

Programação III (PG III)

Semestre de Verão de 2021-2022

1º Trabalho prático

Data de Entrega: 22 de Novembro de 2021

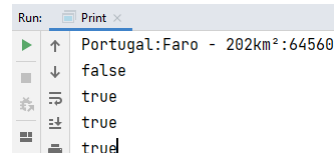
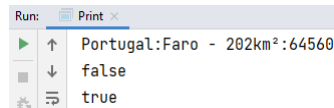
OBJETIVOS: Implementar aplicações simples usando o paradigma da Programação Orientada por Objectos.

NOTA: tem que constar todo o código desenvolvido, incluindo os testes unitários que permitem validar a correcção dos métodos e classes realizadas.

Grupo 1

1. Tendo em conta a listagem de código Java:

- Indique o resultado da execução do programa. Justifique a sua resposta.
- Acrescente o que considerar necessário à classe `City` para que o programa apresente na consola o resultado mostrado na figura. Justifique as alterações.
- Retire o comentário de bloco, o resultado da escrita da expressão deve ser `false`.
- Afete a variável `c3` para que o programa apresente na consola o resultado mostrado na figura ao lado.
- Altere a escrita no `standard output`:
`System.out.println(c1);`
Indique e justifique o resultado da execução explicitando o mecanismo usado.



```
public class City {  
    private final String name, country;  
    private int population;  
    private final int area; // Em km²  
    public City(String nm, int p, int a){  
        this.name = nm; this.contry= "Portugal";  
        this.population = p; this.area = a;  
    }  
}
```

```
public class Print {  
    public static void main(String[] args) {  
        City c1= new City("Faro", 64560, 202);  
        System.out.println( c1.toString() );  
        City c2= new City("Faro", 64560, 202);  
        System.out.println( c1 == c2 );  
        System.out.println( c1.equals( c2 ) );  
        /* City c3= null;  
        System.out.println( c1.equals( c3 ) );  
        if ( c3 != null )  
            System.out.println(c1==c3); */  
    }  
}
```

2. Complete a classe `City`, tendo em conta que deve disponibilizar:

- Construtor com quatro parâmetros o país, o nome, a área e a população.
- Os métodos de instância para obter o nome, o país, a área e a população (*getters*).
- O método instância `populationDensity` que retorna o número de pessoas por km².
- O método de instância `populationChange` que atualiza a população tendo em conta a taxa bruta de crescimento populacional que recebe por parâmetro. Dada uma determinada taxa de crescimento (*rate*) a população no final do ano é obtida pela seguinte fórmula: $\text{newPopulation} = \text{oldPopulation} * e^{\text{rate}}$.
- O método de instância `compareTo` que define a relação de ordem, considerando as áreas, sobre as instâncias da classe `City`. Sejam `c1` e `c2` dois objetos do tipo `City` e `x` um valor inteiro tal que `x = c1.compareTo(c2)`. Se:
 - $x < 0$, significa que a área de `c1` é inferior à área de `c2`;
 - $x > 0$, significa que a área de `c1` é superior à área de `c2`;
 - $x = 0$, significa que a área de `c1` é igual à área de `c2`.
- O método estático `getCity` que recebendo por parâmetro uma instância de `java.lang.String` retorna a correspondente instância de `City`. O formato da *string* recebida por parâmetro é:
`<param>::= <country> ':' <name> '-' <area> 'km²:' <population>`

Usar os métodos de instância da classe `java.lang.String`:

- `indexOf(int ch, int fromIndex)` – para obter os índices dos caracteres de separação;
- `substring(int beginIndex, int endIndex)` – para obter o nome e a área;

