# Linguagem JAVA

## Interfaces

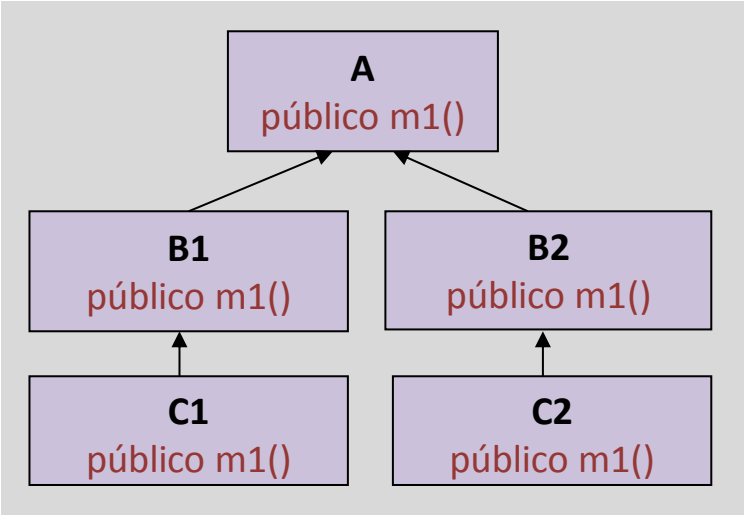
(Livro *Big Java, Late Objects* – Capítulo 9)

# Sumário Geral

# Motivação para usar Interfaces



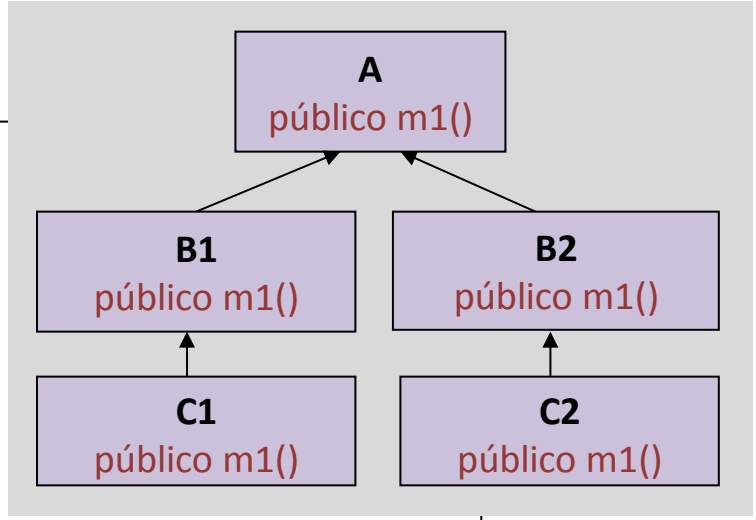
- **Aplicação**
  - Processa instâncias da hierarquia de classes




  - Guarda todas as instâncias num contentor

```
import java.util.ArrayList;
public class DemoInterfaces {
    public static void main(String[] args) {
        ArrayList contentor = new ArrayList();
        contentor.add(new B1());
        contentor.add(new B2());
        contentor.add(new C1());
        contentor.add(new C2());
    }
}
```

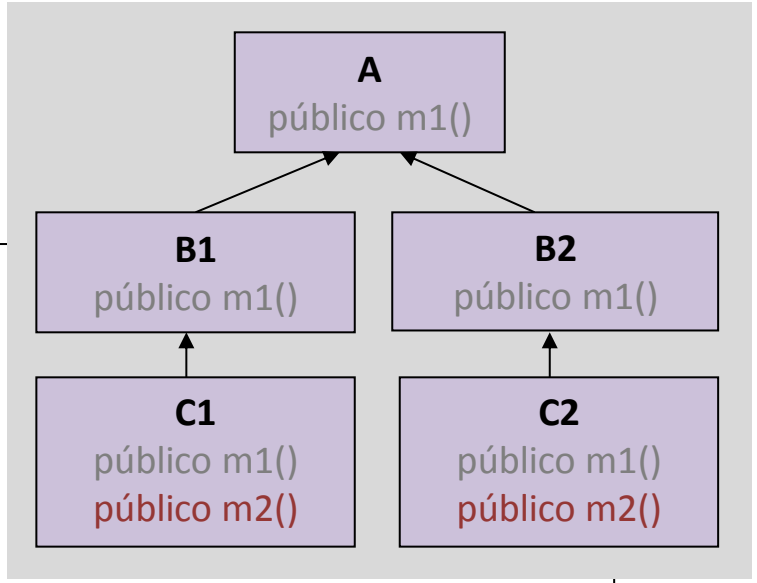
- **Problema 1**
  - Executar m1() das instâncias do contentor
    - Através de varrimento completo do contentor
- **Solução**




```
import java.util.ArrayList;

public class DemoInterfaces {

    public static void main(String[] args) {

        ArrayList contentor = new ArrayList();

        contentor.add(new B1());
        contentor.add(new B2());
        contentor.add(new C1());
        contentor.add(new C2());

        System.out.println("Executar m1() das instâncias do contentor");
        for (Object obj : contentor) {
            System.out.printf("%s m1 = %.2f %n", obj, ((A) obj).m1());
        }
    }
}
```

Tipo **Compatível**
com todos tipos da hierarquia
e ... que possua m1()

- **Problema 2**
  - Executar m2() das instâncias Cx (x:1,2) do contentor
    - Através de varrimento completo do contentor
- **Solução**




```
import java.util.ArrayList;
public class DemoInterfaces {
    public static void main(String[] args) {

        ArrayList contentor = new ArrayList();
        contentor.add(new B1());
        contentor.add(new B2());
        contentor.add(new C1());
        contentor.add(new C2());

        System.out.println("Executar m1() das instâncias do contentor");
        for (Object obj : contentor) {
            System.out.printf("%s m1 = %.2f %n", obj, ((A) obj).m1());
        }

        System.out.println("Executar m2() das instâncias Cx do contentor");
        for (Object obj : contentor) {
            if(obj instanceof     ?     )
                System.out.printf("%s m2= %.2f %n", obj, ((     ?     ) obj).m2());
        }
    }
}
```

Tipo **Compatível** com tipos Cx
e ... que possua m2()

