Programação III (PG III)

Semestre de Inverno de 2025-2026

1º Trabalho prático

Data de Entrega: 30 de outubro de 2025

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

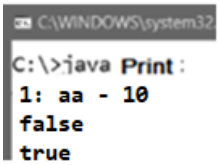
NOTA: tem que constar todo o código desenvolvido, incluindo os testes unitários que permitem validar a correção dos métodos e classes realizadas.

Grupo 1

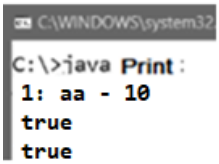
```
public class Student {
 private final String name;
 private final int number; private int grade;
 public Student(int n, String nm, int g) {
   this.name = nm;
   this.number = n; this.grade = g;
 }
 public int getNumber() { return number; }
}
```

```
public class Print {
  public static void main(String[] args) {
    Student s1 = new Student (1, "aa", 10 );
    Student s2 = new Student (1, "aa", 10 );
    Student s3 = s2; // Alterar
    System.out.println( s1.toString() );
    System.out.println( s1 == s3 );
    System.out.print( s1.equals( s3 ) );
  }
}
```

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 `Student` para que o programa apresente na consola o resultado mostrado na figura ao lado. Justifique as alterações, explicitando o mecanismo da linguagem usado.



   - Altere a afetação do método `main` para que o programa apresente na consola o resultado mostrado na figura ao lado.



   - Altere a escrita no *standard output*: `System.out.println(s1);`

   Indique e justifique o resultado da execução.

2. Complete a classe `Student`, tendo em conta que deve disponibilizar:
   - Construtor com dois parâmetros: (1) o nome do tipo `String`; (2) a nota do tipo inteiro. O número do aluno será 1 para o 1º aluno instanciado com este construtor e irá tendo valores consecutivos à medida que este construtor for usado para a instanciação de alunos.
   - Os métodos de instância para obter o nome, o número e a nota (*getters*).
   - O método de instância `toString` que devolve uma *string* com o formato `<numero>: <nome> - <nota>`
   - O método de instância `equals` que retorna `true` se as duas instâncias representarem o mesmo aluno, ou seja, têm o mesmo número e nome.
   - O método de instância `isApproved` que retorna `true` caso o aluno esteja aprovado, isto é, tenha uma nota superior ou igual a 10.
   - O método de instância `setGrade` que, recebendo como parâmetro um valor inteiro correspondente à nota, altera o campo grade caso a nota seja maior ou igual a zero e menor ou igual a 20. Retorna a nota anterior.
   - O método de instância `compareTo` que define a relação de ordem sobre as instâncias da classe `Student`. Sejam `st1` e `st2` dois objetos do tipo `Student` e `x` um valor inteiro tal que `x = st1.compareTo(st2)`. Se:

     `x<0`, significa que o número do aluno `st1` é inferior ao número do aluno st2;
     `x>0`, significa que o número do aluno `st1` é superior ao número do aluno st2;
     `x==0`, significa que o número do aluno `st1` é igual ao número do aluno st2.

   - O método estático `parseStudent` que recebe por parâmetro a descrição, do tipo `String`, e retorna uma instância de `Student`. O formato da *string* passada por parâmetro é:

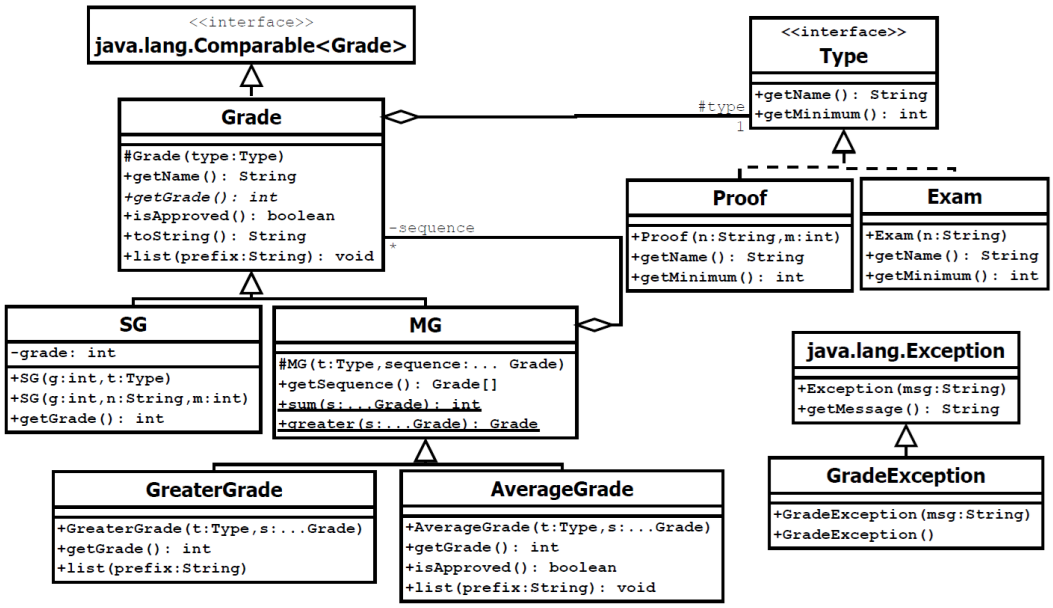
     `<numero>: <name> - <nota>`

     Nota: Usar os métodos de instância da classe `java.lang.String`:
     - `int indexOf( int ch, int fromIndex )` – para obter os índices de fim do número e do nome;
     - `String substring( int beginIndex, int endIndex )` – para obter o número e o nome;
     - `String substring( int beginIndex )` – para obter a nota;

     e o método estático da classe `java.lang.Integer`
     - `int parseInt(String strNumber)` – para obter o número inteiro correspondente ao número e à nota.

Grupo2

Pretende-se implementar uma solução para armazenar a classificação de provas. A nota de uma classificação pode ser: simplesmente a nota dum teste, dum exame, etc. (`SG`); a média aritmética das notas de uma sequência de classificações (`AverageGrade`); ou a maior nota da sequência de classificações (`GreaterGrade`). Para o efeito chegou-se ao diagrama estático de classes:



Tendo em conta o diagrama estático de classes e o código fornecido em anexo:

1. Defina a classe `Grade`. De notar que o método `getGrade`, que retorna a nota, é abstrato, e a classe implementa a interface `Comparable<Grade>` (comparar duas instâncias de `Grade` é comparar as notas). O método `isApproved` verifica se a nota é superior ou igual ao mínimo. Os métodos `toString` e `list` têm a seguinte implementação:

```
public final String toString()   { return getName()+ (isApproved()? " - " : " - R ") + getGrade(); }
public void list( String prefix ){ System.out.println( prefix + this );     }
```

2. Defina a classe `SG`. O construtor com dois parâmetros recebe a nota e o tipo de prova. O construtor com três parâmetros recebe a nota, o nome da prova, e a nota mínima (tem de instanciar uma `Proof`).

3. Defina a classe `GradeException` para que o método `getMessage` retorne a mensagem que é passada por parâmetro no construtor ou, no caso do construtor sem parâmetros, "classificação inválida".

4. Defina a classe abstrata `MG`. O construtor recebe o tipo de prova e a sequência de classificações, lança a exceção `GradeException` caso a dimensão da sequência seja inferior a dois. O método `getSequence` retorna a sequência. Os métodos estáticos `greater` e `sum` recebem por parâmetro uma sequência de classificações `s`, o método `greater` retorna a referência para a maior `Grade` de `s`, e o método `sum` retorna o somatório das notas das classificações de `s`.

5. Defina a classe `AverageGrade`. O método `getGrade` retorna a média aritmética (somatório das notas a dividir pelo número de notas). O método `isApproved` retorna `true` caso exista aproveitamento em todas as classificações da sequência e a média seja maior ou igual do que o mínimo do tipo de prova. O método `list` escreve a classificação e lista as classificações da sequência.

```
Grade[] seq = { new SG( 9, new Proof("1° Teste", 8) ),
                new SG( 8,"2° Teste", 8 ) };
Grade testes = new AverageGrade( new Proof("Testes", 10), seq );
testes.list("");
```

```
Testes – R 9
   1º Teste – 9
   2º Teste – 8
```

6. Defina a classe `GreaterGrade`. O método `getGrade` retorna a maior nota da sequência. O método `isApproved` retorna `true` caso exista uma classificação com aproveitamento. O método `list` lista a classificação e a maior classificação.

```
Grade e1 = new SG(12, new Exam("época normal")),
      e2 = new SG(13, new Exam("época de recurso"));
System.out.println( e1 ); System.out.println( e2 );
Grade total = new GreaterGrade(
   new Proof("Avaliação Teórica", 10), testes, e1, e2);
total.list("");
```

