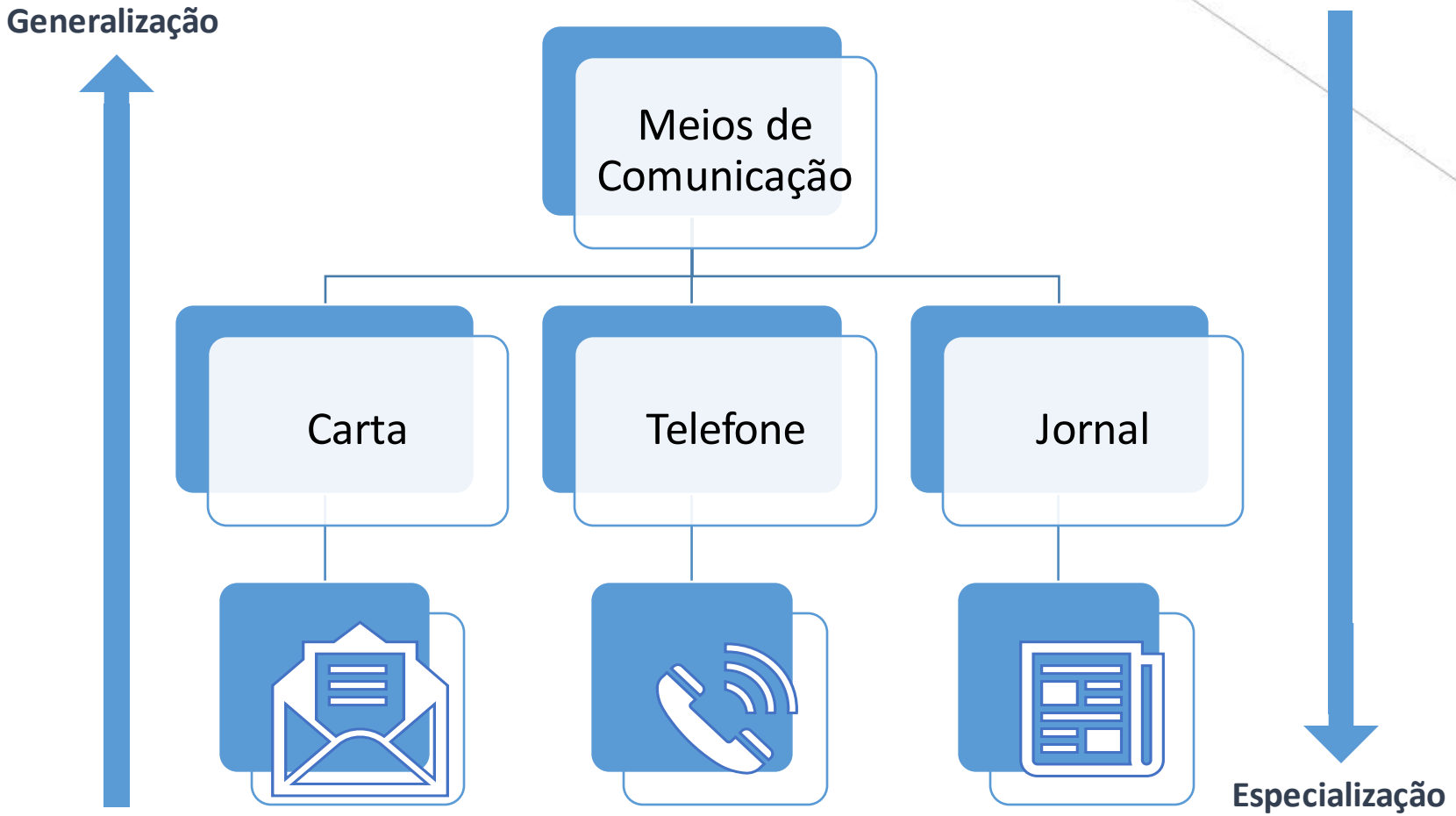
SENAI

CODIFICAÇÃO BACK-END

Classe abstrata e Interface

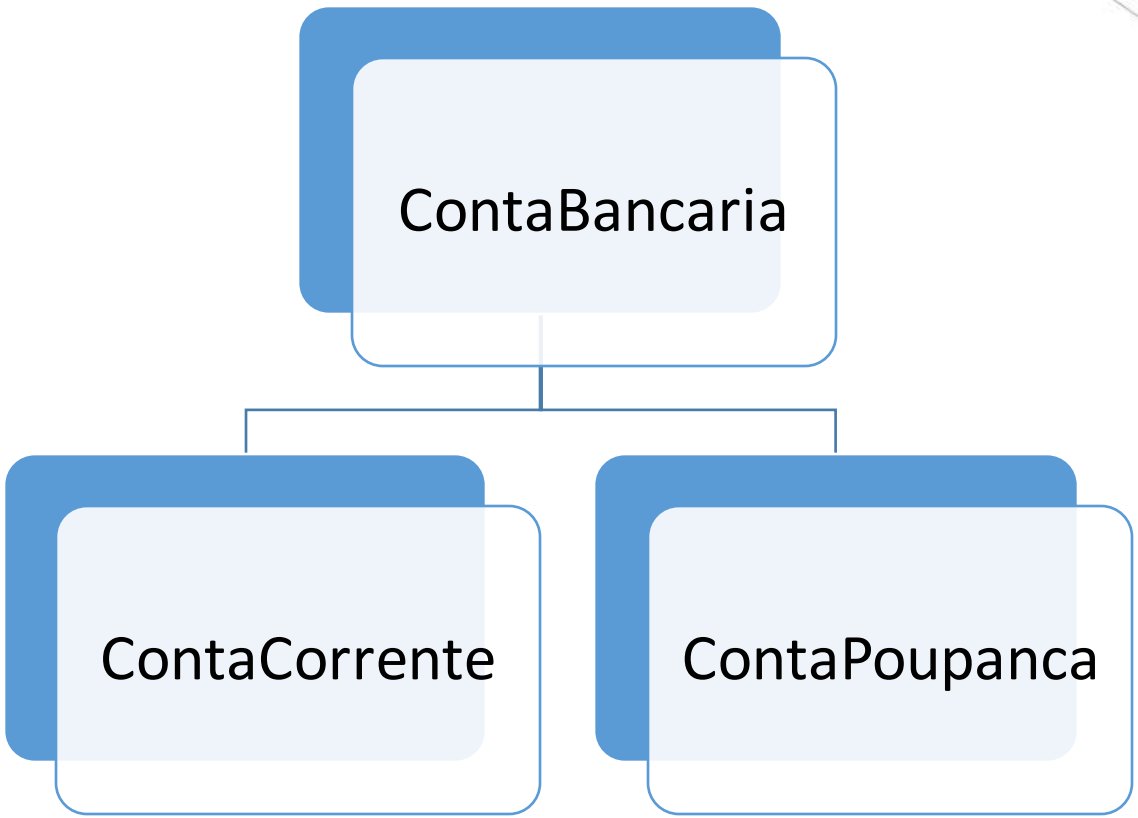
Classe abstrata

- Superclasse genérica com subclasses mais específicas
