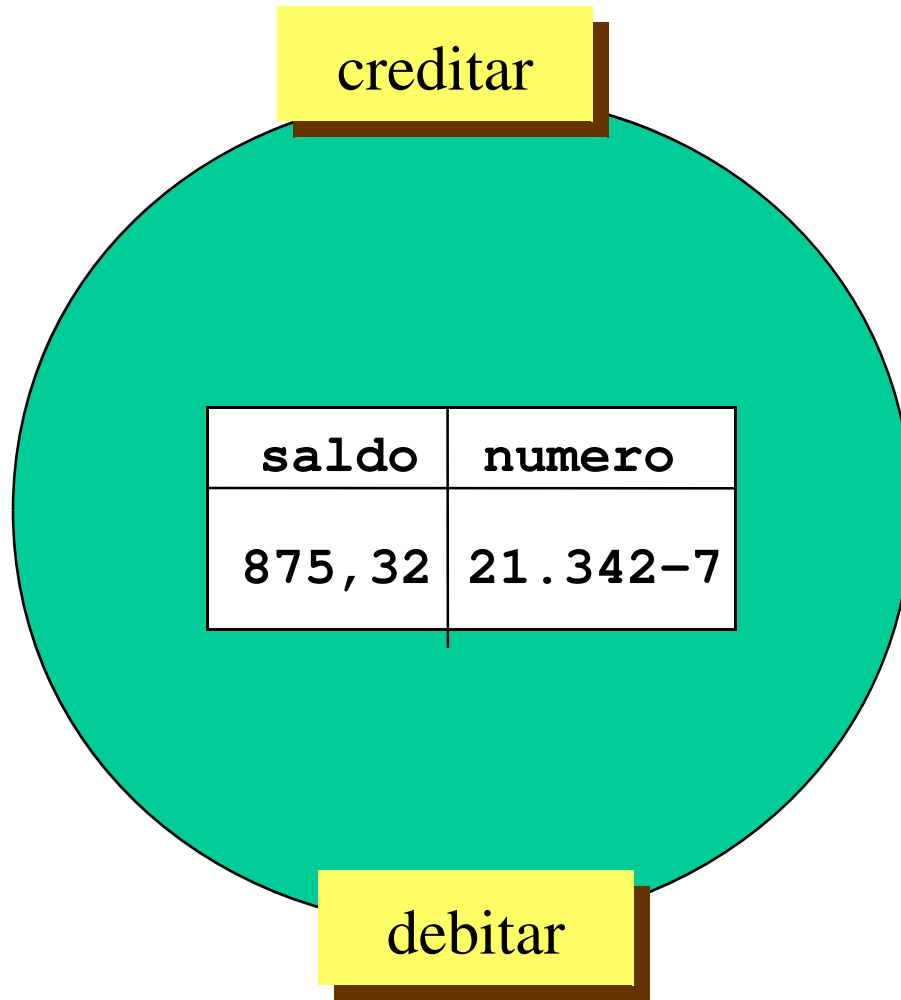
