Programação Orientada pelos Objectos

2º Teste (Duração 2h)

MIEI 2016/2017

Instruções:

- Antes de começar a resolver, **leia o enunciado do princípio até ao fim**.
  - As interfaces e classes do grupo de I e II têm mais métodos do que os que deverá implementar na resolução deste teste. Em cada grupo, tenha o cuidado de **ver com muita atenção quais os métodos que deve implementar**, para **não desperdiçar o seu tempo a implementar métodos que não lhe são pedidos**.
  - Disponibilizamos a descrição sumária de todos os métodos, incluindo os que não tem de implementar, para que os possa usar na sua resolução.
- **Pode** usar caneta ou lápis.
- Não é permitido consultar quaisquer elementos para além deste enunciado.
- **Responda a grupos diferentes em folhas diferentes.**

Introdução aos problemas para os grupos I, II, III e IV:

Nestes 4 grupos vamos implementar parcialmente algumas das classes necessárias à construção de um serviço Web que permita gerir os estágios de verão de um programa nacional para a promoção da ciência pelos alunos das escolas secundárias. No primeiro grupo, faremos a implementação de uma classe que representa a colecção de estágios com uma lista. No segundo grupo, faremos a implementação de uma classe que representa a colecção de estágios com outras colecções do Java. Quer no primeiro, quer no segundo grupo, usamos as classes e interfaces especificadas na primeira parte enunciado. No terceiro grupo é pedida a implementação de um método adicional. No quarto grupo, realizamos alguns testes unitários e praticamos o uso de asserções.

