

# Programação III (PG III)

Semestre de Verão de 2021-2022

## 1º Trabalho prático

Data de Entrega: 29 de Outubro de 2022

**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 correção 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 Query 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 ③ deve ser false.
- Em ② afete a variável c3 para que o programa apresente na consola o resultado mostrado na figura ao lado.
- Altere a escrita no *standard output* em ①: `System.out.println(q1)` ; Indique e justifique o resultado da execução explicitando o mecanismo usado.

Example  
A dimensão do int é 32 bits [5]? yes  
false  
true  
true

Example  
A dimensão do int é 32 bits [5]? yes  
false  
true  
true  
true

```
public class Query{  
    private final String text;  
    private final String correctAnswer;  
    private final int points;  
    public Query( String txt ){  
        this.text= txt;  
        this.points= 5;  
        this.correctAnswer= "yes";  
    }  
}
```

```
public class Example {  
    public static void main(String[] args) {  
        String txt="A dimensão do int é 32 bits";  
        Query q1= new Query(txt);  
        System.out.println(q1.toString()); // ①  
        Query q2= new Query(txt);  
        System.out.println( q1 == q2 );  
        System.out.println( q1.equals( q2 ) );  
        Object o = q2;  
        System.out.println( q1.equals(o) );  
        /* Query q3= null; // ②  
        System.out.println( q1.equals(q3) );// ③  
        if ( q3 != null )  
            System.out.println(q1==q3); */  
    }  
}
```

2. Complete a classe Query, acrescentando:

- Construtor com três parâmetros o texto, a resposta, e número de pontos da questão.
- Construtor com dois parâmetros o texto e um valor booleano que se a true significa que a resposta correta é "yes" a false significa "no".
- Os métodos de instância (getters) para obter o texto e o número de pontos.
- O método instância checkAnswer que recebendo por parâmetro uma *string* com a resposta, retorna o número de pontos da questão se a resposta estiver correta ou zero caso contrário.
- O método de instância compareTo que define a relação de ordem sobre as instâncias da classe Query. Sejam q1 e q2 dois objetos do tipo Query e x um valor inteiro tal que x = q1.compareTo(q2). Se:
  - x<0, significa que o número de pontos da questão q1 é inferior ao número de pontos da questão q2;
  - x>0, significa que o número de pontos da questão q1 é superior ao número de pontos da questão q2;
  - x==0, significa que o número de pontos da questão q1 é igual número de pontos da questão q2.

- O método estático parse que recebendo por parâmetro uma instância de java.lang.String retorna a correspondente instância de Query. O formato da *string* recebida por parâmetro é:

<param>::= <text> '?' <correct answer> | <text> '[' <points> "]"?' <correct answer>

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

- int indexOf(int ch, int fromIndex) – para obter os índices dos caracteres de separação;
- int lastIndexOf(int ch ) – para obter o índice do caractere de separação da resposta;
- String substring(int beginIndex, int endIndex) – para individualizar as *strings* com o texto, a resposta e o número de pontos;