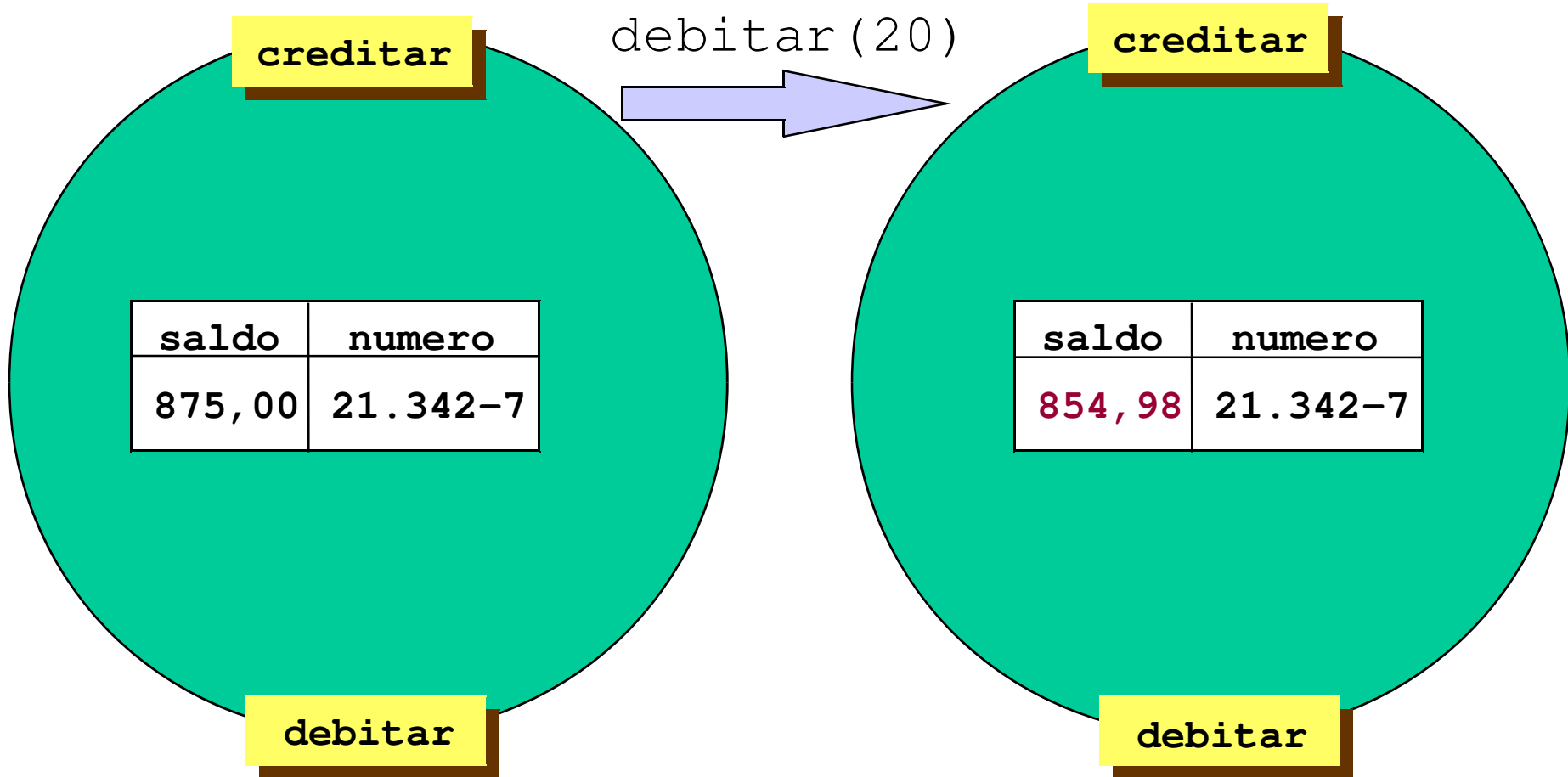


