MC302
Primeiro semestre de 2017

Laboratório 4

Professores: Esther Colombini (esther@ic.unicamp.br) e Fábio Luiz Usberti (fusberti@ic.unicamp.br)
PEDs: (Turmas ABCD) Elisangela Santos (ra149781@students.ic.unicamp.br), Lucas Faloni (lucasfaloni@gmail.com), Lucas David (lucasolivdavid@gmail.com), Wellington Moura (wellington.tylon@hotmail.com)
PEDs (Turmas EF) Natanael Ramos (naelr8@gmail.com), Rafael Arakaki (rafaelkendyarakaki@gmail.com)
PAD: (Turmas ABCD) Igor Torrente (igortorrente@hotmail.com)
PAD: (Turmas EF) Bleno Claus (blenoclaus@gmail.com)

---

1 Objetivo

O objetivo desta atividade consiste na prática de Hierarquias de generalização/especialização através de herança simples e sobrescrita de métodos.

2 Herança

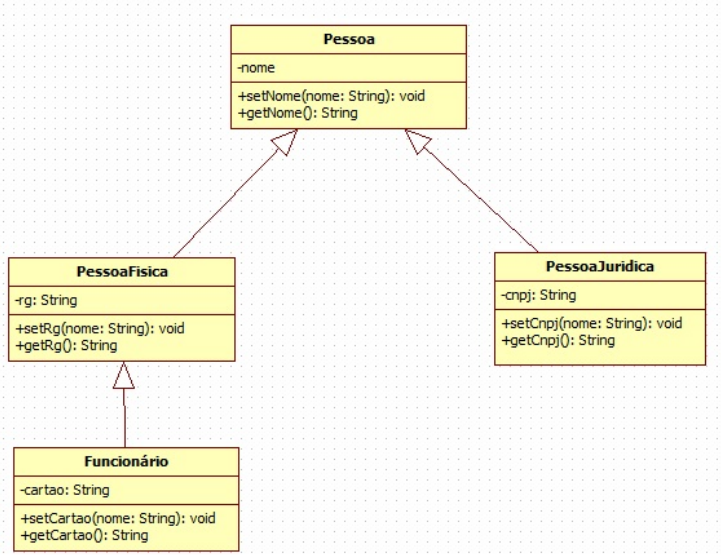

Figura 1: Exemplo de Hierarquia de Classes - Especialização

A Figura 1 apresenta um exemplo típico de hierarquias. A classe **Pessoa** contempla os atributos e métodos comuns às diversas classes que representem pessoas. Por exemplo, diferentes tipos de pessoas possuem a característica **nome** e por isso esta informação deve estar presente na classe mais abrangente (mais genérica). Desta forma, classes mais especializadas podem herdar os atributos da classe **Pessoa**, adicionando atributos não comtemplados na classe principal (denominada superclasse). Por exemplo, a

classe **PessoaJuridica**, que herda de **Pessoa**, terá os atributos **nome** e **cnpj**, sendo que **nome** foi herdado e **cnpj** foi definido na classe atual, pertencendo apenas a esta classe. O código a seguir corresponde à implementação da hierarquia apresentada na Figura 1.

```java
package Exemplos;

public class Pessoa {
    private  String nome;

    public Pessoa(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    @Override
    public String toString() {
        return "Pessoa{" + "nome=" + nome + '}';
    }
}
```

Listing 1: A classe **Pessoa**.

```java
package Exemplos;

public class PessoaJuridica extends Pessoa{
    private String cnpj;

    public PessoaJuridica(String nome) {
        super(nome);
    }

    public String getCnpj() {
        return cnpj;
    }

    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }

    @Override
    public String toString() {
        return "PessoaJuridica{" + "cnpj=" + cnpj + '}';
    }
}
```

Listing 2: A classe **PessoaJuridica**.

```java
package Exemplos;

public class PessoaFisica extends Pessoa{
    private String rg;

    public PessoaFisica(String nome) {
        super(nome);
```

```
8       }
9
10      public String getRg() {
11          return rg;
12      }
13
14      public void setRg(String rg) {
15          this.rg = rg;
16      }
17
18      @Override
19      public String toString() {
20          return "PessoaFisica{" + "rg=" + rg + '}';
21      }
22  }
```

Listing 3: A classe **PessoaFisica**.

```
1   package Exemplos;
2
3   public class Funcionario extends PessoaFisica {
4       private String cartao;
5
6       public Funcionario(String cartao, String nome) {
7           super(nome);
8           this.cartao = cartao;
9       }
10
11      public String getCartao() {
12          return cartao;
13      }
14
15      public void setCartao(String cartao) {
16          this.cartao = cartao;
17      }
18
19      @Override
20      public String toString() {
21          String s = super.toString();
22          s += "\n Funcionario{" + "cartao=" + cartao + '}';
23          return s;
24      }
25  }
```

Listing 4: A classe **Funcionario**.

