Ciências ULisboa | Informática

Programação Centrada em Objetos

2021/2022

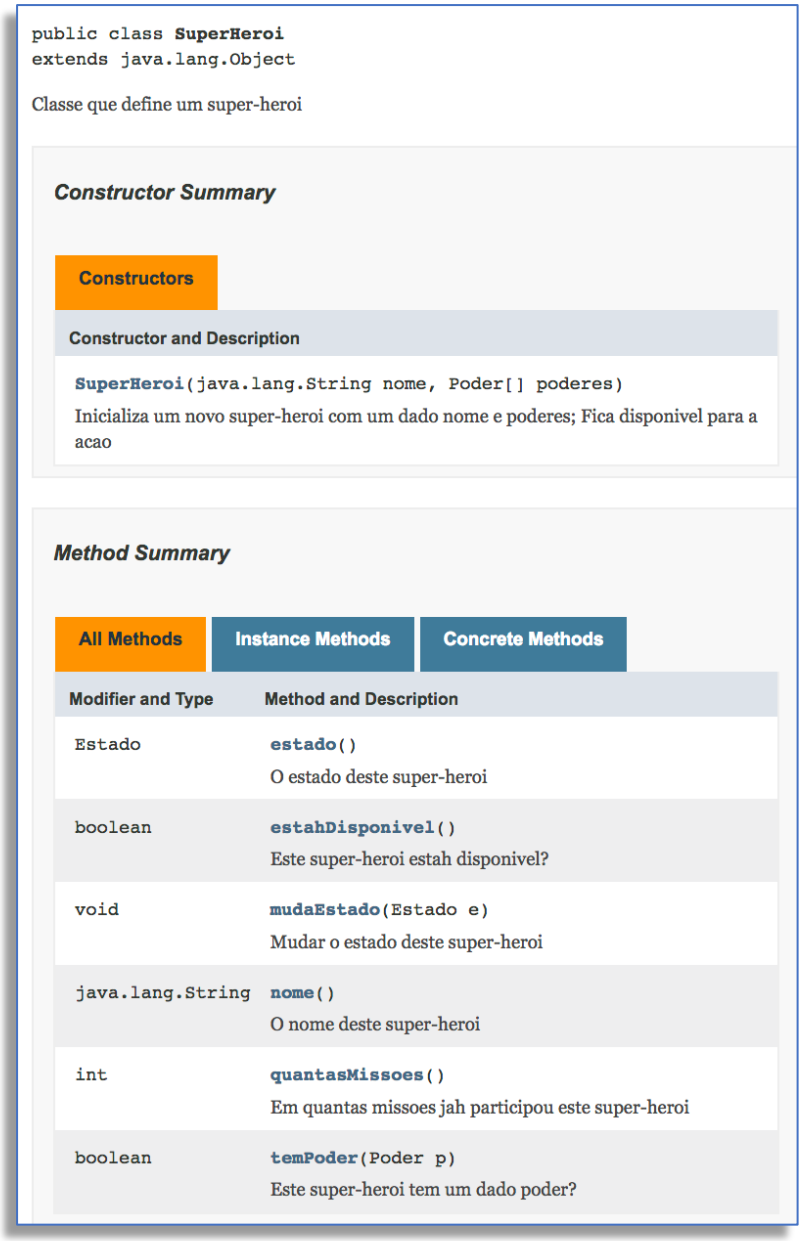
Série 4

Ainda as classes fornecedoras, classes clientes e enumerados

Exercícios

1. A novíssima Liga de Heróis *Pharvell* quer um *software* simples que lhe permita controlar os serviços dos seus super-heróis.
   Em relação a cada super-herói, é necessário saber qual o seu nome, os super-poderes que tem, se está disponível para a ação ou não.
   A Liga terá um certo número de super-heróis que, estando disponíveis, podem ser "contratados" para missões. Os "clientes" da Liga podem contratar um dado super-herói para uma missão que, quando terminada, deixará disponíveis os heróis nela envolvidos que tenham sobrevivido. Para agilizar estas ações, é importante saber algumas informações sobre cada super-herói, saber quais os super-heróis com dados poderes que estão disponíveis e outro tipo de informações úteis.