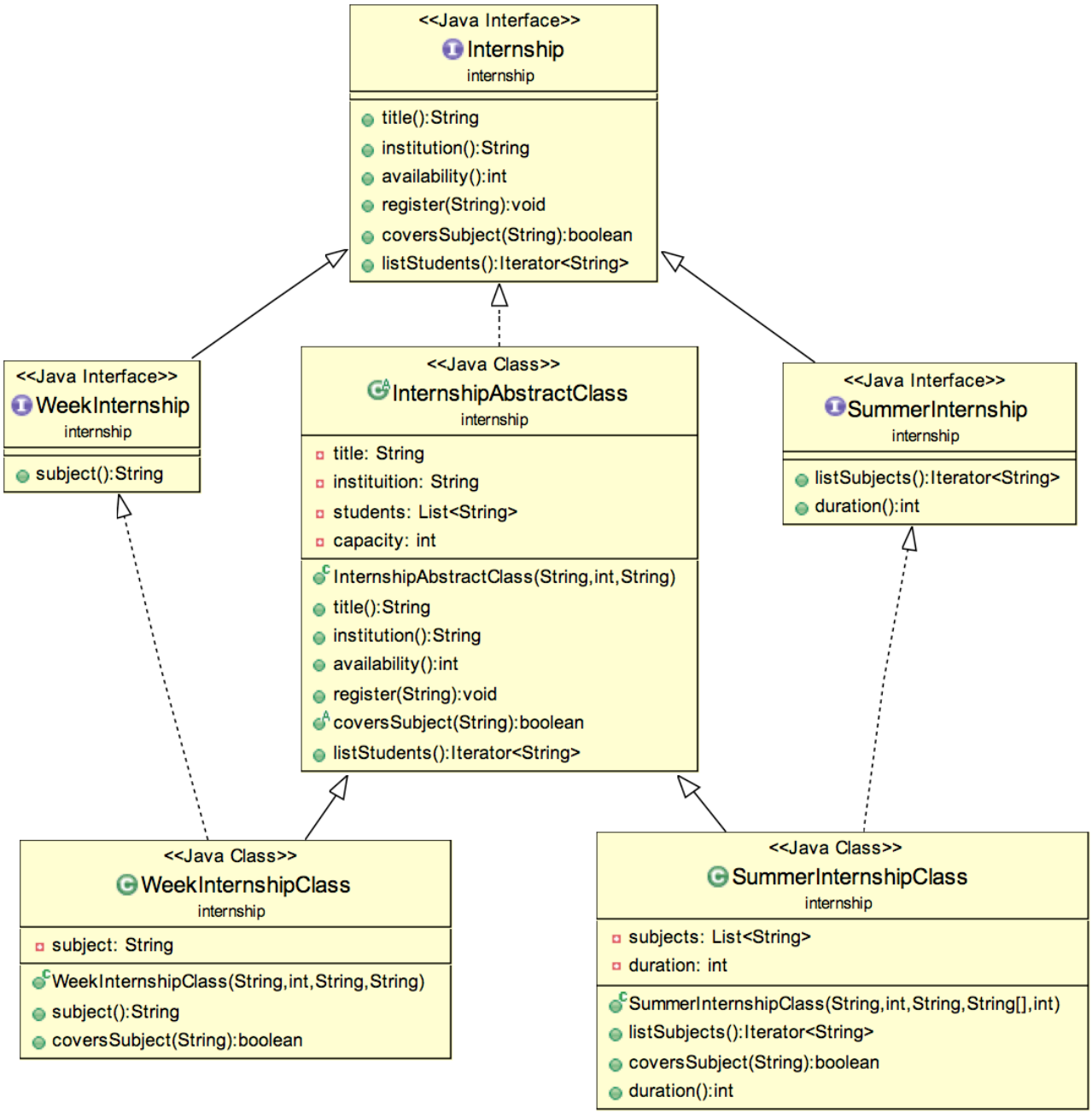

Figura 1: Estágios científicos

A interface `Internship` representa um estágio científico para alunos do ensino secundário. Os objectos da classe `InternshipAbstractClass` mantêm informação sobre o estágio. Na interface `Internship`:

- `title` devolve uma `String` com o título do estágio (identificador único).
- `institution` devolve uma `String` com a instituição responsável pelo estágio (e.g. "NOVA").
- `availability` devolve um inteiro que representa as vagas existentes no estágio.
- `register` regista um novo aluno no estágio com base no email recebido como parâmetro. Esta operação pode lançar as excepções: `StudentAlreadyExistsException` no caso desse aluno já estar registado, `InternshipFullException` no caso do estágio já estar completo.
- `coversSubject` devolve `true` se o tema recebido como argumento é o tema, ou um dos temas, do estágio e `false` caso contrário.

- `listStudents` devolve um iterador dos alunos registados no estágio.

A classe abstracta `InternshipAbstractClass` implementa a interface `Internship`. O construtor desta classe recebe o título, a capacidade e a instituição do estágio. Os métodos implementados nesta classe obedecem à especificação já apresentada durante a descrição da interface. O método abstracto `coversSubject` é implementado nas subclasses `WeekInternshipClass` e `SummerInternshipClass`.

A classe abstracta é especializada em duas classes concretas que representam os dois tipos de estágios disponíveis: estágios semanais sobre um dado tema (e.g. sequenciação de DNA); e estágios de verão de duração variável sobre um conjunto de temas.

A interface `WeekInternship`, que representa um estágio semanal, estende a interface `Internship` com um método adicional:

- `subject` devolve o tema do estágio.

Esta interface é implementa pela classe `WeekInternshipClass`. O construtor recebe os argumentos do construtor da superclasse abstracta, para além do tema do estágio.

A interface `SummerInternship`, que representa um estágio de verão, estende a interface `Internship` com dois métodos adicionais:

- `listSubjects` devolve um iterador para os vários temas do estágio.
- `duration` devolve a duração em dias do estágio.

Esta interface é implementa pela classe `SummerInternshipClass`. O construtor recebe os argumentos do construtor da superclasse abstracta, para além de um vector com os temas do estágio e a sua duração em dias.

