Relatório do Trabalho Prático 1 de ESINF

Guilherme Daniel 1181743
Lucas Sousa 1171589

Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto
29 de Outubro de 2020

Classes definidas:

Classe **Country**- pela informação fornecida dos dados (ficheiro csv) da COVID-19, poderia concluir-se que existiam valores que nunca se alteravam: **iso-code** do país, **continente**, **nome**, **população**, **percentagem de idosos**, **índice de mortalidade por doenças cardiovasculares**, **prevalência da diabetes**, **percentagem de mulheres e homens fumadores**, **número de camas de hospital por mil habitantes** e **esperança média de vida**.

Classe **Data**- sendo que o **total de casos**, **novos casos**, **total de mortes**, **novas mortes**, **novos testes** e **total de testes** estão em constante crescimento face a cada **dia (data)**. Para esses valores mutáveis, criou-se uma classe chamada **Data**.

Classe **CovidInfo**: Para ser possível associar a informação de cada país com as suas respetivas informações e responder às alíneas pedidas, foi criada a classe CovidInfo.

Classe Main: todas as alíneas são impressas de acordo com o desejado.

1. **Carregar e guardar a informação relativa aos países e respetivos dados da pandemia COVID-19 a partir do ficheiro de texto fornecido.**

```
void readCountries(List<String> l) throws Exception {
        Country current = null;

        for (String s : l) {
            String data[] = s.split(",");
            checkNA(data);
            if (current == null || !current.getIsoCode().equals(data[0])) {

                Country c = new Country(data[0], data[1], data[2], Integer.parseInt(data[10]),
Double.parseDouble(data[11]),
                        Double.parseDouble(data[12]), Double.parseDouble(data[13]), Double.parseDouble(data[14]),
                        Double.parseDouble(data[15]), Double.parseDouble(data[16]), Double.parseDouble(data[17]));
                current = c;
                m.put(current, new TreeSet<>());

                insertData(c, data);

            } else {
                insertData(current, data);
            }
        }
    }
```

Imagem 1- Método readCountries que se destina a ler e armazenar no LinkedHashMap m

Na classe **Main** são passados os dados do ficheiro csv para uma **List** do tipo **String**, que será recebida pelo método **readCountry**, que se encarregará de distribuir a informação num Map com key do tipo **Country**, e os respetivos valores do tipo **Data**.

Para guardar a informação vamos, linha a linha, verificar se o iso-code é único a cada país. Se o for, cria-se um novo país dentro do **LinkedHashMap 'm'**, caso contrário, adiciona-se ao país com o respetivo iso-code, a **Data**.

Dentro do método recorremos a um **TreeSet** para ordenar os valores da data de Country em função da data de registo dos dados da COVID-19, sendo que não há datas repetidas.

```
LinkedHashMap<Country, Set<Data>> m;

    public CovidInfo() {
        m = new LinkedHashMap<>();
    }
```

Imagem 2- Instância do LinkedHashMap

```
public int compareTo(Data o) {
        return this.date.compareTo(o.getDate());
    }
```

Imagem 3- Método CompareTo (inserido na classe Data), para ser possível organizar a informação pela data de registo dos dados.

2. **Apresentar uma lista de países ordenados por ordem crescente do número mínimo de dias que foi necessário para atingir os 50.000 casos positivos.**

```
public List<Sorter> ordem50Mil() {
        List<Sorter> sort = new ArrayList<>();

        for (Country co : m.keySet()) {
            LocalDate initialDate = LocalDate.of(2020, Month.JANUARY, 1);

            for (Data d : m.get(co)) {
                if (d.getTotalCases() >= 50000) {

                    int dateSub = (int) Duration.between(initialDate.atStartOfDay(),
d.getDate().atStartOfDay()).toDays();

                    Sorter s = new Sorter(co, d, dateSub);
                    sort.add(s);

                    break;
                }
            }
        }
        Collections.sort(sort);

        return sort;

    }
```

Imagem 4- Método que retorna a lista ordenada, de acordo com o que é pedido na alínea 2

O método **ordem50Mil** vai percorrer o LinkedHashMap por **Country**, e por sua vez, a sua respetiva **Data**. No caso de encontrar países com o total de casos superiores a 50000, irá calcular o número de dias que foi necessário para atingir tal marco.