- **Solução**

```
System.out.println("Executar m2() das instâncias Cx do contentor");
for (Object obj : contentor) {
    if(obj instanceof     ?     )
        System.out.printf("%s m2= %.2f %n", obj, ((    ?    ) obj).m2());
}
```

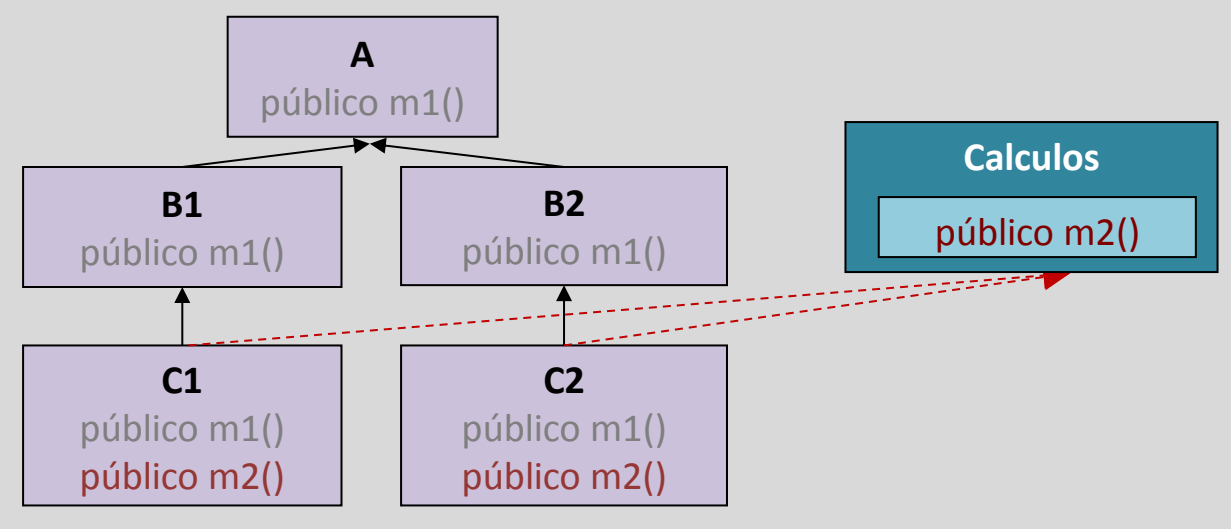
- Preciso tipo de dados
  - Tipo de todas as instâncias Cx // uma instância pode ser de vários tipos
    // ex: C1 é também do tipo B1 e A
  - Declare método m2 // garante m2() em todas as classes Cx
- Tipo A não serve
  - Tipo comum ....mas não declara método m2 // método não-comum a todos obj. da hier.
- Preciso novo tipo de dados
  - Exemplo: Calculos

```
System.out.println("Executar m2() das instâncias Cx do contentor");
for (Object obj : contentor) {
    if(obj instanceof Calculos)
        System.out.printf("%s m2= %.2f %n", obj, ((Calculos) obj).m2());
}
```

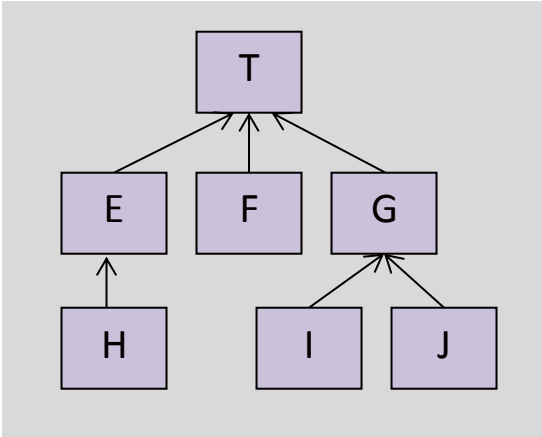
# Motivação para usar Interfaces



- **Solução**




- Em Java, novo tipo Calculos
  - Não pode ser definido por classe
    - Permitida apenas herança de classes simples
    - Subclasse só tem uma superclasse direta
  - Tem de ser definido por Interface Java
    - Alternativa à herança múltipla