   Os enumerados `Poder` e `Estado` e as classes `SuperHeroi` e `LigaDeHerois` são os tipos de dados necessários para o que a *Pharvell* pretende.
   No entanto, neste exercício só lhe pedimos para implementar a classe `LigaDeHerois`.
   Considere que os outros tipos já foram implementados por alguém.



`public class SuperHeroi`
`extends java.lang.Object`

Classe que define um super-heroi

Constructor Summary

Constructors

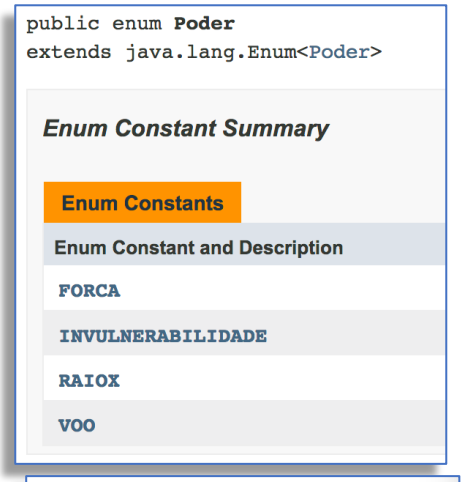
Constructor and Description
`SuperHeroi(java.lang.String nome, Poder[] poderes)` Inicializa um novo super-heroi com um dado nome e poderes; Fica disponivel para a acao

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`Estado`	`estado()` O estado deste super-heroi
`boolean`	`estahDisponivel()` Este super-heroi estah disponivel?
`void`	`mudaEstado(Estado e)` Mudar o estado deste super-heroi
`java.lang.String`	`nome()` O nome deste super-heroi
`int`	`quantasMissoes()` Em quantas missoes jah participou este super-heroi
`boolean`	`temPoder(Poder p)` Este super-heroi tem um dado poder?

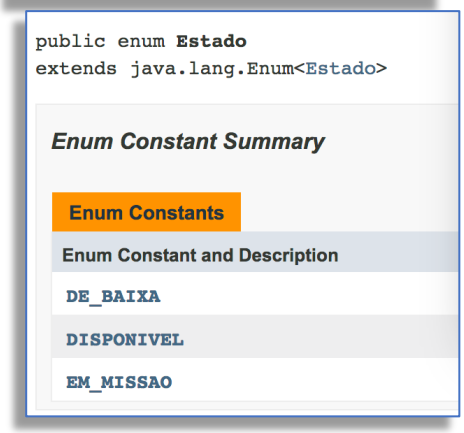
As APIs dos enumerados `Poder` e `Estado` e da classe `SuperHeroi` são apresentados nas figuras seguintes.



public enum **Poder**
extends java.lang.Enum<Poder>

Enum Constant Summary

Enum Constants
Enum Constant and Description
FORCA
INVULNERABILIDADE
RAIOX
VOO



public enum **Estado**
extends java.lang.Enum<Estado>

Enum Constant Summary

Enum Constants
Enum Constant and Description
DE_BAIXA
DISPONIVEL
EM_MISSAO

Então, considerando acessíveis os enumerados `Poder` e `Estado` e a classe `SuperHeroi` descritos atrás, construa a classe `LigaDeHerois` de modo a implementar:

- Um construtor `LigaDeHerois ()` que inicializa uma liga de super-heróis;

e os seguintes métodos (cuja invocação está também exemplificada na classe `TestaLigaDeHerois`):