- Ajuda a lidar com a complexidade
- Modelo para outras classes



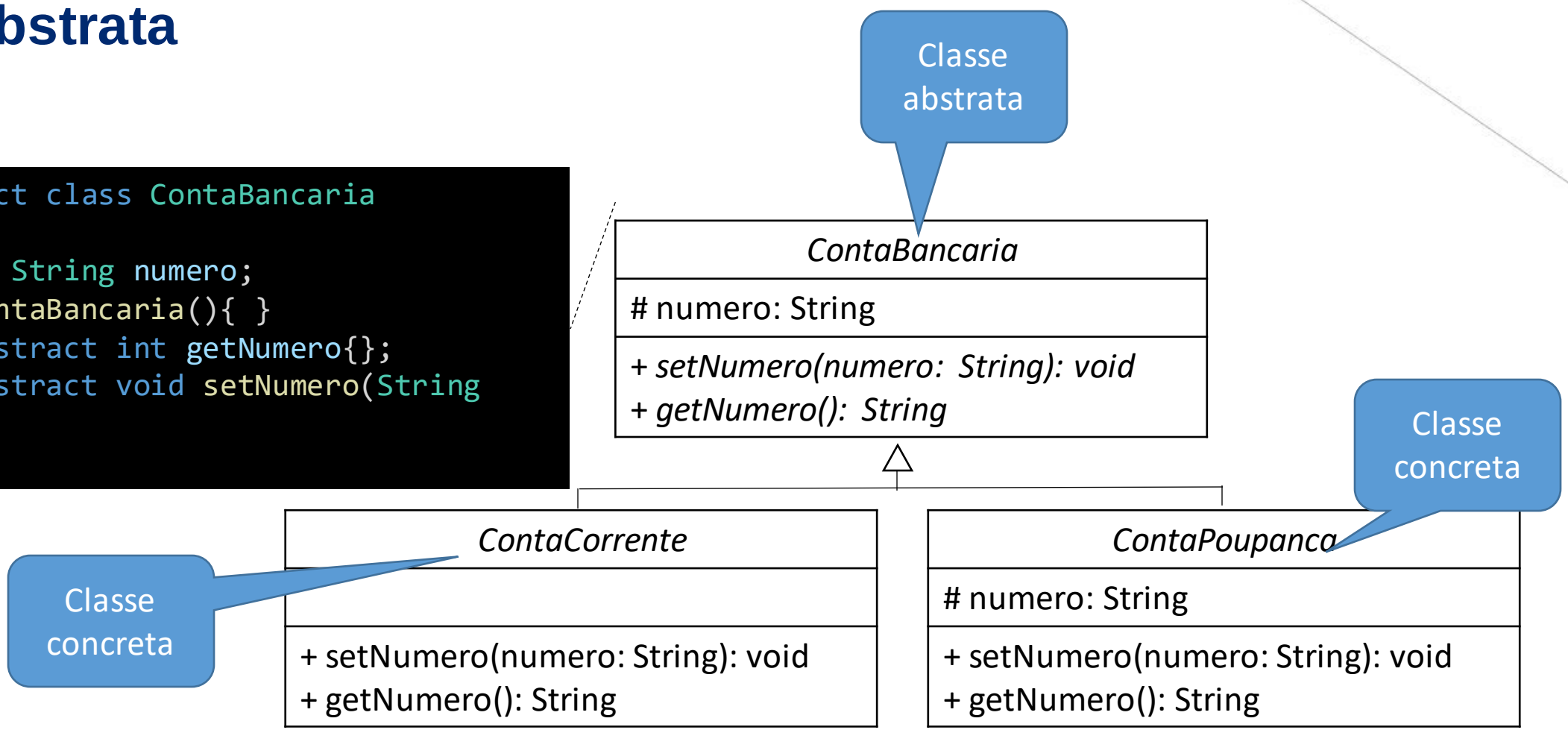
Classe abstrata

- Não tem instância
- Tem pelo menos um método abstrato
- Pode ter métodos concretos também
- Abstrato = itálico



