Instituto Superior de Engenharia do Porto

Projeto 1

ESINF – Licenciatura em Engenharia Informática

Relatório

Turma 2DM e 2DN

Beatriz Santos - 1211178 (2DM)

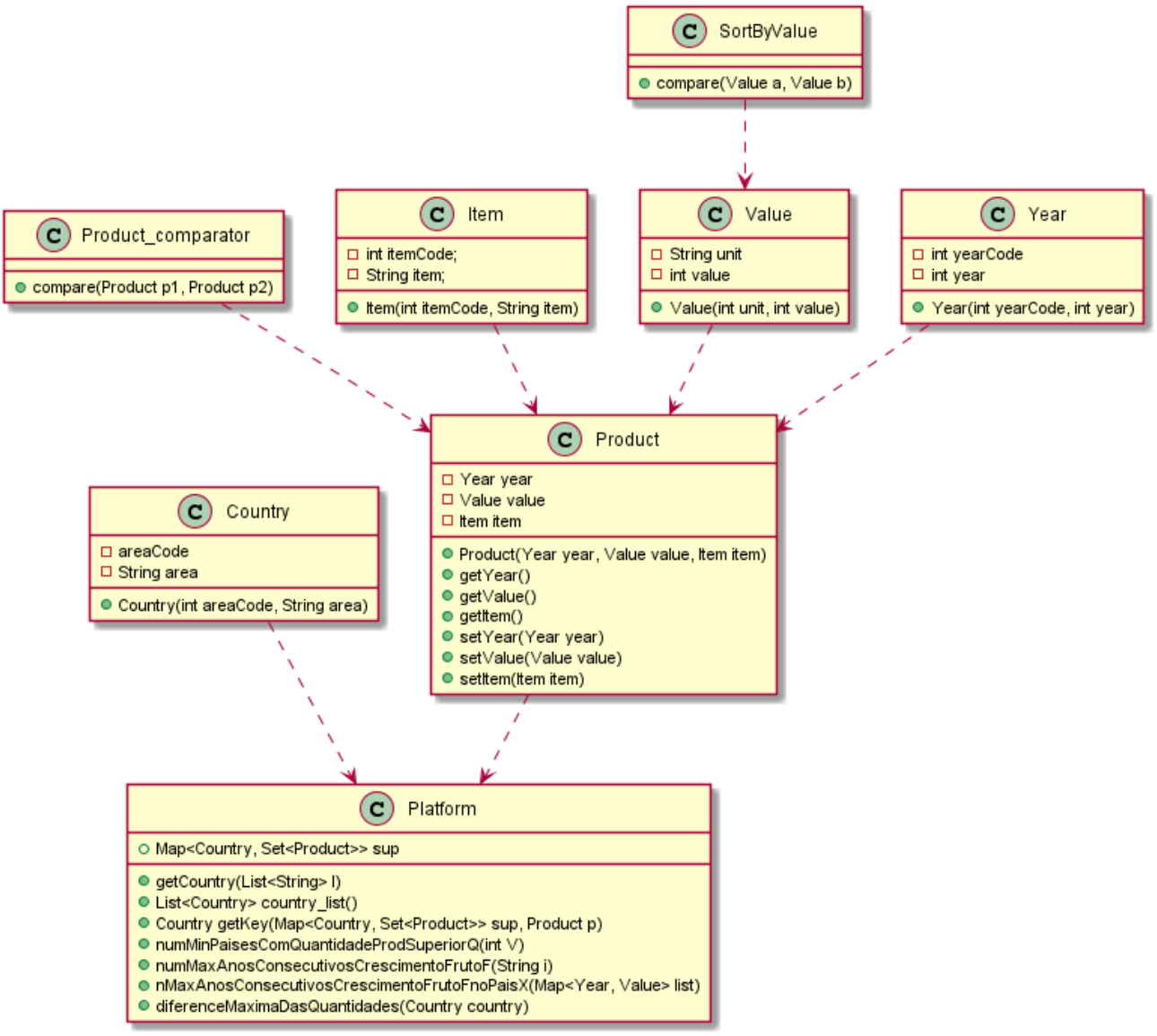
Daniela Soares - 1211229 (2DM)

Ruben Silva - 1200546 (2DM)

Daniel Braga – 1200801 (2DN)

