Aula Prática 9

Objetivos

Composição versus Herança.
Utilização alguns padrões de software: Decorador, Singleton e Iterador.

Problema 9.1

Pretende-se construir um Scanner especial (*ScannerAbeirense*) que forneça apenas alguns dos métodos da classe Scanner e que na leitura transforme todos os caracteres 'v' e 'V' em 'b' e 'B', respectivamente.

```
public class ScannerAbeirense implements Iterator<String>, Closeable
```

Deverá fornecer os métodos das interfaces indicadas e ainda o método nextLine().
Teste a classe lendo do teclado e de um ficheiro de texto.

Problema 9.2

Considere a seguinte classe:

```
public class Gelataria {
   public static void main(String args[]) {
      Gelado ice;
      ice = new GeladoSimples("Baunilha");
      ice.base(2);
      new Copo(ice).base(3);
      new Cone(ice).base(1);
      new Topping(ice, "Canela").base(2);
      ice = new Topping(ice, "Nozes");
      ice.base(1);
      ice = new Topping(new Cone(new GeladoSimples("Morango")), "Fruta");
      ice.base(2);
      ice = new Topping(
              new Topping(
                 new Copo(new GeladoSimples("Manga")), "Chocolate"), "Natas");
      ice.base(4);
      ice = new Topping(ice, "Pepitas");
      ice.base(3);
   }
}
```

Desenvolva classes que representem adequadamente o problema e que para o programa anterior forneça a seguinte saída de dados:

```
 2 bolas de gelado de Baunilha
 3 bolas de gelado de Baunilha em copo
 1 bola de gelado de Baunilha em cone
 2 bolas de gelado de Baunilha com Canela
 1 bola de gelado de Baunilha com Nozes
 2 bolas de gelado de Morango em cone com Fruta
 4 bolas de gelado de Manga em copo com Chocolate com Natas
 3 bolas de gelado de Manga em copo com Chocolate com Natas com Pepitas
```

Problema 9.3

Com base nas classes Pessoa, Data e ListaPessoas desenvolvidas durante as primeiras aulas construa:

a) a classe VectorPessoas que gere uma lista de Pessoas com base num vector que cresce dinamicamente. Inclua os métodos:

```
boolean addPessoa(Pessoa p)
boolean removePessoa(Pessoa p)
int totalPessoas()
```

b) Acrescente à classe VectorPessoas a classe interna VectorIterator que implementa a interface Iterador :

```
public interface Iterator {
      boolean hasNext();
      Object next();
      void remove();
}
```

c) Crie o método Iterator iterador() que cria um iterador a apontar para a primeira posição do vector interno.

d) Repita estes procedimentos para uma nova classe ListaPessoas onde o armazenamento de Pessoas é feito à custa de uma lista ligada.

e) Teste as classes desenvolvidas com um código do tipo:

```
public abstract class TesteIterador {

   public static void main(String[] args) {
      VectorPessoas vp = new VectorPessoas();
      for (int i=0; i<10; i++)
         vp.addPessoa(new Pessoa("Bebé no Vector "+i,
                        1000+i, Data.today()));
      Iterator vec = vp.iterator();
      while ( vec.hasNext() )
         System.out.println( vec.next() );
      ListaPessoas lp = new ListaPessoas();
      for (int i=0; i<10; i++)
         lp.addPessoa(new Pessoa("Bebé na Lista "+i,
                        2000+i, Data.today()));
      Iterator lista = lp.iterator();
      while ( lista.hasNext() )
         System.out.println( lista.next() );
   }
}
```

f) Acrescente às soluções anteriores o seguinte iterador e adapte o *main* para testar também esta solução:

```
public interface BFIterator {
      boolean hasPrevious();
      Object previous();
      boolean hasNext();
      Object next();

}
```