Classe abstrata

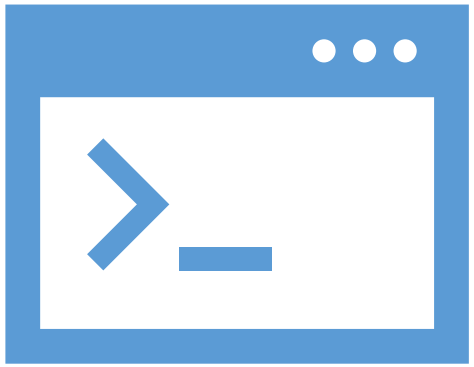
```
public abstract class ContaBancaria
{
    protected String numero;
    public ContaBancaria(){ }
    public abstract int getNumero{};
    public abstract void setNumero(String
numero);
}
```



Classe abstrata

- **Não tem instância**
- **Superclasses / modelos**
- **Genéricas**
- **Assinatura:** `public abstract class` **NomeDaClasse**

Palavra-chave que identifica a classe abstrata



Métodos abstratos

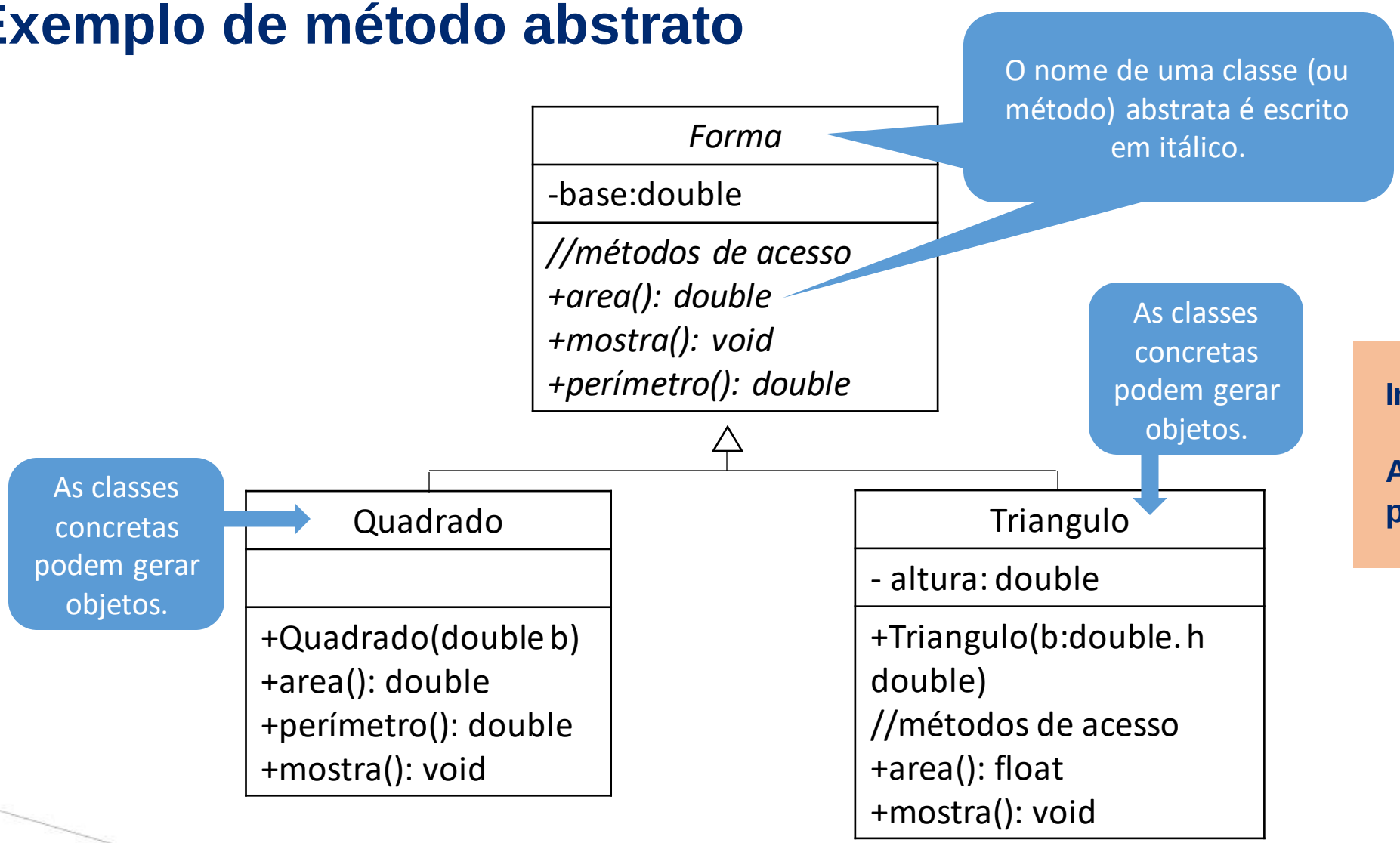
- **Método abstrato → implementado em classe concreta**
- **Método abstrato herdado → não precisa ser implementado**
- **Devem ser reescritos como concreto nas classes filhas (*override*)**
- **Método abstrato não tem corpo, então tem comportamento diferente quando reescrito nas filhas**