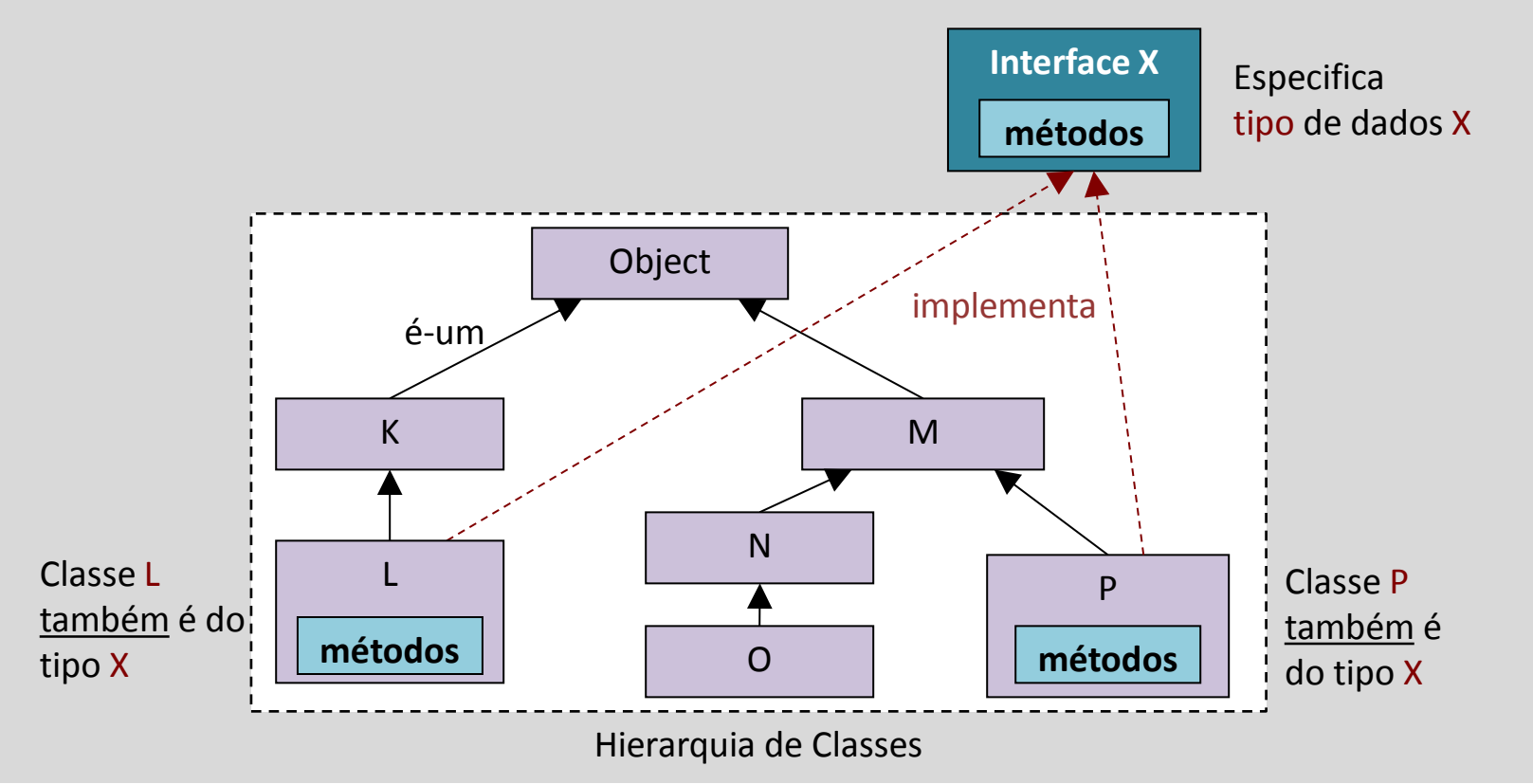

**Herança Simples**

# Motivação para usar Interfaces



- **Motivação Geral para usar Interfaces**
  - Necessidade de garantir que múltiplas classes, não relacionadas hierarquicamente (mesmo ramo), sejam também de um mesmo tipo de dados ...

    ... de modo a assegurar que possuam métodos comuns (interface comum).

