

Material de Apoio Aula 9

Herança

Observe o código das classes Fatorial e Fibonacci apresentados abaixo.

```
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;
    }
}
```

```
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;
        }
    }
}
```

O que estas classes possuem em comum?

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

Herança: hierarquia entre classes para reaproveitamento de código

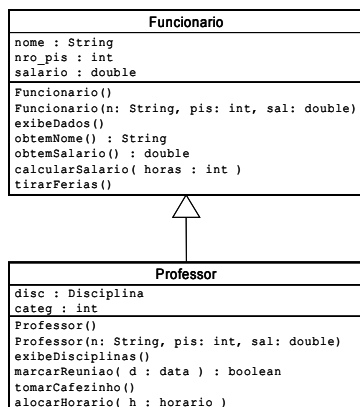
A herança é um recurso de programação do paradigma orientado a objetos que permite o reaproveitamento de esforço já despendido no desenvolvimento de um software. Este recurso permite que uma classe seja definida herdando características de uma outra classe já existente. Da-se a denominação de superclasse 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 regroupados em uma superclasse no software: a classe `Funcionário`. 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

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

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

Exemplo:

Implemente duas classes: Carro e Carrao. Ambas classes devem suportar operações para abastecimento do veículo e deslocamento. No entanto, objetos da classe Carrao possuem ar-condicionado, que pode ser ligado ou desligado através de métodos próprios. Quando o ar estiver ligado, o consumo de combustível aumenta em 10%. Implemente estas duas classes utilizando uma estrutura baseada em herança.

```
class Carro {
    private double comb, // total de combustível no tanque
                cons; // consumo de combustível por quilometro
    private int cont; // contador de quilometragem percorrida

    public Carro() {
        comb = 0; // carro de tanque vazio
        cont = 0; // carro zero quilometro
        cons = 10; // consome 1 litro de gasolina a cada 10 km
    }

    public Carro( double c ) { // recebe quantidade de combustível inicial
        if( c > 55 ) comb = 55; // o reservatorio do tanque e' limitado em 55 litros
        else comb = c; // primeiro abastecimento realizado
        cont = 0; // carro zero quilometro
        cons = 10; // consome 1 litro de gasolina a cada 10 km
    }

    public void setConsumo( double c ) {
        cons = c;
    }

    public double getConsumo() {
        return cons;
    }

    public boolean abastece( double c ) {
        if( (c+comb) > 55 ) return false; // quantidade de combustível cabe no tanque?
        comb += c; // caso caiba, abastece
        return true;
    }
}
```

```

public boolean anda( int q ) {
    if( (comb/cons) > q ) return false; // nao ha combustivel suficiente para andar q quilometros
    comb -= comb/cons; // consome combustivel
    cont += q; // atualiza contador de quilometros
    return true; // o carro andou
}
public String status() {
    String s = new String("Combustivel: " + comb + "\nQuilometros percorridos: " + cont +
        "\nPode andar: " + cons*comb + " quilometros com o combustivel disponivel\n");
}
}

Carrao extends Carro {
    private boolean ar; // ar-condicionado: true para ligado, false para desligado

    public Carrao() {
        super();
        ar = false;
    }

    public Carrao( double c ) {
        super( c );
    }

    public void ligaAr() {
        double novocons;

        if( ar == false ) {
            novocons = getConsumo() * 1.1; // calcula mais 10% do consumo padrao
            setConsumo( novocons ); // seta novo consumo
            ar = true;
        }
    }

    public void desligaAr() {
        double novocons;

        if( ar == true ) {
            novocons = getConsumo() / 1.1; // retorna ao consumo padrao
            setConsumo( novocons ); // seta novo consumo
            ar = false;
        }
    }
}
}

```

Escopo de visibilidade

Importante ressaltar que membros privados em uma classe continuam sendo privados a objetos de outras classes. Assim, sendo criado um objeto de uma classe derivada, os métodos deste objeto não possuem acesso aos membros privados de sua superclasse. Veja no exemplo anterior a implementação do método `ligaAr()`. Este método não acessa diretamente o atributo privado `cons`. O acesso é realizado através de métodos definidos na interface da classe `Carro`. A justificativa desta restrição de visibilidade de escopo está associada ao encapsulamento de dados: o escopo de visibilidade um membro privado deve ser restrita ao objeto que o contém.

Exercícios:

1. Implemente uma classe genérica para reagrupar as partes comuns das classes `Fatorial` e `Finonacci` apresentadas como primeiro exemplo. Reimplemente estas classes.
2. Implemente um novo tipo de veículo: o jipe, o qual não possui ar-condicionado, mas pode ter acionado a tração em quatro rodas. A qual implica em aumentar o consumo de combustível em 10%.
3. Defina a estrutura de classes para representar contas em banco: conta genérica, conta corrente comum, conta corrente especial e conta-poupança. Represente o esquema de relacionamento entre classes e realize sua implementação.
4. Ler o material do prof. Aníbal em <http://www.inf.unisinos.br/~anibal/prog1aula6.pdf> fazer todos os exercícios de sua lista.