o método estático da classe java.lang.Integer

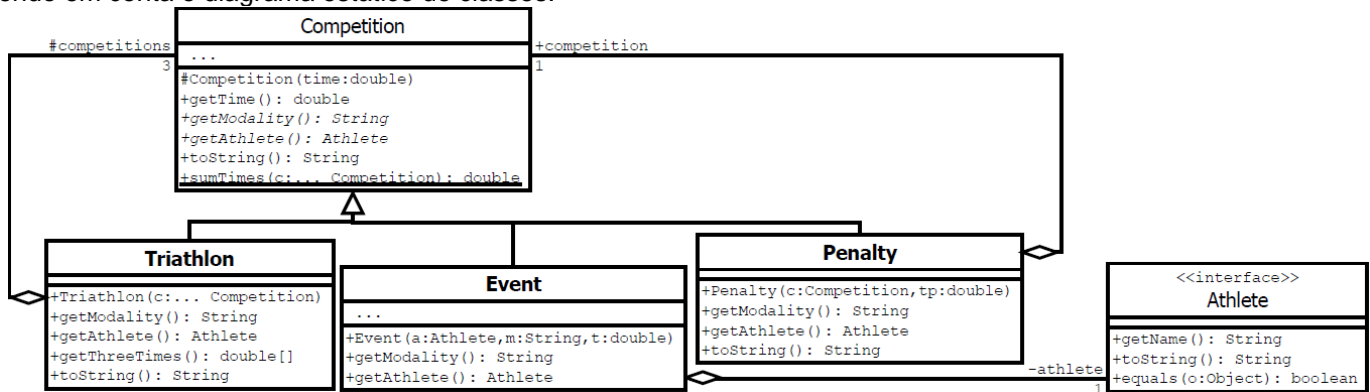
- int parseInt(String strNumber) – para obter o valor inteiro correspondente aos pontos.

- O método estático quiz que recebendo por parâmetro um parâmetro de dimensão variável de elementos do tipo Query, instância um Scanner e para cada Query: faz a pergunta; lê a resposta; e caso esteja correta acumula os pontos. Retorna os pontos acumulados.
- O método estático growingQueries que recebe por parâmetro um *array* de questões produz um *array* ordenado. O *array* recebido por parâmetro é percorrido sequencialmente e a questão é adicionada no fim do novo *array* se: for a 1ª questão; ou se for maior ou igual à última adicionada. Para comparar as Query utilize o método compareTo.

# Grupo2

Pretende-se implementar uma solução para armazenar os tempos obtidos por atletas em competições tendo em conta que posteriormente podem existir penalizações. Para o efeito chegou-se ao seguinte diagrama estático de classes:

Tendo em conta o diagrama estático de classes:



1. Defina a interface Athlete e a classe AthleteTest tendo em conta:

```
Athlete a = new AthleteTest("Diogo Ribeiro");
System.out.println( a );
```

Diogo Ribeiro

2. Defina a classe Competition. O construtor recebe o tempo obtido na competição. O método getTime retorna o tempo recebido no construtor e não pode ser redefinido. Os métodos getModality e getAthlete são abstratos. O método toString retorna a *string* resultante da concatenação da modalidade com o nome do atleta separadas pelo carácter dois pontos, seguidas do tempo. O método estático sumTimes retorna a soma dos tempos das competições recebidas por parâmetro.

3. Defina a classe Event. O construtor recebe o atleta, o nome da modalidade e o tempo obtido na competição.

```
Event e = new Event( new AthleteTest("Diogo Ribeiro"), "50 metros mariposa", 22.96);
System.out.println( e );
```

50 metros mariposa: Diogo Ribeiro - 22.96

4. Defina a classe Penalty. O construtor recebe a referência para a competição penalizada e o tempo de penalização. A variável de instância competition só pode ser afetada no construtor. O tempo de uma competição com penalização é o tempo da competição recebida por parâmetro acrescido do tempo de penalização. O atleta e o nome da modalidade são os da competição recebida por parâmetro. O método toString retorna a *string* retornada pelo método toString herdado concatenada com descrição da penalização entre parênteses retos (ver *output* do exemplo).

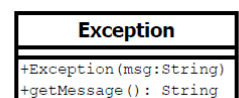
```
Event e = new Event( new AthleteTest("Arnaldo Abrantes"), "100 metros", 10.53);
System.out.println( e );
```

100 metros: Arnaldo Abrantes - 10.53

```
Penalty pe = new Penalty( e, 0.20);
System.out.println( pe );
```

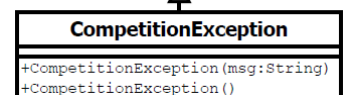
100 metros: Arnaldo Abrantes - 10.73 [10.53+0.20]

5. Implemente a classe CompetitionException para que o método getMessage herdado retorne a mensagem que é passada por parâmetro no construtor, ou no caso do construtor sem parâmetros "Competição inválida".