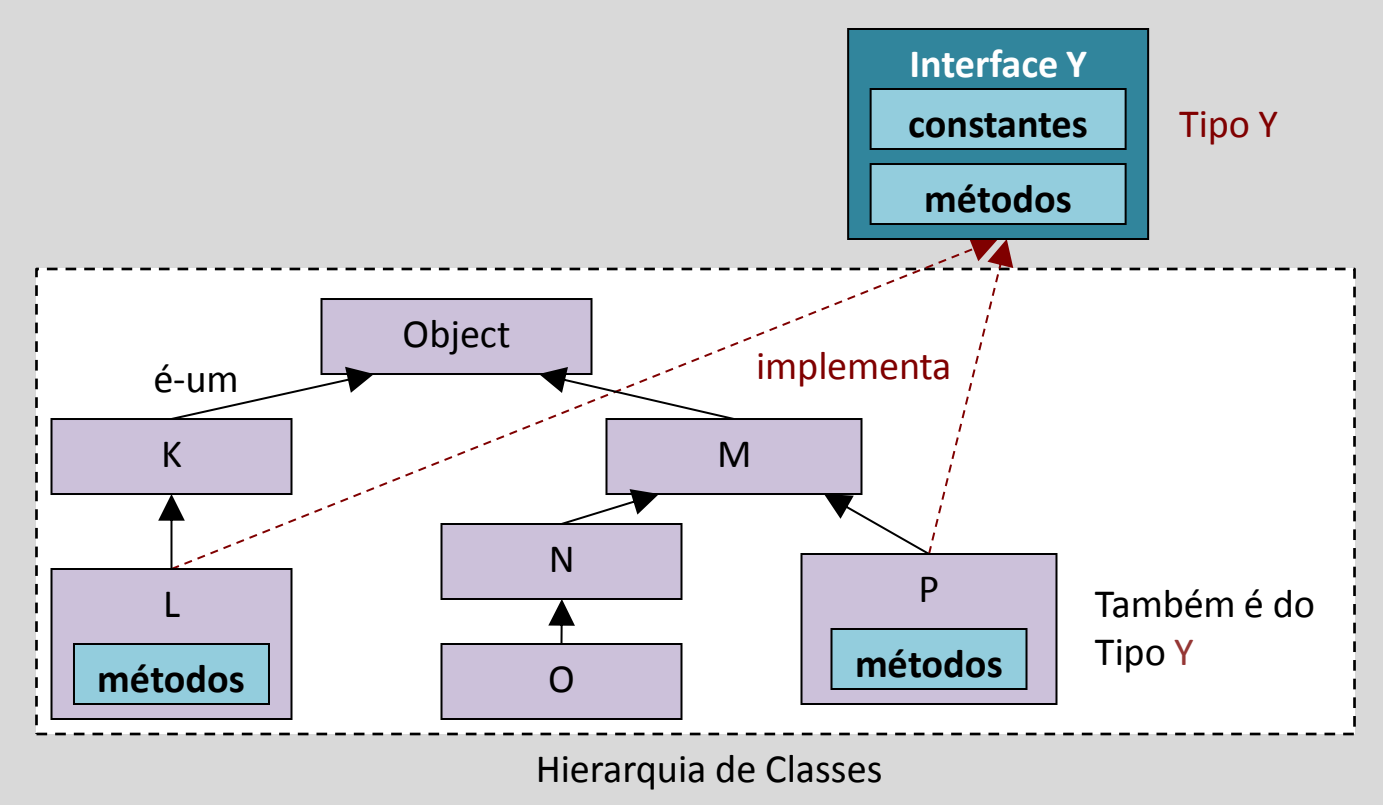



- Alternativa Java ... à Herança de Classes múltipla

# Definição de Interface

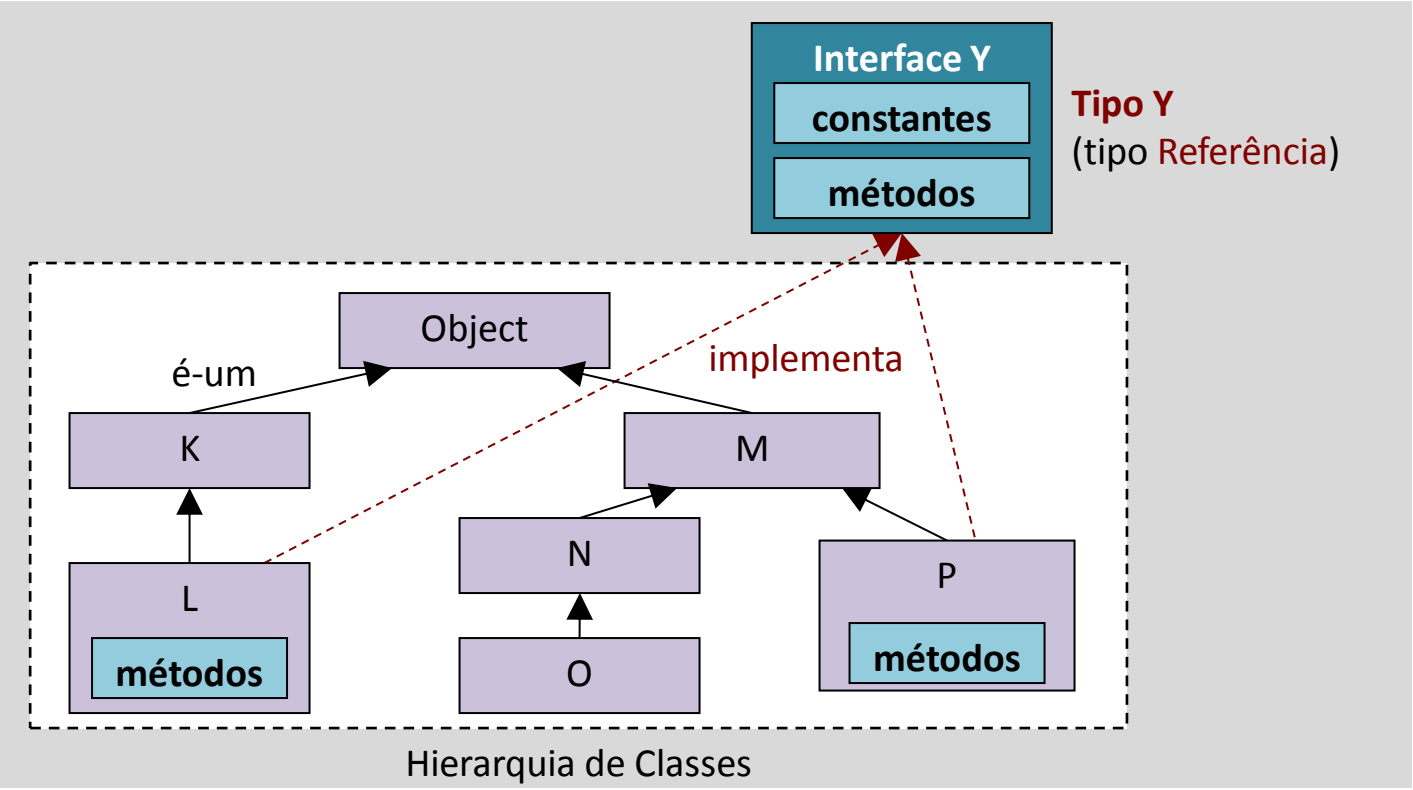
- **Interface é**
  - Uma especificação de um tipo de dados abstrato (sem implementação dos métodos) que ...

    ... qualquer classe pode implementar (classe em qualquer ponto da Hierarquia de Classes Java)
- **Especificação pode definir**
  - Conjunto de métodos de instância abstratos (só cabeçalhos) // opcional
  - Conjunto de constantes // opcional



# Interface define Tipo de Dados

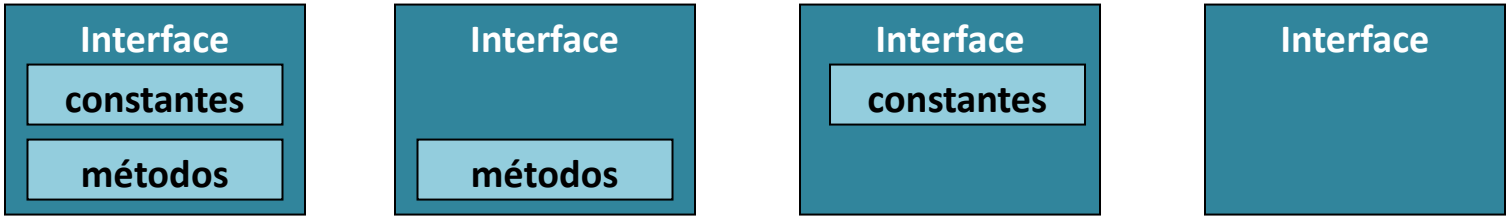
- **Interface Especifica**
  - Tipo de dados referência
- **Tipos de Dados Java**
  - Primitivos
  - Referência
    - Classes
    - Interfaces



# Estruturas de Interfaces

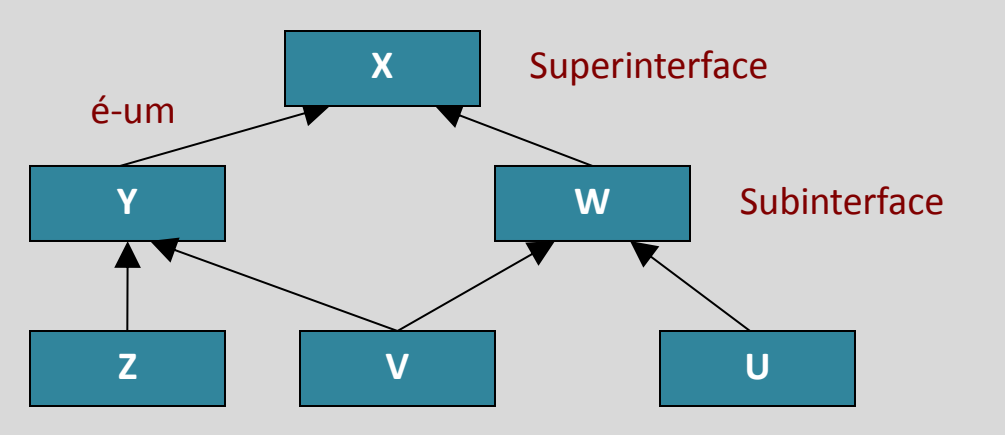
- **Especificações Opcionais**
  - Métodos
  - Constantes

- **Estruturas de Interfaces possíveis**



# Hierarquia de Interfaces

- **Interfaces podem pertencer a Hierarquias de Interfaces**
  - Diferente da Hierarquia de Classes
- **Tipos de Hierarquia (em Java)**
  - Hierarquia de Classes
  - Hierarquia de Interfaces




- **Hierarquia de Interfaces**
  - Estabelece relações hierárquicas entre interfaces
    - Relações do tipo é-um
      - Subinterface estende (aumenta) Superinterface

```
public interface V extends Y, W { ... }
```

  - Permitida herança múltipla
    - Subinterface pode herdar múltiplas Superinterfaces diretas
  - Uma interface pode ter ou não Superinterfaces
    - Exemplos
      - Interface X // não tem Superinterface
      - Interface Z // tem Superinterfaces Y e X

# Declaração de uma Interface

- **Sintaxe**

```
[modificador de acesso] interface nomeInterface [ extends Interface1, ..., InterfaceN ] {

    // Declarações de constantes
    [public static final] tipo nomeConstante1 = value1;
    ...
    [public static final] tipo nomeConstanteN = valueN;

    // Declarações de métodos (só cabeçalhos)
    [public abstract] tipoRetorno nomeMetodo1( listaParâmetros );
    ...
    [public abstract] tipoRetorno nomeMetodoM( listaParâmetros );
}
```

Constantes
- Por omissão (implicitamente)
  public static final

Métodos
- Por omissão
  public abstract
- Só de instância

- **Exemplos**
  - Definido pelo utilizador (programador)

```
public interface Calculos { double m2(); }
```

Uma interface é armazenada num ficheiro com o mesmo nome e extensão java
- Exemplo: Calculos.java

  - Nativas do Java

```
public interface Serializable { }                          // abordada em Ficheiros
public interface Comparable { int compareTo( Object o ); }
```

# Uso de Interfaces



- **Interfaces podem ser usadas para**
  - Implementar numa classe
  - Declarar
    - Variável
    - Parâmetro de método

- **Implementação de uma interface numa classe**
  - Depende do tipo de classe
    - Abstrata: não é obrigatório implementar qualquer método da interface usada
    - Concreta: se não implementar todos os métodos da interface usada, tem de passar a classe abstrata
  - Declaração de classe concreta que implementa (usa) interfaces

```
[modificador de acesso] class nome [extends Classe]  implements  Interface1, ..., InterfaceN {
        ...
        // implementação obrigatória de todos os métodos dos N interfaces
        ...
}
```

  - Declaração de classe abstrata que implementa (usa) interfaces

```
[modif. de acesso] abstract class nome [extends Classe]  implements  Interface1, ..., InterfaceN {
        ...
        // implementação dos métodos dos N interfaces não é obrigatória
        ...
}
```

  - Exemplo
    - Próximo slide

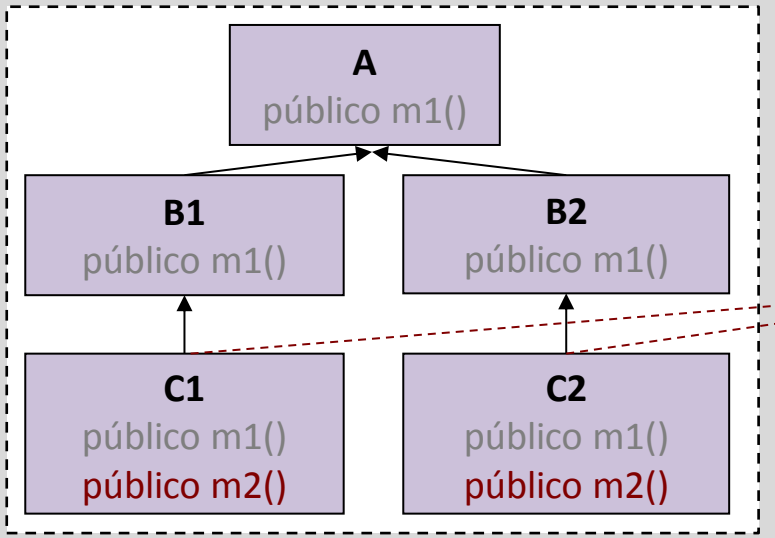
# Uso de Interfaces



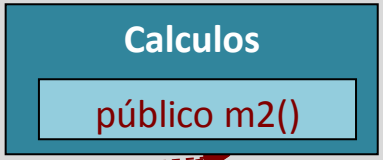
- **Exemplo de implementação da interface Calculos**

```
public class C1 extends B1 implements Calculos {
    ...
    ...
   public double m2() {
      ...
   }
}
```

```
public class C2 extends B2 implements Calculos {
   ...
   ...
   public double m2() {
      ...
   }
}
```




Hierarquia de Classes




implementa

Classes **C1** e **C2** implementam método especificado no tipo Calculos

# Uso de Interfaces



- **Na declaração de variáveis de tipo Interface**
  - Sintaxe

```
nomeInterface  nomeVariável;          // variável tipo simples
nomeInterface[ ]  nomeVariável;       // variável tipo array
```

  - Exemplo

```
Calculos x1;                          // variável tipo simples
x1 = new C1();
Calculos x2 = new C2();

Calculos[] a = new Calculos[10];      // variável tipo array
a[0] = x1;
a[1] = x2;
```

# Uso de Interfaces



- **Na declaração de parâmetros de métodos**
  - Sintaxe

```
nomeInterface  nomeParâmetro;          // parâmetro tipo simples
nomeInterface[ ]  nomeParâmetro;       // parâmetro tipo array
```

  - Exemplo

```
public  void  m( Calculos  c ){                    // parâmetro tipo simples
        // Corpo do método
}
```

```
public  static  void  ordenar( Calculos[ ] a ){    // parâmetro tipo array
        // Corpo do método
}
```

# Comparação entre Interfaces e Classes Abstratas

- **Têm propósitos diferentes**
  - Classes abstratas: Permitir criação de hierarquias de classes
  - Interfaces: Especificar e garantir implementação de funcionalidades adicionais comuns a classes não relacionadas hierarquicamente
- **Há muitas diferenças entre interfaces e classes abstratas**
  - Exemplos

| Interface | Classe Abstrata |
|---|---|
| 100% abstrato (abstração total) | Pode não ser 100% abstrato (abstração parcial ou total) |
| Classe pode implementar vários interfaces | Classe só pode herdar uma classe abstrata |

