MC302
Primeiro semestre de 2017

Laboratório 1

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

---

1 Objetivo

O objetivo desta atividade consiste na prática de Hierarquias de generalização/especialização e Herança Simples e Múltipla. Por meio de encapsulamento e herança novas técnicas podem ser adicionadas as classes. Enquanto a herança é um poderoso mecanismo de especialização, o polimorfismo oferece um mecanismo para a generalização. No Java uma subclasse somente pode exttender diretamente uma outra subclasse, não sendo permitido a herança múltipla, como ocorre em C++. Ou seja, uma determinada classe em Java pode implementar múltiplas interfaces e fazer uso de agregação para responder por chamadas a métodos que tecnicamente só poderiam acontecer se existisse a "herança múltipla". Nesta aula treinaremos especialização e heranças. Interfaces e generalização serão assuntos futuros.

2 Atividade

Continuaremos trabalhando com classes baseadas no jogo chamado Hearthstone[1] ©.Nesta atividade o principal foco será a familiarização com generalização e herança em java e a programação da classe chamada **Carta** . Crie um projeto chamado Lab4. No projeto crie os mesmos pacotes da aula passada com.seuPrimeiroNome.Util e com.seuPrimeiroNome.base. Cole a classe CartaLacaio (utilizada no Lab3), a classe CartaMagia (utilizada no Lab2), a classe Baralho (utilizada no Lab3) e a classe BaralhoArrayList dentro do projeto no pacote base. No pacote Util cole a classe Util.java.

3 Classe Carta

Crie a classe Carta no pacote base com os seguintes atributos: A classe Carta deve ter os seguintes atributos:

- ID (número inteiro)
- nome (cadeia de caracteres - *String*)
- tipo (TipoCarta) - Enum[2] ©: TipoCarta.LACAIO, TipoCarta.MAGIA
- ataque (número inteiro)

[1] http://us.battle.net/hearthstone/pt
[2] Enumeração será assunto dos próximos capitulos, portanto não usaremos este atributo agora

- vida (número inteiro)
- vidaMesa (número inteiro)
- mana (número inteiro)
- magiaTipo (TipoMagia) - Enum[3] ©: TipoMagia.ALVO, TipoMagia.AREA
- magiaDano (número inteiro)
- turno (número inteiro)

O exemplo abaixo apresenta a declaração da classe Carta e seus atributos. Note que todas as variáveis são declaradas como privadas (**private**). Note também a implementação dos métodos de acesso get() e set(), esses métodos são comumente utilizados na linguagem Java para acessar os atributos dos objetos.

```java
public class Carta{

    private int ID ;
    private  String nome;
    private int ataque;
    private int vida;
    private int vidaMesa;
    private int mana;
    private int magiaDano;
    private int  turno;

    // Metodo construtor aqui


    // Demais metodos aqui
    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

}
```

Carta.java

Além disso a classe Carta deve conter um método construtor, o método construtor deve receber como argumentos os atributos para inicializar o objeto. Para ilustrar esse conceito melhor, veja o exemplo abaixo.

```java
public Carta(int ID, String nome, int ataque, int vida, int vidaMesa, int mana, int
    magiaDano, int turno) {
        this.ID = ID;
        this.nome = nome;
        this.ataque = ataque;
        this.vida = vida;
        this.vidaMesa = vidaMesa;
        this.mana = mana;
        this.magiaDano = magiaDano;
        this.turno = turno;
    }
```

MetodoConstrutor.java

[3]Enumeração será assunto dos próximos capitulos, portanto não usaremos este atributo agora

Também é necessário que a classe Carta possua uma função **toString()** que devolve uma String contendo uma descrição geral dos atributos da carta. Veja o exemplo abaixo:

```java
@Override
    public String toString() {
        return "Carta{" + "ID=" + ID +
                          ", nome=" + nome +
                          ", ataque=" + ataque +
                          ", vida=" + vida +
                          ", vidaMesa=" + vidaMesa +
                          ", mana=" + mana +
                          ", magiaDano=" + magiaDano +
                          ", turno=" + turno + '}';
    }
```

toString.java

Como já aprendemos são utilizados os métodos de get e set. Precisamos programar estes métodos antes de utilizá-los, para cada atributo da classe Carta deve existir um método get e set correspondente. O formato do método toString() a ser implementado é aquele que ja aprendemos.

Faça a implementação do método **construtor**, métodos **get()** e **set()** de todos os atributos e do método **toString()** para a classe **Carta**.