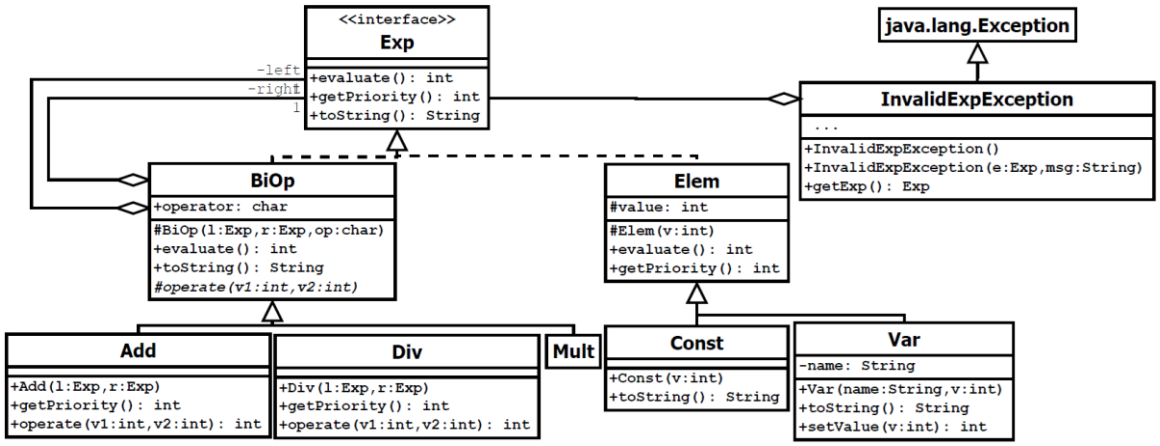
```
Exame de época normal – 12
Exame de época de recurso – 13
```

```
Avaliação Teórica – 13
  Exame de época de recurso – 13
```

7. Altere o tipo do campo `grade` da classe `Student` para `Grade`, e os métodos que esta alteração provoca.

Grupo3

Para suporte de expressões aritméticas em memória foi elaborado o conjunto de tipos ilustrado no diagrama estático de classes. Por simplicidade do exercício, as expressões suportadas só têm os operadores binários para somas (`Add`), para divisões (`Div`), e para multiplicações (`Mult`) e valores inteiros constantes (`Const`) ou variáveis (`Var`). O método `evaluate` avalia a expressão e retorna o valor. O método `toString` retorna a *string* que representa a expressão. O método `getPriority` permite que a representação da expressão apenas contenha os parêntesis quando a prioridade da operação da sub-expressão (inferior) for menor ou igual que a da operação corrente.



Tendo em conta o diagrama estático de classes, e os *outputs* da execução do seguinte troço de código:

```
try {
  Var v = new Var("A", 5); System.out.println( v+"->"+v.evaluate() );
  Const c = new Const(2); System.out.println( c+"->"+c.evaluate() );
  Exp e1 = new Mult(c, v); System.out.println( e1+"->"+e1.evaluate() );
  Exp e2 = new Add(v, c); System.out.println( e2+"->"+e2.evaluate() );
  Exp e3 = new Div(e1, e2); System.out.println( e3+"->"+e3.evaluate() );

  v.setValue( -2 );
  int r = e3.evaluate();
} catch ( InvalidExpException ex ) {
    System.out.println(ex.getMessage());}
```

```
A->5
2->2
2*A->10
A+2->7
(2*A)/(A+2)->1
```

```
Divide by zero -> (2*A)/(A+2)
```

1. Implemente a exceção `InvalidExpException`. O método `getMessage` no caso de ter sido instanciada: com o construtor com os dois parâmetros retorna a mensagem concatenada com a representação da expressão; com o construtor sem parâmetros retorna "*Invalid expression*".

2. Implemente a interface `Exp`. O método `evaluate` pode lançar a exceção `InvalidExpException` caso a operação não seja possível.

3. Implemente a classe abstrata `Elem` que implementa os métodos `evaluate` e `getPriority`. O método `evaluate` retorna o valor passado no construtor. O método `getPriority` não pode ser redefinido e retorna a maior prioridade que é zero. A variável de instância `value` pode ser alterada por classes que estendem `Elem`. A variável de instância `operator` só pode ser afetada no construtor.

4. Implemente a classe `Const`.

5. Implemente a classe `Var`. O método `setValue` permite alterar o valor inicialmente atribuído à variável de instância `value` e retorna o valor anteriormente atribuído.