- **Daí**
  - Interfaces não podem ser considerados substitutos de classes abstratas
- **Não há regras para decidir entre classes abstratas e interface**
  - Algumas orientações:
    - Classe abstrata não é apropriada para definir métodos que não são comuns a todas as subclasses duma hierarquia de classes
      - Porque obriga implementação dos seus métodos abstratos nas classes instanciáveis
    - Usar interfaces para evitar criação de hierarquias de classes artificiais e a reestruturação de hierarquias

# Sumário Geral

# Interfaces Java Nativas

- **Interfaces**
  - Disponibilizados pelo JDK
- **Exemplos**
  - Comparable
  - Comparator
- **Interesse de Comparable e Comparator**
  - Ordenação
  - Pesquisa

  } de Contentores
    - Realizada por Métodos das Classes Nativas
      - Arrays
      - Collections
- **Nesta Aula**
  - Abordada a Ordenação de Contentores

# Métodos de Ordenação de Contentores

- **Disponibilizados pelas Classes** // package java.util
  - Arrays // prestadora de serviços para arrays
  - Collections // prestadora de serviços para coleções (Ex: ArrayList)

- **Métodos de Ordenação da Classe Arrays**
  - Aplicam-se a Contentores Tipo Array
    - Com Objetos Comparable // requer interface Comparable
    - Com Objetos Não-Comparable // requer interface Comparator

- **Métodos de Ordenação da Classe Collections**
  - Aplicam-se a Contentores Tipo ArrayList
    - Com Objetos Comparable // requer interface Comparable
    - Com Objetos Não-Comparable // requer interface Comparator

# Métodos de Ordenação de Contentores Tipo Array

- **Métodos da Classe Arrays**
- **Para arrays com Objetos Comparable**

```
public  static  void  sort( Object[]  a )                    // método de classe
```

  - Obrigatório
    - Objetos do array a … sejam do tipo Comparable
      - Cujas classes implementam interface Comparable
    - Podem implicar a alteração de classes existentes
  - Ordenação ascendente
    - Critério de ordenação …definido nas classes dos objetos

- **Para arrays com Objetos Não-Comparable**

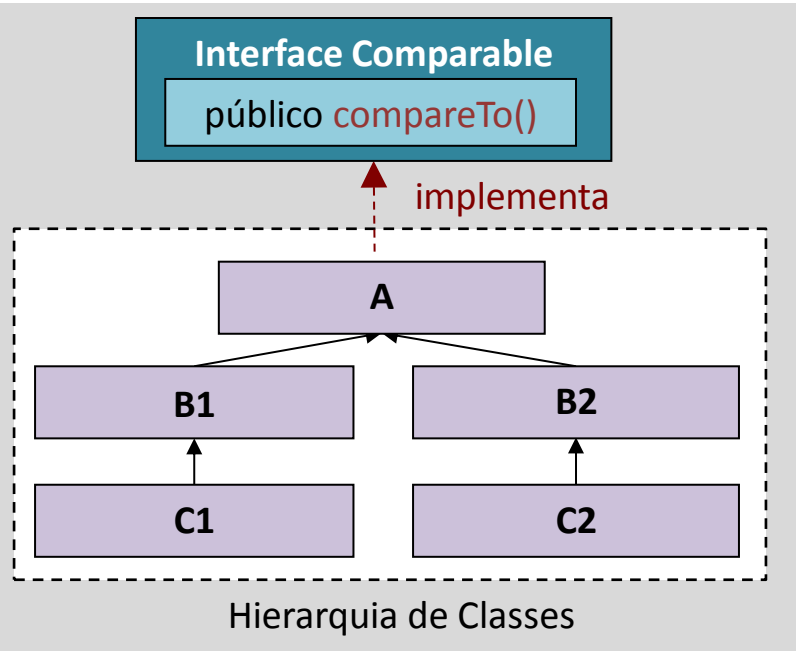
```
public  static  void  sort( T[]  a, Comparator  c )          // método de classe
```

  - Objetos não-Comparable
    - Cujas classes não implementam interface Comparable
    - Não obrigam alteração de classes existentes
  - Ordenação ascendente
    - Critério de ordenação … definido no objeto Comparator c passado por parâmetro

# Ordenação de Array com Objetos Comparable

- **Exemplo**
  - Ordenação de array com instâncias da Hierarquia de Classes abaixo apresentada
    - Critério de ordenação: resultado de m1()
    - Ordem: ascendente
- **Para tornar Objetos Comparable**
  - Classes têm de implementar interface Comparable
- **Critério de Ordenação**
  - Definido no método compareTo() dos objetos do array
- **Método de Ordenação**
  - Arrays.sort( array )

```
import java.util.Arrays;
public class DemoInterfaces {
        public static void main(String[] args) {
                A[] a = new A[10];
                a[0] = new B1();
                a[1] = new B2();
                  ...
                Arrays.sort( a );                    // por m1()
                for(int i=0; i<a.length; i++){
                     System.out.println( a[i] ); // objetos ordenados
                }
        }
}
```



Hierarquia de Classes

# Interface Comparable



- **Declaração**

```
public interface Comparable { public int compareTo( Object obj ); }
```

- **Método compareTo**
  - Requisitos
    - Comparar objeto parâmetro obj com objeto recetor da mensagem
    - Retornar valor inteiro
      - Negativo (<0 : típico -1) // valor do recetor < valor do parâmetro
      - Igual a Zero (0) // valor do recetor = valor do parâmetro
      - Positivo (>0 : típico 1) // valor do recetor > valor do parâmetro