- `int quantosHerois()` que retorna o número de super-heróis existentes nesta liga;
- `boolean heroiPertence(String nome)` que retorna `true` se um super-herói com nome `nome` pertence a esta liga e `false` caso contrário;
- `void maisUmHeroi(String nome, Poder[] poderes)` que, assumindo que `nome` e `poderes` são diferentes de `null` e o herói de nome `nome` ainda não pertence a esta liga, cria e adiciona a esta liga um super-herói de nome `nome` com os super-poderes indicados em `poderes`; este novo super-herói estará disponível para a ação;
- `boolean estahDisponivel(String nome)` que, assumindo que um super-herói de nome `nome` pertence a esta liga, retorna `true` se ele está disponível para a ação e `false` caso contrário;
- `Estado estadoHeroi(String nome)` que, assumindo que um super-herói de nome `nome` pertence a esta liga, retorna o estado em que se encontra;
- `void contrataHeroi(String nome)` que, assumindo que um super-herói de nome `nome` pertence a esta liga e está disponível para a ação, muda-lhe o estado para `EM_MISSAO`;
- `void libertaHeroi(String nome)` que, assumindo que um super-herói de nome `nome` pertence a esta liga e não está disponível para a ação, muda-lhe o estado para `DISPONIVEL`;
- `String heroiMaisRequisitado()` que retorna o nome do herói que mais vezes esteve em missão; se existirem vários super-heróis com o mesmo número de missões, devolve o nome do mais antigo deles nesta liga, ou seja, do que entrou para a liga em primeiro;
- `List<String> disponiveisComPoderes(Poder[] poderes)` que retorna uma lista com os nomes dos super-heróis desta liga que estão disponíveis e têm todos os poderes indicados em poderes;
- `int[] quantosPorPoder()` que retorna um *array* com tantos elementos quantos os elementos do enumerado `Poder`; o *k*-ésimo elemento deste vetor deve conter o número de super-heróis que possuem o poder cujo número de ordem no enumerado `Poder` é *k*.

Exemplifica-se abaixo a utilização da classe `LigaDeHerois`, num programa que cria uma instância de `LigaDeHerois`, lhe adiciona vários super-heróis e invoca vários métodos.

```
public class TestaLigaDeHerois {

   public static void main(String[] args) {

      LigaDeHerois liga = new LigaDeHerois();

      Poder[] hyperPoderes = {Poder.VOO,Poder.RAIOX,Poder.FORCA};
      liga.maisUmHeroi("Hyperman", hyperPoderes);

      Poder[] rulkPoderes = {Poder.FORCA,Poder.INVULNERABILIDADE};
      liga.maisUmHeroi ("Rulk", rulkPoderes);

      Poder[] phorPoderes = {Poder.FORCA,Poder.VOO,Poder.INVULNERABILIDADE};
      liga.maisUmHeroi ("Phor", phorPoderes);

      System.out.println("Herois disponiveis com voo e forca:");
```

```
        Poder[] procura = {Poder.VOO,Poder.FORCA};
        List<String> herois = liga.disponiveisComPoderes(procura);
        for(String nome : herois){
            System.out.println(nome);
        }

        if(liga.estahDisponivel("Hyperman")){
            System.out.println("Hyperman estah disponivel");
            liga.contrataHeroi("Hyperman");
        }
        if(!liga.estahDisponivel("Hyperman")){
            System.out.println("Hyperman jah nao estah disponivel");
        }

        System.out.println("O heroi mais requisitado eh: " +
                           liga.heroiMaisRequisitado());

        Poder[] novaProcura = {Poder.FORCA};
        herois = liga.disponiveisComPoderes(novaProcura);
        System.out.println("Herois disponiveis com forca:");
        for(String nome : herois){
            System.out.println(nome);
        }

        System.out.println("Quantos herois existem na Liga, com cada Poder?");
        int[] quantosCada = liga.quantosPorPoder();
        for(int i = 0 ; i < quantosCada.length ; i++){
            System.out.println(Poder.values()[i].name() + ": " + quantosCada[i]);
        }
    }
}
```