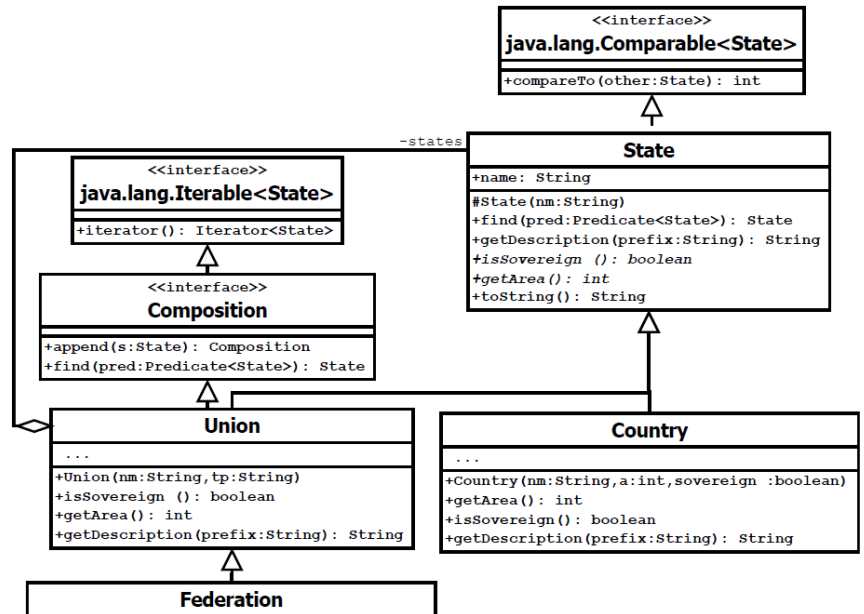
6. Defina a classe Triathlon. Tendo em conta que:

- No triatlo o mesmo atleta participa em três competições de modalidades distintas. O construtor lança a exceção CompetitionException caso: a dimensão do *array* seja diferente de três (mensagem: "Triatlo: Número de competições inválido"); as três competições não correspondam a modalidades distintas (mensagem: "Triatlo: Modalidades inválidas"); ou não sejam do mesmo atleta (mensagem: "Triatlo: Atleta inválido").
- O tempo do triatlo é o somatório dos tempos das três competições.
- O método getModality retorna "Triatlo";
- O método getAthlete retorna o atleta que realizou as três competições.
- O método getThreeTimes retorna um *array* com o tempo de cada modalidade pela ordem inversa que as competições foram dadas no construtor.
- O método toString retorna uma *string* com a descrição da competição triatlo seguida da descrição da competição de cada modalidade, as descrições são separadas pelo carácter fim de linha.



# Grupo3

Pretende-se implementar uma solução simplificada para armazenar as entidades territoriais existentes a nível internacional, assim como as suas constituições. Um estado (State) é uma entidade territorial que tanto pode ser soberana como autónoma. Uma união (Union) é uma entidade que associa entidades distintas. Uma federação (Federation) é uma união composta por entidades territoriais autónomas, dotadas de governo próprio, em que a soberania é transferida para a união federal. Para o efeito chegou-se ao diagrama estático de classes:



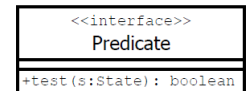
Tendo em conta o diagrama estático de classes, os troços de código e respetivo *output* que exemplificam o que se pretende que o método `getDescription` retorne:

1. Implemente a classe abstrata State. O construtor recebe por parâmetro o nome do estado. O campo name só pode ser afetado no construtor. Os métodos `getArea` e `isSovereign` são abstratos. O método `compareTo` compara as áreas dos estados é maior o que tiver maior área. Os métodos `find`, `getDescription` e `toString` têm a seguinte implementação:

```

public State find( Predicate<State> pred ) { return pred.test(this) ? this: null; }
public String getDescription(String prefix) { return prefix + name + " - "; }
public final String toString() { return getDescription( "" ); }