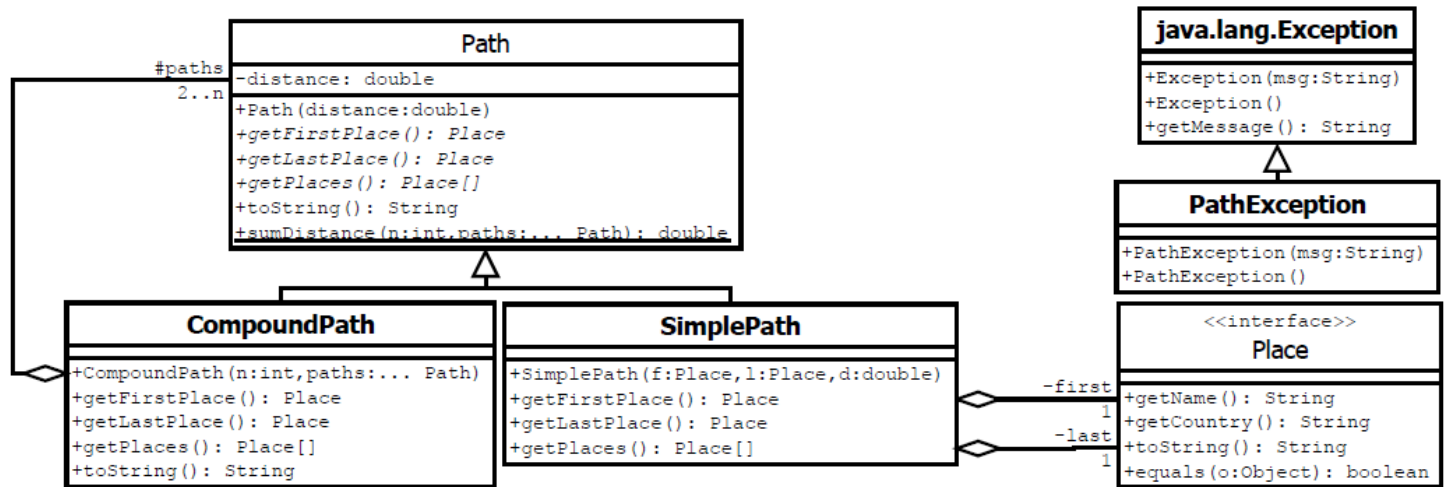
e o método estático da classe `java.lang.Integer`

- `parseInt(String strNumber)` – para obter os valores inteiros correspondentes à área e à população.

- O método estático `getCountryCitiesCount` que recebendo por parâmetro um *array* de elementos do tipo `City` e o nome de um país retorna o número de cidades cujo nome do país é passado por parâmetro.
- O método estático `smallerCities` que recebendo um parâmetro de dimensão variável de elementos do tipo `City` retorna a `City` que tem menor área. Para comparar as `City` utilize o método `compareTo`.
- O método estático `getTop10` que recebendo por parâmetro um *array* de elementos do tipo `City` retorna um novo *array* com as dez cidades que têm maior área. A ordem dos elementos no *array* original pode ser alterada.