Para ser possível ordenar os países por ordem crescente do número mínimo de dias que foi necessário atingir os 50.000 casos positivos de **COVID-19**, foi criada uma classe **Sorter,** em que os atributos são constituídos por **Country**, **Data** e número de dias para chegar aos 50.000 casos. Com a criação desta classe, e a utilização da interface **Comparable**, organizamos a lista pela ordem pedida. A informação referente aos países é obtida através do **Map** anteriormente criado.

Por fim, retorna a lista ordenada, que na classe **Main** será impressa através dum método void ***print50mil***.

```
class Sorter implements Comparable<Sorter> {

        private Country country;
        private Data data;
        private int mindays;

        public Sorter(Country c, Data d, int mindays) {
            this.country = c;
            this.data = d;
            this.mindays = mindays;
        }

        @Override
        public String toString() {

            String s = String.format("%-15s%-20s%-30s%-20s%-10s%-15s", country.getIsoCode(), country.getContinent(),
country.getName(), data.getDate(), data.getTotalCases(), (mindays + " days"));
            String x = s.replaceAll("\"", "");
            return x;
        }

        @Override
        public int compareTo(Sorter o) {
            return this.mindays - o.mindays;
        }

    }
```

Imagem 5- Classe Sorter usada na List a ser retornada

3. **Devolver o total de novos_casos/novas_mortes por continente/mês, ordenado por continente/mês.**

```java
public Map<String, Map<Integer, Data>> continenteMes() {
        Map<String, Map<Integer, Data>> mCM = new HashMap<>();
        String currentCont = null;
        boolean flag;

        for (Country co : m.keySet()) {
            currentCont = co.getContinent();
            if (!mCM.containsKey(currentCont)) {
                mCM.put(currentCont, new HashMap());
            }
        }

        for (String s : mCM.keySet()) {

            for (int i = 1; i <= 12; i++) {
                int sumNewCases = 0, sumNewDeaths = 0;
                flag = false;
                for (Country co : m.keySet()) {
                    for (Data d : m.get(co)) {
                        if (d.getDate().getMonthValue() == i && co.getContinent().equals(s)) {
                            sumNewCases += d.getNewCases();
                            sumNewDeaths += d.getNewDeaths();
                            flag = true;
                        }
                    }
                }
                if (flag == true) {
                    Data d = new Data(LocalDate.now().toString(), sumNewCases, sumNewDeaths);

                    mCM.get(s).put(i, d);
                }
            }
        }

        return mCM;
    }
```

Imagem 6- Método que retorna os novos casos/novas mortes, por continente em cada mês

Com o intuito de resolver corretamente esta alínea, recorremos à criação de um novo **Map**, dentro de outro **Map** (nested map), com chave principal uma String que corresponde ao **Continente**, e uma segunda chave correspondente a cada mês com o valor **Data**. Essa Data é o incremento dos novos casos/novas mortes de todos os países de um dado continente. O output será ordenado por ordem alfabética consoante o continente, e ascendente relativamente ao mês.

O método retorna o **nested map** criado, que depois será transformado em String na classe **Main**, através do método ***printContinenteMes***.

4. **Devolver para cada dia de um determinado mês e para um dado continente, os países ordenados por ordem decrescente do número de novos casos positivos.**

```
public Map<Integer, LinkedHashMap<String, Integer>> novosCasos(int month, String continent) {

        LocalDate date = LocalDate.of(2020, month, 1);
        int nDays = date.lengthOfMonth();
        LinkedHashMap<String, Integer> map = new LinkedHashMap<>();
        Map<Integer, LinkedHashMap<String, Integer>> mapDays = new HashMap<>();

        for (int i = 1; i <= nDays; i++) {
            map = new LinkedHashMap<>();
            mapDays.put(i, new LinkedHashMap());
            for (Country co : m.keySet()) {
                for (Data da : m.get(co)) {

                    if (da.getDate().getDayOfMonth() == i && da.getDate().getMonthValue() == month &&
co.getContinent().equalsIgnoreCase("\"" + continent + "\"")) {
                        mapDays.get(i).put(co.getName(), da.getNewCases());
                        map.put(co.getName(), da.getNewCases());
                    }

                }
            }
            LinkedHashMap mapSorted = sortMap(map);
            mapDays.put(i, mapSorted);

        }

        return mapDays;
    }
```

Imagem 7- Método que retorna cada dia de um determinado mês e para um dado continente, os países ordenados por ordem decrescente do número de novos casos positivos.