O resultado de executar este método `main` é:

```
Herois disponiveis com voo e forca:
Hyperman
Phor
Hyperman estah disponivel
Hyperman jah nao estah disponivel
O heroi mais requisitado eh: Hyperman
Herois disponiveis com forca:
Rulk
Phor
Quantos herois existem na Liga, com cada Poder?
RAIOX: 1
VOO: 2
FORCA: 3
INVULNERABILIDADE: 2
```

Herança

Exercícios

2. Considere as seguintes classes. Identifique os vários problemas que o compilador assinalará. Justifique a sua resposta.

```
public class A {
    private int v;

    public A(int i) {
        this.v = i;
    }

    public int v() {
        return this.v + this.x;
    }
}
```

```
public class B extends A {
    private int x;

    public B() {
        this.x = 1.0;
    }

    public B(int i, int j) {
        this.v = i;
        this.x = j;
    }
}
```

3. Considere as seguintes classes.

```
public class A {
    private int v;
    public static final int FIRST = 3;

    public A() {
       this.v = FIRST;
    }

    public A(int i) {
       this.v = i;
    }

    public int value() {
       return this.v;
    }
}
```

```
public class B extends A {
    private double x;

    public B() {
        this.x = this.value() * 2;
    }

    public B(int i) {
        super();
        this.x = i;
    }

    public B(int i, int j) {
        super(i);
        this.x = j;
    }

    public double x() {
        return this.x;
    }
}
```

O que apareceria no ecrã caso se executasse cada uma das sequências de instruções que se seguem:

a) `B b = new B(); System.out.println(b.value() + "   " + b.x());`

b) `B b = new B(4); System.out.println(b.value() + "   " + b.x());`

c) `B b = new B(3, 2); System.out.println(b.value() + "   " + b.x());`

4. Considere as classes `Adega` e `AdegaEspecial` definidas abaixo.

```
public class Adega {
       private int[] quantidades;

       /**
        * Inicializa o novo objeto
        * @param nCubas O numero de cubas que a nova adega vai ter
        * @requires nCubas > 0
        */
       public Adega (int nCubas) {
              this.quantidades = new int [nCubas];
       }

       /**
        * Quantidade de líquido numa dada cuba
        * @param nCuba Numero de ordem da cuba
        * @return Quantidade de líquido na cuba numero nCuba
        * @requires nCuba >= 0 && nCuba < this.totalCubas()
        */
       public int quantidade(int nCuba) {
              return this.quantidades[nCuba];
       }

       /**
        * Acrescenta liquido a uma dada cuba
        * @param nCuba Numero de ordem da cuba
        * @param litros Numero de litros a acrescentar
        * @requires nCuba >= 0 && nCuba < this.totalCubas() && litros > 0
        */
       public void acrescenta(int nCuba, int litros) {
              this.quantidades[nCuba] += litros;
       }

       /**
        * Quantas cubas tem a adega?
        * @return
        */
       public int totalCubas() {
              return this.quantidades.length;
       }
```

```
}

public class AdegaEspecial extends Adega {

    /**
     * Inicializa o novo objeto
     * @param nCubas O numero de cubas que a nova adega vai ter
     * @requires nCubas > 0
     */
    public AdegaEspecial (int nCubas) {
          super(nCubas);
    }

    /**
     * Acrescenta 'as cubas desta adega quantidades de liquido dadas
     * @param litros As quantidades a acrescentar 'as cubas, pela mesma
     *               ordem da ordem das cubas
     * @requires litros != null && litros.length == this.totalCubas()
     */
    public void acrescentaACada(int[] litros) {
          for(int i = 0 ; i < this.totalCubas() ; i++) {
                this.acrescenta(i, litros[i]);
          }
    }
}
```

a) Apresente um diagrama UML que capture a estrutura dessas classes e a forma como se relacionam.