- **Assinatura:** `public abstract tipoRetorno NomeDoMetodo();`

Palavra-chave que identifica o método como abstrato

Implementação indefinida

Exemplo de método abstrato



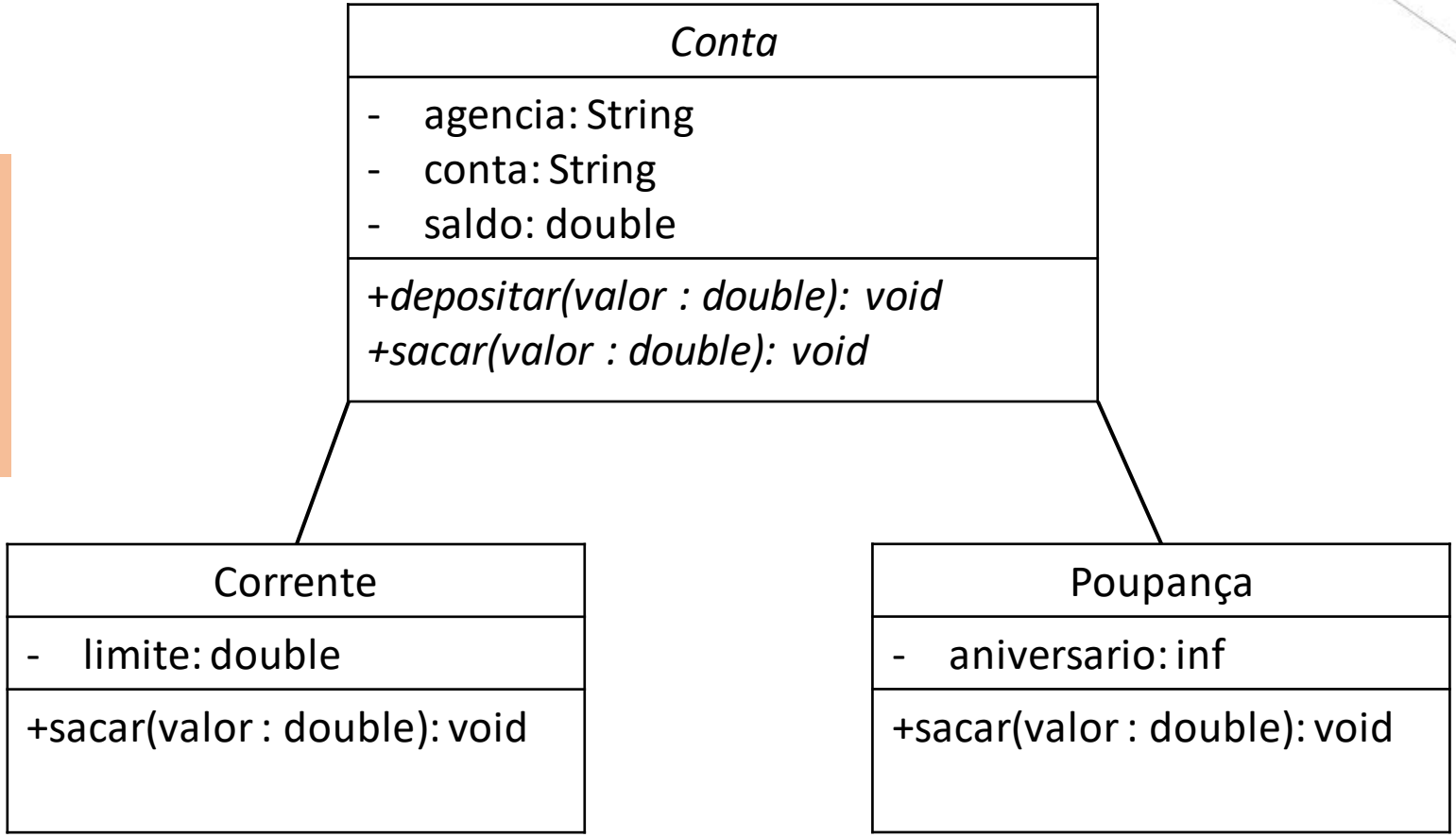
Importante!

As classes abstratas não podem ser instanciadas.