Grupo2

Pretende-se implementar uma solução para um sistema de gestão duma companhia de transportes. Foram modelados os tipos Path, SimplePath e CompoundPath, para representar, respetivamente, uma ligação direcionada genérica, uma ligação simples que liga duas localidades, ou uma ligação composta que liga três ou mais localidades. Cada ligação contém informação da primeira localidade (getFirstPlace), da última localidade (getLastPlace) e a distância entre estas em Km.



```

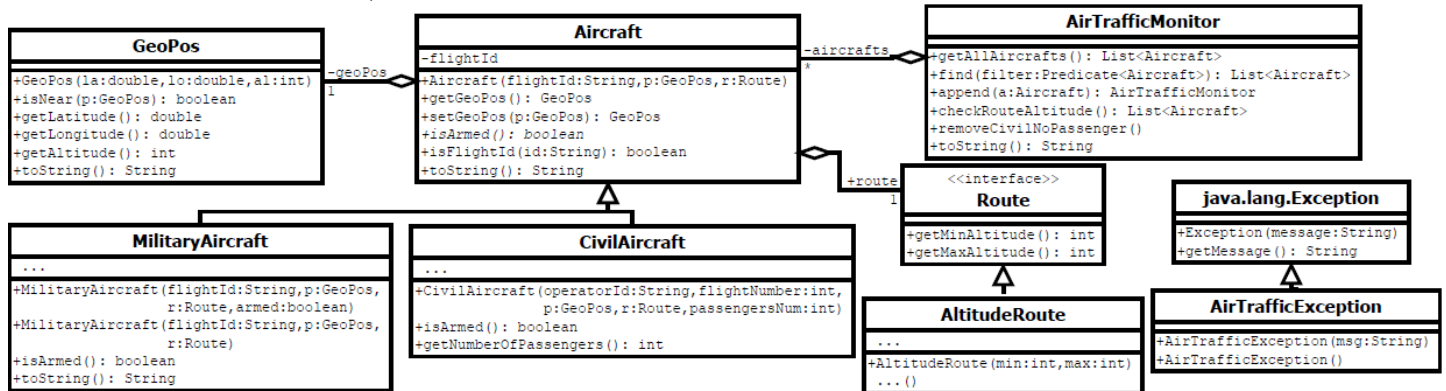
public abstract class Path {
    private final double distance; // Distância entre as Localidades terminais
    protected Path( double distance ) { this.distance = distance; }
    public abstract Place getFirstPlace(); // Localidade inicial da ligação
    public abstract Place getLastPlace(); // Localidade final da ligação
    public abstract Place[] getPlaces(); // Array (sem repetições) com as Localidades da ligação
    public String toString( )
    { return getFirstPlace().getName()+" -> "+ getLastPlace().getName()+" : "+distance+"Km"; }
    public static double sumDistances( int n, Path ... paths) throws PathException { ... }
}
  
```

Sem alterar o diagrama estático de classes sugerido nem a definição da classe Path, implemente:

- Defina a interface Place e coloque a classe City do Grupo 1 a implementar Place.
- O método estático sumDistances da classe Path. Este método retorna a soma das distâncias das n primeiras ligações. Lança a exceção PathException se o n for maior do que a dimensão do array ("Número de ligações inválido") ou se a sequência de ligações não for válida ("Ligação inválida"). A sequência de ligações é inválida se o array de ligações paths não é composto por ligações contíguas.
 - O array de ligações [A → B], [B → C], [C → D] é válido, pois é composto por ligações contíguas.
 - O array de ligações [A → B], [B → C], [D → E] não é válido pois C ≠ D
- A classe SimplePath. O construtor recebe por parâmetro a primeira localidade, a última localidade, e a distância entre elas. O método getPlaces retorna um array com as duas localidades (first e last).
- A classe PathException para que o método getMessage herdado retorne a mensagem que é passada por parâmetro no construtor, ou no caso do construtor sem parâmetros "Ligação inválida".
- A classe CompoundPath. Tendo em conta que:
 - O construtor constrói a ligação composta com as n primeiras ligações do array paths. Lança a exceção PathException caso o n seja inferior a 2 ou superior à dimensão do array paths, ou as n primeiras ligações não sejam contíguas.
 - O método getPlaces retorna um array de todas as localidades contidas na ligação.
 - O método toString retorna a informação da primeira e da última localidade e a distância total da ligação, seguido da lista do nome das localidades que compõem a ligação entre parenteses. Exemplo:
 localidade1-> localidadeN: 100.0Km (localidade1->localidade2->localidade3-> ... ->localidadeN).

Grupo 3

Pretende-se implementar uma solução para o controlo de tráfego aéreo (AirTrafficMonitor). Existem dois tipos de aeronaves, as militares (MilitaryAircraft) e as civis (CivilAircraft). Todas as aeronaves (Aircraft) são caracterizadas pelo número de voo, posição geográfica (GeoPos) e o corredor aéreo (Route) em que estão autorizadas a navegar (altitudes mínima e máxima). O centro de controlo tem a capacidade de adicionar e excluir aeronaves, de determinar o conjunto das aeronaves que se encontram fora do corredor aéreo atribuído, o conjunto de aeronaves em risco de colisão, etc.



