudar o usuário a diagnosticar, reconhecer e corrigir problemas” (9)



Atividade de Desenvolvimento

- Iniciaremos o desenvolvimento propriamente dito (implementação) do software a partir da sua arquitetura do sistema (que definimos no Diagrama de Classes UML na atividade anterior: “Atividade de Projeto”).

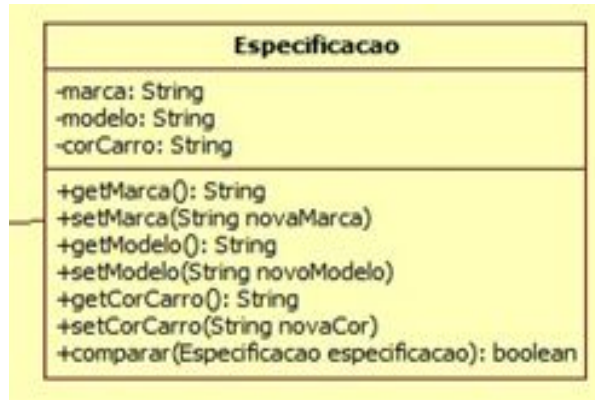


Atividade de Desenvolvimento

- Para a edição de código, compilação, execução, construção (build do projeto) e etc do projeto, utilizamos uma ferramenta CASE ou IDE de desenvolvimento.
- Aqui, vamos utilizar o Eclipse (mas você poderia escolher o IntelliJ, por exemplo).
- Coloquei no moodle um pdf que oferece uma introdução ao Eclipse (para quem perder algo que estou explicando aqui no VNC).



```
public class Carro {  
  
    private String placa;  
    private Especificacao espec;  
  
    public Carro(String placa, Especificacao espec) {  
        this.placa = placa;  
        this.espec = espec;  
    }  
  
    public String getPlaca(){  
        return placa;  
    }  
  
    public void setPlaca(String novaPlaca){  
        placa = novaPlaca;  
    }  
  
    public Especificacao getEspec() {  
        return espec;  
    }  
  
    public void setEspec(Especificacao espec) {  
        this.espec = espec;  
    }  
  
}
```



```
public class Especificacao {

    private String marca;
    private String modelo;
    private String corCarro;

    public Especificacao(String marca, String modelo, String corCarro) {
        this.marca = marca;
        this.modelo = modelo;
        this.corCarro = corCarro;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String novaMarca) {
        this.marca = novaMarca;
    }

    public String getModelo() {
        return modelo;
    }

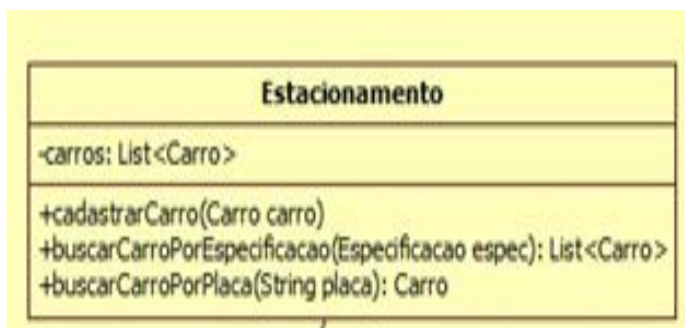
    public void setModelo(String novoModelo) {
        this.modelo = novoModelo;
    }

    public String getCorCarro() {
        return corCarro;
    }

    public void setCorCarro(String novaCor) {
        this.corCarro = novaCor;
    }

    public boolean comparar(Especificacao especificacao){
        if(this.marca.equals(especificacao.marca) &&
this.modelo.equals(especificacao.modelo)
&&
this.corCarro.equals(especificacao.corCarro)){
            return true;
        } else {
            return false;
        }
    }

}
```



```
import java.util.LinkedList;
import java.util.List;

public class Estacionamento {

    private List<Carro> carros = new LinkedList<Carro>();

    public void cadastrarCarro(Carro carro){
        carros.add(carro);
    }

    public List<Carro> buscarCarroPorEspecificacao(Especificacao espec){
        List<Carro> carrosEncontrados = new LinkedList<Carro>();
        for(Carro carro:carros){
            if(carro.getEspec().comparar(espec)) carrosEncontrados.add(carro);
        }
        return carrosEncontrados;
    }

    public Carro buscarCarroPorPlaca(String placa){
        for(Carro carro:carros){
            if(carro.getPlaca().equals(placa)) return carro;
        }
        return null;
    }

    public List<Carro> getCarros(){
        return carros;
    }
}
```

Atividade de Desenvolvimento

- Implementamos juntos no VNC
- Agora, precisamos testar!