Classes abstratas

Atributos da Classe Corrente:

- **Limite (atributo exclusivo)**
- **Agencia, saldo e conta (atributos da super classe).**



Classes abstratas

```
namespace projeto
{
    1 reference
    public abstract class Conta
    {
        0 references
        public string agencia {get;set}
        0 references
        public string conta {get;set}
        6 references
        public double saldo {get;set}
        1 reference
        public abstract void Sacar(double valor)
    }
}
```

```
// Corrente.cs – classe filha

namespace projeto
{
    2 references
    public class Corrente : Conta
    {
        0 references
        double limite {get;set}
        1 reference
        public override void Sacar(double valor)
        {
            base.saldo = base.saldo - valor;
        }
    }
}
```

Interfaces ou contrato da aplicação

- Classe abstrata pura
- Mecanismo de reuso de código
- Funcionalidades especificadas, mas sem implementação
- Não tem atributo (exceto constante), nem definição de método e nem construtor (não instancia)
- Apenas constantes e métodos públicos, não precisa da palavra-chave abstract
- Não podemos usar private / protected
- Palavra-chave: interface → public interface nomeInterface {...}

Interface vs. Classe abstrata

Interface

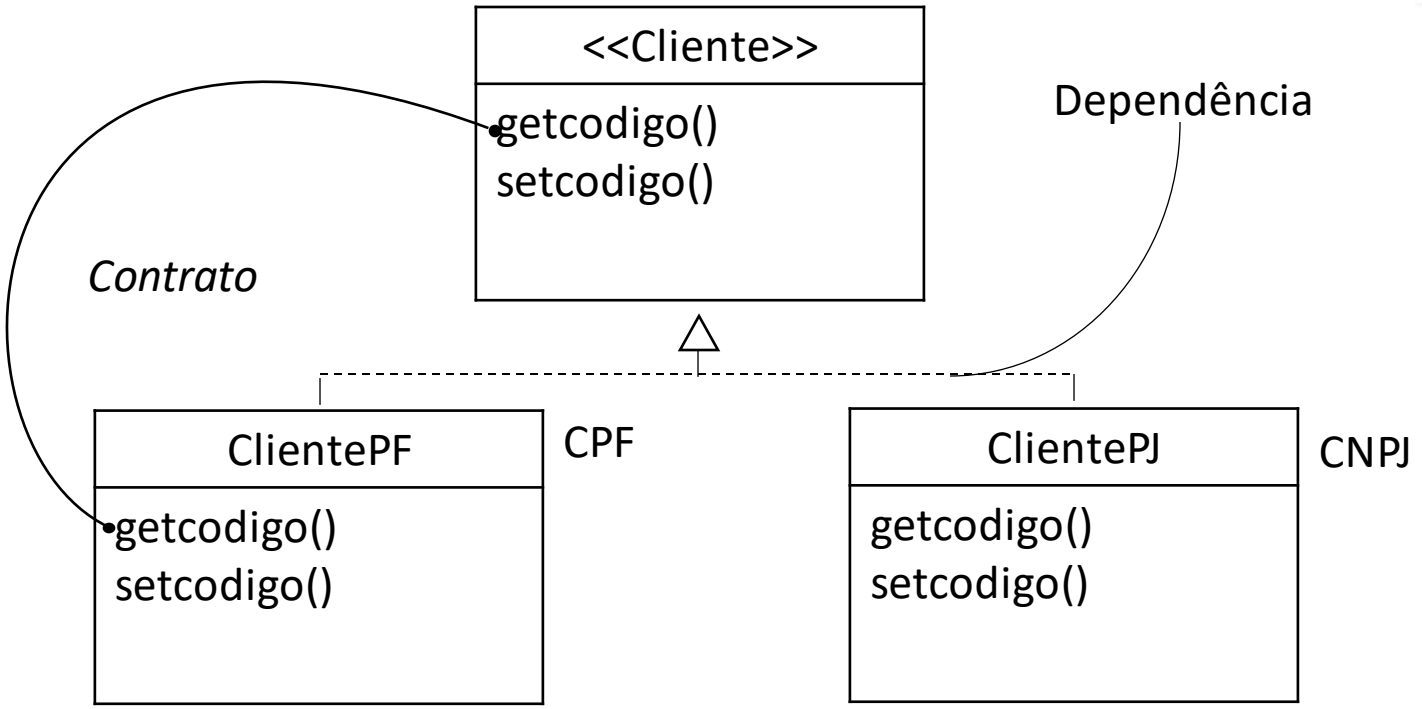
- Não tem corpo
- Implementada por outras classes
- Uma classe pode implementar várias interfaces (herança múltipla)
- Obriga a implementar todos os métodos

Classe abstrata

- Pode ter corpo
- Estendida por classes derivadas
- Uma classe só pode ser derivada de uma classe abstrata
- Não é obrigatório implementar todos os métodos