2022

Diagrama de Classes



Funcionalidade 1:

```
void getCountry(List<String> l) throws Exception {
    Set<Product> sp = null;
    Country c = null;
    Year y = null;
    Value v = null;
    Item i = null;
    Product p = null;
    int x = 0;

    for (String s : l) {
        if (x == 0) {
            x++;
            continue;
        }
        if (s.charAt(0) == '\"') {
            String[] lin = s.split( regex: "\",\"");
            c = new Country(Integer.parseInt(lin[2]), lin[3]);
            y = new Year(Integer.parseInt(lin[8]), Integer.parseInt(lin[9]));
            v = new Value(lin[10], Integer.parseInt(lin[11]));
            i = new Item(Integer.parseInt(lin[6]), lin[7]);
            p = new Product(y, v, i);
            if (!sup.containsKey(c)) {
                sp = new HashSet<>();
                sp.add(p);
                sup.put(c, sp);
            } else {
                Set<Product> set = sup.get(c);
                set.add(p);
                sup.put(c, set);
            }
            break;
```

O método getCountry() tem como a função ler os dados do ficheiro que irão ser recebidos no parâmetro como uma lista de String. Foi utilizado o Map para os dados serem interpretados da maneira mais intuitiva possível. Onde a chave seria o Country e o valor um Set<Product>. O Product teria dentro de si as classes Year, Value e Item. Com yearCode e year, unit e value e, por fim, itemCode e item.

Como temos dois tipos de ficheiros com pormenores ligeiramente diferentes, é importante saber diferenciá-los e, só depois começar a guardar os dados.

Primeiro de tudo, foi implementar uma variável denominada de x, com valor igual a zero. Bem no início da primeira iteração nós saltamos uma linha. Isto porque todos os ficheiros terão um cabeçalho que deverá ser ignorado.

No ficheiro “BIG” os dados encontram-se separados por uma vírgula e dentro de aspas. Logo, o passo correto é identificar se o primeiro caracter é equivalente a uma aspa (“). Então, dividimos a string pela ocorrência de uma virgula entre aspas (“,”),

No ficheiro “SMALL” já não irá começar por uma aspa, mas por um outro caracter qualquer, uma letra é a opção mais comum. Neste caso só é preciso dividir a informação por virgulas (,).

A seguir de identificarmos a forma como a informação é dividida, começamos por armazenar os dados nos seus lugares respetivos. Adicionando o country ou areaCode como a chave, e preenchendo a Product numa HashSet, pois a ordenação não foi considerada.

Se por acaso a chave já é existente, nós simplesmente adicionamos os valores da Product a chave.

```
        } else if (s.charAt(0) != '\"') {
            String[] lin = s.split( regex: ",");
            c = new Country(Integer.parseInt(lin[2]), lin[3]);
            y = new Year(Integer.parseInt(lin[8]), Integer.parseInt(lin[9]));
            v = new Value(lin[10], Integer.parseInt(lin[11]));
            i = new Item(Integer.parseInt(lin[6]), lin[7]);
            p = new Product(y, v, i);
            if (!sup.containsKey(c)) {
                sp = new HashSet<>();
                sp.add(p);
                sup.put(c, sp);
            } else {
                Set<Product> set = sup.get(c);
                set.add(p);
                sup.put(c, set);
            }
            break;
        } else {
            throw new Exception("Line not recognized");
        }
    }
}
```

Se no caso dos if's não serem correspondidos com nada, o método lança uma Exeption com a mensagem "Line not recognized".

Funcionalidade 2:

```java
// Exercício 2
1 usage  Daniela_Soares +1
public List<Country> country_list(String fruit, int value) {
    LocalDate current_date = LocalDate.now();
    List<Product> productList = new ArrayList<>();
    List<Country> countryList = new ArrayList<>();
    for (Country c : sup.keySet()) {
        Set<Product> sp = sup.get(c);
        for (Product p : sp) {
            if (p.getItem().getItem().equals(fruit) && (current_date.getYear() - p.getYear().getYear()) >= 1 && p.getValue().getValue() >= value) {
                productList.add(p);
            }
        }
    }
    productList.sort(new Product_comparator());
    for (Product p : productList) {
        countryList.add(getKey(sup, p));
    }

    return countryList;
}
```

Para esta funcionalidade, inicialmente será criada uma lista de produtos e uma lista de países (por enquanto será vazia). De seguida os ciclos for irão percorrer cada produto dentro do set de produtos de cada país dentro do map e irá ser feita uma condição. Esta condição será verificada caso a produção do fruto "fruit" seja maior ou igual a "value"(*p.getItem().getItem().equals(fruit)* e *p.getValue().getValue() >= value*) e caso a produção tenha pelo menos 1 ano (*current_date.getYear()-p.getYear().getYear()>=1*). Ao ser verificada a condição o produto será adicionado à lista.