b) Diga, justificando, se as seguintes instruções são aceites pelo compilador.

```
0.  int[] v = {25, 12, 30, 22, 60, 55};
1.   Adega a1 = new Adega(6);
2.   AdegaEspecial ea1 = new AdegaEspecial (4);
3.   Adega a2 = new AdegaEspecial (7);
4.   AdegaEspecial ea2 = new Adega (3);
5.   ea1.acrescenta(1,25);
6.   ea1. acrescentaACada(v);
7.   a2.acrescenta(3,10);
8.   a1.acrescentaACada(v);
9.   a2. acrescentaACada(v);
10.  a2 = ea1;
11.  ea1 = a2;
12.  int x = a1.quantidade(1);
13.  x = a2.quantidade(1);
14.  x = ea1. quantidade(1);
15.  a1.acrescenta(3,-10);
```

c) Contando somente com as instruções corretas, represente o estado do sistema à medida que as instruções são executadas.

d) Que métodos podem ser invocados sobre `a1`, `ea1` e `a2`? Justifique.

5. Considere de novo a classe `Adega` definida no exercício 4.

(a) Fazendo uso da herança, crie uma classe que represente uma adega comunitária (ver API) em que quando se acrescenta vinho a uma cuba, se o resultado de adicionar essa nova quantidade à quantidade já existente nessa cuba for superior à quantidade máxima existente nalguma cuba da adega, a quantidade a acrescentar deverá ser distribuída por todas as cubas da adega.

(b) Analise a bondade dos contratos oferecidos pela classe `Adega` do ponto de vista dos seus clientes. A solução da alínea a) seria válida se os contratos de `Adega` assegurassem, por ex., que quando se acrescenta vinho a uma cuba todas as outras permanecem no estado em que estavam?

`public class` **EcoAdega**
`extends Adega`

Representa adegas comunitarias em que as quantidades são por vezes distribuidas pelas varias cubas

Author:
in

Constructor Summary

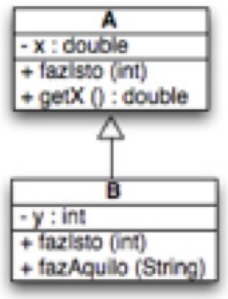
Constructors	
Constructor and Description	
EcoAdega`(int nCubas)`	Inicializa o novo objeto

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	**acrescenta**`(int nCuba, int litros)` Acrescenta liquido a uma dada cuba ou uma parte desse liquido a todas as cubas no caso de, com esse liquido extra, essa cuba se tornar a cuba com maior quantidade

6. Considere a hierarquia de classes apresentada na figura. A classe `B` redefine o método `fazIsto` e acrescenta um novo método `fazAquilo` e um novo atributo `y`.



Indique, para cada uma das instruções seguintes, se seriam ou não aceites pelo compilador.

a) `A p1 = new B ();`
b) `A p2 = new A ();`
c) `double r = p2.getX();`
d) `p1.fazAquilo("ola");`
e) `B p3 = new B ();`
f) `B p4 = new A ();`
g) `p3 = p1;`
h) `p1.fazIsto(55);`
i) `double s = p3.getX();`
j) `((B) p1).fazAquilo("bom dia");`
k) `B p5 = p3;`
l) `System.out.println(p5.toString());`

7. Considere as classes `Adega` e `AdegaEspecial` definidas anteriormente e a classe:

```
public class AdegaPlus extends AdegaEspecial {
	private int nRecargas;

	/**
	 * Inicializa o novo objeto
	 * @param nCubas O numero de cubas que a nova adega vai ter
	 * @requires nCubas > 0
	 */
	public AdegaPlus (int nCubas) {
		super(nCubas);
	}

	/**
	 * Acrescenta liquido a uma dada cuba ou uma parte desse liquido
	 * a todas as cubas no caso de, com esse liquido extra, essa cuba
	 * se tornar a cuba com maior quantidade
	 * @param nCuba Numero de ordem da cuba 'a qual acrescentar
	 * @param litros Numero de litros a acrescentar
	 * @requires nCuba >= 0 && nCuba < this.totalCubas() && litros > 0
	 */
	@Override
	public void acrescenta(int nCuba, int litros) {
		super.acrescenta(nCuba, litros);
		this.nRecargas++;
	}

	/**
	 * Quantas recargas ja' foram feitas nesta adega
	 */
	public int quantasRecargas() {
		return this.nRecargas;
	}
}
```