```



2. Defina a classe Country. No construtor recebe por parâmetro o nome, a área e se é soberano. O método `getArea` retorna a área passada no construtor. O método `isSovereign` retorna o valor passado no construtor. O seguinte troço de código e o respetivo *output* exemplificam o que se pretende que o método `getDescription` faça:

```

State p = new Country("Portugal", 92391, true); //Estado soberano
System.out.println( p );
State g = new Country("Geórgia", 154077, false); //Estado autónomo
System.out.println( g );

```

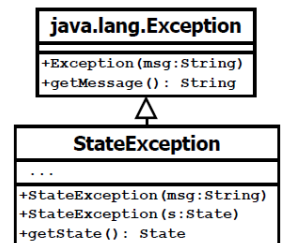
```

Portugal - Estado soberano (92391 km²)
Geórgia - Estado autónomo (154077 km²)

```

3. Defina a classe `StateException` para que o método `getMessage` herdado de `Exception` retorne:

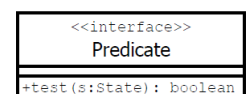
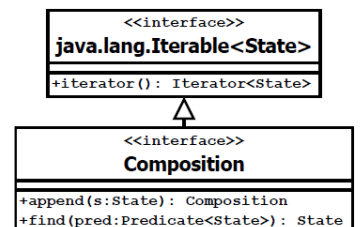
- a *string* que é passada por parâmetro no construtor, caso tenha sido instanciado com o construtor com parâmetro do tipo `String`;
- o nome do State passado por parâmetro no construtor concatenado com a string " – Estado inválido", caso tenha sido instanciado com o construtor com parâmetro do tipo `State`.
- O método `getState` retorna o State passado por parâmetro no construtor.



4. Defina a interface `Composition`. O método `append` pode lançar `StateException`.

5. Defina a classe `Union` que agrega uma lista de estados soberanos ou autónomos.

- O construtor recebe por parâmetro o nome e o tipo da união.
- O método `isSovereign` retorna `false`.
- O método `getArea` retorna o somatório das áreas dos estados que agrega.
- O método `find` retorna a referência para a própria união caso a própria união satisfaça o predicado. Caso contrário se encontrar nos estados que agrega um estado que satisfaça o predicado retorna a referência para esse estado, se não encontrar retorna `null`.
- O método `iterator` retorna um `Iterator` para os estados que agrega.
- O método `append` adiciona o estado passado por parâmetro à união caso não encontre um estado com o mesmo nome na união. Usar o método `find` para verificar se existe um estado com o mesmo nome. Retorna a própria `Union`.



- O seguinte troço de código e o respetivo *output* exemplificam o que se pretende que o método `getDescription` faça:

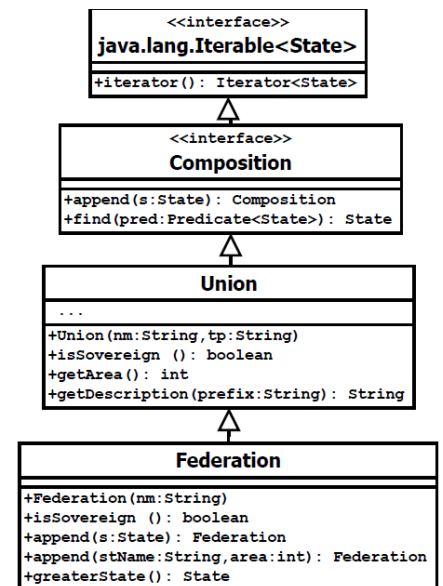
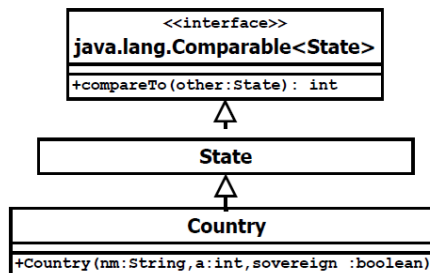
```
State p = new Country("Portugal", 92391, true);
State f = new Country("França", 154077, true);
String onuName = "Organização Nações Unidas";
String onuType = "Organização Internacional";
Union onu = new Union( onuName, onuType );
```

```
onu.append( p ).append( f ).append( p );
System.out.println( onu );
```

```
Organização Nações Unidas - Organização Internacional
Portugal - Estado soberano (92391 km²)
França - Estado soberano (154077 km²)
```

