Programando em Java
(Iteração com filtros)

Mestrado Integrado em Engenharia Informática FCT UNL
http://ctp.di.fct.unl.pt/miei/ip/

Corpo Docente 2020/2021
António Ravara, Artur Miguel Dias, Bernardo Toninho,
Ema Vieira, Inês Fernandes, Margarida Mamede,
Miguel Monteiro, Rui Nóbrega

Relembre o problema do Laboratório

O laboratório ***X*** gere projectos de cientistas; cada projecto necessita de vários investigadores; cada investigador tem um *email* único e pode propor projectos no laboratório; cada projecto é identificado por um número único e é caracterizado por uma descrição, pelo *email* do seu proponente (que é um investigador), pela quantidade de investigadores colaboradores necessários no projeto e pelos *emails* dos colaboradores efectivos (os colaboradores que existem em cada momento).

Nenhum investigador pode ser simultaneamente proponente e colaborador no mesmo projeto.

Classes relevantes

"O laboratório ***X*** gere projectos de cientistas; cada projecto necessita de vários investigadores"

Claramente, são entidades:

- o laboratório, que contém projectos
- os projectos, que contém colaboradores (outros investigadores, além do proponente)
- os investigadores (são só Strings - o *email*)

Pretende-se listar:

- todos os projectos do laboratório
- todos os projectos do laboratório **propostos por dado investigador**
- todos os projectos do laboratório **com vagas para colaboradores**

Listagens

1. Todos os projectos do laboratório - um simples iterador (classe `IteratorProjs`)
2. Todos os projectos do laboratório propostos por dado investigador
   - “filtram-se” os projectos, procurando os que têm como proponente o investigador em causa
3. Todos os projectos do laboratório com vagas para colaboradores
   - “filtram-se” os projectos, procurando os que têm vagas para colaboradores

Listagens - todos os projectos de dado investigador

Uma hipótese é:

- fazer a filtragem no método da classe Laboratorio, construindo um vector auxiliar só com os projectos propostos pelo investigador em causa (precisamos dum contador de elementos nesse vector)

```
int countP = 0; // contador de projectos do investigador

Proj[] projectsProp = new Proj[count]; // vector que conterá os projectos

// no máximo, terá todos os projectos (podia-se contar antes quantos projecto tem
// dado investigador, mas para poupar espaço, gastava-se tempo)

for (int i = 0; i < count; i++) // selecciona-se os projectos do investigador

    if (projects[i].getProp().equals(prop))

        projectsProp[countP++] = projects[i];
```

- passar esse vector e o contador de elementos nesse vector ao iterador "normal"

```
return new IteratorProjs(projectsProp, countP);
```

Listagens - todos os projectos de dado investigador

```
// @pre findProjProp(prop) - para nao passar ao iterador um vector sem elementos
public IteratorProjs listProjectsProp(String prop) {
    int countP = 0;
    Proj[] projectsProp = new Proj[count];
    for (int i = 0; i < count; i++)
        if (projects[i].getProp().equals(prop))
            projectsProp[countP++] = projects[i];
    return new IteratorProjs(projectsProp, countP);
    }
```

Listagens - todos os projectos com vagas para colaboradores

Fazemos agora de outra forma:

- construímos outro iterador (classe `ItProjsAvalPos`)
- o método `next` devolve o projecto corrente que tem vagas e "salta" para o próximo com vagas, se houver

```
Proj toReturn = projects[pointer++]; // guarda o corrente (que tem vagas)
moveToNext(); // avança o pointer ate' ao proximo projecto com vagas (se existir)
return toReturn; // devolve o corrente
```

- Como "saltar" nos projectos até ao próximo com vagas?

  ciclo que avança enquanto houver projectos e o corrente não tiver vagas

```
private void moveToNext() {
    while ( pointer < count && !projects[pointer].hasPositions() )
        pointer++;
}
```

- Como garantir que o método `next` devolve sempre um projecto com vagas?

  o construtor tem que avançar o `pointer` até ao primeiro que tenha (se houver)

Listagens - todos os projectos com vagas para colaboradores

```
public class ItProjsAvalPos {
    private Proj[] projects;
    private int count, pointer;

    public ItProjsAvalPos(Proj[] projects, int count) {
     this.projects = projects; this.count = count; pointer = 0;
     moveToNext();
    }

    public boolean hasNext() { return pointer < count; }

    private void moveToNext() {
     while ( pointer < count && !projects[pointer].hasPositions() ) pointer++;
    }

    // @pre hasNext()
    public Proj next() { Proj toReturn = projects[pointer++]; moveToNext();
         return toReturn;
     }

}
```