Atividade de Teste

- O que é testar? É (tentar) garantir que o software faz o que deveria fazer.
- É possível testar tudo? Não! Por que não?

```
funcao(int a){  
  
    b = a + 1 //deveria ser a - 1  
    c = b / 30000  
    return c  
  
}
```

Vou considerar apenas inteiros no range -32768 até +32767

Dentre estes 65536 números, apenas 4 revelam o defeito do software (-29999, 29999, -30000, 30000)

Atividade de Teste

- Nunca testar aleatoriamente, mas sim utilizar técnicas de teste de software.
- Por exemplo, no caso anterior uma técnica chamada “Teste de Valor Limite” poderia ser utilizada.
 - Logo, eu encontraria os 4 números mais facilmente testando valores “em cima” e próximos de 30000 (por que a divisão antes do retorno é por este número).

Atividade de Teste

- Para este caso do estacionamento, podemos utilizar uma técnica chamada “Testes por Classes de Equivalência”.
- Vamos começar pelo método de “cadastrarCarro”:
 - Se eu adiciono 5 carros, certamente adiciono 6 (5 e 6 fazem parte da mesma classe de equivalência)
 - Vamos definir as classes de equivalência neste caso:
 - 1 classe de equivalência: 0 carros (lista vazia)
 - 1 classe de equivalência: 1 carro (guardando 1 objeto)
 - 1 classe de equivalência: n carros (utilizando a Lista para guardar n carros)

Atividade de Teste

- Método “buscarCarroPorEspecificacao”:
 - 1 classe de equivalência: 0 carros encontrados (estou testando “lista vazia”)
 - 1 classe de equivalência: 1 carro (guardando 1 objeto)
 - 1 classe de equivalência: n carros (encontrando n carros)
- Método “buscarCarroPorPlaca”:
 - 1 classe de equivalência: retorna 1 carro
 - 1 classe de equivalência: retorna null

Atividade de Teste

- Toda linguagem de programação possui bibliotecas para auxiliar no teste. No nosso caso, vamos utilizar o Junit. No Eclipse: File-> New -> Junit Test Case.

Atividade de Teste

```
import static org.junit.Assert.*;

import org.junit.Test;

public class Teste {

    @Test
    public void test() {

        Estacionamento estacionamento = new Estacionamento();

        //Testes de Classes de Equivalencia para o método cadastrarCarro

        //1 classe de equivalência: 0 carros (lista vazia)

        assertEquals(estacionamento.getCarros().size(), 0);

        //1 classe de equivalência: 1 carro (guardando 1 objeto)

        estacionamento.cadastrarCarro(new Carro("AAA1111", new Especificacao("VW", "fusca", "verde")));

        assertEquals(estacionamento.getCarros().size(), 1);

        //1 classe de equivalência: n carros (utilizando a Lista para guardar n carros)

        estacionamento.cadastrarCarro(new Carro("BBB1111", new Especificacao("VW", "fusca", "amarelo")));
        estacionamento.cadastrarCarro(new Carro("CCC1111", new Especificacao("VW", "variant", "azul")));

        assertEquals(estacionamento.getCarros().size(), 3);

    }

}
```

Atividade de Teste

//Testes de Classes de Equivalencia para o método buscarCarroPorEspecificacao

//1 classe de equivalência: 0 carros encontrados (estou testando “lista vazia”)

List<Carro> encontrados = estacionamento.buscarCarroPorEspecificacao(new Especificacao("fiat", "fusca", "azul"));

assertEquals(encontrados.size(), 0);

//1 classe de equivalência: 1 carro (guardando 1 objeto)

List<Carro> encontrados2 = estacionamento.buscarCarroPorEspecificacao(new Especificacao("VW", "fusca", "verde"));

assertEquals(encontrados2.size(), 1);

