Lucas Fonseca dos Santos
+55 (16) 99169-9662

PROJETO TESTE PETSHOP DTI

O presente documento visa apresentar a execução e todos os detalhes pertinentes ao projeto teste Petshop proposto.

Foi proposta a construção de um programa que objetiva solucionar qual escolha de opção de petshop seria mais viável frente ao critério de preços menores. Estes preços variam de acordo com o dia da semana, podendo aos finais de semana sofrer variações. Em caso de empate, o critério usado para o desempate é a distância do pet shop. Para além deste parâmetro, não foi proposto outro critério.

As informações utilizadas para a tomada de decisão são em parte pré inseridas no código fonte (a lista dos petshops e suas propriedades) e inseridas pelo usuário ao momento da execução (data requerida, quantidade de cachorros grandes e quantidade de cachorros pequenos).

1. CONSTRUÇÃO

A aplicação foi construída utilizando linguagem Java versão 8 sobre paradigma da Orientação a Objetos. Não foi utilizado nenhuma biblioteca externa conforme especificado na descrição do projeto.

1.1. Arquitetura

No projeto construído não foi utilizado algum padrão arquitetural específico, mas diversas estratégias foram tomadas buscando alcançar alguns princípios conhecidos da engenharia de software, S.O.L.I.D. e facilitar processos evolutivos futuros. O projeto também contou com a presença do padrão de projeto build, no caso em questão utilizado para melhorar a experiência de legibilidade e escrita do código, dado a quantidade de parâmetros envolvidos na instanciação do objeto em questão (ilustrado na imagem abaixo) e o padrão de design template method, para adicionar uma modificação de comportamento pequena em cada instância de petshop (no caso em questão, alterando a política de preços para dias que caiam em finais de semana, conforme

requisito apresentado do projeto).

```
new Petshop()
        .setName("Meu Canino Feliz")
        .setDistance(2.0)
        .setSmallDogBathPrice(20.00)
        .setBigDogBathPrice(40.00)
        .setWeekendPrices(new PricePolicy(){

            @Override
            public double getBigDogBathPriceOnWeekend(double value) {

                return (value + (.20 * value));

            }

            @Override
            public double getSmallDogBathPriceOnWeekend(double value) {

                return (value + (.20 * value));

            }

    })
        .build(),
```

Imagem 1.1: Builder design pattern sendo utilizado na aplicação para garantir melhor legibilidade e escrita no momento da instanciação de um objeto da classe Petshop.

```
.setWeekendPrices(new PricePolicy(){

    @Override
    public double getBigDogBathPriceOnWeekend(double value) {

        return (value + (.20 * value));

    }

    @Override
    public double getSmallDogBathPriceOnWeekend(double value) {

        return (value + (.20 * value));

    }
```

Imagem 1.2: Template method implementado através da interface PricePolicy que provém métodos com comportamentos sobrescritos aos objetos instanciados da classe Petshop. Neste caso, uma diferente política de preços para fins de semana.

1.2. Estrutura

A estrutura do projeto conta com 5 pacotes e 12 classes conforme abaixo listado::

```
dev
├── core
│   ├── ApplicationBootstrap.java
│   ├── ApplicationDataLoader.java
│   ├── ApplicationProcessor.java
│   ├── ApplicationValidations.java
│   └── Main.java
├── handler
│   └── DateHandler.java
├── model
│   ├── Input.java
│   ├── Petshop.java
│   ├── PetshopOption.java
│   └── PricePolicy.java
├── tool
│   └── Parser.java
└── view
        └── Cli.java
```

Observando tal estrutura, já é possível denotar uma certa unicidade de responsabilidade de classe, como no pacote core, onde a classe application tem suas responsabilidade divididas em 4 outras classes, a ApplicationBootstrap, responsável pela linkagem entre os módulos do programa, quase de certa forma encapsulando a aplicação, a classe ApplicationDataLoader, responsável únicamente por prover acesso a dados, abrindo um horizonte de evolução para o projeto (até o momento, os dados utilizados pelo programa são pré definidos no código fonte e não lidos de alguma base).

A classe ApplicationProcessor é responsável pelo definição das regras de negócio da aplicação. Nela, estão descrito as decisões que levarão a busca da solução que o usuário precisa. Todos as informações cedidas por meio de entradas pelo usuário, são validadas na ApplicationValidations, que se baseia nas regras de negócio para aferir suas validações.

No pacote handler, estão situados os manipuladores do projeto, no caso, temos apenas uma classe manipuladora, de data, responsável por manusear e organizar informações relacionadas a datas.

No pacote model, encontramos o modelo das entidades pertencentes ao nosso projeto, no caso a entidade Input que representa os dados inseridos pelo usuário e trafegados na aplicação, a interface PricePolicy que provém diferenças nos comportamentos das entidades Petshop, essas, representando a abstração física de um petshop com suas propriedades (informadas via requisito na descrição do projeto) com distância e preços de banho para animais pequenos e grandes, incluindo sua variação aos finais de semana. A abstração petshop option diz respeito a execução do algoritmo que busca a opção de petshop mais viável frente aos requisitos de competição propostos pelo projeto. Ela armazena o petshop e os valores totais do custo dos banhos, baseados na quantidade de animais informados pelo usuário.