Por parâmetro, são recebidos os atributos mês (em número) e continente.

Para a resolução deste exercício, foi criado um **Map** com um **LinkedHashMap** dentro. Como chave principal, o **Map** tem o Integer correspondente a cada dia do mês. O **LinkedHashMap**, terá o país como chave principal, e como valor os novos casos relativos a esse dia, nesse mesmo país. O método retorna o **Map** ordenado, para no **Main** ser transformado numa String e, consequentemente, impresso.

O **LinkedHashMap** vai introduzir os dados pela ordem sequencial de entrada do mesmo.

Para ser possível ordenar os países por número de novos casos, em cada dia, foi necessário criar um método **sortMap**, que irá, com recurso ao **Collection.sort**, ordenar os países (key), em função dos values (novos casos). O método retornará o **LinkedHashMap** ordenado.

```
public static LinkedHashMap<String, Integer> sortMap(LinkedHashMap<String, Integer> map) {
        List<Map.Entry<String, Integer>> lista = new LinkedList<>(map.entrySet());

        Collections.sort(lista, (o1, o2) -> o2.getValue().compareTo(o1.getValue()));

        LinkedHashMap<String, Integer> resultado = new LinkedHashMap<>();
        for (Map.Entry<String, Integer> entry : lista) {
            resultado.put(entry.getKey(), entry.getValue());
        }
        return resultado;
    }
```

Imagem 8- Método que ordena o LinkedHashMap em função dos novos casos

5. **Devolver numa estrutura adequada, todos os países com mais de 70% de fumadores, ordenados por ordem decrescente do número de novas mortes.**

```java
public List<SortNewDeaths> novasMortes() {
        LocalDate lastDate = LocalDate.of(2020, Month.SEPTEMBER, 29);
        List<SortNewDeaths> list = new ArrayList<>();

        for (Country co : m.keySet()) {
            for (Data da : m.get(co)) {
                double soma = co.getFemaleSmokers() + co.getMaleSmokers();

                if (soma > 70 && da.getDate().isEqual(lastDate)) {
                    SortNewDeaths sND = new SortNewDeaths(co.getName(), soma, da.getTotalDeaths());
                    list.add(sND);
                }
            }
        }

        Collections.sort(list);

        return list;
    }
```

Imagem 9- Método que retorna uma lista de todos os países com mais de 70% de fumadores, ordenados por ordem decrescente do número de novas mortes.

O método **novasMortes** irá verificar, no total de países, quais têm percentagem de fumadores superior a 70%. Ao efetuar tal verificação, irá adicionar a uma List do tipo SortNewDeaths, o nome do país, a percentagem de fumadores e o total de mortes.

SortNewDeaths é uma classe criada para ordenar a lista em prol do número de novas mortes para um determinado país, que tem mais do que 70% de fumadores. O método **novasMortes** irá retornar esta lista, que no **Main** irá ser transformada em String e impressa.

Por fim, é retornada a lista ordenada.

```java
class SortNewDeaths implements Comparable<SortNewDeaths> {

        String str;
        double dbl;
        int in;

        @Override
        public String toString() {
            String formatDbl = String.format("%.1f", dbl);
            String fmtPoint = formatDbl.replaceAll(",", ".");

            String st = "[" + str + " , " + fmtPoint + " , " + in + "]";
            String x = st.replaceAll("\"", "");
            return x;
        }

        public SortNewDeaths(String str, double dbl, int in) {
            this.str = str;
            this.dbl = dbl;
            this.in = in;
        }

        @Override
        public int compareTo(SortNewDeaths o) {
            return o.in - this.in;
        }

    }
```

Imagem 10- Classe SortNewDeaths que ordenará a lista em prol do total de mortes

Diagrama de Classes: