

Material de Apoio Aula 1

Herança (Revisão)

Observe o código das classes Fatorial e Fibonacci apresentados abaixo.

<pre>class Fatorial { private int n, res; public Fatorial(int aux) { n = aux; res = 1; } public void calcula() { int i; for(i = 1 ; i < n ; i++) res = res * i; } int getRes() { return res; } }</pre>	<pre>class Fibonacci { private int n, res; public Fibonacci(int aux) { n = aux; res = 0; } public void calcula() { int i, t, a = 0, b = 1; for(i = 1 ; i < n ; i++) { res = a + b; a = b; b = res; } } int getRes() { return res; } }</pre>
---	--

Exercício: O que estas classes possuem em comum?

- 1) _____
- 2) _____
- 3) _____

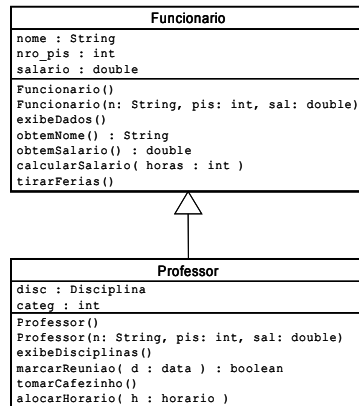
Herança: hierarquia entre classes para reaproveitamento de código (Revisão)

A herança é um recurso de programação do paradigma orientado a objetos que permite o reaproveitamento de esforço já despendido no desenvolvimento de um software. Este recurso permite que uma classe seja definida herdando características de uma outra classe já existente. Dá-se a denominação de superclasse a classe original e de subclasse a nova classe. Outras nomenclaturas populares: classe base e classe derivada, classe mãe e classe filha (note que não existem outros parentescos, tipo: classes irmãs). Uma das grandes vantagens da herança é de diminuir a necessidade de replicar código em um programa, permitindo que trechos de códigos definidos para uma classe sejam reaproveitados na construção de outras.

É importante ressaltar que quando uma classe é herdada, mais do que atributos e métodos, também é herdada a estrutura desta classe. Assim, a herança entre duas classes define uma relação de **é um**. Ou seja, considere que a classe Y seja subclasse da classe X. Caso seja criado um objeto y1 da classe Y, este objeto é um objeto da classe Y e também é um objeto da classe X. O inverso, no entanto, não é verdadeiro: um objeto criado da classe X não é necessariamente um objeto da classe Y.

Tomemos como exemplo, a necessidade de definir a classe `Professor` em software de gestão de recursos humanos. Objetos da classe `Professor` necessitam manipular atributos específicos deste tipo de funcionário de uma empresa, como `categoria` (para enquadramento funcional) e `disciplinas` que leciona. Também são necessários alguns métodos: `alocarHorario`, `tomarCafezinho` e `marcarReuniao`. No entanto, objetos da classe `Professor` também necessitam manipular informações referentes a funcionários em geral, tal como `nome` e `nro_pis` e da mesma forma responder por ações genéricas a todos funcionários de uma empresa, como `tirarFerias` e `calcularSalario`. Estas características (atributos e métodos) podem ser regrupados em uma superclasse no software: a classe `Funcionario`. Desta forma, toda especificação desenvolvida para `Funcionario` seria reaproveitada na classe `Professor` e em qualquer outra classe necessária para outro tipo de funcionário da empresa.

Em uma forma gráfica, representamos a herança desta forma:



Sintaxe da herança em Java (Revisão)

Em Java, a herança pode ser utilizada através da palavra reservada **extends**, da seguinte forma:

```

class NovaClasse extends ClasseJaExistente {
    ...
}
  
```

Classe abstrata

Continuando a análise do primeiro exemplo deste material. Já foi observado que a estrutura das classes são comuns. Elas, portanto, “respondem” pela mesma estrutura, no caso, uma estrutura para representar cálculo de valores para posições de séries. Esta estrutura comum faz parte da **concepção** das classes que devem representar o cálculo de séries genéricas. No entanto, uma questão deve ser colocada: *como se calcula o valor de uma série genérica?* Em uma visão abstrata do problema, não é possível definir valores para uma série genérica. Mas isto não impede que a estrutura seja definida. Mais do que isto, pode ser necessário especificar uma estrutura e alocar a tarefa de implementação para a equipe de implementação. Assim, no projeto de um software que necessite cálculos de séries, uma classe genérica poderia ser dada por:

```

abstract class Serie { // abstract indica que existe pelo menos um método abstrato
    protected int n, res;

    public Serie( int aux ) { // Inicialização default
        n = aux;
        res = 1;
    }

    public Serie( int aux, int aux2 ) { // Informando o valor inicial
        n = aux;
        res = aux2;
    }

    abstract public void calcula(); // Método abstrato puro

    int getRes() { // Serviço que pode ser reaproveitado tal e qual
        return res;
    }
}
  
```

Desta forma, as classes Fibonacci e Fatorial poderiam ser reimplementadas como segue.

