



Instituto Federal de Santa Catarina – IFSC
Campus São José

Programação Orientada a Objetos

Herança

Prof. Francisco de Assis S. Santos, Dr.

São José, 2014.

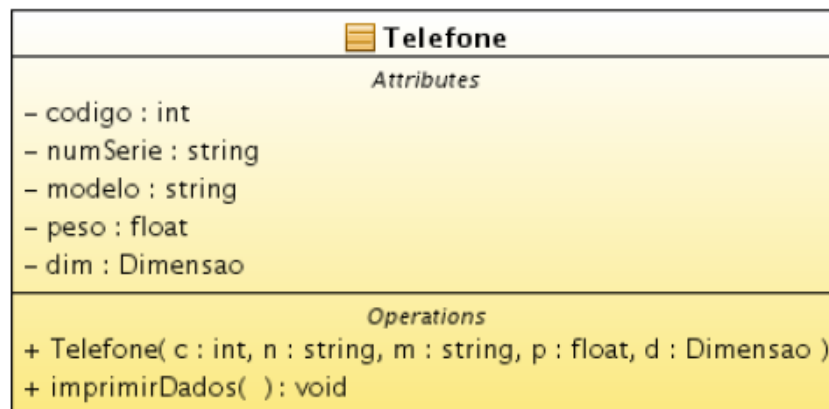
Herança em POO

- O conceito de herança torna mais rápido o desenvolvimento de softwares complexos
 - Novas classes são criadas baseadas em classes existentes
- Classe filha, subclasse ou classe derivada
 - A classe que herda os atributos e funções de outra classe
- Classe pai, superclasse ou classe base
 - A classe cujo membros são herdados por outras classes

Ideal para casos onde são necessárias classes distintas para atacar problemas específicos. Porém, tais classes necessitam compartilhar um núcleo comum

Herança – Exemplo: Sistema para cadastro de produtos

- Uma industria da área de telecomunicações necessita de um sistema para cadastrar os produtos que fabrica
- Aparelho telefônico
- As informações necessárias para o cadastro são:
- código, número de série, modelo, cor, peso, dimensões (AxLxP)



Herança – Exemplo: Sistema para cadastro de produtos

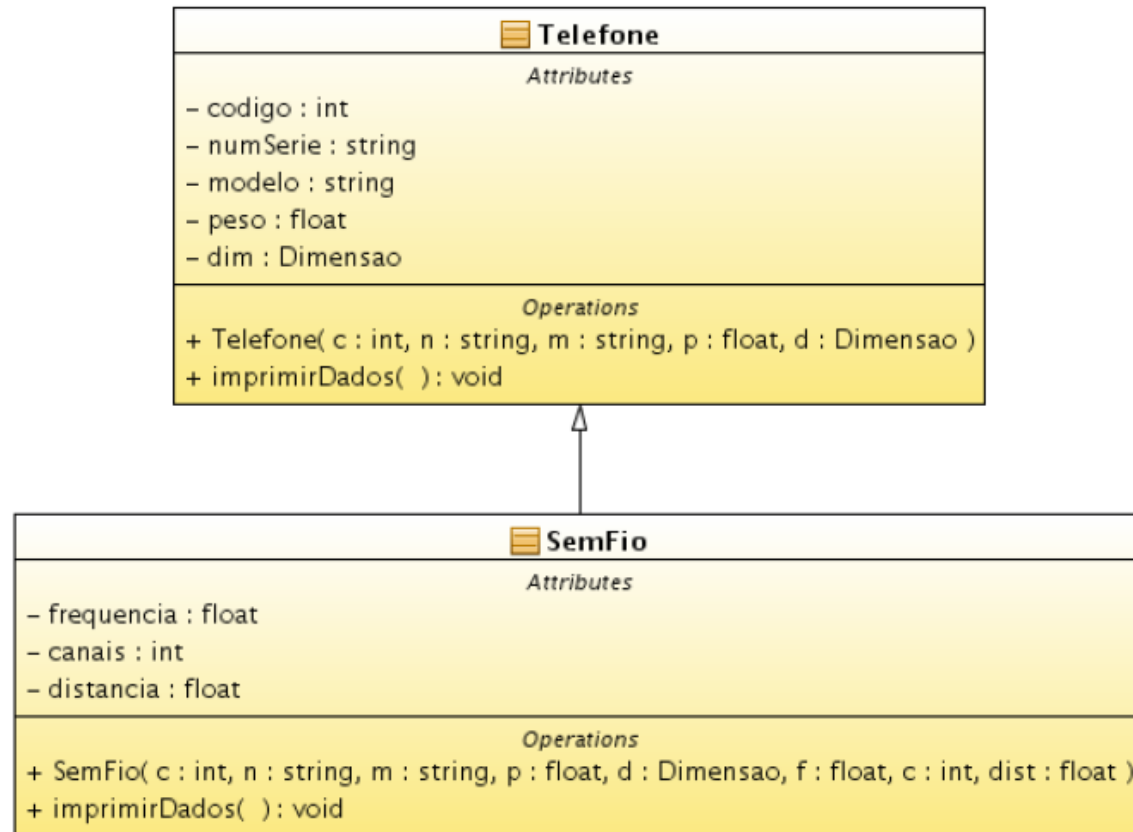
- A empresa começou a fabricar também telefones sem fio
- Os telefones sem fio compartilham todas as características de um telefone, porém possuem novas características
- frequência, quantidade de canais, distância de operação
- O atual sistema não permite cadastrar essas novas informações

O que fazer?

1 - Criar uma nova classe telefone sem fio e colocar nela tudo o que tem na classe telefone mais as características do telefone sem fio?

2 - Herdar as características comuns da classe telefone e adicionar as particulares do telefone sem fio

Herança – Exemplo: Sistema para cadastro de produtos



Herança : Superclasse

```
public class Telefone{
    private int codigo;
    private String numSerie, modelo;
    private float peso;
    private Dimensao dim;

    public Telefone(int c, String s, String m, float p, Dimensao d)
    {
        this.codigo = c; this.peso = p; this.dim = d;
        this.numSerie = s;this.modelo = m;
    }

    public void imprimirDados(){
        System.out.println("Codigo: " + this.codigo);
        ...
        this.dim.imprimirDados();
    }
}
```

Herança : Subclasse

```
public class Semfio extends Telefone{
    private float frequencia, distancia;
    private int canais;

    public SemFio(int c, String s, String m, float p, Dimensao d,
        int ca, float f, float dis){
        super(c, s, m, p, d); // invocando o construtor da
            superclasse
        this.frequencia = f;
        this.distancia = dis;
        this.canais = ca;
    }

    // sobreescrita do metodo da superclasse
    public void imprimirDados(){
        super.imprimirDados(); // invocando o metodo de mesmo nome da
            superclasse
        System.out.println("Freq: " + this.frequencia);
        ...
    }
}
```

Herança: Criando instâncias

```
public class Principal{  
    public static void main(String[] args){  
        Telefone t = new Telefone(1,"ABC123","MesaTel",0.5, new  
            Dimensao(10,10,5));  
  
        SemFio sf = new SemFio(2,"DEF456","LivreTel",0.7,new Dimensao  
            (20,8,7), 11, 2400,100);  
  
        t.imprimirDados();  
        sf.imprimirDados();  
    }  
}
```


Membros públicos, privados e protegidos

- Os membros privados de uma classe só podem ser acessados pelos demais membros desta mesma classe
- Os membros públicos de uma classe podem ser acessados por qualquer outra classe
- O modificador de acesso *protected* apresenta uma restrição intermediária entre o *private* e o *public*
- Membros protegidos podem ser acessados pelos demais membros da classe e pelos membros das classes derivadas

Modificador de acesso protected: exemplo

```
public class Telefone{
    private String marca;
    protected String modelo;
    public float peso;
}

public class SemFio extends Telefone{
    private float frequencia;
    public void modificador(){
        this.frequencia = 900; // acesso ok
        this.modelo = "ABC"; // acesso ok
        this.peso = 0.5; // acesso ok
        this.marca = "GrandTel"; // erro! Nao permitido
    }
}
```

Modificador de acesso protected: exemplo

```
public class Principal{
    public static void main(String[] args){
        Telefone t = new Telefone();
        SemFio sf = new SemFio();

        // invocando um membro public
        t.peso = 0.6; // acesso ok
        // invocando um membro protected
        t.modelo = "DEF"; // erro!
        // invocando um membro private
        t.marca = "GT"; // erro!
    }
}
```

Herança múltipla

- No desenvolvimento de softwares complexos poderemos nos deparar com situações onde uma nova classe possui características semelhantes com duas ou mais classes existente
- A linguagem C++ possui o conceito de herança múltipla permitindo que uma classe seja derivada de varias classes base

Em Java uma classe só pode derivar de uma classe. O conceito de herança múltipla pode ser obtido em Java fazendo uso de Interfaces

Exercícios

1) Para um aplicativo que trabalha com desenhos

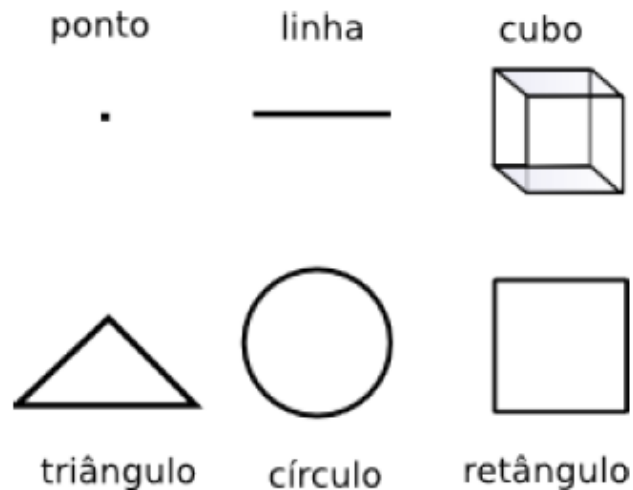


Diagrama de classes UML

Crie classes para representar as formas ao lado. Deve-se indicar as associações entre as classes (agregação, composição e herança)

- Um ponto é representado no plano cartesiano pelas coordenadas X e Y
- Deve possuir um método para calcular a área das formas que a possuem

Exercícios

2) Desenvolva em Java classes para um aplicativo bancário. Deve conter as classes de conta bancária, poupança, corrente, pessoa, cliente, bancário e gerente. Identifique os atributos e métodos essenciais para o funcionamento do aplicativo. Na classes principal devem ser instanciados objetos e realizadas as operações convencionais de uma conta poupança, conta corrente e atribuições dos funcionários na manipulação das contas dos clientes do banco. Exemplos: Gerente atribui limites de crédito ao cliente e funcionário realiza operações de empréstimos. Lembrando que as operações básicas das contas são impressões de saldos, extratos, compensação de cheques, depósitos e transferências.

Referências

Notas de aula do Prof. Emerson Ribeiro de Mello