Implementação de Interfaces

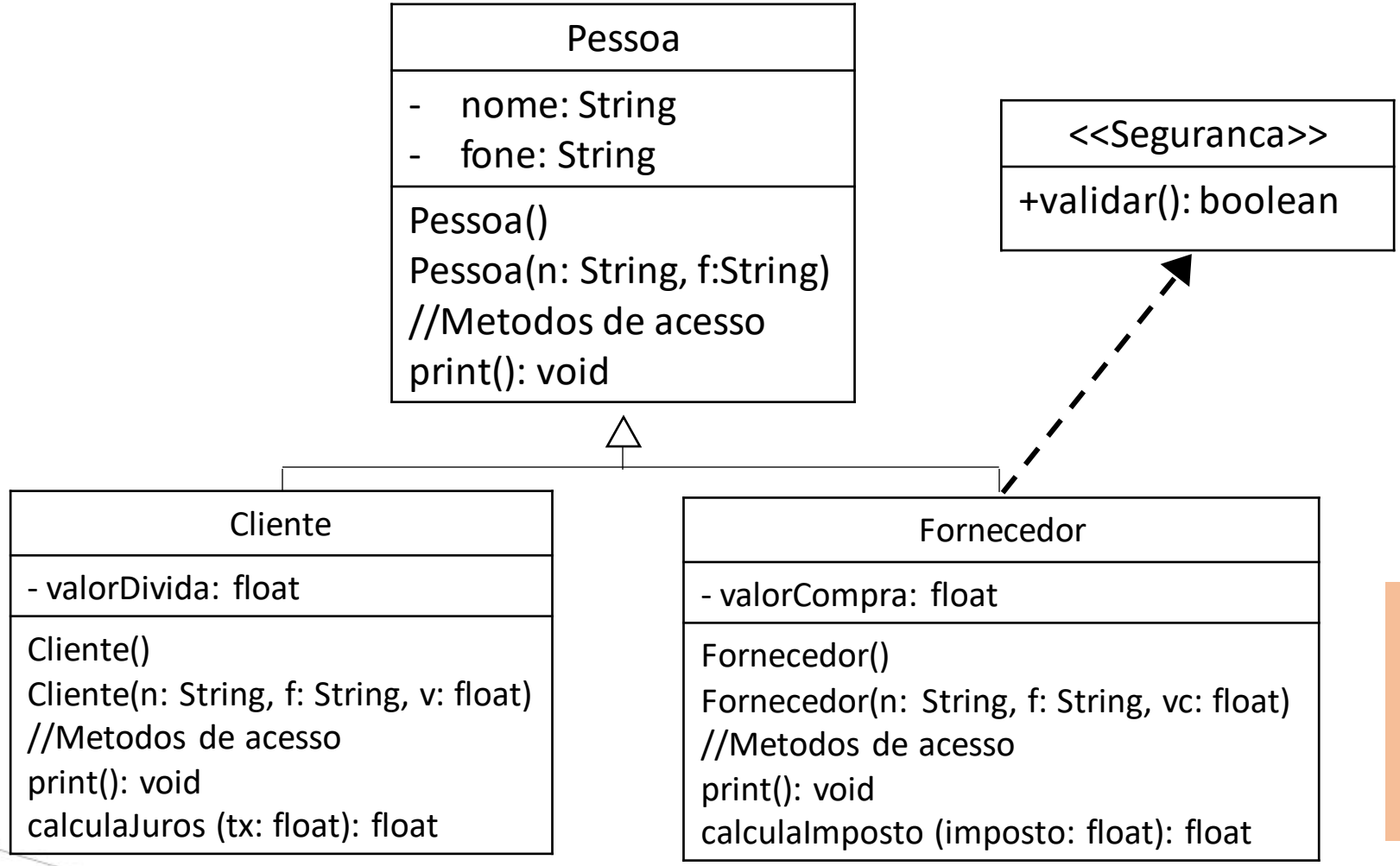
- Interface → contrato
- Classe deve implementar todos os métodos da interface
- Mecanismo para herança múltipla



