Projeto 3

Pilhas

Unidade Curricular de
Laboratório de Programação

2020/2021

Objetivos

- Utilizar a estrutura de dados Pilha;
- Utilizar o interface `Stack` e a classe `ArrayStack` desenvolvidos na disciplina de AED.

Antes de Começar

Criar um projeto java e importar os ficheiros

- Fazer *download* do ficheiro `alunosProjeto3.zip` na página de LabP;
- No Eclipse:
  - Escolher **File > Import > General > Existing Projects int Workspace**
  - De seguida **Select Archive File** e selecionar `alunosProjeto3.zip`
    Deverá passar a ter um projeto chamado `Projeto3` contendo:
    - Na pasta `src`, os ficheiros `RunOrders.java`, `TestsWarehouse.java`, `Stack.java` e `ArrayStack.java`;
    - Na raiz do projeto, vários ficheiros de dados.

Algumas informações úteis

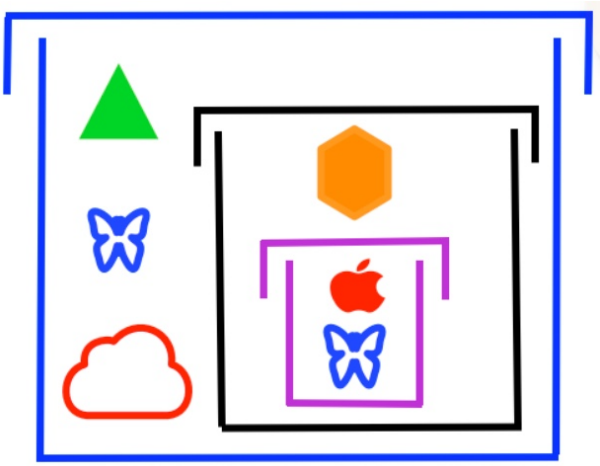
De modo a poder realizar este projeto deverá recordar o tipo de dados Pilha e o interface `Stack` e a classe `ArrayStack` desenvolvidas na disciplina de AED.

Enunciado

O sr. Aurélio quer automatizar o processo de preparação de encomendas no armazém da sua empresa.

Os alunos de LabP fazem parte da equipa que vai fornecer o equipamento, computadores, etc, e respetivas aplicações, para que seja possível, a partir da descrição de uma encomenda, organizar automaticamente todas as partes dessa encomenda para posterior envio ao cliente.

Considere a figura ao lado.



Se codificarmos os artigos nesta encomenda da seguinte forma:

- caixas: `Blue Box-bx1` ; `Black Box-bx2` e `Pink Box-bx3`;
- tampas das caixas: `1xb` a `3xb`, de forma correspondente;
- outros objetos: `Triangle-Tri` ; `Butterfly-anim1` ; `Cloud-Clo` ; `Hexagon-Geom3` ; `Apple-Fr2`

uma possível *descrição* desta encomenda seria:

```
bx1 Clo anim1 bx2 bx3 anim1 Fr2 3xb Geom3 2xb Tri 1xb
```

A partir de uma descrição deste tipo, queremos produzir uma sucessão de instruções que servirão para saber que artigos usar e como os dispor.

Para este exemplo, essas instruções seriam:

```
Go fetch Blue Box and put it on the table
Go fetch Cloud and put it in Blue Box
Go fetch Butterfly and put it in Blue Box
Go fetch Black Box and put it in Blue Box
Go fetch Pink Box and put it in Black Box
Go fetch Butterfly and put it in Pink Box
Go fetch Apple and put it in Pink Box
Go fetch the cover of Pink Box and close it
Go fetch Hexagon and put it in Black Box
Go fetch the cover of Black Box and close it
Go fetch Triangle and put it in Blue Box
Go fetch the cover of Blue Box and close it
```

Para que isto seja possível, as descrições fornecidas têm que constituir encomendas válidas. Vamos considerar as seguintes regras:

- Todas as caixas deverão ter um código contendo um prefixo comum. No exemplo acima o prefixo comum é "`bx`". As tampas são representadas por um código que é o inverso do código da caixa correspondente. Os códigos dos outros artigos são diferentes dos códigos das caixas e seus inversos.
- Todas as encomendas têm que incluir pelo menos uma caixa em primeiro lugar e a tampa correspondente em último.
- As tampas devem sempre fechar as caixas correspondentes.
- Cada novo item (caixa, tampa ou objeto) é colocado na caixa mais recentemente aberta que ainda não foi fechada.

- Cada artigo tem determinados peso e volume. O volume de uma caixa deve ser suficiente para conter todos os itens indicados. Por exemplo, o volume da caixa lilás adicionado ao volume de um hexágono não deve exceder o volume da caixa preta.
- Todos os artigos referidos na descrição devem existir em *stock*.