Observe que, para que a classe especializada (i.e. subclasse) consiga herdar todas as características da classe original (i.e. superclasse), é necessário que na declaração da subclasse seja acrescentada a palavra chave *extends*.

As subclasses (classes especializadas) não herdam os construtores da superclasse (classe original). Logo, é necessário chamar explicitamente o construtor da superclasse como a primeira ação a ser realizada no construtor da subclasse. Isso acontece através do comando *super*.

3 Atividade

A Figura 2 apresenta um exemplo de hierarquias que iremos construir na aula de hoje. Continuaremos trabalhando com classes baseadas no jogo chamado Hearthstone[1] ©. Nesta atividade o principal foco

[1] http://us.battle.net/hearthstone/pt

será a familiarização com generalização e herança em java e a programação de algumas classes: **Carta**, **Baralho**, **Dano**, **Buff**, **DanoArea**.

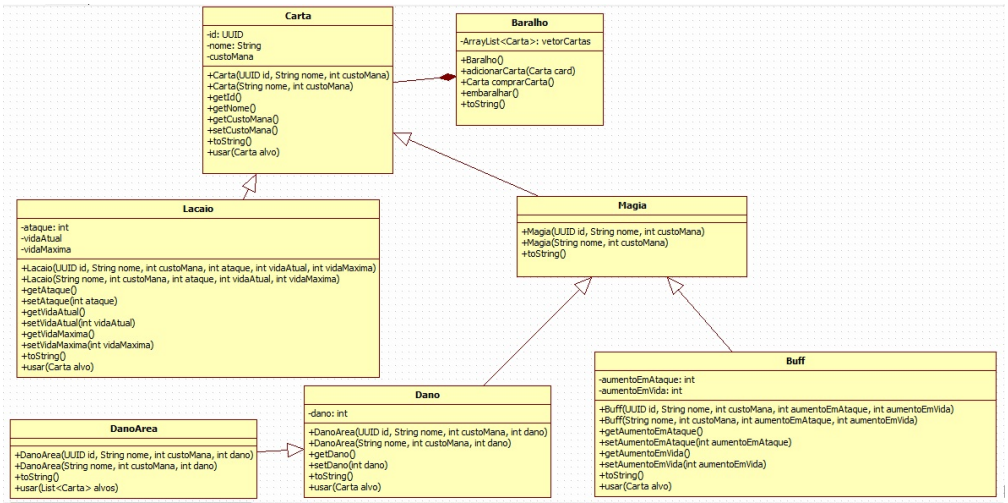

Figura 2: Exemplo de Hierarquia de Classes - Especialização da classe Carta

3.1 Primeiramente...

1. Crie um projeto chamado Lab4 e, dentro dele, os pacotes *base*, *base.cartas*, *base.cartas.magias* e *util*.

2. Reutilize as classes **BaralhoArrayList** e **CartaLacaio** criadas nos laboratórios anteriores, colando-as no pacote *base* e *base.cartas*, respectivamente. Renomeie a classe **BaralhoArrayList** para **Baralho** e **CartaLacaio** para **Lacaio**.

3. Acrescente uma classe vazia **Magia** no pacote *base.cartas.magias* e copie a classe **Util** já criada para o pacote *util*.

3.2 A classe base Carta

1. Crie a classe **Carta** no pacote base com no diagrama representado na Fig. 2.

2. O método **toString()** da classe **Carta** deve retornar um objeto da classe **String** contendo uma descrição geral dos atributos da carta.

3. A classe **Carta** deverá possuir um método **usar(Carta alvo)** vazio que será sobrescrito nas subclasses para modificar os atributos da carta **Lacaio**.

3.3 A classe Lacaio

1. Faça **Lacaio** extender **Carta**. Remova todos os atributos repetidos (i.e., todos os atributos de **Lacaio** que já foram implementados por **Carta**).

2. Modifique os construtores de **Lacaio** de tal forma que estes reutilizem um ou mais construtores da superclasse **Carta**.