Após percorrer o map, a lista de produtos será organizada pelos seguintes critérios:

```java
3 usages  Daniel Braga
public class Product_comparator implements Comparator<Product> {

    Daniel Braga
    @Override
    public int compare(Product p1, Product p2) {
        if(p1.getYear().compareTo(p2.getYear()) == 0){
            if(p1.getValue().compareTo(p2.getValue()) > 0) return -1;
            else if(p1.getValue().compareTo(p2.getValue()) < 0) return 1;
            else return 0;
        }else if(p1.getYear().compareTo(p2.getYear()) > 0) return 1;
        else return -1;
    }
}
```

Caso os anos de produção dos produtos sejam iguais, será comparada a quantidade produzida dos mesmos, retornando –1 caso a quantidade produzida do primeiro for maior ou 1 caso a quantidade produzida do segundo for maior, sendo assim ordenada por ordem decrescente. No caso dos anos de produção serem diferentes, será retornado o valor 1 caso o ano de produção do primeiro for maior, ou –1 se acontecer o oposto, ficando assim ordenada por ordem crescente.

Depois da lista de produtos estar ordenada, a lista de países inicialmente criada será então preenchida pela ordem que a lista de produtos tem, não adicionando países que já estejam presentes na lista:

```
productList.sort(new Product_comparator());
for (Product p : productList) {
    Country c1 = getKey(sup, p);
    if(!countryList.contains(c1)){
        countryList.add(c1);
    }
}
```

Gerando assim a lista de países ordenada como pedido. O método utilizado para obter o país a partir do produto foi o seguinte:

```
private Country getKey(Map<Country, Set<Product>> sup, Product p) {
    for (Map.Entry<Country, Set<Product>> entry : sup.entrySet()) {
        if(entry.getValue().contains(p)){
            return entry.getKey();
        }
    }
    return null;
}
```

Irá ser feito um ciclo for que irá percorrer todas as combinações de país e set de produtos existente e se o produto p estiver presente no set de produtos de um determinado país, esse país será adicionado à lista.

Funcionalidade 3:

```
//Exercício 3
public int numMinPaisesComQuantidadeProdSuperiorQ(int V) {
    Set<Country> res = new HashSet<>();
    int somaQuantidades = 0;
    int limit = 0;
    int lowestValue = 99999999;
    Country pais = null;
    while (somaQuantidades < V) {
        for (Country c : sup.keySet()) {
            for (Product p : sup.get(c)) {
                if (p.getValue().getValue() > limit && p.getValue().getValue() < lowestValue) {
                    lowestValue = p.getValue().getValue();
                    pais = c;
                }
            }
        }
        somaQuantidades = somaQuantidades + lowestValue;
        res.add(pais);
        limit = lowestValue;
        lowestValue = 99999999;
    }
    return res.size() - 1;
}
```

Esta funcionalidade vai retornar o número mínimo de países que em conjunto consegue ter uma quantidade de produção maior ou igual a Q (quantidade fornecida por parâmetro).

Primeiro é criado um *HashSet* onde vão ficar guardados os países que irão fazer parte do conjunto de países necessários. Foi escolhido o Set porque só é guardado um item e porque não queremos ter países repetidos dentro da lista, em particular o *HashSet* porque não nos interessa a ordem.

De seguida, através de dois ciclos for vamos percorrer o *sup*, que contém todos os valores do ficheiro, e encontrar a quantidade mais baixa, juntamo-lo à *somaQuantidades* e adicionamos o país à lista *res*. De seguida, e enquanto *somaQuantidades* for inferior a Q, vãos encontrar o segundo valor mais baixo, o terceiro, etc. Para isso temos de definir um *limit*, que será sempre o valor da última quantidade mais baixa encontrada.

No fim, é retornado o tamanho – 1 da lista que contém os países.

Funcionalidade 4:

O objetivo é devolver os países agrupados por maior número de anos consecutivos em que houve crescimento de quantidade de produção para a fruta passada por parâmetro. Como a ordem é insignificante e queremos associar dois dados usamos o HashMap, em que a chave é o país.

```
//Exercício 4
Map<Country, Integer> numMaxAnosConsecutivosCrescimentoFrutoF(String i) {

    Map<Country, Integer> res = new HashMap<>();
    for (Country c : sup.keySet()) {
        Map<Year, Value> temporaryList = new TreeMap<>();
        for (Product p : sup.get(c)) {
            if (i.equals(p.getItem().getItem())) {
                Year year = p.getYear();
                Value value = p.getValue();
                temporaryList.put(year, value);
            }
        }
        int n = nMaxAnosConsecutivosCrescimentoFrutoFnoPaisX(temporaryList);
        res.put(c, n);
    }
    return res;
}
```

