Exercício 5

Expressões Regulares e Testes

Cadeira de
Laboratórios de Programação

2021/2022

Objetivos

- Uso de expressões regulares para reconhecimento de padrões;
- Criação de testes.

Antes de Começar

De modo a poder realizar este exercício, constituído por 2 partes, deverá ter lido os guiões sobre expressões regulares e construção de testes.

O que fazer

Parte I.

Comece por descarregar o arquivo *alunosExercicio5.zip* da página de LabP. Nele pode encontrar:

- O ficheiro *RunRecognisePatterns.java* que deverá ser usado para verificar o comportamento pedido da classe *RecognisePatterns* a ser desenvolvida neste exercício;
- O ficheiro *TestEmail.java* que contém um esqueleto parcialmente preenchido, de uma classe de testes JUnit, com o javadoc da classe e dois exemplos de testes concretos.
- Os ficheiros *emails.txt*, *IPs.txt*, *dates.txt* e *parkingReservations.txt* com exemplos de input para os métodos da classe *RecognisePatterns.java*.
- O ficheiro *expectedOutput.txt* que contém o output esperado pela execução da classe *RunRecognisePatterns.java* com os ficheiros de input acima.

Crie uma classe *RecognisePatterns* que inclua a definição dos seguintes métodos:

1. `public static boolean isValidStudentEmail(String s)` – recebe uma *string* s e verifica se s é um endereço de email de alunos da FCUL válido. Lembre-se que o formato esperado para este tipo de endereço é o seguinte: fcXXXXX@alunos.fc.ul.pt onde cada X é um algarismo entre 0 e 9.

   Este método só deve retornar true se a string s for um endereço válido de aluno da FCUL.

2. `public static boolean isValidIPAddress(String s)` – recebe uma *string* s e verifica se s representa um endereço IPv4 válido. Lembre-se que um endereço IPv4 é representado por um conjunto de quatro números entre 0 e 255 separados por pontos "a.b.c.d", por exemplo, 192.168.1.10. Assim, o intervalo de endereços válidos vai do endereço 0.0.0.0 ao endereço 255.255.255.255.
   Este método só deve retornar true se a string s for um endereço IPv4 válido.

3. `public static String matchDate(String s)` – recebe uma string *s* e verifica se s contém alguma data no formato *`NomeMês Dia, Ano`*, por exemplo, *`Jan. 6, 1983`*,

onde *nomeMês* é uma abreviatura (constituída pelas 3 primeiras letras e terminada por um ponto) do nome do mês em inglês (por exemplo, Jan., Feb., Mar., etc.), e *Ano* representa um ano entre 19XX e 20XX.

Este método deve retornar uma string que contenha a primeira sequência de caracteres com a forma indicada acima, ou null se a string *s* não tiver uma data no formato indicado.

4. `public static boolean verifyParkingReservations(String s)` – recebe uma string s que contém uma reserva de um único veículo para um parque de estacionamento (reserva tal como descrita no projeto 3) e verifica se a reserva tem o formato esperado.
   Uma reserva deve ter um formato "matricula-tipo (hh:mm,x.y)". A falta de um ou mais dos quatro elementos da descrição, ou seja, matricula, tipo, hh:mm, x.y, representa uma descrição mal formatada. Os elementos seguem as seguintes regras:

   a. A *matricula* deve ser composta por 6 carateres, 4 algarismos seguidos de duas letras minúsculas.

   b. O *tipo* deve ser composto de, no mínimo, três e no máximo vinte letras minúsculas.

   c. A hora *hh:mm* deve ser composta por 5 caracteres, dois para indicar a hora, dois para indicar os minutos e os dois pontos ":" como separador. As horas podem ter valores entre 00 e 23, os minutos podem ter valores entre 00 e 59.

   d. O comprimento *x.y* deve ser um número decimal composto por apenas dois algarismos separados por um ponto.

   e. Entre o par *matricula-tipo* e o par *(hh:mm,x.y)* é aceite apenas um espaço, e os outros separadores "-" "(" "," ")" devem estar todos presentes em uma reserva bem formatada.

   Este método devolve o valor true quando a representação seja bem formatada e false no caso contrário.

Na implementação dos métodos acima deve-se usar expressões regulares através das classes *java.util.regex.Pattern* e *java.util.regex.Matcher* da API do Java.
É dada uma classe *RunRecognisePatterns* que já inclui um método *main* para ler os vários ficheiros de input e usar os métodos da classe *RecognisePatterns* pedidos acima. Também é dado o output esperado na consola com a execução deste programa num ficheiro "expectedOutput.txt" que se encontra dento do arquivo alunosExercicio5.zip.