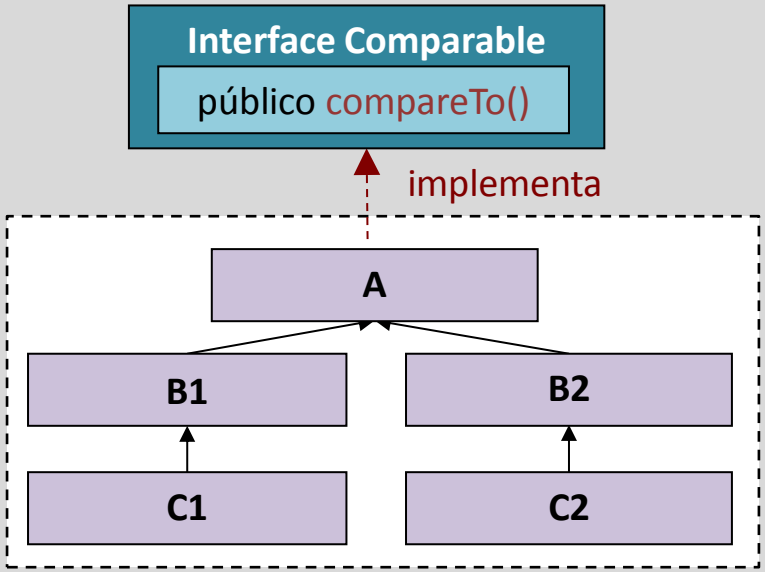
- **Implementado por algumas classes nativas do Java**
  - Objetivo
    - Ter uma ordem nas suas instâncias // designada ordem natural
  - Exemplos
    - String
    - Date

- **Implementação desta interface numa classe garante a quem a usa**
  - Ordem nas suas instâncias
    - Segundo critério (método) de ordenação definido no método compareTo

# Interface Comparable



- **Exemplo**
  - Implements na superclasse
    - Garante implementações em todas as classes instanciáveis da hierarquia



Hierarquia de Classes

```
public class A implements Comparable {
  ...
  public int compareTo( Object obj ) {
     A o = (A) obj;
     if ( m1() > o.m1() )
       return 1;
     else if (m1() < o.m1() )
       return -1;
     else
       return 0;
   }
}
```

```
public class B1 extends A {
   ...
  // Herda método compareTo de A
}
```

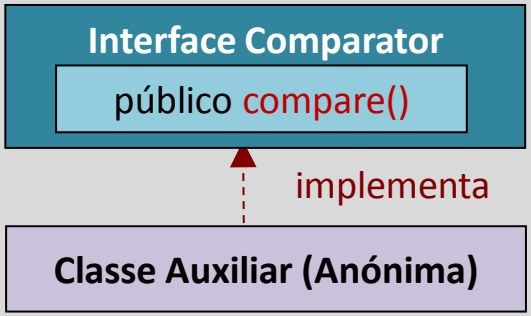
```
public class B2 extends A {
   ...
  // Herda método compareTo de A
}
```

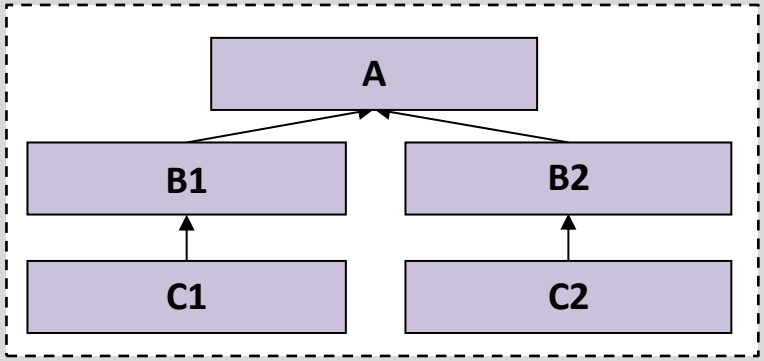
# Ordenação de Array com Objetos não-Comparable

- **Exemplo**
  - Anterior
- **Método de Ordenação**
  - Arrays.sort( array , objetoComparator )
- **Critério de Ordenação**
  - Definido no método compare() de um objeto auxiliar do tipo Comparator (objetoComparator)
  - Vantagens
    - Não obriga a modificar classes dos objetos do array
    - Permite ordenar mesmo array por critérios diferentes

```
import java.util.Arrays;
public class DemoInterfaces {
        public static void main(String[] args) {
                A[] a = new A[10];
                a[0] = new B1();
                a[1] = new B2();
                  ...
                Comparator criterio = new Comparator() { ? };
                Arrays.sort( a, criterio );          // por m1()
                for(int i=0; i<a.length; i++){
                    System.out.println(a[i]);  // obj.  ordenados
                }
        }
}
```

Slides Seguintes




Hierarquia de Classes

# Interface Comparator



- **Declaração**

```
public interface Comparator { int compare( Object o1, Object o2 ); }
```

- **Semelhante ao interface Comparable**
  - Especifica método com
    - Sintaxe diferente
    - Semântica igual

- **Método compare**
  - Requisitos
    - Comparar os objetos parâmetro (o1 e o2)
    - Retornar como resultado um valor inteiro
      - Negativo (<0 : típico -1) // valor do parâmetro 1 < valor do parâmetro 2
      - Igual a Zero (0) // valor do parâmetro 1 = valor do parâmetro 2
      - Positivo (>0 : típico 1) // valor do parâmetro 1 > valor do parâmetro 2

# Interface Comparator



```
public class DemoInterface {                          // Exemplo de uso da interface Comparator
    public static void main(String[] args) {
        A[] a = new A[10];
        a[0] = new B1();
        a[1] = new B2();
         ...
        // Objeto para definir critério de ordenação das instâncias
        Comparator criterio = new Comparator() {      // classe anónima cria só 1 objeto
            public int compare( Object o1, object o2 ) {   // classe anónima = classe interna
                double r1 = ( (A) o1 ).m1();
                double r2 = ( (A) o2 ).m1();
                if ( r1 == r2 )  return 0;
                else if ( r1 > r2  ) return 1;
                else  return -1;
            } } ;
        // Ordenação do array segundo critério fornecido em criterio
        Arrays.sort( a,  criterio );                  // Ordenação ... pelo m1()
        for(int i=0; i<a.length; i++){
            System.out.println( a[i] );               // instâncias ordenadas
        }
    }
}
```

# Métodos de Ordenação de Contentores Tipo ArrayList

- **Métodos da Classe Collections**
  - Semelhantes aos da classe Arrays // parâmetro tipo List em vez de array
- **Para ArrayLists com Objetos Comparable**

```
public  static  void  sort( List lista )                    // método de classe
```