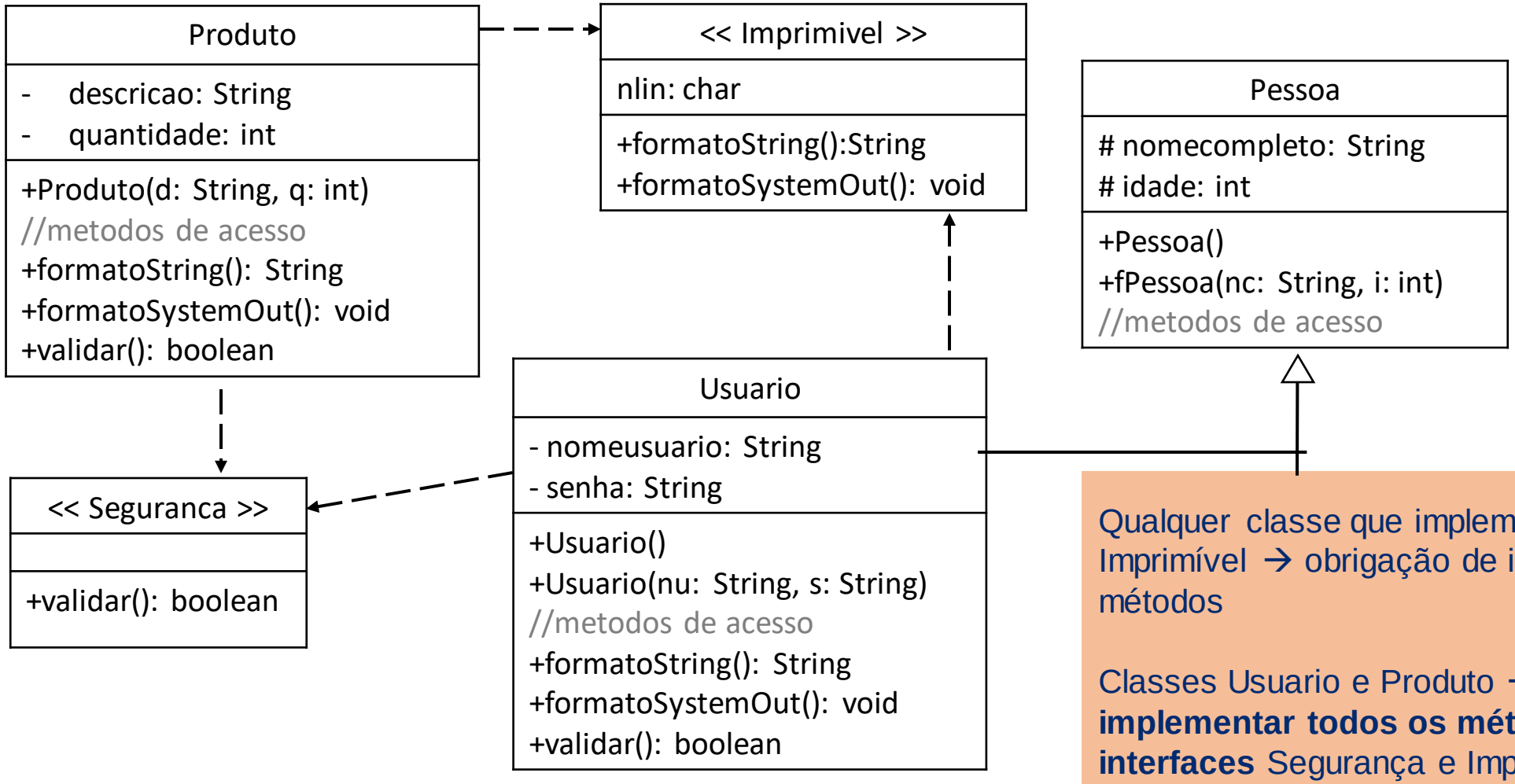
Vantagens da Interface

- Implementa similaridades sem forçar hierarquia
- Define métodos comuns a várias classes
- Revela apenas a interface → esconde a complexidade da arquitetura
- Facilidade para implementar herança múltipla
- Facilita entendimento → separa o que a classe “é” do que ela “faz”

Exemplos

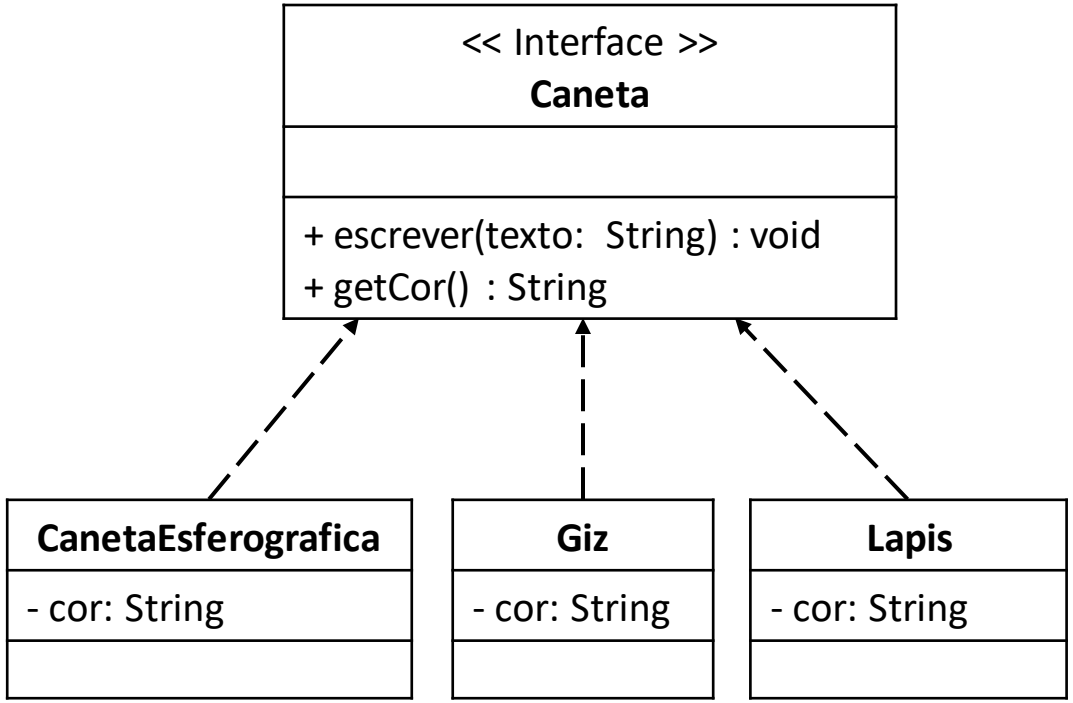


A classe fornecedor implementa a interface segurança→ é obrigada a implementar o método validar();



