LARBACK

HERANÇA

Professor Leonardo Cabral da Rocha Soares

Lattes: **http://lattes.cnpq.br/3184602538494393**

HERANÇA

A Herança, em programação orientada a objetos (POO), é a possibilidade de escrever novas classes partindo de classes já existentes, ou seja, podemos construir objetos especializados que herdam as características de objetos mais generalistas.

O conceito de herança mimetiza as características reais de diversos sistemas, como por exemplo a classificação biológica de animais:

- Animais (conceito geral)
- Mamíferos e Aves (características mais específicas)

HERANÇA

Em Java, identificamos o relacionamento de herança entre classes (*é-um*) é feito utilizando-se a *keyword* extends:

```
public class Animal {
	public String nomeCientifico;
}
public class Mamifero extends Animal {
}
```

Todos os atributos, métodos e constantes não privados de Animal serão herdados por Mamifero.

A classe Animal é chamada de super classe ou classe Pai.

A classe Mamifero é chamada de subclasse ou classe derivada.

HERANÇA

Em Java, todas as classes são implicitamente subclasses de Object. Sempre que escrevemos uma classe sem utilizar a *keyword* extends em sua assinatura, o compilador java adiciona "extends Object" – Assim, se voltarmos a declaração de Animal, no exemplo anterior, poderíamos lê-la como:

```
public class Animal [extends Object]{
    public String nomeCientifico;
}
```

* Os [] são utilizados apenas para mostrar que este trecho foi adicionado pelo compilador e não pelo desenvolvedor

HERANÇA – MÉTODOS CONSTRUTORES

LARBACK

Métodos construtores não são herdados. Entretanto, como são eles que fornecem o caminho para a criação dos objetos e o objeto sendo criado neste momento (sub-classe) depende dos recursos de um objeto anterior (super-classe) é necessário que o construtor da super-classe seja executado.

Assim, o primeiro comando de um método construtor deverá ser sempre super() ou this(). O this() para chamarmos outro construtor da própria classe ou o super() para executarmos um construtor da super-classe. Se não iniciarmos o construtor com um desses comandos (ou não escrevermos um construtor) o compilador irá adicionar super().

HERANÇA – MÉTODOS CONSTRUTORES

LARBACK

```
public class Animal {
        public String nomeCientifico;
        public Animal(){
                System.out.println("Animal criado");
        }
}
public class Mamifero extends Animal {
        public Mamifero() {
                System.out.println("Mamifero criado");
}
```

Se criarmos um objeto de Mamifero (new Mamifero()) veremos que as duas Strings serão impressas – "Animal criado" e "Mamifero criado" – Isso ocorre porque o compilador Java adicionou ao construtor de Mamifero uma chamada ao construtor de Animal – super()

HERANÇA – SOBRESCRIÇÃO (@OVERRIDE)

Os métodos herdados da super-classe podem ter seu comportamento alterado (desde que não sejam declarados como ***final***) para algo mais específico. Este processo é feito criando-se um novo método na sub-classe com a mesma assinatura do método sobrescrito – Se a assinatura do método for alterada estaremos gerando uma nova sobrecarga e não uma sobrescrição.

HERANÇA – SOBRESCRIÇÃO (@OVERRIDE)


```
public class Animal {
        public void dormir(){
                System.out.println("zzzzZZZZZzzzZZZZzzz");
        }
}
public class Mamifero extends Animal {
        public void dormir(){
                System.out.println("RonronronronZZZRonronron");
        }
}
```

Embora os objetos de Animal e Mamifero compartilhem o método dormir, o comportamento *dormir()* em Mamifero foi alterado para uma versão mais específica.

HERANÇA – CLASSES ABSTRATAS

LARBACK

Algumas generalizações criam representações abstratas demais para possuírem objetos. Embora essas classes devam existir para evitar a repetição de código nas sub-classes, é inadimissível que um objeto deste tipo seja criado.

Por exemplo, imagine um sistema onde os clientes são agrupados em Clientes Pessoas Físicas e Clientes Pessoas Jurídicas. Ambos os tipos de clientes possuem características em comum, entretanto, não há uma relação direta entre as classes. Para resolvermos este problema, criamos uma classe mais genérica que represente os clientes. Esta, deverá ser herdada por ClientesPessoasFisicas e ClientesPessoasJuridicas.

HERANÇA – CLASSES ABSTRATAS

LARBACK

Se criassemos uma classe concreta para representar Clientes, esta poderia possuir objetos e, no nosso caso, isso não é possível – O objeto deve ser uma pessoa física ou jurídica. Para evitar isso, criamos uma classe abstrata:

```
public abstract class Cliente {

	/* Aributos e métodos comuns a todos clientes

}
```

A classe Cliente não poderá gerar objetos. Ela deverá ser herdada por classes concretas e objetos destas poderão ser criados.

HERANÇA – CLASSES ABSTRATAS

LARBACK

```
public abstract class Cliente {
	/* Aributos e métodos comuns a todos clientes
}
public class ClientePessoaFisica extends Cliente {
}
public class ClientePessoaJuridica extends Cliente {
}
```

Continuando o exemplo, imaginemos que todos os clientes (pessoas físicas - PF ou jurídicas) possuam um método para calcular o limite de crédito. PF e PJ possuem regras diferentes para calcular este limite mas ambos devem possuí-lo, assim, este método deve ser definido em Cliente.

HERANÇA – CLASSES ABSTRATAS

Mas qual será o comportamento deste método em cliente? As regras são definidas apenas nas sub-classes, não há um comportamento padrão para implementarmos na super-classe, assim, definiremos, na super-classe, um método abstrato:

```
public abstract class Cliente {
	/* Aributos e métodos comuns a todos clientes
	public abstract long calcularLimiteCredito();
}
```

O método abstrato não possuirá corpo e deverá, obrigatoriamente, ser sobrescrito (implementado) pela primeira classe não abstrata que herdá-lo.

HERANÇA – CLASSES ABSTRATAS

LARBACK

```
public abstract class Cliente {
	/* Aributos e métodos comuns a todos clientes
	public abstract long calcularLimiteCredito();
}
public class ClientePessoaFisica{
	public long calcularLimiteCredito() {
		return 20;
	}
}
public class ClientePessoaJuridica{
	public long calcularLimiteCredito(){
		return 42;
	}
}
```