6. Implemente a classe abstrata `BiOp` que guarda as referências para a expressão esquerda e direita e o operador. O método `evaluate` é implementado nesta classe e deve chamar o método `operate` para executar a operação sobre os resultados das avaliações das sub-expressões. O método `operate` é abstrato (é implementado nas classes que estendem `BiOp`) e pode lançar exceção `InvalidExpException` caso a operação não seja possível. De notar que caso as sub-expressões (direita ou/e esquerda) tenham menor ou igual prioridade do que a expressão corrente o método `toString` deverá colocar a representação das sub-expressões entre parênteses.

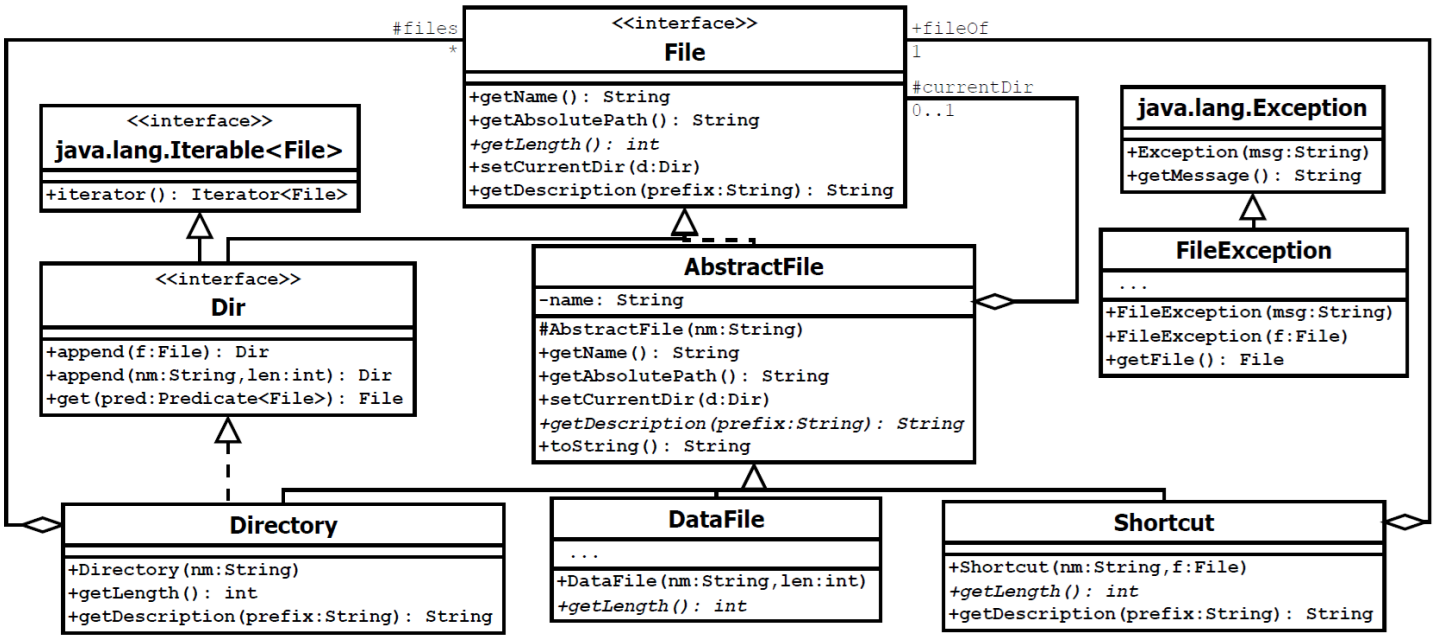
7. Implemente as classes `Add`, `Mult` e `Div`. A prioridade das operações de multiplicação e divisão é 4 e das operações de adição é 5. Caso o divisor seja zero o método `operate` da classe `Div` deve lançar a exceção `InvalidExpException` e dependendo do valor do dividendo a mensagem passada no construtor deverá ser:

   - "*Divide by zero*" se o dividendo for diferente de zero;
   - "*Indeterminate*" se o dividendo for zero (0/0).

Grupo 4

Pretende-se implementar uma solução para um sistema de ficheiros. Um ficheiro (`File`) pode conter dados (`DataFile`), ficheiros (`Directory`), ou pode ser um atalho para outro ficheiro (`Shortcut`). Sobre qualquer ficheiro (`File`) é possível obter o nome (`getName`), o *pathname* absoluto (`getAbsolutePath`), a dimensão (`getLength`) e a descrição (`getDescription`). Um ficheiro pode ser colocado numa determinada diretoria (`setCurrentDir`) ou estar na raiz. O *pathname* absoluto é a concatenação dos nomes das diretorias até à raiz.

Para o efeito chegou-se ao diagrama estático de classes:



Tendo em conta o diagrama estático de classes:

1. Implemente a classe abstrata `AbstractFile`. No construtor recebe por parâmetro o nome do ficheiro. O método `getName` retorna o nome passado no construtor e não pode ser redefinido. O método `setCurrentDir` afeta a variável de instância `currentDir` com a diretoria passada por parâmetro. O método `getAbsolutePath` retorna o *pathname* absoluto da diretoria corrente concatenado com o caráter '\' e com o nome do ficheiro, ou só o nome caso a diretoria corrente seja a raiz (`currentDir == null`). Os métodos `toString` e `getDescription` têm a seguinte implementação:

```
public String getDescription( String prefix ) { return prefix + this ;   }
public final String toString() { return name +" - " + getLength()+ "KB"; }
```

2. Defina a classe `FileException` 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 *pathname* absoluto do `File` passado por parâmetro no construtor concatenado com a string *" - invalid access"* (caso tenha sido instanciado com o construtor com parâmetro do tipo `File`). O método `getFile` retorna o `File` passado por parâmetro no construtor.

3. Defina a classe `DataFile`. No construtor recebe por parâmetro o nome e a dimensão, caso a dimensão seja menor que zero lança a exceção `FileException` em que a mensagem é *" Not create DataFile - Invalid length "*. O método `getLength` retorna a dimensão passada no construtor.

```
File f = new DataFile("file1", 100);
System.out.println( f + "  -> "+ f.getAbsolutePath() );
f.setCurrentDir( new Directory("dir1") );
System.out.println( f + "  -> " + f.getAbsolutePath() );
```

```
file1 - 100KB  -> file1
file1 - 100KB  -> dir1\file1
```

4. Defina a classe `Shortcut`. O construtor recebe o nome e o `File` do qual é atalho. A variável de instância `fileOf` só pode ser afetada no construtor. O método `getLength` retorna a dimensão do ficheiro do qual é atalho. O método `getDescription` retorna o `getDescription` da classe base concatenado com a string "*shortcut of* " e com o *pathname* absoluto do ficheiro do qual é atalho entre parênteses retos.

```
File f1 = new DataFile("file1", 100); f1.setCurrentDir( new Directory("dir1") );
File f2 = new Shortcut("file2", f1);
System.out.println( f2.getDescription("") );
```

```
file2 - 100KB  [shortcut of dir1\file1]
```

5. Defina a interface `Dir`. O método `append` com dois parâmetros pode lançar uma `FileException`.

6. [7] Defina a classe `Directory` que contém uma lista de instâncias de `File`.

- O método `append` com um parâmetro adiciona o `File f` à lista `files` e evoca sobre `f` o método `setCurrentDir` passando-lhe por parâmetro a diretoria. Retorna a própria diretoria.
- O método `append` com dois parâmetros instancia um `DataFile` com o nome e a dimensão recebidos por parâmetro e evoca o método `append` com o parâmetro do tipo `File`. De notar que ao instanciar um `DataFile` pode ser lançada uma exceção que se quer que seja propagada.
- O método `getLength` retorna o somatório das dimensões dos ficheiros contidos.
- O método `get` retorna a referência para um `File` contido na diretoria ou em subdiretorias que obedece aos requesitos do predicado passado por parâmetro, caso não exista retorna `null`.
- O método `iterator` retorna um `Iterator` para os objetos `File` existentes na diretoria.

O seguinte troço de código e o respetivo *output* exemplificam o que se pretende que os métodos `getDescription` e `get` façam.

```
Dir d1 = new Directory("dir1");
d1.append("file1", 100).append(new Directory("dir2") );

File f = d1.get( ft -> ft instanceof Dir);
if ( f != null )
  ((Dir) f).append("file2", 100).append("file3",300);

f = d1.get( ft-> ft.getName().equals("file2"));
d1.append( new Shortcut("file4", f));

System.out.println( d1.getDescription("") );
```

```
dir1
 file1 - 100KB
 dir2
  file2 - 100KB
  file3 - 300KB
 file4 - 100KB [shortcut of dir1\dir2\file2]
```

Bom trabalho