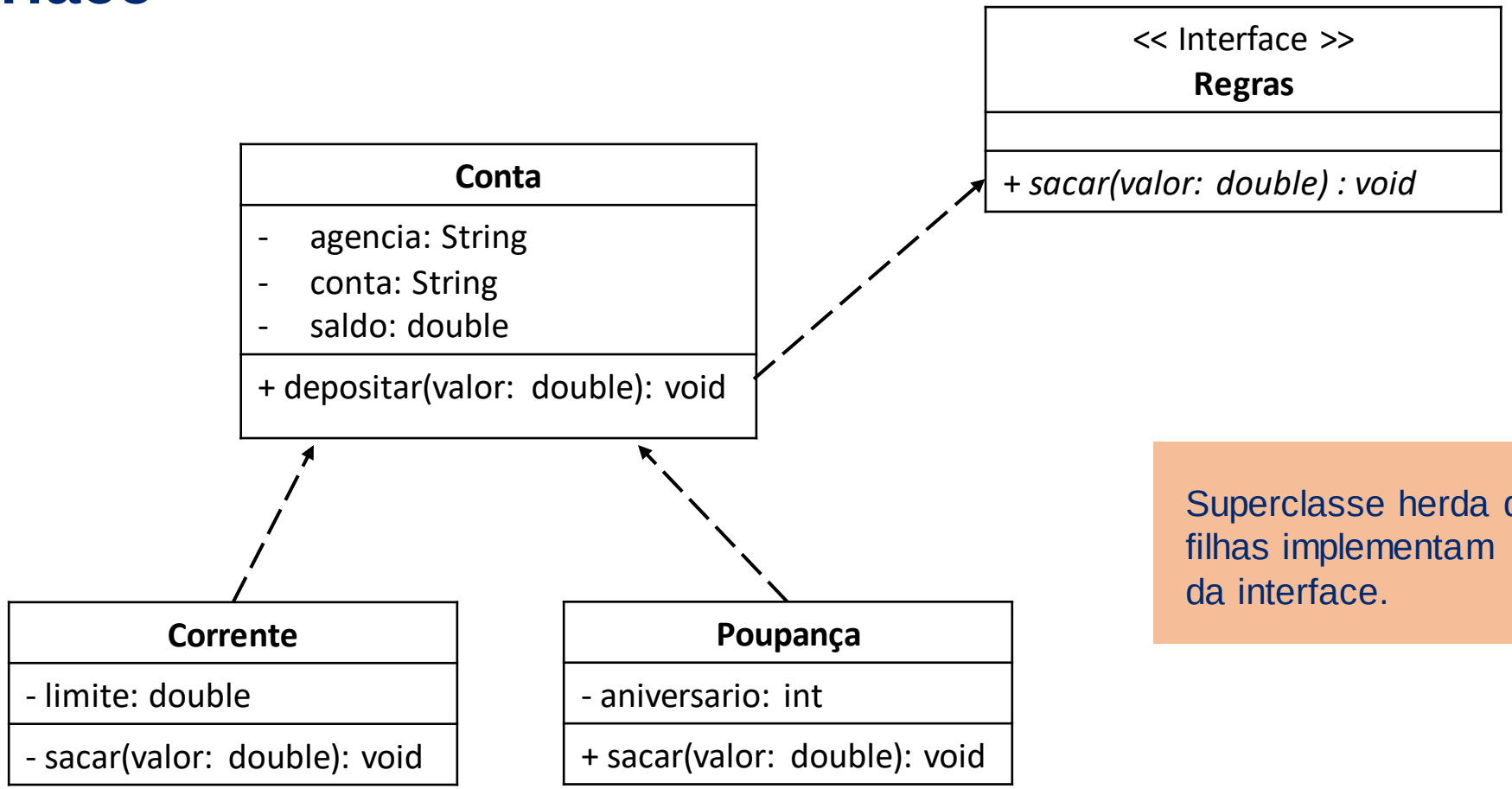
Qualquer classe que implementa a interface Imprimível → obrigação de implementar seus métodos

Classes Usuario e Produto → **obrigadas a implementar todos os métodos das interfaces** Segurança e Imprimível → padrão.



As classes **CanetaEsferografica**, **Giz** e **Lapis** implementam a interface **Caneta**, logo, implementam todos os métodos da interface.

Interface



Superclasse herda da interface, e as filhas implementam o método sacar da interface.

Interface

Classe abstrata Conta herda a interface Regras → herda o método Sacar();

```
// Conta.cs

namespace projeto
{

    public abstract class Conta : Regras
    {
        public string agencia {get; set;}
        public string conta {get; set;}
        public string saldo {get; set;}

        pubic void Depositar (double valor) {
            this.saldo = this.saldo + valor;
        }

        // método virtual pode ser substituído por qualquer classe
que o herde
        public virtual void Sacar(double valor)
        {
            base.saldo = base.saldo - valor;
        }
    }
}
```

Classe Corrente (filha de Conta) → implementa o método Sacar()

```
// Corrente.cs – classe filha

namespace projeto
{
    public class Corrente : Conta
    {
        double limite {get;set}

        public override void Sacar(double valor)
        {
            base.saldo = base.saldo - valor;
        }
    }
}
```

Interface: implementação

- **Criação da interface Regras**

```
// Regras.cs

namespace projeto
{
    public interface Regras
    {
        void Sacar(double valor);
    }
}
```

- **Classe Corrente**

```
// Program.cs

using projeto;

Corrente cc = new Corrente();
cc.Depositar(5000);
Console.WriteLine("O saldo é: "+ cc.saldo);
cc.Sacar(1000);
Console.WriteLine("O saldo é: "+ cc.saldo);
```

Interface vs. Classes abstratas

Classe abstrata

Template: modelo para subclasses.

Interfaces

Regra: contrato independente da hierarquia de classe.

Bons estudos!