Classes Abstratas e Interfaces

Objeto Conta Imposto



Estados do Objeto Conta Imposto



Desconta 0,1% sobre o saque

Classe ContaImposto

```
public class ContaImposto extends Conta {  
    public static final double TAXA = 0.001; //0,1%  
  
    public ContaImposto (String n, Cliente c) {  
        super (n, c);  
    }  
  
    public void debitar(double valor){  
        double imposto = valor * TAXA;  
        super.debitar(valor + imposto)  
    }  
}
```

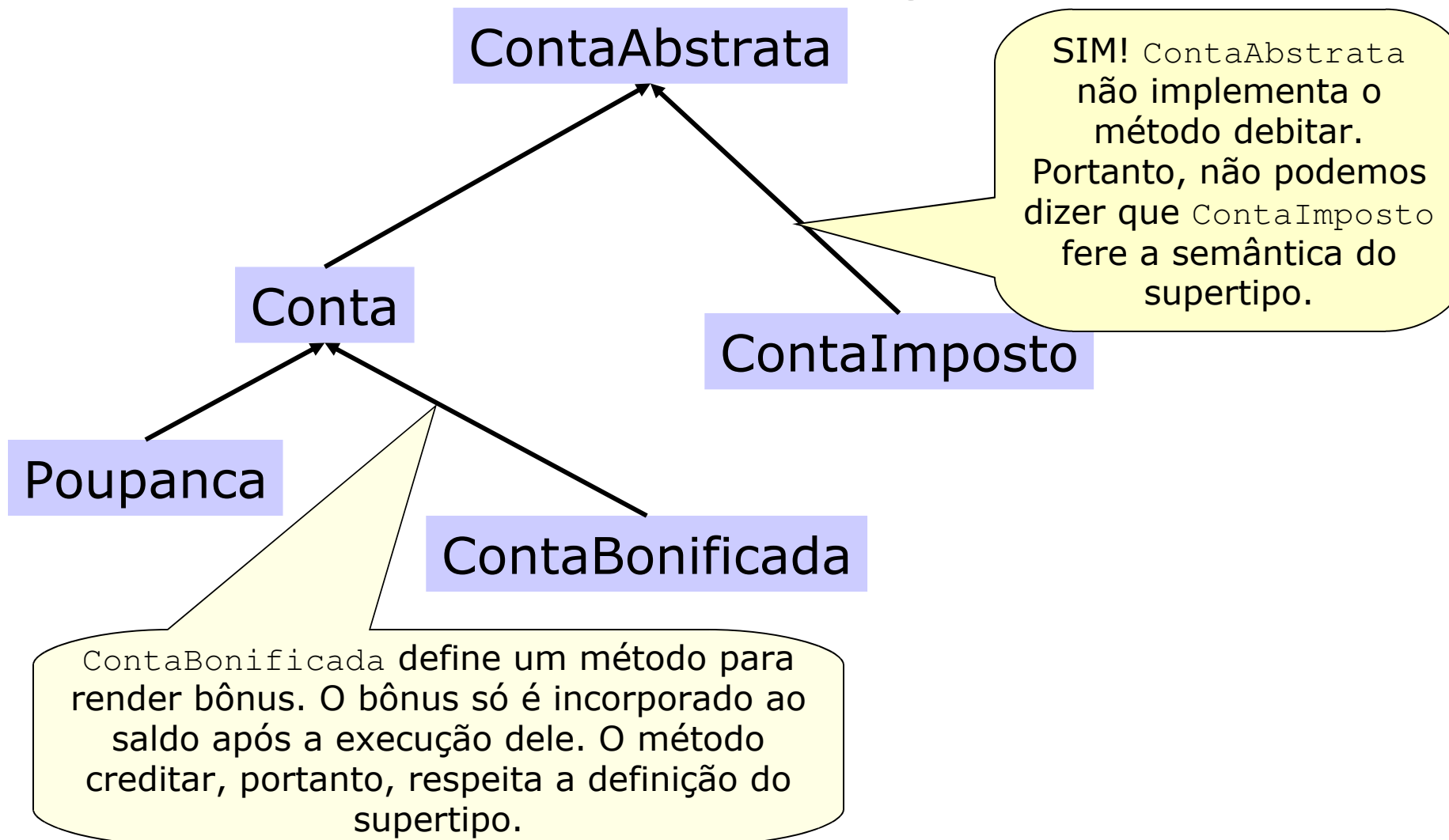
Subclasses e Comportamento

- Objetos da **subclasse** devem se comportar como os objetos da **superclasse**
- Redefinições de métodos devem preservar o comportamento (semântica) do método original
- Grande impacto sobre manutenção e evolução de software

Subclasses e Evolução de Software



Subclasses e Evolução de Software



Classes Abstratas

- Devem ser declaradas com a palavra-chave *abstract*
- Podem declarar métodos abstratos
 - Métodos sem implementação
 - Implementação fornecida na subclasse
- Podem declarar métodos concretos
 - Métodos com implementação

Definição de Classes Abstratas

- Uma classe abstrata é declarada com o modificador **abstract**
- Um método é definido abstrato também usando o modificador **abstract**

```
abstract  
public abstract class Nome_da_Classe {  
    atributo1;  
    atributo2;  
    ...  
  
    public void metodo1() {  
        // código do método 1  
    }  
    ...  
    public abstract void metodoAbstrato();  
}
```

Definindo uma Conta abstrata

```
public abstract class ContaAbstrata {  
    private String numero;  
    private double saldo;  
    private Cliente cliente;  
  
    public ContaAbstrata(String num, Cliente c) {  
        numero = num;  
        cliente = c;  
    }  
    public ContaAbstrata(String num, double s, Cliente c)  
{  
        numero = num;  
        saldo = s;  
        cliente = c;  
    }  
}
```

Definindo uma Conta abstrata (cont...)

```
public Cliente getCliente(){
    return cliente;
}
public String getNumero() {
    return numero;
}
public double getSaldo() {
    return saldo;
}
public void setCliente(Cliente cliente) {
    this.cliente = cliente;
}
public void setNumero(String num) {
    this.numero = num;
}
public void setSaldo(double valor) {
    saldo = valor;
}
```

Definindo uma Conta abstrata (cont...)

```
public void creditar(double valor){
    saldo = saldo + valor;
}
public abstract void debitar(double valor) ;

public void transferir(ContaAbstrata c, double v){

    this.debitar(v);
    c.creditar(v);
}
}
```

Modificando a classe Conta

```
public class Conta extends ContaAbstrata {  
  
    public Conta(String n, Cliente c) {  
        super (n,c);  
    }  
  
    public void debitar(double valor){  
        double saldo = getSaldo();  
        if(valor <= saldo){  
            setSaldo(saldo - valor);  
        } else {  
            System.out.println("Saldo insuficiente");  
        }  
    }  
}
```

Modificando a classe ContaImposto

```
public class ContaImposto extends ContaAbstrata {  
    public static final double TAXA = 0.001;  
  
    public ContaImposto(String n, Cliente c) {  
        super (n, c);  
    }  
  
    public void debitar(double valor) {  
        double imposto = valor * TAXA;  
        double saldo = this.getSaldo();  
        if (valor + imposto <= saldo) {  
            setSaldo(saldo - (valor + imposto));  
        } else {  
            System.out.println("Saldo Insuficiente");  
        }  
    }  
}
```

Classes abstratas: propriedades

- Uma classe abstrata não pode ser instanciada
- Mesmo sem poder ser instanciada, pode definir construtores para permitir reuso
- Qualquer classe com um ou mais métodos abstratos é automaticamente uma classe abstrata
- Se uma classe herdar de uma classe abstrata, deve redefinir todos os métodos abstratos. Caso contrário, deve ser declarada como abstrata.

Classes abstratas: propriedades

- Métodos **private**, **static** e **final** não podem ser abstratos.
 - Métodos declarados com estes modificadores não podem ser redefinidos
- Uma classe **final** não pode conter métodos abstratos
 - Métodos definidos em classes declaradas com este modificador não podem ser redefinidos
- Uma classe pode ser declarada como **abstract**, mesmo se não tiver métodos abstratos
 - Permite somente o reuso, mas não permite instâncias dessa classe

Classes abstratas: polimorfismo

```
public static void main(String args[]) {  
    ...  
    ContaAbstrata ca1, ca2;  
    ca1 = new ContaBonificada("21.342-7");  
    ca2 = new ContaImposto("21.987-8");  
  
    ca1.debitar(500);  
    ca2.debitar(500);  
  
    System.out.println(ca1.getSaldo());  
    System.out.println(ca2.getSaldo());  
    ...  
}
```

Classes abstratas: utilização

- Ajudam a estruturar sistemas definindo hierarquias consistentes de classes.
- Simplificam o reuso de código
- Definem "contratos" a serem realizados por subclasses
- Tornam o polimorfismo mais claro

Interfaces

Auditor de Banco de Investimentos

```
class AuditorBI {  
    final static double MINIMO = 500.00;  
    private String nome;  
    /* ... */  
  
    public boolean investigaBanco(BancoInvest b) {  
        double sm;  
        sm = b.saldoTotal() / b.numContas() ;  
        return (sm > MINIMO) ;  
    }  
}
```

Auditor de Banco de Seguros

```
class AuditorBS {  
    final static double MINIMO = 500.00;  
    private String nome;  
    /* ... */  
  
    public boolean investigaBanco(BancoSeguros b)  
    {  
        double sm;  
        sm = b.saldoTotal() / b.numContas() ;  
        return (sm > MINIMO) ;  
    }  
}
```

Problema

- Duplicação desnecessária de código
- O mesmo auditor deveria ser capaz de investigar qualquer tipo de banco que possua operações para calcular
 - o número de contas no banco, e
 - o saldo total de todas as contas
- Casos em que as classes envolvidas não estão relacionadas através de uma hierarquia de herança

Interfaces

- Caso especial de classes abstratas
- Definem tipo de forma abstrata, apenas indicando a **assinatura dos métodos** suportados pelos objetos do tipo
 - Os métodos declarados em uma interface não têm implementação
- Os métodos são implementados pelos subtipos (classes que implementam interfaces)
- Não têm construtores. Não se pode criar objetos já que métodos não estão implementados

Uma interface define um “contrato” a ser seguido

Definição de Interfaces

```
interface Nome_Interface {
```

```
    /*... assinaturas de novos métodos... */
```

```
}
```

```
interface QualquerBanco {
```

```
    double saldoTotal();
```

```
    int numContas();
```

```
}
```


Utilização de Interfaces

```
class Nome_classe implements Nome_Interface
{
    /*... Implementação dos métodos
        declarados na interface
        ...
    */
}
```

Utilização de Interfaces

```
class BancoInvest implements QualquerBanco {  
    ...  
    double saldoTotal() {  
        /* código específico para BancoInvest */  
    }  
    int numContas() {  
        /* código específico para BancoInvest */  
    }  
    ...  
}
```

Utilização de Interfaces

```
class BancoSeguros implements QualquerBanco {  
    ...  
    double saldoTotal() {  
        /* código específico para BancoSeguros */  
    }  
    int numContas() {  
        /* código específico para BancoSeguros */  
    }  
    ...  
}
```

Auditor Genérico

```
class Auditor {  
    final static double MINIMO = 500.00;  
    private String nome;  
    /* ... */  
  
    boolean investigaBanco(QualquerBanco b) {  
        double sm;  
        sm = b.saldoTotal() / b.numContas();  
        return (sm > MINIMO);  
    }  
}
```

Usando do Auditor Genérico

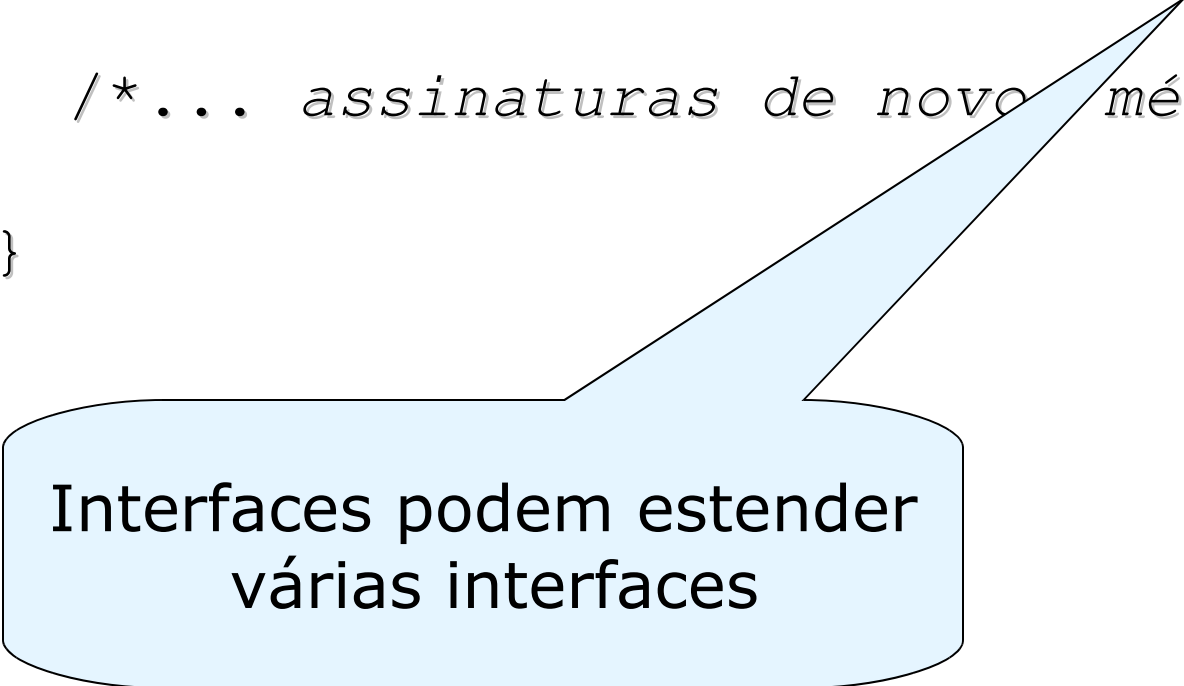
```
QualquerBanco bi = new BancoInvest();  
QualquerBanco bs = new BancoSeguros();  
Auditor a = new Auditor();  
  
/* ... */  
boolean res1 = a.investigaBanco(bi);  
boolean res2 = a.investigaBanco(bs);  
/* ... */
```