<pre> class Fatorial extends Serie { public Fatorial(int aux) { super(aux, 1); } public void calcula() { int i; for(i = 1 ; i < n ; i++) res = res * i; } } </pre>	<pre> class Fibonacci extends Serie { public Fibonacci(int aux) { super(aux, 0); } public void calcula() { int i, t, a = 0, b = 1; for(i = 1 ; i < n ; i++) { res = a + b; a = b; b = res; } } } </pre>
---	--

Exercício: Por que os atributos **n** e **res** foram definidos como **protected** na classe **Serie** ?

Exercício: Não é possível criar um objeto da classe `Serie`, pois esta classe é uma *classe abstrata pura*. Você saberia dizer por quê?

Polimorfismo

Do grego: *poli* = muitas, *morphos* = formas. Muitas formas.

Deve ser salientado que, as classes `Fibonacci` e `Fatorial` herdam e especializam a classe `Serie`. Assim, todo objeto que for criado da classe `Fibonacci` (ou `Fatorial`), além de ser um objeto `Fibonacci` (ou `Fatorial`), também é um objeto da classe `Serie`. Isto faz com que uma referência a um objeto `Serie` possa referenciar objetos `Fatorial` e `Fibonacci`. De forma prática, o código abaixo é válido:

<pre>class Teste { public static void main(String args) { Fibonacci f; Calculadora c; c = new Calculadora(); f = new Fibonacci(5); System.out.print(c.CalculaSerie()); } }</pre>	<pre>class Calculadora { public int CalculaSerie(Serie s) { s.calcula(); return s.getRes(); } }</pre>
--	---

Observe que o resultado da aplicação da linha “`s.calcula();`” no código da classe `Calculadora` é polimórfico. Quer dizer, o serviço a ser de fato executado é definido conforme o objeto real passado (no caso do exemplo trata-se de um objeto da classe `Fibonacci`). Isto é feito em tempo de execução, sendo seu suporte associado à **ligação dinâmica**.

Exercícios

1. Crie novas extensões para a classe `Serie` considerando as séries de Lucas, Padovan, Perrin.

Lucas	Padovan	Perrin
$U(0) = 0$ $U(1) = 1$ $U(n + 2) = PU(n + 1) - QU(n)$, com $n > 1$, com P e Q arbitrários (Note que <code>Fibonacci</code> é o caso $P = 1, Q = -1$)	$P(0) = P(1) = P(2) = 1$, $P(n) = P(n - 2) + P(n - 3)$, para $n > 1$	$P(0) = 3$ $P(1) = 0$ $P(2) = 2$ $P(n) = P(n - 2) + P(n - 3)$ para $n > 2$

2. Defina uma estrutura de classes para representar alunos. Nesta estrutura, concebida de forma simplista, alunos possuem apenas 3 atributos, o primeiro uma `String` representando o nome do aluno e dois numéricos (`double`), representando suas notas, por exemplo, GA e GB. Deve ser possível calcular a média dos alunos segundo diferentes critérios: para alunos da Unisinos a média é ponderada, onde GA tem peso 0,33 e GB tem peso 0,67, alunos da instituição Alfa tem a média calculada por média aritmética simples $(GA + GB)/2$, alunos da instituição Beta utilizam apenas a nota mais alta.
3. Defina uma estrutura de classes para representar as seguintes figuras: retângulo, quadrado, elipse e círculo. A operação comum que todas as classes devem implementar é o cálculo da sua área.

Retângulo	Quadrado	Elipse	Círculo
$A = h * l$	$A = l^2$	$A = \Pi ab$	$A = \Pi a^2$

4. Defina uma estrutura de classes para representar datas nos formatos brasileiro e americano.
5. No método `main`, crie um array de séries de 10 posições. Nas posições pares insira objetos `Fatorial` e nas ímpares objetos `Fibonacci`. Cada objeto deve ser inicializado com o valor n correspondendo a sua posição no vetor. Imprima os valores na computados.
6. Definir a estrutura de dados para pacotes de mensagens. Um pacote é composto por 3 campos: dois numéricos (`int`) representando a origem e o destino do pacote e um `String`, representando o dado a ser enviado. O serviço relevante é o envio do pacote, para tanto, pacotes devem ser capazes de “empacotar” os dados em um único string: neste string, a primeira informação é o endereço do destino, a segunda é o endereço da origem do pacote e a terceira é a mensagem propriamente dita. Implemente uma estrutura de classes onde diferentes métodos de criptografia são aplicados. Por exemplo: trocar as letras a por x e x por a, em outro substitui cada letra pelo seu ASCII correspondente.