Universidade do Estado de Santa Catarina
Curso de Bacharelado em Ciência da Computação
Disciplina: Programação Orientada a Objetos (POO)

Herança

A herança é um conceito da orientação a objetos que dispõe as classes relacionadas entre si de maneira hierárquica. Sendo assim os descendentes de uma classe herdam todos os atributos e métodos de seus ancestrais e podem também criar seus próprios atributos e métodos. As classes descendentes são chamadas de subclasses. A classe pai de uma determinada subclasse é chamada de superclasse.

A herança é utilizada quando o programador deseja especializar uma classe anteriormente criada. Considere a classes *Cliente* abaixo e a classe *ClientePreferencial* no Exemplo 1a. Observe que ela possui os mesmos atributos da classe *Cliente* acrescido com novos atributos e métodos (em vermelho). No entanto a definição desta classe não é adequada, visto que acaba por gerar uma certa redundância na definição de atributos e métodos em relação à classe *Cliente* previamente definida.

A solução ideal é mostrada pelo Exemplo 1b. Nele a classe *ClientePreferencial* estende a classe *Cliente* através do uso da palavra chave *extends* logo após o nome da classe. Neste caso é preciso apenas definir os atributos e métodos específicos da classe *ClientePreferencial* enquanto que os demais atributos e métodos são herdados da classe *Cliente*.

Podemos ter uma série de extensões sucessivas de classes, isto é, podemos formar uma hierarquia de tamanho ilimitado. No entanto Java não permite a herança múltipla, isto significa dizer que uma mesma subclasse não pode ter mais de uma superclasse.

```
public class Cliente {
   private String nome;
   private int cpf;

   public Cliente(String _nome, int _cpf) {
         nome = _nome;
         cpf = _cpf;
   }

   public void setNome(String _nome){
         nome = _nome;
   }
   public void setCpf(int _cpf){
         cpf = _cpf;
   }
   public String getNome(){
         return nome;
   }
   public int getCpf(){
         return cpf;
   }
}
```

Exemplo 1a:

```
public class ClientePreferencial {
   private String nome;
   private int cpf;
   private Date desde;
```

```
    private double desconto;

    public ClientePreferencial(String nome, int cpf, Date desde,
    double desconto) {
          this.nome = nome;
          this.cpf = cpf;
          this.desde = desde;
          this.desconto = desconto
    }
    public void setNome(String nome){
          this.nome = nome;
    }
    public void setCpf(int cpf){
          this.cpf = cpf;
    }
    public String getNome(){
          return nome;
    }
    public int getCpf(){
          return cpf;
    }
    public void setDesde(Date desde) {
          this.desde = desde;
    }
    public Date getDesde() {
          return this.desde;
    }
    public void setDesconto(double desconto) {
          this.desconto = desconto;
    }
    public double getDesconto() {
          return this.desconto;
    }
    public double calcularDesconto(double valor) {
          return this.desconto * valor;
    }
}
```

Exemplo 1b:

```
public class ClientePreferencial extends Cliente {
    private Date desde;
    private double desconto;

    public ClientePreferencial(String nome, int cpf, Date desde,
    double desconto) {
          super(nome, cpf);
          this.desde = desde;
          this.desconto = desconto
    }
    public void setDesde(Date desde) {
          this.desde = desde;
    }
    public Date getDesde() {
          return this.desde;
    }
    public void setDesconto(double desconto) {
          this.desconto = desconto;
    }
    public double getDesconto() {
          return this.desconto;
    }
```

```
    public double calcularDesconto(double valor) {
          return this.desconto * valor;
    }
}
```

Propriedades da Herança

a) Utiliza-se a palavra chave ***super*** quando no código da subclasse desejarmos nos referir a superclasse;
b) A classe java.lang.Object é a classe pai de todas as classes Java;
c) Os construtores da superclasse não são herdados;
d) Antes de executar qualquer um dos construtores definidos em uma subclasse, é preciso executar um dos construtores da superclasse ("É preciso existir um objeto pai antes de existir um objeto filho!"). Se a chamada ao construtor da superclasse não é efetuada explicitamente pelo programador o Java irá executar o construtor default da superclasse (se ele existir). Observe o Exemplo 2a, se mandarmos compilá-lo um erro será gerado, pois o construtor default da classe *Cliente* não existe! Uma solução é apresentada pelo Exemplo 2b, onde uma chamada a um dos construtores da classe *Cliente* é efetuada. Outra solução seria definirmos o construtor default (sem parâmetros) explicitamente para a classe *Cliente*.
Importante: A chamada a um dos construtores da superclasse deve ser o primeiro comando a ser executado por um construtor de subclasse. Lembre-se que é preciso existir pai antes de existir filho!