4 Herança

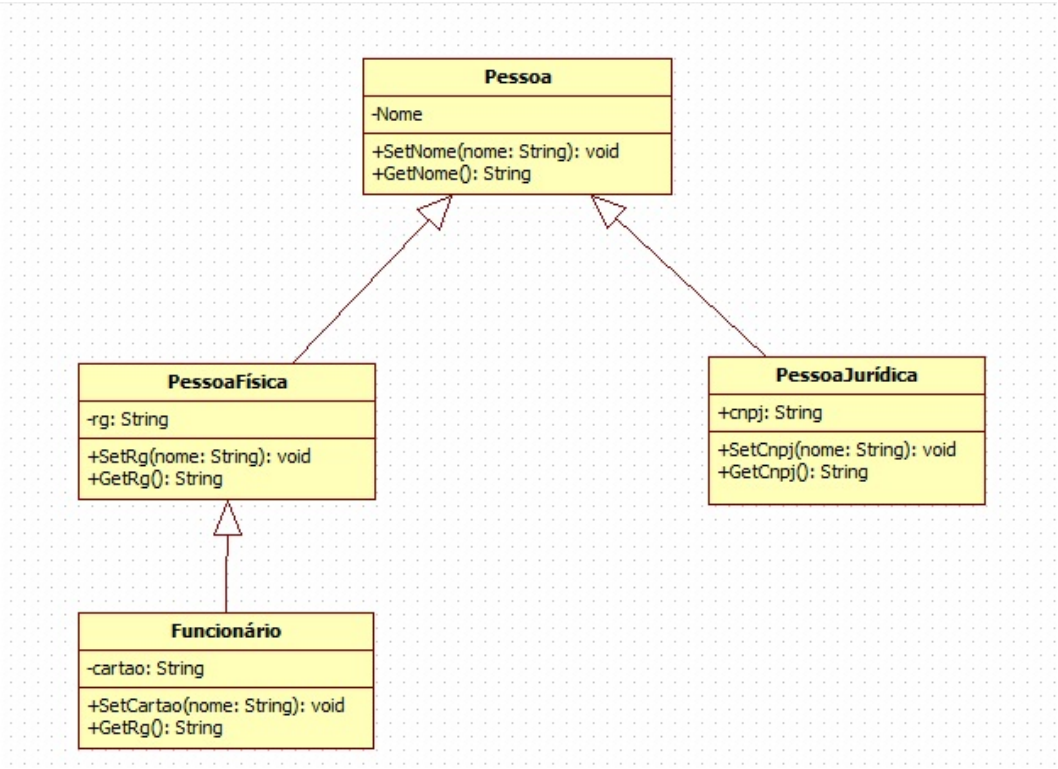

Figura 1: Exemplo de Hierarquia de Classes - Especialização

Diferentes tipos de classes podem ter algo em comum entre elas. Por exemplo diferentes tipos de pessoas podem ter as mesmas característica (nome, sobrenome), a esta classe podemos chamar classe Pessoa. Classes mais especializadas podem herdam os mesmos atributos da classe Pessoa e podem ter atributos não comtemplados na classe principal (denominada superclasse).Por exemplo: A classe PessoaJuridica pode terá o atributo nome e cnpj, nome foi herdado mais cnpj pertence somente a esta classe. Observe na figura 1, o diagrama de hierarquia de classes especializadas. Veja a seguir o código do diagrama de hierarquia de classes, demonstrado 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 + '}';
    }



}
```

Pessoa.java

```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 + '}';
    }


```

```
24
25 }
```

PessoaJuridica.java

```
1  package Exemplos;
2
3  public class PessoaFisica extends Pessoa{
4   private String rg;
5
6      public PessoaFisica(String nome) {
7          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
23
24
25 }
```

PessoaFisica.java

```
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
20 }
```

Funcionario.java

4.1 Adicionando Herança

Para que a classe especializada (denominada subclasse) consiga herdar todas as características da classe original (denominada superclasse), é necessário que a subclasse seja acrescida da palavra *Extends*. Veja

o exemplo a seguir:

```
1
2 package Exemplos;
3
4 public class CartaLacaio extends Carta {
5
6
7 }
```

Extends.java

4.2 Construtores das classes especializada (denominada subclasse)

As subclasses (classes especializadas) não herdam os construtores da superclasse (classe original), então é necessário chamar explicitamente o construtor da superclasse. Isso acontece através do comando *super*. Os parâmetros necessários são passados dentro dos parênteses que seguem essa palavra-chave (*super*). Veja o exemplo a seguir:

```
1 public CartaLacaio(int vidaAtual, int vidaMaxima, int custoMana, int ID, String nome,
2                     int ataque, int vida, int vidaMesa, int mana, int magiaDano, int
                          turno) {
3         super(ID, nome, ataque, vida, vidaMesa, mana, magiaDano, turno);
4         this.vidaAtual = vidaAtual;
5         this.vidaMaxima = vidaMaxima;
6         this.custoMana = custoMana;
7     }
```

construtor.java

5 Tarefas

- Modifique a classse CartaLacaio e CartaMagia para que elas tenham todos os atributos e características da classe Carta (denominada superclasse).
- Refaça a programação dos métodos construtores das classes CartaLacaio e CartaMagia.
- Refaça a programação dos métodos get e set das classes CartaLacaio e CartaMagia.

6 Submissão

Para submeter a atividade utilize o Moodle (`https://www.ggte.unicamp.br/ea`). Salve os arquivos dessa atividade em um arquivo comprimido no formato .tar.gz e nomeie-o **Lab1-000000.tar.gz** trocando '000000' pelo seu número de RA. Submeta o arquivo na seção correspondente para esse laboratório no moodle da disciplina MC302.