1. Defina a interface Route e a classe AltitudeRoute.
2. Implemente a classe AirTrafficException. O método getMessage herdado de Exception tem que retornar: a string "Aviso: " concatenada com a mensagem passada no construtor; ou "Aviso: aeronave fora da rota" se a exceção for estanciada com o construtor sem parâmetros.
3. Implemente a classe abstrata Aircraft sendo a implementação do método toString apresentada em baixo. No construtor recebe por parâmetros: o identificador do voo flightId; a posição geográfica p; e o corredor aéreo r. O construtor lança uma exceção do tipo AirTrafficException quando a altitude associada à posição geográfica p não está entre o intervalo de valores definidos no corredor aéreo r. O campo route é iniciado no construtor e não pode ser alterado. O método isFlightId retorna true caso o campo flightId seja igual à string passada por parâmetro e não pode ser redefinido. O método isArmed é abstrato.

```
public String toString() { return flightId + " at " + geoPos; }
```

4. Implemente a classe CivilAircraft tendo em conta que um avião civil não está armado. No construtor recebe por parâmetros: o identificador da operadora da aeronave operatorId, o número do voo flightNumber; a posição geográfica p; e o corredor aéreo r; e o número de passageiros passengersNum.

```
GeoPos pos = new GeoPos(32.4534, 85.5687, 10457);
System.out.println("GeoPos -> " + pos );
```

```
GeoPos -> 32.4534-85.5687, 10457m
```

```
try {
    Aircraft plain = new CivilAircraft("Tap", 320, pos, new AltitudeRoute(10000, 11500), 100);
    System.out.println( plain );
} catch (AirTrafficException e) { ... }
```

```
Tap320 at 32.4534-85.5687, 10457m
```

5. Implemente a classe MilitaryAircraft. Os três primeiros parâmetros dos dois construtores são: o identificador do voo flightId; a posição geográfica p; e o corredor aéreo r. Caso o construtor tenha o quarto parâmetro este indica se está ou não armado, caso contrário não está armado.

```
GeoPos pos1 = new GeoPos(32.4534, 85.5687, 10100);
Aircraft m1 = new MilitaryAircraft("FAPT357", pos1, new AltitudeRoute(10000, 11500), true);
System.out.println( m1 );
```

```
Military FAPT357 at 32.4534-85.5687, 10100m (armed)
```

```
GeoPos pos2 = new GeoPos(32.5534, 87.5689, 10250);
Aircraft m2 = new MilitaryAircraft("FAPT380", pos2, new AltitudeRoute(10000, 11500) );
System.out.println( m2 );
```

```
Military FAPT380 at 32.5534-87.5689, 10250m
```

```
System.out.println( m2.isFlightId("FAPT380") );
System.out.println( m2.isFlightId("Military FAPT380 ") );
```