No pacote tool, estão presentes ferramentas que auxiliam o trabalho do programa, como o Parser, responsável por validar e construir sintaticamente e semanticamente, a entrada do

usuário, provida através de uma Command Line Interface, definida na classe Cli no pacote view. Se um dia, houver um projeto para construir uma interface gráfica na aplicação, basta construir um módulo a parte no pacote view e integrá-lo ao core da aplicação.

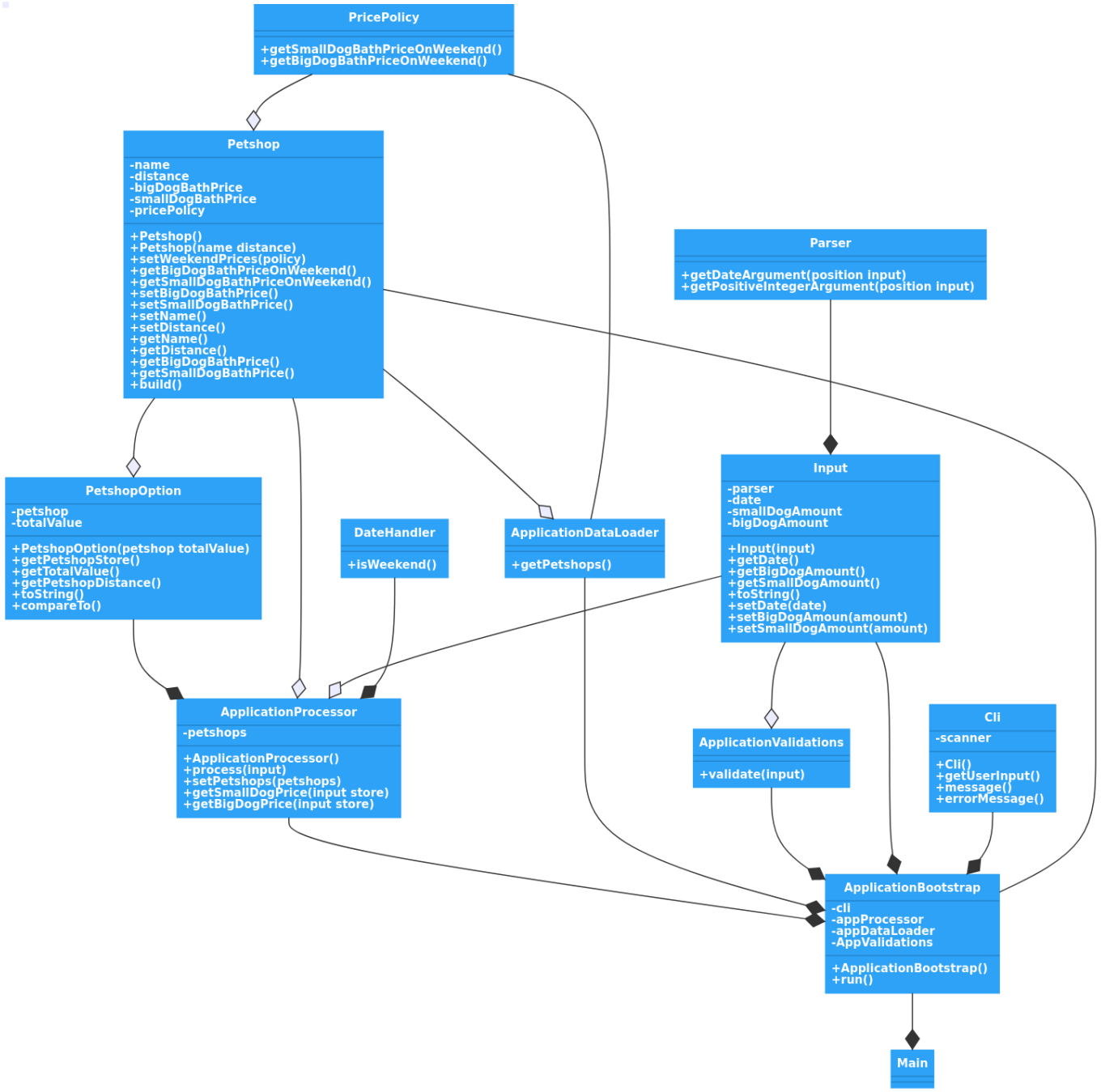

Imagem 1.3:Diagrama final UML de classes do projeto.

2. EXECUÇÃO

Para facilitar a execução do projeto, foi escrito um script em bash (GNU/Linux) que compila todos os códigos fontes .java recursivamente nos pacotes e depois executa o código objeto em um diretório build, para tal, basta:

```
$ ./init.sh
```

Se ocorrer qualquer tipo de problema relacionado a permissão, basta alterar tais propriedades a partir do seguinte comando:

```
$ chmod +x
```

Feito o passo anterior, será executado a aplicação, com o seguinte padrão de entrada (este fornecido via requisito na descrição do teste)

```
<data> <quantiade_de_animais_pequenos> <quantidade_de_animais_grandes>
```

A saída esperada será uma resposta textual informando a melhor opção de petshop dentre os critérios pré estabelecidos para a tomada de decisão, juntamente ao valor final de tudo isso.

3. ADENDOS

Foi utilizado um repositório público no github para exposição do projeto em questão para os avaliadores da DTI, entretanto, não foi utilizado algum fluxo de git como gitflow, que poderia facilmente ter sido implantado no projeto, com pré definições de responsabilidade para branchs e nomenclaturas específicas.

Link do repositório: https://github.com/LucasFonsecadosSantos/test