6. [4] Defina a classe `Federation` que é um estado soberano que agrega uma lista de estados autónomos.

- O método `isSovereign` retorna `true`.
- O método `append` com um parâmetro caso o estado não seja um país autónomo lança um `StateException`, caso contrário chama o método `append` da classe base para o adicionar. Retorna a própria `Federation`.
- O método `append` com dois parâmetros instância um `Country` autónomo com o nome e a área recebidos por parâmetro e evoca o método `append` com um parâmetro. Retorna a própria `Federation`. Este método não tem a cláusula `throws` na assinatura.
- O método `greaterState` retorna o maior estado da federação ou `null` caso ainda não tenham sido adicionados estados. Caso existam dois ou mais estados iguais retorna o último adicionado.



Exemplo:

```
State g = new Country("Geórgia", 154077, false);
Federation usa = new Federation("Estados Unidos");
try {
    usa.append( g ).append("Flórida",170451);
    System.out.println( usa );
    usa.append( p );
} catch ( StateException e ) {
    System.out.println(e.getMessage() );
}
```

```
onu.append( usa );
System.out.println( onu );
```

```
State res;
if ( onu.find( s ) -> s.name.equals(onuName)) == onu &&
    (res= onu.find( s ) -> (s instanceof Country) && s.getArea()==170451)) !=null )
    System.out.println(res);
```

```
Estados Unidos - Estado federal
Geórgia - Estado autónomo (154077 km²)
Flórida - Estado autónomo (170451 km²)
```

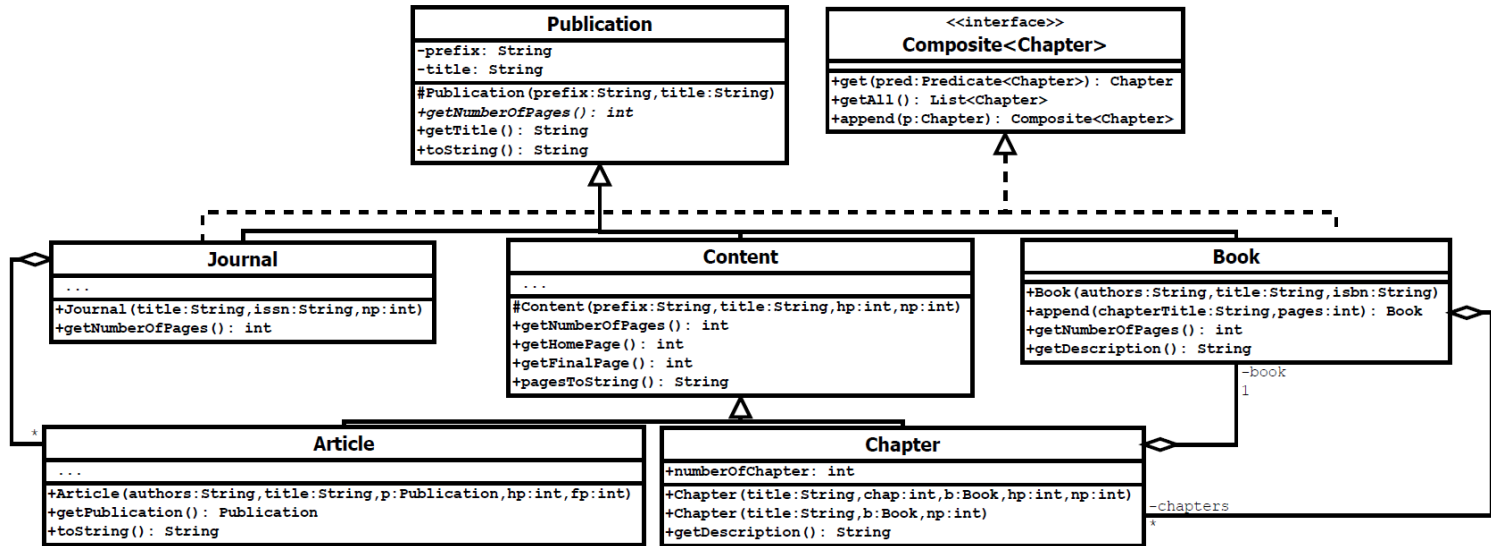
```
Portugal - Estado inválido
```

```
Organização Nações Unidas - Organização Internacional
Portugal - Estado soberano (92391 km²)
França - Estado soberano (154077km²)
Estados Unidos - Estado federal
Geórgia - Estado autónomo (154077 km²)
Flórida - Estado autónomo (170451 km²)
```

```
Flórida - Estado autónomo (170451 km²)
```