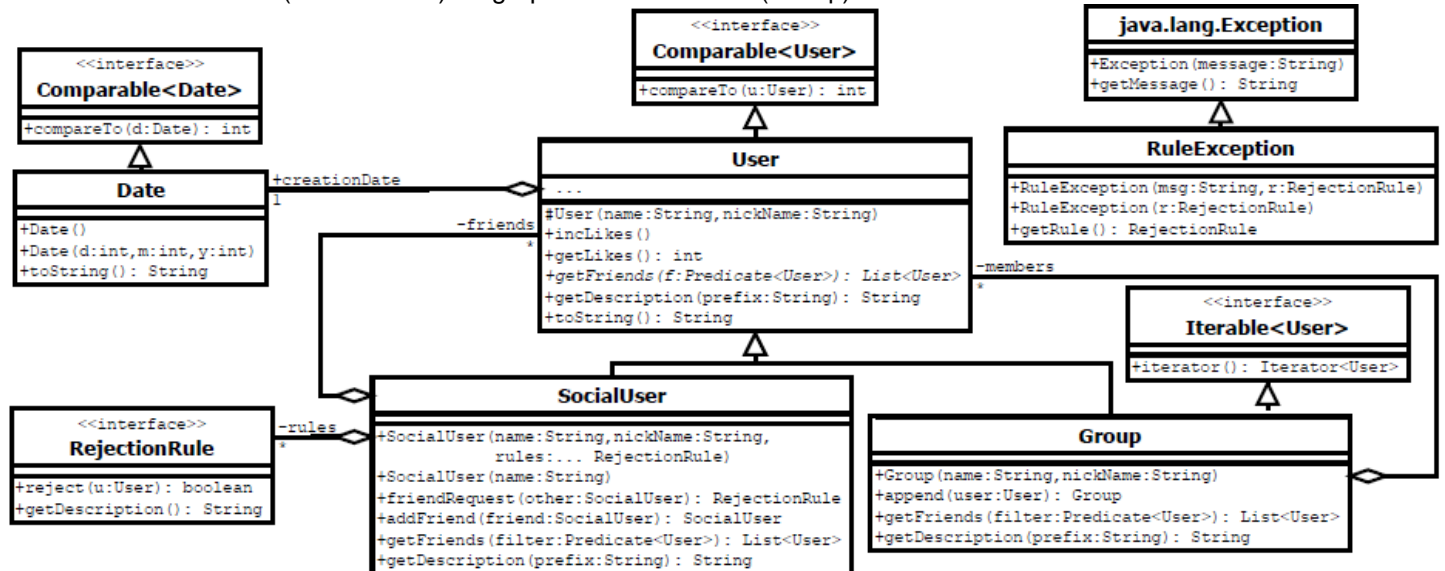
```
true
false
```

6. Implemente a classe AirTrafficMonitor, tendo em conta os seguintes aspetos:

- O método getAllAircrafts devolve a coleção de aeronaves;
- O método find devolve a coleção de aeronaves que obedecem ao predicado filter;
- O método append acrescenta uma aeronave caso não exista uma aeronave de em que a posição esteja próxima da posição da aeronave passada por parâmetro (método isNear de GeoPos). Caso não adicione lança a exceção AirTrafficException (a string retornada pelo método getMessage deve ser "Aviso: não foi adicionada porque está próxima de uma das aeronave monitorizada"). Caso adicione retorna o próprio monitor.
- O método checkRouteAltitudes devolve o conjunto de aeronaves, cujas altitudes estão fora dos limites impostos pelos respetivos corredores aéreos, ordenado por ordem decrescente de altitudes;
- O método remove todas as aeronaves civis sem passageiros. Classes que implementam a interface List<Aircraft> disponibilizam o método removeIf.

Grupo4

Pretende-se desenvolver uma aplicação de suporte a uma rede social. Com esse objetivo, foi desenhada a hierarquia de classes, apresentada à frente, que representa as várias entidades envolvidas. Os utilizadores (User) podem ser utilizadores individuais (SocialUser) ou grupos de utilizadores (Group).



Tendo em conta o diagrama estático de classes e o output dos troços de código:

1. Defina a interface RejectionRule e implemente as classes RejectionRule:

- RejectName rejeita os User que contenham na descrição a *string* que é passada no construtor e tem como descrição "Reject name" seguida da string recebida no construtor.
- RejectDate rejeita os User cuja data de criação seja inferior à que é passada por parâmetro no construtor e tem como descrição "Creation date less than" seguida da data recebida no construtor..

2. Implemente a classe RuleException. O método getRule retorna a RejectionRule recebida por parâmetro no construtor. O método getMessage herdado de Exception tem que retornar a string "Reject user by rule:" ou a string recebida por parâmetro no construtor (conforme construtor com um ou dois parâmetros) concatenadas com a descrição da regra entre aspas.