3. Implemente o método **toString()** em **Lacaio**. Esta sobrescrita deve retornar uma descrição de todos os atributos do lacaio, reutilizando o método **toString()** da superclasse.

4. Sobrescreva o método **Carta.usar(Carta alvo)** em **Lacaio**. Na classe **Lacaio**, este método deve aplicar dano sobre o alvo considerando os fatores como *ataque* e *vidaAtual*.

4 As classes Buff, Dano e DanoArea

1. Faça **Magia** extender **Carta**.

2. Implemente os construtores em Magia de tal forma que estes reutilizem um ou mais construtores da superclasse **Carta**.

4.1 Classe Buff

1. Crie a classe **Buff** no pacote *base.cartas.magias* como descrito no diagrama representado pela Fig. 2.

2. Faça **Buff** extender **Magia**.

3. Implemente os construtores de **Buff** de tal forma que estes reutilizem um ou mais construtores da superclasse **Carta**.

4. Implemente o método **toString()** em **Buff**. Esta sobrescrita deve retornar uma descrição de todos os atributos do buff, reutilizando o método **toString()** da superclasse.

5. Sobrescreva o método **Carta.usar(Carta alvo)** em **Buff**. Na classe **Buff**, este método deve aumentar as vidas e o ataque de um lacaio.

4.2 Classe Dano

1. Crie a classe **Dano** no pacote *base.cartas.magias* como descrito no diagrama representado pela Fig. 2.

2. Faça **Dano** extender **Magia**.

3. Implemente os construtores de **Dano** de tal forma que estes reutilizem um ou mais construtores da superclasse **Carta**.

4. Implemente o método **toString()** em **Dano**. Esta sobrescrita deve retornar uma descrição de todos os atributos do dano, reutilizando o método **toString()** da superclasse.

5. Sobrescreva o método **Carta.usar(Carta alvo)** em **Dano**. Aqui, este método deve danificar um alvo considerando aspectos como a vidaAtual e o dano.

4.3 Classe DanoArea

1. Crie a classe **DanoArea** no pacote *base.cartas.magias* como descrito no diagrama representado pela Fig. 2.

2. Faça **DanoArea** extender **Dano**.

3. Implemente os construtores de **DanoArea** de tal forma que estes reutilizem um ou mais construtores da superclasse **DanoArea**.

4. Implemente o método **toString()** em **DanoArea**. Esta sobrescrita deve retornar uma descrição de todos os atribudos do dano, reutilizando o método **toString()** da superclasse.

5. Implemente o método **Carta.usar(List<Carta> alvos)** em **DanoArea**. Aqui, este método deve danificar igualmente uma sequência de alvos, considerando aspectos como a vidaAtual e o dano.

5 Baralho com objetos do tipo Carta, Lacaio, Magia, Dano e DanoArea

Vamos ilustrar um cenário onde herança faça sentido: refatore **Baralho** para que este contenha qualquer tipo de cartas.

6 Tarefas

Responda as questões a seguir sucintamente em um arquivo texto que deverá ser submetido juntamente com o código:

1. Crie uma nova classe **Main** que contenha o método **main**. Instancie um **Baralho** e adicione várias cartas de diferentes tipo à ele. Apresente na tela as informações de todas as cartas atualmente no baralho.

2. A instanciação de objetos do tipo **Carta** faz sentido neste exemplo? Por quê?

3. O método **usar(List<Carta> alvo)** da classe **DanoArea** está sobrescrevendo algum método? Explique.

4. É possível acessar o atributo nome diretamente na classe **Lacaio**? Explique.

5. Imagine que existe um método estático **Carta.meuMetodoEstatico**. Seria possível chamá-lo nas subclasses? Em caso afirmativo, que instrução deveria ser empregada neste caso?

6. Quais os benefícios de implementar herança neste cenário?

7 Submissão

Para submeter a atividade, utilize o Moodle (`https://www.ggte.unicamp.br/ea`). Crie um arquivo texto com as respostas para cada item da seção tarefas e as saídas geradas pelo código. Compacte o código-fonte contido no diretório **src** juntamente com arquivo de respostas no formato .zip ou similar e nomeie-o **Lab4-000000.zip**, trocando '000000' pelo seu número de RA. Submeta o arquivo na seção correspondente para esse laboratório no moodle da disciplina MC302.

Datas de entrega

- Dia **10/04** Turma **ABCD** até às 23:55h
- Dia **07/04** Turma **EF** até às 23:55h