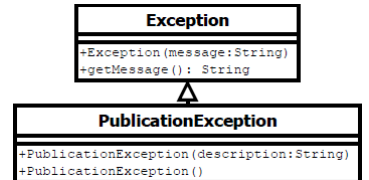
# Grupo4

Com o objetivo de gerir as publicações de uma biblioteca, foi criada a seguinte hierarquia. Uma publicação (Publication) pode ser um artigo (Article), um capítulo dum livro (Chapter), um livro (Book) ou um jornal (Journal). Todas as publicações têm um título e um determinado número de páginas. Os artigos e os capítulos estão inseridos em determinadas páginas duma publicação. Os livros são constituídos por uma sequência de capítulos, e os jornais por um conjunto de artigos.



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

1. Implemente a classe abstrata Publication. Ao construtor é passado o título e um prefixo ao título. O método getTitle retorna o título recebido por parâmetro no construtor e não pode ser redefinido. O método getNumberOfPages é abstrato. O método toString retorna o prefixo seguido do título entre *aspas*.
2. Implemente a classe PublicationException. O método getMessage herdado de Exception tem que retornar a *string* "Invalid publication" caso a exceção tenha sido instanciada com o construtor sem parâmetros ou a *string* "Error: " concatenada com a descrição passada por parâmetro caso tenha sido instanciada com o construtor com um parâmetro.
3. Implemente a classe abstrata Content. Ao construtor é passado o prefixo, o título, a página de início e o número de páginas. Os métodos getNumberOfPages, getHomePage e getFinalPage não podem ser redefinidos, o primeiro retorna o número de páginas recebido por parâmetro no construtor, o segundo retorna a página inicial, e o terceiro a página final (calculada tendo em conta a página inicial e o número de páginas). O método pagesToString retorna uma *string* contendo a descrição das páginas, caso o número de páginas seja 1 a descrição contém unicamente o número da página inicial (Ex: "6"), caso contrário a descrição contém a página inicial e a página final separadas pelo carácter '-' (Ex: "10-20").
4. Implemente a classe Article. Na instanciação é passado os autores, o título, a publicação em que está inserido, a página inicial e a página final. Ao construtor da classe base deve ser passado como parâmetro os autores, o título a página inicial e o número de páginas calculado tendo em conta as páginas inicial e final. Se o número da página inicial for maior do que o número da página final lança uma exceção com a mensagem por omissão. O método toString acrescenta à *string* retornada pelo método toString da classe base o título da publicação e a descrição das páginas retornada pelo método pagesToString (ver exemplo de *output*).



```

Journal j= new Journal("Expresso", "656756788989", 30);
Article a1 = new Article("Tavares, Miguel Sousa", "E desembarcam nas praias", j, 5, 5);
System.out.println(a1);

```

Tavares, Miguel Sousa "E desembarcam nas praias", Expresso, 5

```

Article a2=new Article("Salvador, João Miguel","A falar é que a gente se entende",j, 34,38);
System.out.println(a2);

```

Salvador, João Miguel "A falar é que a gente se entende", Expresso, 34-38



5. Implemente a classe Chapter. Ao construtor com 5 parâmetros é passado o título, o número do capítulo, o livro, a página inicial e o número de páginas. No caso do construtor com 3 parâmetros o número do capítulo e a página inicial não são passados porque são os dois 1. Ao construtor da classe base deve ser passado como primeiro parâmetro (prefix) a string "Cap. " seguida de dois dígitos correspondentes ao número (Ex: "Cap. 03 –"). O campo público numberOfChapter só pode ser iniciado no construtor. O método getDescription acrescenta à *string* retornada pelo método toString uma linha contendo a *string* " in " seguida do livro e da descrição das páginas retornada pelo método pagesToString (ver exemplo de *output*).

```
Book book= new Book("Walter J. Savitch, Kenrick Mock",  
                    "Java: An Introduction to Problem Solving & Programming", "0132162709");  
System.out.println( book.toString() );
```

Walter J. Savitch, Kenrick Mock "Java: An Introduction to Problem Solving & Programming"

```
Chapter c = new Chapter("Introduction to Computers and Java", book, 11);  
System.out.println( c.getDescription() );
```

Cap. 01 - "Introduction to Computers and Java"  
in Walter J. Savitch, Kenrick Mock "Java: An Introduction to Problem Solving & Programming", 1-11

```
c= new Chapter("Basic Computation", 2, book, 11, 14);  
System.out.println(c.getDescription());
```

Cap. 02 - "Basic Computation"  
in Walter J. Savitch, Kenrick Mock "Java: An Introduction to Problem Solving & Programming", 11-24

6. Implemente a classe Book que implementa Composite<Chapter>, tendo em conta o seguinte troço de código:

```
Book b = new Book("Walter J. Savitch, Kenrick Mock",  
                  "Java: An Introduction to Problem Solving & Programming", "0132162709");  
b.append("Introduction to Computers and Java", 11).append("Basic Computation", 15);  
System.out.println( b.toString() );
```

Walter J. Savitch, Kenrick Mock "Java: An Introduction to Problem Solving & Programming"

```
System.out.println( b.getDescription() );
```

Walter J. Savitch, Kenrick Mock "Java: An Introduction to Problem Solving & Programming"  
ISBN:0132162709, 26 pages  
Cap. 01 - "Introduction to Computers and Java", 1-11  
Cap. 02 - "Basic Computation", 12-26

- O método get retorna o último capítulo que obedece ao predicado pred, ou null caso não exista.
- O método getAll retorna uma nova lista de capítulos ordenados por título.
- O método append com um parâmetro lança a exceção de *runtime* UnsupportedOperationException.
- O método append com dois parâmetros instancia um capítulo com o título e número de páginas passadas no construtor, sendo o número do capítulo e o número da primeira página os que se seguem ao último adicionado ou 1 caso ainda não tenham sido adicionados capítulos. Se já existir um capítulo com o mesmo nome lança uma exceção com a mensagem "Invalid chapter" (usar o método get para verificar se existe).
- O método getNumberOfPages retorna o somatório das páginas dos capítulos.
- O método getDescription retorna uma *string* com a descrição do livro (ver o exemplo de *output*).

```
<<interface>>  
Predicate<Chapter>  
+test(p:Chapter): boolean
```

Bom trabalho