(a) Considere as seguintes declarações e inicializações de variáveis. Indique, em cada um dos casos, se a instrução é correta e quais os métodos que pode invocar sobre a variável declarada.

```
1.  Adega a1 = new Adega(6);
2.  Adega ae1 = new AdegaEspecial(23);
3.  Adega ap1 = new AdegaPlus(15);
```

```
4.  AdegaEspecial a2 = new Adega(4);
5.  AdegaEspecial ae2 = new AdegaEspecial(10);
6.  AdegaEspecial ap2 = new AdegaPlus(8);
7.  AdegaPlus a3 = new Adega(11);
8.  AdegaPlus ae3 = new AdegaEspecial(9);
9.  AdegaPlus ap3 = new AdegaPlus(17);
```

(b) Qual é o resultado produzido pelo seguinte pedaço de código? Justifique, descrevendo as instruções que vão sendo executadas e o estado dos objetos.

```
1.  AdegaPlus ap = new AdegaPlus(6);
2.  ap.acrescenta(2, 25);
3.  int[] v = {25, 12, 30, 22, 60, 55};
4.  ap.acrescentaACada(v);
5.  System.out.println(ap.quantasRecargas());
```

8. Considere as seguintes classes:

```
public abstract class A {
    private int x;
    private StringBuilder sb;
    public A (int val){
        x = val;
        sb = new StringBuilder();
    }
    public int getX(){ return x; }
    public void addText(int v, String s){
        if (isValid(v))
            sb.append(s);
    }
    protected void incrementX(int i, int v){
        if (i < v) x += i;
    }
    public String contents(){
        return sb.toString();
    }
    protected abstract boolean isValid(int i);
}
```

```
public class B extends A {
    private String t;
    public B (int x, String s){
        super (x);
        t = s;
    }
    public B (String s){
        super (s.length());
        t = s;
    }
    public B (){
        this (0,"");
    }
    @Override
    public boolean isValid(int i) {
        return i > getX();
    }
    @Override
    public String contents() {
        return super.contents() + t;
    }
}
```

a) O compilador dá erro em ?

```
B b0 = new B("PCO");
```

b) O que aparece no ecrã após a execução de

```
B b0 = new B("PCO");
System.out.println(b0.getX());
```

c) O que aparece no ecrã após a execução de

```
B b1 = new B(2, "P");
b1.addText(5, "C");
System.out.println(b1.contents());
```

d) O que aparece no ecrã após a execução de

```
B b2 = new B("Ola!");
b2.incrementX(2,6);
if (b2.isValid(3)){
    System.out.println(b2.contents());
} else {
    System.out.println(b2.getX());
}
```

e) O que aparece no ecrã após a execução de

```
B b3 = new B();
System.out.println(b3.contents());
```

9. Considere as seguintes classes:

```
public class Person {
    private String name, surname;

    public void setFirstname(String n) {
        this.name = n;
    }

    public void setSurname(String n) {
        this.surname = n;
    }

    public String getFirstname() {
        return this.name;
    }

     public String getSurname() {
        return this.surname;
    }

    public String getName() {
        return getFirstname() + " " +
                              getSurname();
    }

    public String toString() {
        return "My name is " + getName();
    }
}
```

```
public class Agent extends Person {

    public String getFirstname() {
        return "";
    }
}
```

(a) Apresente o "*output*" produzido pela seguinte sequência de comandos:

```
1.  Agent a = new Agent();
2.  a.setSurname("Bond");
3.  a.setFirstname("Jaime");
4.  System.out.println(a.toString());
```

(b) Altere a classe `Agent`, sem remover nem alterar o método `getFirstname` que aí se encontra implementado, por forma a que o "*output*" da sequência de instruções apresentada na alínea anterior seja "My name is Bond, Jaime Bond".