//1 classe de equivalência: n carros (encontrando n carros)

estacionamento.cadastrarCarro(new Carro("FFF1111", new Especificacao("VW", "fusca", "amarelo")));

estacionamento.cadastrarCarro(new Carro("ABC1111", new Especificacao("VW", "fusca", "amarelo")));

List<Carro> encontrados3 = estacionamento.buscarCarroPorEspecificacao(new Especificacao("VW", "fusca", "amarelo"));

assertEquals(encontrados3.size(), 3);

//Testes de Classes de Equivalencia para o método buscarCarroPorPlaca

//1 classe de equivalência: retorna 1 carro

Carro carro1 = estacionamento.buscarCarroPorPlaca("ABC1111");

assertEquals(carro1.getEspec().getModelo(), "fusca");

assertEquals(carro1.getEspec().getCorCarro(), "amarelo");

//1 classe de equivalência: retorna null

Carro carro2 = estacionamento.buscarCarroPorPlaca("DSA7878");

assertEquals(carro2, null);

workspace - Java - estacionamento/src/Teste.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit

Finished after 0,017 seconds

Runs: 1/1 Errors: 0 Failures: 0

Teste [Runner: JUnit] (0,001s)

Failure Trace

Carro.java Especificacao.java Estacionamento.java Teste.java

```
1 import static org.junit.Assert.*;
2
3 import org.junit.Test;
4
5 public class Teste {
6
7     @Test
8     public void test() {
9
10
11         Estacionamento estacionamento = new Estacionamento();
12
13         //Testes de Classes de Equivalencia para o método cadastrarCarro
14
15         //1 classe de equivalência: 0 carros (lista vazia)
16
17         assertEquals(estacionamento.getCarros().size(), 0);
18
19         //1 classe de equivalência: 1 carro (guardando 1 objeto)
20
21         estacionamento.cadastrarCarro(new Carro("AAA1111", new Especificacao("VW", "fusca", "verde")));
22
23         assertEquals(estacionamento.getCarros().size(), 1);
24
25         //1 classe de equivalência: n carros (utilizando a lista para guardar n carros)
26
27         estacionamento.cadastrarCarro(new Carro("BBB1111", new Especificacao("VW", "fusca", "amarelo")));
28         estacionamento.cadastrarCarro(new Carro("CCC1111", new Especificacao("VW", "variant", "azul")));
29
30         assertEquals(estacionamento.getCarros().size(), 3);
31
32     }
33 }
34
35 }
36
```

Connect Mylyn

Connect to your t
ALM tools or crea
task.

Teste

test()

Problems Javadoc Declaration Console

<terminated> Teste [JUnit] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (2 de abr de 2018 20:15:26)

Picked up _JAVA_OPTIONS: -Xms256m -Xmx512m

Writable Smart Insert 20 : 39

Iniciar

Cido de Vida do Dese...

workspace - Java - ...

Calculadora

20:17
02/04/2018

Atividade de Teste

- Com relação à atividade de teste em si, conseguimos validar os métodos “cadastrarCarro”, “buscarCarroPorEspecificacao” e “buscarCarroPorPlaca” de acordo com a técnica de “Classes de Equivalência”
- Com relação ao Ciclo de Vida como um todo, o teste automatizado me ajudou (tanto com relação à custo quanto a tempo) a iterar entre as etapas do ciclo de vida (ou seja, consigo voltar na prática para atividades anteriores que posso ter errado) e principalmente eu consigo alterar código (Atividade de Manutenção) por exemplo colocando novos requisitos e validar se esse acréscimo manteve o software funcionando corretamente ou não.
- O teste não serve apenas para validar a funcionalidade, mas também para auxiliar na atividade de manutenção (que faz parte do Ciclo de Vida).
- Refatoração de código só é possível com boas baterias de testes automatizados.

Concluindo o Desenvolvimento da Arquitetura

Terminamos a implementação e os testes do “Model” da nossa arquitetura.

Agora faremos o Controller (REST web services)

E por fim a View (interação com o usuário via web app)

<https://github.com/giulianobertoti/softwareengineering>