Objetivos

No contexto de um armazém onde existem vários artigos, queremos que seja possível fazer várias ações a partir de uma descrição de encomenda:

- Saber se é uma descrição correta, ou seja, se é possível obter os materiais necessários e dispô-los da forma indicada.

  Exemplos de descrições incorretas:

  - `bx1 obj1 bx2 obj2 1xb 2xb` - a tampa da caixa 1 aparece para tapar a caixa 2;
  - `obj1 bx2 obj2 2xb` - a encomenda não começa com uma caixa;
  - `bx2 obj2 obj1` - a caixa ficou por fechar;
  - `bx1 obj33 1xb` em que o produto `obj33` não existe, ou existe mas não há em *stock*;
  - `bx1 obj2 obj2 1xb` em que só existe uma unidade do produto `obj2` em *stock*;
  - `bx1 obj2 obj1 obj3 1xb` em que a soma dos volumes dos três objetos excede o volume da caixa;

- Efetivar a encomenda correspondente a uma descrição válida, ou seja, retirar de *stock* as unidades necessárias de cada artigo, e obter uma lista ordenada das tarefas a fazer para preparar a encomenda;

- Saber o volume e o peso da encomenda correspondente a uma descrição válida. O volume da encomenda é o volume da caixa que a contém. O peso da encomenda é a soma dos pesos de todos os artigos que a compõem.

O que fazer

Comecemos por raciocinar sobre os tipos de dados que precisamos de criar e de utilizar para obtermos o efeito desejado.

Temos *artigos*, com características que os definem e os distinguem uns dos outros:

- Um código (no exemplo `Tri`, `anim1`, `Geom3`, etc);
- Uma descrição (no exemplo `Triangle`, `Butterfly`, `Hexagon`, etc);
- Um volume e um peso.

Temos *armazéns* com artigos usados em encomendas e caixas para os colocar. Uma caixa também tem as características acima, por isso vamos considerar que também é um artigo.

No contexto deste problema, não precisamos de distinguir um artigo de outro perfeitamente igual; só precisamos de saber quantos existem em *stock*.

Então, podemos associar a cada artigo uma nova característica:

- Número de exemplares em *stock*.

Um armazém tem então:

- um conjunto de artigos;
- um prefixo que é comum aos códigos associados às caixas (`bx` no exemplo).

Temos também o conceito de *encomenda*. Mas, na verdade, o que nos interessa é:

- saber se uma descrição de uma encomenda é válida, ou seja, se está bem construída e se os artigos que refere existem em stock;
- se a descrição válida, obter a sucessão de instruções de montagem da encomenda que ela descreve;
- retirar de *stock* os artigos referidos numa descrição válida.

Ou seja, nada nos faz sentir a necessidade de ter objetos que representem encomendas.

Então, faz sentido criarmos as duas classes seguintes:

- `Article`, cujas instâncias representam artigos. Deve definir:
  - atributos que representem as características indicadas anteriormente;
  - um construtor que inicialize os atributos de um novo artigo com os valores dados nos parâmetros;
  - métodos que permitam revelar os valores dos atributos;
  - métodos que permitam retirar um dado número de unidades do *stock*.
- `Warehouse`, cujas instâncias representam armazéns. Deve definir:
  - atributos que representem as entidades indicadas anteriormente;
  - um construtor que receba como parâmetros o número máximo de artigos diferentes que o armazém poderá ter e o prefixo comum a todas as caixas;

  Para estar de acordo com o código da classe `RunOrders` por nós fornecida, que é cliente de `Warehouse`, esta classe deve oferecer também os seguintes métodos públicos:
  - `boolean inCatalog(String code)` que retorna `true` se um artigo com código `code` está já definido no armazém, mesmo que tenha *stock* zero;
  - `void addArticle(String code, String description, double volume, double weight, int quantityInStock)` que, assumindo que um artigo com o código `code` não existe já definido no armazém, adiciona um novo artigo com as características dadas;
  - `double[] articleVolumeWeight(String code)` que, assumindo que um artigo com o código `code` existe no armazém, devolve o seu volume e peso;
  - `OrderStatus validOrder(String description)`, que verifica se `description` é uma descrição válida de uma encomenda neste armazém; como resultado terá um dos seguintes valores:
    - `VALID_ORDER` – descrição válida segundo as regras já descritas anteriormente;
    - `NO_BOX_START` – se a descrição não se inicia com o código de uma caixa;

        - `ORDER_VOLUME_EXCEEDS_BOX` – o volume total dos artigos a meter numa dada caixa excede o volume da caixa;
        - `MISMATCHED_BOX_COVER` – um código que não corresponde a nenhum artigo também não corresponde à tampa para a última caixa;
        - `NO_AVAILABLE_STOCK` – um código corresponde a um artigo que não tem stock;
        - `UNCLOSED_BOX` – se alguma caixa fica por fechar.
    - `String putOrder(String description)` que, assumindo que `description` é uma descrição válida, retira uma unidade de *stock* dos artigos que aparecem na `description` e cria e devolve uma *string* contendo a sucessão de instruções correspondentes (ver pág. 2);
    - `double[] orderVolumeWeight(String description)` que, assumindo que `description` é uma descrição válida, devolve o volume e o peso da encomenda correspondente.

Os tipos de dados seguintes são fornecidos por nós e devem ser usados na vossa solução:

- `OrderStatus`, enumerado cujos valores representam resultados possíveis na validação de uma descrição de encomenda.
- `Stack`, o interface construído em AED que define o tipo de dados Pilha; deverão usar este tipo de dados na implementação de alguns métodos da classe `Warehouse`, como por exemplo, os métodos `putOrder` e `validOrder`.
- `ArrayStack`, uma classe que implementa o interface `Stack`.

Como exemplo de utilização das classes que precisam de criar, vejam o método `main` da classe `RunOrders` fornecida por nós. Devem executá-lo para testarem a vossa solução. Esse método `main`:

- Cria uma instância da classe `Warehouse`, que representa um armazém e adiciona-lhe vários artigos lidos a partir do ficheiro `articlesIn.txt`. Este ficheiro tem na primeira linha um inteiro que representa o número de artigos que se se seguem, um por linha. Cada linha tem o código do artigo, uma descrição, o volume, o peso e o número de unidades em *stock*.
- Para cada uma de várias descrições lidas a partir do ficheiro `ordersIn.txt`:
    - Verifica se a descrição é válida.
    - Se for válida, efetiva a encomenda correspondente e obtém informações sobre ela.
    - Se não for válida, imprime qual o problema detetado.

Para o ficheiro `ordersIn.txt` contendo

```
bx1 Clo anim1 bx2 bx3 anim1 Fr2 3xb Geom3 2xb Tri 1xb
bx1 Clo anim1 bx2 bx3 anim1 Fr2 3xb Geom3 2xb Tri 1xb
bx1 Clo Clo bx2 bx3 anim1 Fr2 2xb Geom3 3xb Tri 1xb
Clo anim1 bx2 bx3 anim1 Fr2 3xb Geom3 2xb Tri
bx3 Clo Fr2 Clo 3xb
bx1 Clo bx2 anim1 2xb Fr2
```

e `articlesIn.txt`, contendo

```
8
bx1;Blue Box;50;0.73;4;
bx2;Black Box;35;0.56;3;
bx3;Pink Box;20;0.38;5;
Tri;Triangle;2;2;1;
anim1;Butterfly;1.5;0.5;3;
Clo;Cloud;10;0.5;4;
Geom3;Hexagon;8;5;3;
Fr2;Apple;4;2;3;
```

o *output* do `main` da classe `RunOrders` deverá ser:

```
From the articles input file:
bx1  Blue Box  50  0.73  4
bx2  Black Box  35  0.56  3
bx3  Pink Box  20  0.38  5
Tri  Triangle  2  2  1
anim1  Butterfly  1.5  0.5  3
Clo  Cloud  10  0.5  4
Geom3  Hexagon  8  5  3
Fr2  Apple  4  2  3
============================================================

Asking the warehouse for information about articles:
Article bx1:   Volume: 50.0    Weight: 0.73
Article bx2:   Volume: 35.0    Weight: 0.56
Article bx3:   Volume: 20.0    Weight: 0.38
Article Tri:   Volume: 2.0    Weight: 2.0
Article anim1:   Volume: 1.5    Weight: 0.5
Article Clo:   Volume: 10.0    Weight: 0.5
Article Geom3:   Volume: 8.0    Weight: 5.0
Article Fr2:   Volume: 4.0    Weight: 2.0
============================================================

The description 'bx1 Clo anim1 bx2 bx3 anim1 Fr2 3xb Geom3 2xb Tri 1xb'
gives the following:
Go fetch Blue Box and put it on the table
Go fetch Cloud and put it in Blue Box
Go fetch Butterfly and put it in Blue Box
Go fetch Black Box and put it in Blue Box
Go fetch Pink Box and put it in Black Box
Go fetch Butterfly and put it in Pink Box
Go fetch Apple and put it in Pink Box
Go fetch the cover of Pink Box and close it
Go fetch Hexagon and put it in Black Box
Go fetch the cover of Black Box and close it
Go fetch Triangle and put it in Blue Box
Go fetch the cover of Blue Box and close it


Order volume: 50.0  and weight: 12.17
============================================================

The description 'bx1 Clo anim1 bx2 bx3 anim1 Fr2 3xb Geom3 2xb Tri 1xb'
gives the following:
Article not in stock
============================================================
```

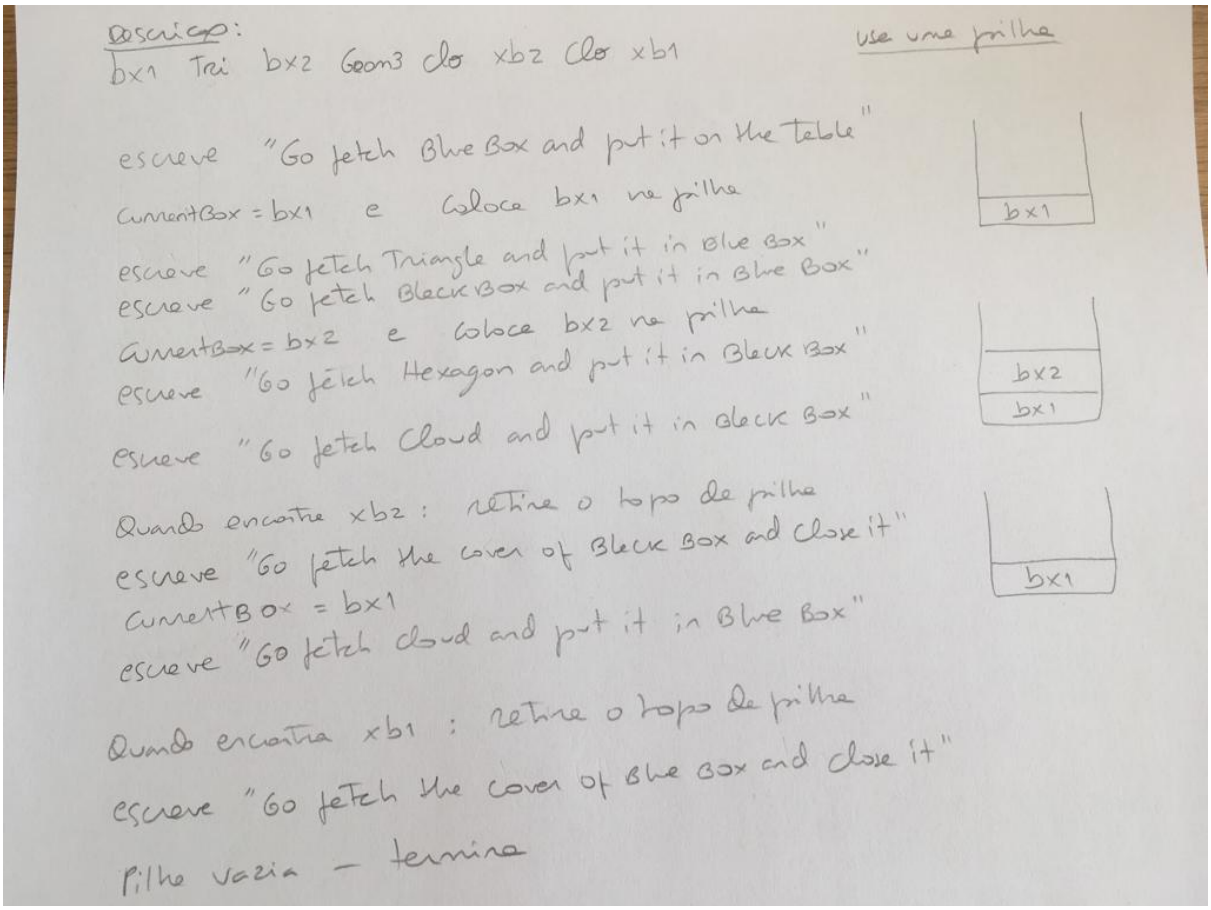
```
The description 'bx1 Clo Clo bx2 bx3 anim1 Fr2 2xb Geom3 3xb Tri 1xb' gives
the following:
The cover does not match the box
============================================================

The description 'Clo anim1 bx2 bx3 anim1 Fr2 3xb Geom3 2xb Tri' gives the
following:
The order does not begin with a box
============================================================

The description 'bx3 Clo Fr2 Clo 3xb' gives the following:
The contents volume exceeds the box volume
============================================================

The description 'bx1 Clo bx2 anim1 2xb Fr2' gives the following:
A box is not closed
============================================================
```

Uma dica para o método `putOrder`:



Claro que, para cada artigo (incluindo caixas), há que retirar uma unidade de *stock* de cada vez que aparece.

Entenda-se a palavra "escreve" como escrever no `StringBuilder` onde se está a construir o resultado do método.

Antes de Entregar

Antes de entregar, certifique-se da correção da formatação e inclua comentários *javadoc* para todos os métodos.

Na linha do ficheiro contendo a *tag* `@author` indique o seu nome e número de aluno.

Entrega

Deve submeter no Moodle os dois ficheiros `Article.java` e `Warehouse.java`