

Material de Apoio 5

Sumário

Este material apresenta conceitos gerais de herança. Em aula também serão abordados outros conceitos: classe abstrata, polimorfismo e ligação dinâmica.

Motivação

M1: 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) _____

M2: É possível desenhar uma *figura geométrica*?

Herança: hierarquia entre classes para reaproveitamento de código

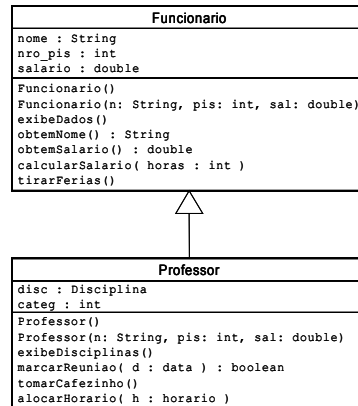
A herança é um recurso de programação do paradigma orientado a objetos que permite o reaproveitamento de esforço já despendido no desenvolvimento de um software. Este recurso permite que uma classe seja definida herdando características de uma outra classe já existente. Dá-se a denominação de superclasse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

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

```
class NovaClasse extends ClasseJaExistente {
    ...
    // definição dos membros
    ...
}
```

Exemplo 1:

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 combustivel no tanque
                 cons; // consumo de combustivel 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 combustivel 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 combustivel 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;
        }
    }
}
}
```

Exemplo 2:

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.

Atenção: A solução que será proposta propõe uma abordagem didática: queremos verificar os conteúdos relacionados à matéria tratada em aula; em um problema real um estudo mais aprofundado sobre o negócio (no caso, contas em banco) deverá ser realizado.

Solução: O primeiro problema a ser atacado é a modelagem da solução em termos do relacionamento de classes. Para tanto, inicie identificando os atributos e as ações comuns a todas as possíveis contas de banco. Em uma classe *genérica* para tal problema, identificamos como atributos mais evidentes os seguintes:

- Número da conta
- Nome do proprietário da conta
- Endereço do proprietário da conta
- Saldo da conta

Embora esta lista de atributos não esteja completa, ela é suficiente para nossos propósitos. Os serviços que uma conta qualquer deve prestar são:

- Permitir depósito
 - Fazer com que um valor depositado seja incorporado ao saldo total.

- Permitir saque
 - Fazer com que seja retirado um valor do saldo total. Observe que, em uma conta genérica, a situação *default* é não permitir que o saldo fique negativo.
- Apresentar o status da conta
 - Apresentar dados referentes à conta (nome, endereço, saldo, ...) conforme necessidade.
- Retornar o saldo da conta
 - Permitir a leitura do valor que a conta possui em saldo.

O passo seguinte consiste em identificar quais são as classes que podem ser definidas a partir desta classe já existente. No nosso exercício, estas classes já foram nomeadas: poupança, conta comum e conta especial. A terceira etapa consiste, portanto, na identificação de como pode ser configurada a estrutura de heranças entre classes.

Conta Poupança. Uma conta poupança possui todas as características de uma conta genérica. Os atributos são os mesmos e os serviços definidos para as contas genéricas podem ser aproveitados. No entanto, contas poupança possuem uma particularidade: ao final do mês, é aplicado um cálculo sobre a rentabilidade do saldo. Desta forma, uma nova classe pode ser definida para classes poupança, herdando toda a estrutura de serviços da classe genérica para conta em banco, adicionando um serviço extra:

- Calcula rendimento
 - Aplica o cálculo de rendimento do saldo da poupança considerando a taxa do mês.

Conta Comum. Uma conta comum corresponde a uma conta genérica onde é possível fazer retiradas através de cheques, no entanto, sendo passado um cheque cujo montante seja maior que o saldo em conta, a conta comum entra no vermelho, ou seja, em uma situação que o correntista deve procurar seu gerente e regularizar a situação. Portanto, em adição aos atributos definido para a classe genérica para contas em banco, objetos da conta comum devem também possuir o atributo:

- Situação da conta
 - Indicando se ela está ou não em situação regular.

E, em relação aos serviços, três novos métodos são identificados:

- Emite talão de cheques
 - Permite a emissão de um talonário de cheques
- Compensa cheque
 - Implementa uma retirada através de um cheque emitido. Caso o cheque possua um valor maior que o saldo em conta, o saque não é efetuado, a conta entre em situação irregular e nenhum outro talão de cheque pode ser emitido nem outro cheque (mesmo de menor valor) pode ser compensado.
- Regulariza a situação
 - Permite que um gerente, após conversar com o correntista, permita novamente a geração de talonário e compensação de cheques.

Conta Especial. Uma conta especial difere de uma conta comum por permitir que o saldo fique negativo, considerando o limite que o correntista possui. Portanto, em adição aos atributos definidos para a classe de conta comum, objetos da conta especial devem também possuir o atributo:

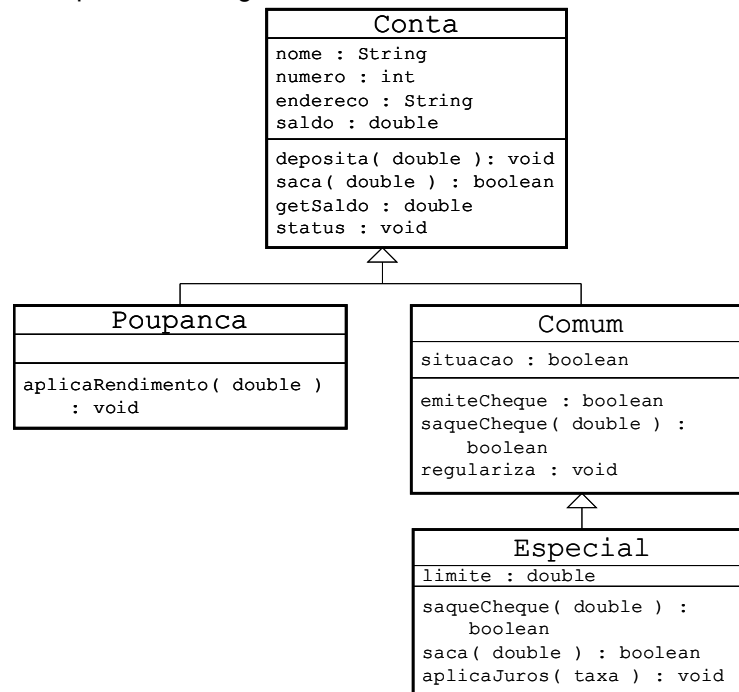
- Limite da conta
 - Indicando quanto negativo a conta pode ficar.

Os serviços já identificados para conta comum são necessários a uma conta especial, no entanto, pode ser herdada apenas a estrutura, o comportamento destes deve ser diferenciado. Um novo método deve ser incorporado para calcular os juros que devem ser cobrados do saldo negativo.

- Permitir saque
 - Fazer com que seja retirado um valor do saldo total. Observe que, em uma o saldo pode ficar negativo, dentro do limite especificado para ela.
- Compensa cheque

- Permite que seja efetuada uma compensação de cheque, deixando a conta no negativo. A situação da conta ficará irregular apenas se o montante do cheque for maior que o saldo corrente mais o limite.
- Cálculo de juros
 - Obtém o valor dos juros a serem cobrados da conta pelo valor que se encontra negativo.

A visão gráfica do problema é apresentada na figura abaixo.



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. Justifique o uso do mecanismo de herança na programação orientada a objetos.
2. O que é uma classe abstrata? É possível criar um objeto de uma classe abstrata? E é possível ter uma referência de uma classe abstrata?
3. Implemente uma classe abstrata para reagrupar as partes comuns das classes `Fatorial` e `Finonacci` apresentadas como primeiro exemplo. Reimplemente estas classes.
4. Implemente e execute o exemplo de código apresentado em Exemplo 1.
5. Implemente uma classe abstrata para representar, de forma genérica, todas as funcionalidades dos carros apresentados no Exemplo 1.
6. 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%.
7. Implemente a estrutura de classes do Exemplo 2, assumindo que a classe `Conta` é uma classe abstrata.