  - Obrigatório
    - Objetos do arraylist lista sejam do tipo Comparable
      - Cujas classes implementam interface Comparable
    - Podem implicar a alteração de classes existentes
  - Ordenação ascendente
    - Critério de ordenação … definido nas classes dos objetos
- **Para ArrayLists com Objetos Não-Comparable**

```
public  static  void  sort( List  lista, Comparator  c )    // método de classe
```

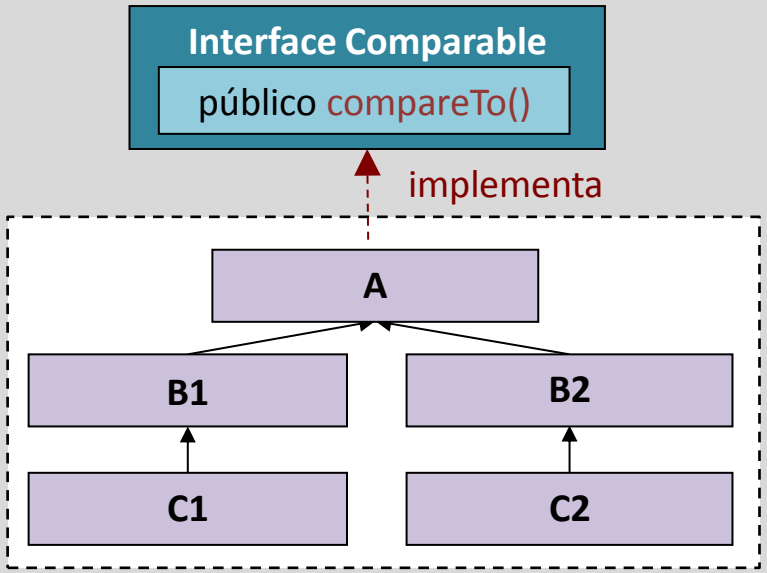
  - Objetos não-Comparable
    - Cujas classes não implementam interface Comparable
    - Não obrigam alteração de classes existentes
  - Ordenação ascendente
    - Critério de ordenação … definido no objeto Comparator c passado por parâmetro

# Ordenação de ArrayList com Objetos Comparable

- **Exemplo**
  - Obrigatório contentor de instâncias Comparable

```
public class DemoInterfaces {
        public static void main(String[] args) {
                ArrayList a = new ArrayList();
                a.add( new B1() );
                a.add( new B2() );
                ...
                Collections.sort( a );          // por m1()
                for( Object obj : a ){
                   System.out.println( obj );
                }
        }
}
```

```
public class A  implements  Comparable {
   ...
   public int compareTo( Object o ) {
       if ( m1() > ( (A) o).m1() )  return 1;
       else if (m1() < ( (A) o).m1() )  return -1;
       else return 0;
    }
}
```

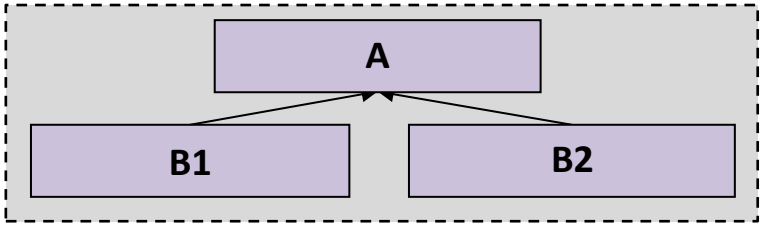



Hierarquia de Classes

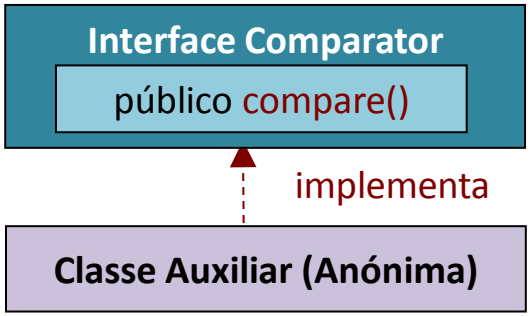
# Ordenação de ArrayList com Objetos não-Comparable

- **Exemplo**

```
public class DemoInterfaces {
    public static void main(String[] args) {
        ArrayList a = new ArrayList();
        a.add( new B1() );
        a.add( new B2() );
          ...
        Comparator criterio = new Comparator() {          // classe anónima cria só 1 objeto
            public int compare(Object o1, object o2) {    // para definir critério de ordenação das figuras
                double r1 = ( (A) o1 ).m1();
                double r2 = ( (A) o2 ).m1()
                if ( r1 == r2 )  return 0;
                else if ( r1 > r2  ) return 1;
                else  return -1;
            } } ;
        // Ordenação do array segundo critério fornecido em criterio
        Collections.sort( a ,  criterio );                // Ordenação por m1()
        for( Object obj : a ){
            System.out.println( obj );                    // mostra instâncias ordenadas
        }
    }
}
```

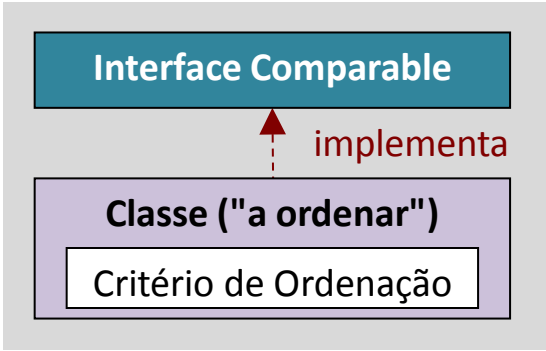


Hierarquia de Classes



# Comparação de Interfaces: Comparable e Comparator

- **Comparator**
  - Permite solução mais flexível do que solução Comparable
- **Solução Comparable**
  - Obriga implementação dessa interface
    - Em todas as classes dos objetos a ordenar
      - Inaceitável quando se pretende código estável, genérico e incremental
    - Ou então, numa nova subclasse
      - Apenas para ordenar
  - Permite só 1 critério de ordenação
    - Definido nas classes dos objetos a ordenar




- **Solução Comparator**
  - Não obriga implementação da interface nas classes dos objetos a ordenar
    - i.e., não requer a modificação de classes existentes
  - Permite múltiplos critérios de ordenação
    - Definidos em classes auxiliares tipo Comparator