Exemplo 2a:

```
public class ClientePreferencial extends Cliente {
    private Date desde;
    private double desconto;

    public ClientePreferencial(String nome, int cpf, Date desde,
    double desconto) {
          this.nome = nome;
          this.cpf = cpf;
          this.desde = desde;
          this.desconto = desconto
    }

    public void setDesde(Date desde) {
          this.desde = desde;
    }
    public Date getDesde() {
          return this.desde;
    }
    public void setDesconto(double desconto) {
          this.desconto = desconto;
    }
    public double getDesconto() {
          return this.desconto;
    }
    public double calcularDesconto(double valor) {
          return this.desconto * valor;
    }
}
```

Exemplo 2b:

```
public class ClientePreferencial extends Cliente {
    private Date desde;
    private double desconto;
```

```
    public ClientePreferencial(String nome, int cpf, Date desde,
    double desconto) {
          super(nome, cpf);
          this.desde = desde;
          this.desconto = desconto
    }

    public void setDesde(Date desde) {
          this.desde = desde;
    }
    public Date getDesde() {
          return this.desde;
    }
    public void setDesconto(double desconto) {
          this.desconto = desconto;
    }
    public double getDesconto() {
          return this.desconto;
    }
    public double calcularDesconto(double valor) {
          return this.desconto * valor;
    }
}
```

e) É possível definir atributos nas subclasses com os mesmos nomes dos atributos da superclasse. Podemos também criar métodos nas subclasses com os mesmos nomes dos métodos da superclasse, porém neste caso ocorre o que é chamado de **Sobreposição de Métodos** (*Overriding*). Observe o Exemplo 2c. O método *getNome()* é sobrescrito para conferir um tratamento especial ao nome do cliente preferencial. O método *getNome()* de qualquer objeto do tipo *ClientePreferencial* irá sempre retornar o nome do cliente precedido de "Excelentíssimo". Porém o mesmo método quando utilizado por um objeto do tipo *Cliente* irá apenas retornar o nome do cliente, conforme definido no método *getNome()* da classe pai.

A Sobreposição de métodos é mais uma característica de polimorfismo.

Existem algumas regras para a sobreposição de métodos:

- O tipo de retorno do método que sobrescreve deve ser igual ao do método subscrito;
- O método que sobrescreve não pode lançar exceções não compatíveis com as exceções lançadas pelo método subscrito;
- O método que sobrescreve não pode utilizar um modificador de acesso mais restrito que o utilizado pelo método subscrito.

Exemplo 2c:

```
public class ClientePreferencial extends Cliente {
    private Date desde;
    private double desconto;

    public ClientePreferencial(String nome, int cpf, Date desde,
    double desconto) {
          super(nome, cpf);
          this.desde = desde;
          this.desconto = desconto
    }

    public String getNome() {
          return "Excelentíssimo(a) " + super.getNome();
    }
```

```
    public void setDesde(Date desde) {
          this.desde = desde;
    }
    public Date getDesde() {
          return this.desde;
    }
    public void setDesconto(double desconto) {
          this.desconto = desconto;
    }
    public double getDesconto() {
          return this.desconto;
    }
    public double calcularDesconto(double valor) {
          return this.desconto * valor;
    }
}
```

Pacotes

A declaração de pacote (*package*) deve ser o primeiro comando de um código fonte Java. O pacote especifica a localização da classe em questão, na estrutura de diretórios a partir do diretório de trabalho Java (*classpath*).

Pacotes são utilizados para agrupar classes relacionadas.

Exemplo 3:

```
package lpg2.aulas;

public class ClientePreferencial extends Cliente {
...
```

Import

A instrução de *import* é semelhante aos includes da linguagem C. Sempre que necessitarmos de uma classe que não está dentro do pacote da classe em que estamos trabalhando, é necessário utilizarmos o import. O import especifica o pacote em que a classe que estamos precisando está. Na classe *ClientePreferencial* estamos utilizando a classe *Date*, uma classe da api Java. Se tentarmos executar esta classe como ela está teremos um erro pois o compilador não encontrará a definição da classe *Date*. Para isto no Exemplo 4a, definimos um import para que a classe *Date* possa ser utilizada pela classe *ClientePreferencial*.

Existem duas formas de declaração de imports. A primeira (Exemplo 4a), é o caminho absoluto, onde especificamos a estrutura de diretório dos pacotes até chegarmos à classe que almejamos. E a segunda (Exemplo 4b), é o caminho relativo, onde especificamos apenas a estrutura de diretórios do pacote da classe que necessitamos, colocando um *. O * permite que todas as classes do referido pacote (diretório em questão) sejam importadas. Esta é a forma mais recomendada para declaração de imports.

Importante:

- O * importa todas as classes de um determinado diretório, por exemplo todas as classes que estão no diretório java.util. As classes que se encontram em sub-diretórios não são importadas, por exemplo se tivéssemos classes em java.util.io elas não seriam importadas.
- Por *default* o pacote java.lang é automaticamente importado para qualquer classe java criada.

Exemplo 4a:

```
package lpg2.aulas;
```

```
import java.util.Date;

public class ClientePreferencial extends Cliente {
...
```

Exemplo 4b:

```
package lpg2.aulas;
import java.util.*;

public class ClientePreferencial extends Cliente {
...
```