Tipos e Subtipos

- Classes e Interfaces são **tipos**
- Os termos **subtipo** e **supertipo** também são usados
 - supertipo : interface
 - subtipo: classe que implementa a interface ou interface que estende outra interface
- Interfaces utilizam o conceito de herança múltipla
 - Herança múltipla de assinatura

Herança múltipla de assinatura

```
interface I extends I1, I2, ..., In {  
    /*... assinaturas de novo métodos... */  
}
```



Interfaces podem estender
várias interfaces

Múltiplos supertipos

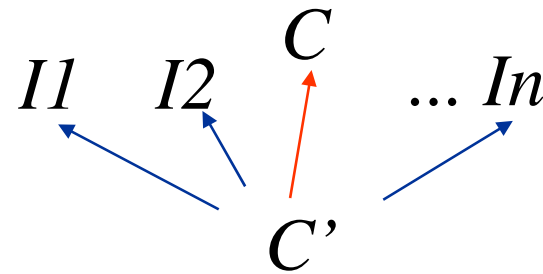
- Classe que implementa uma interface deve definir os métodos da interface
 - classes concretas têm que implementar todos os métodos da interface
 - caso algum método da interface não seja implementado, a classe deve ser declarada `abstract`
- Apesar de não suportar herança múltipla de classes, uma classe pode implementar mais de uma interface (pode ter vários **supertipos**)

```
class XXX implements interface1, interface2 {  
  
    ...  
}
```


Definição de classes: forma geral

```
class C'  
  extends C  
  implements I1, I2, ..., In {  
    /* ... */  
  }
```

A classe C' é **subtipo** de $C, I1, I2, \dots, In$



Interfaces e métodos

- Todos os métodos de uma interface são implicitamente **abstratos e públicos**
- métodos de interfaces não podem ser :
 - static
 - final
 - private
 - protected

Interfaces e atributos

- Interfaces não podem conter atributos
- A única exceção são os atributos **static final** (constantes).
- Interfaces são excelentes repositórios para constantes

```
public interface CoresBasicas {  
    public static final int AZUL = 0;  
    public static final int VERMELHO = 1;  
    public static final int VERDE = 2;  
    public static final int PRETO = 3;  
    public static final int AMARELO = 4;  
}
```

```
janela.alterarCor(CoresBasicas.AZUL);
```

Interfaces e Reusabilidade

- Evita duplicação de código através da definição de um tipo genérico, tendo como subtipos várias classes não relacionadas
- Uma interface agrupa objetos de várias classes definidas independentemente, sem compartilhar código via herança, tendo implementações totalmente diferentes

Subtipos e instanceof

```
class Circulo extends Forma implements
    Selecao, Rotacao, Movimentacao {
    ...
}
```

```
...
Circulo c = new Circulo();
res1 = c instanceof Circulo;
res2 = c instanceof Selecao;
res3 = c instanceof Rotacao;
res4 = c instanceof Movimentacao;
...
```

res1, res2, res3 e res4 são **true**!

Classes abstratas X Interfaces

Classes (abstratas)

- Agrupa objetos com implementações compartilhadas
- Define novas classes através de herança (simples) de **código**
- Só uma classe pode ser supertipo de outra **classe**

Interfaces

- Agrupa objetos com implementações diferentes
- Define novas interfaces através de herança (múltipla) de **assinaturas**
- Várias interfaces podem ser supertipo do mesmo **tipo**