3. Implemente a classe User. No construtor é passado por parâmetro o nome e o *nickname*. O campo público creationDate deve ser iniciado com a data corrente (construtor sem parâmetros de Date) e só pode ser iniciado no construtor. O método incLikes permite contar o número de *likes*. O método getLikes retorna o número de vezes que o método incLikes foi chamado. O método getDescription retorna o prefixo recebido como parâmetro no construtor concatenado com o nome e o *nickname* todo em maiúsculas entre parênteses. A comparação natural de dois User é o número de *likes* que têm, é maior o que tiver mais *likes*. O método getFriends é abstrato. O método toString tem a seguinte implementação:

```
public final String toString() { return getDescription(""); };
```

4. Implemente a classe SocialUser. No construtor com três parâmetros é passado o nome, o *nickname* e as regras de rejeição dos amigos. No construtor com um parâmetro só é passado o nome, o *nickname* é construído com as três primeiras letras do nome e não rejeita pedidos de amizade. O método friendRequest adiciona o amigo caso não seja rejeitado por alguma das RejectionRules recebidas por parâmetro no construtor e retorna null, caso contrário retorna a regra que o rejeitou. O método addFriend adiciona o amigo caso este não seja rejeitado por alguma das suas RejectionRules e o amigo aceite o pedido de amizade (chamada a friendRequest), caso o amigo seja rejeitado lança a exceção RuleException passando por parâmetro a regra que o rejeitou. O método getFriends(Predicate<User> filter) retorna a lista de amigos que obedecem ao predicado filter. O método getDescription retorna um *string* que à descrição da classe base acrescenta o número de amigos ou a string "not have friends" conforme tenha ou não amigos (ver exemplo de output).

```
try{
    RejectionRule[] r = {new RejectName( "Joana"), new RejectDate(new Date(1,1,2000))};
    SocialUser u1= new SocialUser("Joaquim");
    SocialUser u2=new SocialUser("Carlos","Carl", r );
    SocialUser u3=new SocialUser("Joana","Jo",r[1]);
    System.out.println( u1 );
```

descrição da classe base acrescenta

Joaquim (JOA) - not have friends

```
u1.addFriend( u2 ).addFriend( u3 );
System.out.println(u1 + "\n" + u2 + "\n" + u3);
u2.addFriend( u3 ); // Lança exceção
}catch(RuleException e)
{ System.out.println(e.getMessage());}
```

Joaquim (JOA) - 2 friends
Carlos (CARL) - 1 friend
Joana (JO) - 1 friend

Reject user by rule: "Reject name Joana"

5. Implemente a classe Group. No construtor é passado por parâmetro o nome, o *nickname* e os membros do grupo.
- O método `append` acrescenta o utilizador à lista `members` caso o utilizador ainda não esteja contido na lista e retorna o próprio Group. Caso o utilizador já esteja contido na lista lança uma `RuleException` passando-lhe por parâmetro uma instância de `RejectName` em que a string passada ao construtor é o retornado pelo método `toString`.
 - O método `iterator` retorna um `Iterator<User>` para os elementos da lista `members`. Nota: Qualquer List implementa a interface `Iterable` e disponibiliza o método `iterator`.
 - O método `getFriends` retorna a união dos amigos, que obedecem ao predicado, de todos os membros.
 - O método `getDescription` retorna uma string contendo a descrição de todos os membros ordenados por data de criação (ver exemplo de output).

```
try{
    SocialUser u1= new SocialUser("Joaquim");
    RejectionRule[] r = {new RejectName( "Joana"), new RejectDate(new Date(1,1,2000) )};
    SocialUser u2=new SocialUser("Carlos","Carl", r );
    SocialUser u3=new SocialUser ("Joana","Jo",r[1]);
    u1.addFriend( u2 ).addFriend( u3 );
    Group gjf = new Group("JavaFellows", "jf");
    gjf.append(u1).append(u3);
    System.out.println(gjf);

    Group gte = new Group("Teachers", "te");
    gte.append(u1).append(u2).append(u3).append(gjf);
    System.out.println(gte);
} catch( RuleException e ) {}
```

```
group JavaFellows (JF) {
    Joaquim (JOA) - 2 friends
    Joana (JO) - 1 friend
}
```

```
group Teachers (TE) {
    Joaquim (JOA) - 2 friends
    Carlos (CARL) - 1 friend
    Joana (JO) - 1 friend
    group JavaFellows (JF) {
        Joaquim (JOA) - 2 friends
        Joana (JO) - 1 friend
    }
}
```

Bom trabalho