Parte II.

Construa 4 classes de testes, que permitam testar os métodos da classe `RecognisePatterns.java` usando testes `JUnit`.

Consideremos, por exemplo, o primeiro método da lista anterior, `isValidStudentEmail`. O endereço aluno FCUL válido tem que começar pelas letras minúsculas "*fc",* depois conter 5 dígitos e terminar com a sequência de caracteres "@alunos.fc.ul.pt". Segundo o indicado no guião de testes, temos que ver quais as características úteis e a partir delas definir regiões que particionem o espaço:

Características	**Nome da característica**	**Regiões**
o nome começa pelas duas letras *fc*	**s** (de **s**tarts with)	true ou false (sT, sF)
o nome tem 5 dígitos	**d** (de contains **d**igits)	true ou false (dT, dF)
o nome termina com a sequencia @alunos.fc.ul.pt	**e** (de **e**nds with)	true ou false (eT, eF)

Poderíamos ter considerado mais (ou menos) características.

Quanto mais características considerarmos, maior será o número de combinações de regiões a testar. Temos que optar por um conjunto de características que seja razoável, isto é, que nos permita testar os vários aspetos considerados relevantes sem, no entanto, ser excessivo.

Para o método `isValidStudentEmail`, com as 3 características indicadas acima, o número total de combinações é 8 (=2*2*2) que estão listadas na tabela seguinte :

Combinação de regiões	**Caso de teste**	**Resultado esperado**
(sT, dT, eT)	“fc12345@alunos.fc.ul.pt”	true
(sT, dT, eF)	“fc12345@alunos.fc.ul”	false
(sT, dF, eT)	“fc1234@alunos.fc.ul.pt”	false
(sT, dF, eF)	“fc1345@alunos.fc.pt”	false
(sF, dT, eT)	“cc12345@alunos.fc.ul.pt”	false
(sF, dT, eF)	“cc12345@alunos.ul.pt”	false
(sF, dF, eT)	“cc1234@alunos.fc.ul.pt”	false
(sF, dF, eF)	“cc1245@alunos.fc.pt”	false

A partir da tabela acima poderíamos então definir os métodos de teste. Por exemplo, para o 1º e 3º casos de teste:

```
@Test
/**
* Test an email showing the right format
*   starts with the lower-case sequence 'fc': true
*   contains 5 digits : true
*   ends with the sequence '@alunos.fc.ul.pt': true
*/
public void testIsValidStudentEmail1 () {
     assertTrue(RecognisePatterns.
          isValidStudentEmail(”fc12345@alunos.fc.ul.pt”));
}

@Test
/**
* Test an email with the wrong number of digits
*   starts with the lower-case sequence 'fc': true
*   contains 5 digits : false
*   ends with the sequence '@alunos.fc.ul.pt': true
*/
public void testIsValidStudentEmail3 () {
  assertFalse(RecognisePatterns.
          isValidStudentEmail (”fc1234@alunos.fc.ul.pt”));
}
```

Deve proceder da mesma forma para os outros métodos a testar. Para cada método desenvolvido na classe *RecognisePatterns*, pede-se para criar uma classe de Teste que apresenta apenas **4** dos testes resultantes das características que escolheu.
E pede-se ainda que:
(i) os vários testes para um método devem considerar as mesmas características;
(ii) o javadoc de cada um desses testes deve deixar absolutamente claro qual a combinação de regiões (dessas características) que está a ser testada por esse teste.

Note que o javadoc de cada uma dessas classes deve indicar explicitamente quais as características consideradas e as suas regiões. Deve ainda indicar o número total de combinações viáveis de regiões, resultantes dessas características. Como indicado acima, só precisará de implementar 4 desses testes (à sua escolha).

Damos o esqueleto da classe *TestEmail.java*, já parcialmente preenchida com o respetivo javadoc da classe e os dois exemplos de teste que são apresentados aqui no enunciado.
Sugerimos que adaptem esse esqueleto para as outras 3 classes de testes pedidas, que devem chamar-se *TestIP.java*, *TestDate.java* e *TestReservation.java*.

Entrega

Deve entregar um *zip* de nome `E5fcxxxxx.zip`, onde `xxxxx` é o seu número de aluno, contendo somente os ficheiros **`RecognisePatterns.java, TestEmail.java, TestIP.java, TestDate.java`** e **`TestReservation.java.`**

Data e hora limite: 13 de Maio às 23h55

ATENÇÃO: Antes de submeter o trabalho, verifique que documentou cada um dos métodos da sua classe e incluiu o valor da *tag* `@author` com o seu número de aluno, no comentário da classe.