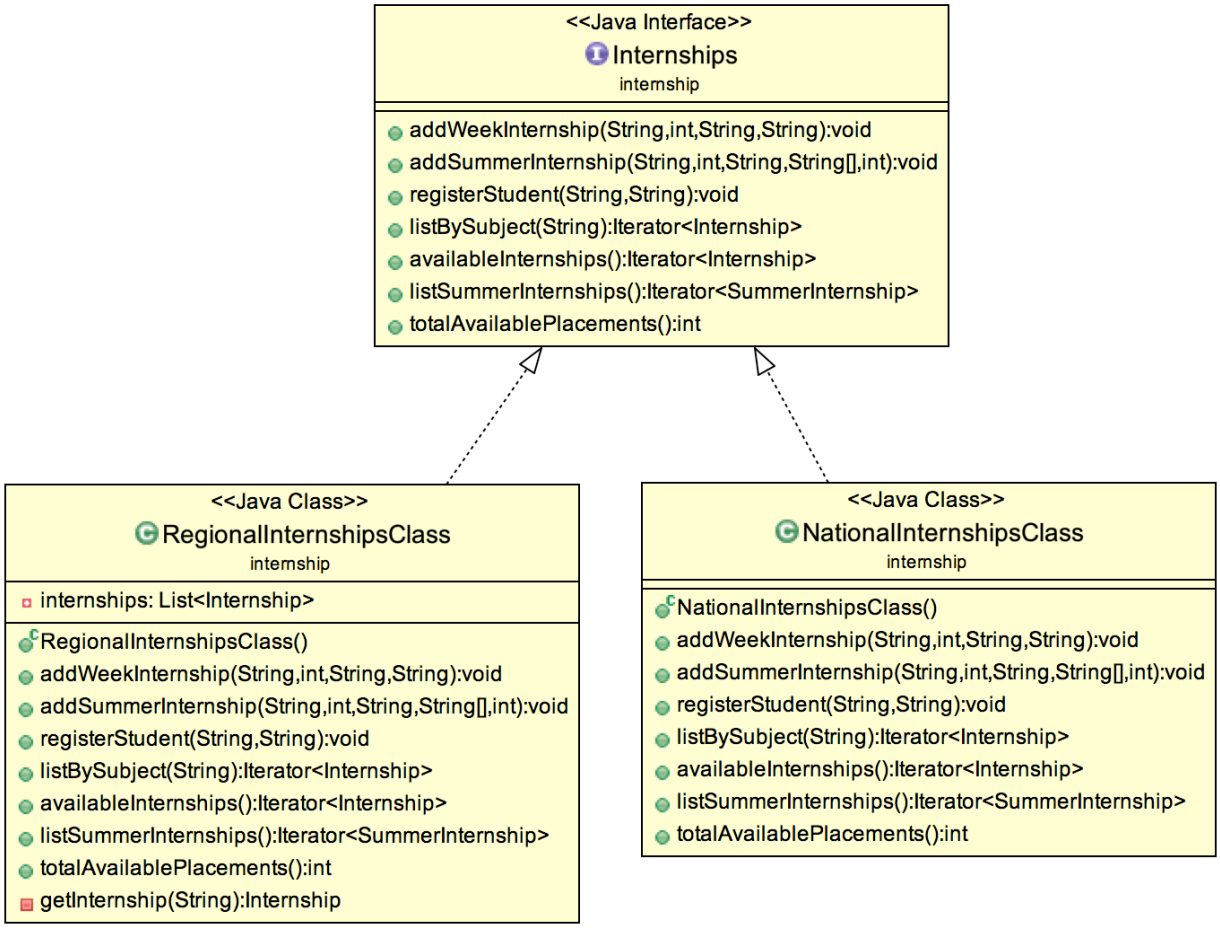

Figura 2: Colecção de estágios nacionais e regionais

A Fig. 2 apresenta a interface e as classes necessárias à resolução dos grupos I e II. A interface `Internships` representa uma colecção de estágios e tem os seguintes métodos:

- `addWeekInternship` adiciona um novo estágio semanal, recebendo, por esta ordem, o título, a capacidade, a instituição e o tema do estágio. A operação lança a excepção `InternshipAlreadyExistsException` se já existir um estágio com o título dado.
- `addSummerInternship` adiciona um novo estágio de verão, recebendo, por esta ordem, o título, a capacidade, a instituição, o vector com os temas do estágio e a duração em dias. A operação lança a excepção `InternshipAlreadyExistsException` se já existir um estágio com o título dado.
- `registerStudent` regista um novo aluno num estágio e recebe o título do estágio e o email do aluno. Esta operação pode lançar as excepções: `InternshipDoesNotExistException` no caso do estágio não existir, `StudentAlreadyExistsException` no caso desse aluno já estar registado e `InternshipFullException` no caso do estágio já estar completo.
- `listBySubject` devolve um iterador para todos os estágios sobre o tema recebido como argumento. Os estágios devem ser ordenados por ordem alfabética de título. Caso não existam estágios nesse tema, o método deve lançar a excepção `SubjectDoesNotExistException`.
- `availableInternships` devolve um iterador para todos os estágios com vagas ainda disponíveis. Os estágios devem ser ordenados por ordem decrescente de vagas e depois por ordem alfabética de título. Caso não existam estágios com vagas disponíveis, o método deve lançar a excepção `NoAvailableInternshipsException`.
- `listSummerInternships` devolve um iterador para todos os estágios de verão sem nenhuma ordem em particular. Caso não existam estágios de verão, o método deve lançar a excepção `NoSummerInternshipsException`.
- `totalAvailablePlacements` devolve o número total de vagas disponíveis em todos os estágios.

Grupo I – Colecção pequena (`RegionalInternshipsClass`)

A classe `RegionalInternshipsClass` destina-se à gestão de estágios locais (na ordem das dezenas) pelo que vamos usar uma lista denominada `internships` para guardar a colecção de estágios. Tendo em conta que neste grupo não pode adicionar variáveis de instância à classe, e que tem apenas acesso à variável `internships`, implemente os seguintes métodos:

a) O construtor de modo a que, ao ser criada a lista, esteja vazia. Apresente também a declaração da variável `internships` (coloque a declaração acima do construtor).

b)
```
void addWeekInternship(String title, int capacity, String institution,
      String subject) throws InternshipAlreadyExistsException;
```

c)
```
void registerStudent(String title, String email)
     throws InternshipDoesNotExistException,
            StudentAlreadyExistsException, InternshipFullException;
```

d)
```
Iterator<Internship> listBySubject(String subject)
      throws SubjectDoesNotExistException;
```

e)
```
Iterator<SummerInternship> listSummerInternships()
      throws NoSummerInternshipsException;
```

Grupo II – Colecção grande (`NationalInternshipsClass`)

Pretende-se implementar uma classe para a gestão de estágios a nível nacional (na ordem dos milhares). A implementação da classe deve ter em conta a eficiência das **pesquisas por título do estágio**, a eficiência de **todas as listagens** e a eficiência do **método** `totalAvailablePlacements`.

Na alínea a) deve definir as variáveis de instância e nas seguintes alíneas deve implementar alguns dos métodos da classe `NationalInternshipsClass`:

a) Defina as variáveis de instância e escolha as estruturas de dados que achar mais adequadas de acordo com os requisitos de eficiência mencionados acima.

b) Defina o construtor de modo a que, ao ser criado não existam estágios.

c)
```
void addWeekInternship(String title, int capacity, String institution,
      String subject) throws InternshipAlreadyExistsException;
```

d)
```
void registerStudent(String title, String email)
     throws InternshipDoesNotExistException,
            StudentAlreadyExistsException, InternshipFullException;
```

f)
```
Iterator<Internship> availableInternships()
      throws NoAvailableInternshipsException;
```

Assuma que está disponível uma classe `AvailabilityComparator` (que implementa a interface `Comparator<Internship>`) que estabelece uma relação de ordem entre estágios com base nas vagas disponíveis e depois por ordem alfabética de título.

e)
```
int totalAvailablePlacements()
```

Grupo III

Considere que existe um método estático adicional da classe `NationalInternships`:

```
public static int containsAllStudents(Internship i1, Internship i2).
```

Este método verifica se o estágio `i1` contém todos os alunos do estágio `i2`, não obrigatoriamente pela mesma ordem de registo, e devolve o primeiro número de registo de um aluno de `i2` no estágio `i1`. Caso o estágio `i1` não contenha todos os alunos do estágio `i2`, o método devolve o valor zero.

Implemente este método.

Exemplo:

Alunos estágio `i1 ["joao@fct.pt", "matias@fct.pt", "ana@fct.pt", "luis@fct.pt"]`

Alunos estágio `i2 ["luis@fct.pt", "matias@fct.pt"]`

Alunos estágio `i3 ["luis@fct.pt", "ines@fct.pt"]`

A invocação `containsAllStudents(i1, i2)` iria devolver o valor 2 porque `"matias@fct.pt"` foi o segundo a registar-se em `i1`, enquanto o `"luis@fct.pt"` foi o quarto a registar-se estágio `i1`.

A invocação `containsAllStudents(i1, i3)` devolveria o valor 0, porque `"ines@fct.pt"` não está registada no estágio `i1`.

Grupo IV – Testes unitários e asserções

a) Implemente uma operação de teste ao método estático `containsAllStudents` da classe `NationalInternships` apresentado no Grupo III, usando o JUnit. O seu teste deve construir estágios de diferentes tipos (de verão e semanais) e testar quatro casos de utilização distintos (o exemplo do Grupo III apresenta dois casos de teste). Para cada um dos casos de utilização verifique que obtém o resultado esperado.

b) Considere o seguinte método:

```
public int foo(int y)
{
      return y * 2;
}

public void bar(int z)  {
      assert foo(z) != 10 : z;

      switch (z % 2)
      {
      case 0: System.out.println("Zero"); break;
      case 1: System.out.println("One"); break;
      default: System.out.println("Other"); assert false : z % 2;
      }
}
```

Assumindo que o mecanismo de asserções está activo, diga qual o output produzido na consola pelo método `bar` quando chamado com cada um dos seguintes valores: 5, 4, -4, 0, 7, -7.

Algumas das interfaces abaixo reproduzidas poderão ser úteis na resolução deste teste: