Linguagem Java

Continuação do estudo sobre coleções de dados

Ordenamento.

Interfaces Comparable e Comparator

Exercício

- Resolução do exercício 19 da ficha

19. Após o desenvolvimento da aplicação do exercício 13 e depois de conhecer melhor o modo de funcionamento de uma biblioteca, verificou-se que a aplicação desenvolvida precisa de ser expandida. A biblioteca tem livros muito antigos, cuja manipulação só é permitida aos funcionários. As pessoas que frequentam a biblioteca têm acesso apenas a reproduções destas obras antigas. Também foi verificado que todos os restantes livros (mais recentes) possuem um ISBN e existe informação sobre o seu preço, pretendendo-se que ambos sejam guardados.

Exercício

- Resolução do exercício 19 da ficha

a. Defina uma classe `OldBook` como uma especialização de um livro genérico (`Book`) que permita definir um livro antigo. Este tipo de livro possui, adicionalmente à informação base de um livro, o número de cópias existentes (`int`).
b. Defina uma classe `RecentBook` como especialização de um livro genérico (`Book`) que permita definir um livro recente. Este tipo de livro possui, adicionalmente à informação base de um livro, o código ISBN (`String`) e o preço (`double`).
c. Garanta que o pressuposto inicial, em que dois livros são considerados iguais se possuírem o mesmo código, continua a ser cumprido (independentemente de serem livros antigos ou recentes).

Exercício

- Resolução do exercício 19 da ficha

d. Com a definição dos novos tipos de livro, deixou de fazer sentido serem criados livros genéricos.
    i. Altere a classe `Book` de modo que não possam ser criadas instâncias desse objeto base, mas garantindo que o normal funcionamento da aplicação.
    ii. Altere a interface com o utilizador de modo a ser possível indicar o tipo de livro, antigo ou recente, aquando da sua criação, com a indicação dos dados adicionais necessários para caracterizar cada um deles.
e. Adicione um método `toStringSorted()` à biblioteca que permita devolver informação similar ao `toString()` previamente desenvolvido, mas garantido que os livros são listados por ordem alfabética do título. Altera a interface com o utilizador para passar a usar esta nova versão.
    i. Implemente a ordenação recorrendo à interface `Comparable<T>`
    ii. Implemente a ordenação recorrendo à interface `Comparator<T>`

Comparable<T>

- A interface `Comparable` permite definir um ordenamento por omissão para um determinado tipo de objeto
  - Esse ordenamento é usado quando é solicitado um ordenamento de elementos pertencentes a uma determinada coleção de dados

- A implementação da *interface* `Comparable<T>` permite indicar o tipo de dados com o qual se pretende fazer essa comparação
  - O habitual é escolher como tipo para comparação o próprio tipo de dados onde se implementa a *interface*
- Necessário disponibilizar o método `int compareTo(T obj)` que deverá retornar um inteiro
  - <0, quando o próprio objeto (`this`) é considerado menor do que `obj`
  - 0, se for considerado igual a `obj`
  - >0, quando o próprio objeto (`this`) é considerado maior do que `obj`

Comparable<T>

- Ex.:

```
public class Book implements Comparable<Book>{
   // ...

   @Override
   public int compareTo(Book o) {
      return id - o.id;
      //Another example: return title.compareTo(o.title);
   }
}

// ...
ArrayList<Book> lstBooks = new ArrayList<>(originalCollection);
// ...
Collections.sort(lstBooks);
// ...
```

Comparator<T>

- A interface `Comparator` , em comparação com a `Comparable`, permite definir um número mais alargado de ordenamentos
  - Normalmente definido através da criação de um nova classe, com um nome representativo
  - Tal como a interface `Comparable<T>`, permite definir o tipo de dados cujas instâncias se pretendem comparar

- Necessário disponibilizar o método

```
int compare(T obj1, T obj2)
```

  ... que deverá retornar um inteiro:
  - <0, quando `obj1` é considerado menor do que `obj2`
  - 0, se `obj1` for considerado igual a `obj2`
  - >0, quando `obj1` é considerado maior do que `obj2`

Comparator<T>

- Ex.:

```
public class BookComparator implements Comparator<Book> {

    @Override
    public int compare(Book o1, Book o2) {
        return o1.getTitle().compareTo(o2.getTitle());
    }

}

// ...
ArrayList<Book> lstBooks = new ArrayList<>(originalCollection);
// ...
Collections.sort(lstBooks, new BookComparator());
// ...
```