Começamos por percorrer todos os países. Para cada um deles vamos criar uma lista temporária onde vamos guardar os anos e as quantidades de produção ordenadas por ordem crescente dos anos. Esta lista será um TreeMap, onde a chave serão os anos. Escolhemos esta opção uma vez que queremos comparar dois anos consecutivos e por isso precisamos que estejam ordenados. Nessa lista vamos adicionar todos os produtos daquele país em que o fruto seja igual ao fornecido.

```java
public int nMaxAnosConsecutivosCrescimentoFrutoFnoPaisX(Map<Year, Value> list) {
    int totalAnos = 0;
    int maxTotalAnos = 0;
    Iterator<Map.Entry<Year, Value>> i = list.entrySet().iterator();
    for (Year y : list.keySet()) {
        int valueX = list.get(y).getValue();
        int valueX1 = i.next().getValue().getValue();
        if (valueX1 > valueX) {
            totalAnos++;
        } else {
            maxTotalAnos = totalAnos;
            totalAnos = 0;
        }
    }
    return maxTotalAnos;
}
```

Depois num método à parte vamos calcular o número de anos. Através de um *iterator* vamos comparar o um ano com o seguinte e enquanto houver crescimento vamos somando os anos. Sempre que s verifica um decréscimo de produção, se o número de anos com crescimento for superior ao *maxTotalAnos* guardamos esse valor e começa uma nova soma quando se voltar a registar crescimento. Após percorrer toda a lista é retornado o número máximo de anos consecutivos em que houve crescimento da quantidade de produção.

O número de anos e o país são acrescentados à lista criada inicialmente e no fim essa lista é retornada.

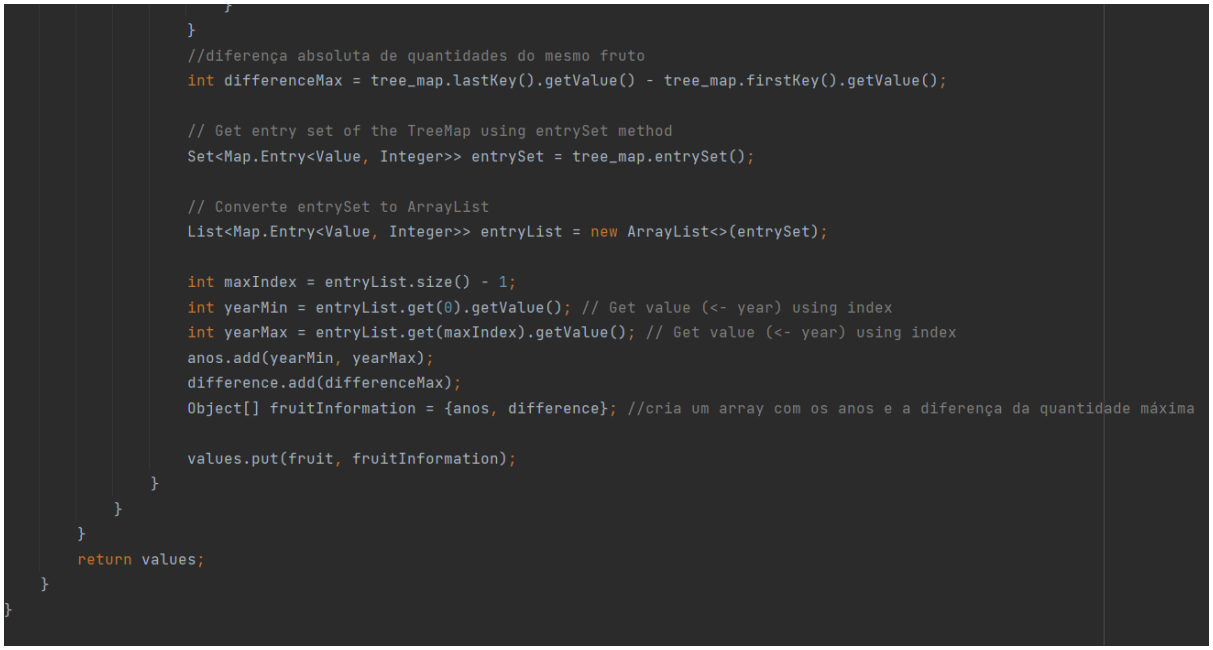
Funcionalidade 5:

Dado um país o que o método faz é o seguinte: calcula a diferença máxima de produção para cada fruto produzido naquele país e devolve o par de anos correspondente à diferença máxima e o fruto.

A estrutura de dados escolhida foi um *hasMap*, pois cada fruto para aquele país não se iria repetir não ocorrendo o risco de ser eliminado, daí a chave ser do tipo *Item* e os valores associados àquela chave são do tipo *Object[]* pois assim permite que sejam guardados duas listas de valores, uma com o par de anos e outra com a diferença da quantidade máxima. Desta forma é fácil mapear aqueles tipos de dados.

```java
//Exercício 5
Map<Item, Object[]> diferenceMaximaDasQuantidades(String country) {
    Map<Item, Object[]> values = new HashMap<>();
    TreeMap<Value, Integer> tree_map = new TreeMap<>(new SortByValue());
    List<Integer> anos = new ArrayList<>();
    ArrayList<Integer> difference = new ArrayList<>();

    for (Country c : sup.keySet()) {
        if (c.getAreaCountry().equalsIgnoreCase(country)) {
            Set<Product> sp = sup.get(c);
            Iterator<Product> it = sp.iterator();

            for (Product p : sup.get(c)) {
                Item fruit = p.getItem();
                while (it.hasNext()) {
                    Product aux = it.next();
                    if (fruit == aux.getItem()) {
                        tree_map.put(aux.getValue(), aux.getYear().getYear());
                    }
                }
                //diferença absoluta de quantidades do mesmo fruto
                int differenceMax = tree_map.lastKey().getValue() - tree_map.firstKey().getValue();

                // Get entry set of the TreeMap using entrySet method
                Set<Map.Entry<Value, Integer>> entrySet = tree_map.entrySet();

                // Converte entrySet to ArrayList
                List<Map.Entry<Value, Integer>> entryList = new ArrayList<>(entrySet);
```

Em primeiro lugar, são percorridas as chaves do *Map sup*, o qual contém os países carregados do ficheiro, depois se o país dado, ou seja, o *country*, for igual ao país percorrido, então vou usar o *Set<Product> sp* ao qual se associa um *Iterator<Product>* para guardar os valores referentes àquele país e percorre-los. A seguir é percorrido os produtos associados ao país através de um *ciclo for* com o *sup.get(c)* dentro do qual através de um *ciclo while* e do um *Iterator<Product>*, utilizado para para guardar os valores referentes àquele país, através do *Set<Product> sp,* percorre-se os anos, os frutos e as quantidades daquele país. Caso sejam frutas iguais, condição imposta pelo *if(fruit == aux.getItem())*, então é guardada a quantidade daquela fruta e o ano num tree_map que através da classe SortByValue permite ordenar os valores, que são a chave da map, por ordem crescente. Desta forma, com os valores todos ordenados, é possível ir buscar a quantidade de maior valor, que corresponde à última posição do treeMap e a quantidade de menor valor que corresponde à primeira posição. Após o cálculo, através do *Set<Map.Entry<Value, Integer>> entryList = tree_map.entrySet()* obtém-se o conjunto de entradas do *TreeMap* usando o método *entrySet* e através do List<Map.Entry<Value, Integer>> entryList = *new ArrayList<>(entrySet)* é possível converter o *entrySet* num *ArrayList* assim é possível obter o ano que corresponde à maior quantidade, o yearMax, e o ano que corresponde à menor quantidade, o yearMin.



```
                }
                //diferença absoluta de quantidades do mesmo fruto
                int differenceMax = tree_map.lastKey().getValue() - tree_map.firstKey().getValue();

                // Get entry set of the TreeMap using entrySet method
                Set<Map.Entry<Value, Integer>> entrySet = tree_map.entrySet();

                // Converte entrySet to ArrayList
                List<Map.Entry<Value, Integer>> entryList = new ArrayList<>(entrySet);

                int maxIndex = entryList.size() - 1;
                int yearMin = entryList.get(0).getValue(); // Get value (<- year) using index
                int yearMax = entryList.get(maxIndex).getValue(); // Get value (<- year) using index
                anos.add(yearMin, yearMax);
                difference.add(differenceMax);
                Object[] fruitInformation = {anos, difference}; //cria um array com os anos e a diferença da quantidade máxima

                values.put(fruit, fruitInformation);
            }
        }
    }
    return values;
  }
}
```

Alcançando o valor do par de anos correspondente à diferença máxima de produção, estes são adicionados numa *List anos* e a diferença corresponde é adicionada a um *ArrayList difference*. Para ser possível colocar estes valores no Map<Item, Object[]) que irá retornar estes valores é criado um array Object, o qual guardar a lista com os anos e o arrayList com as diferenças máximas. O values.put(fruit, fruitInformation) guardar o par de anos correspondente à máxima diferença abosulta das quantidades de produção para cada fruto presente naquele país.