Interfaces

Professoras:
Ariane Machado Lima
Fátima L. S. Nunes

Interface

- O que significa?

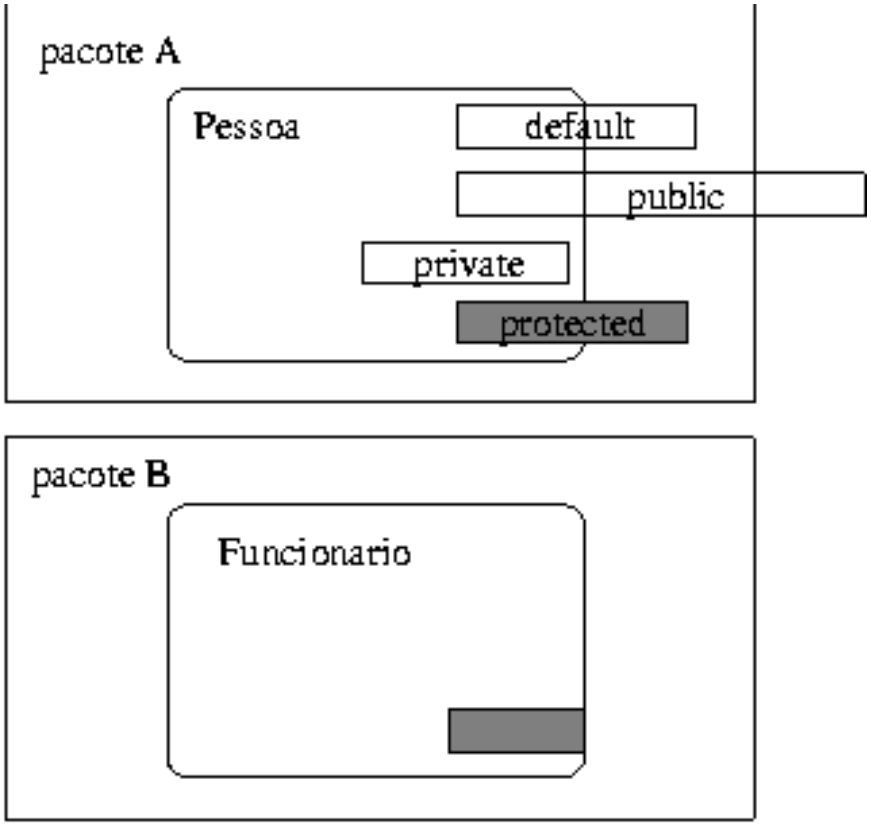
Interface

- Definição:

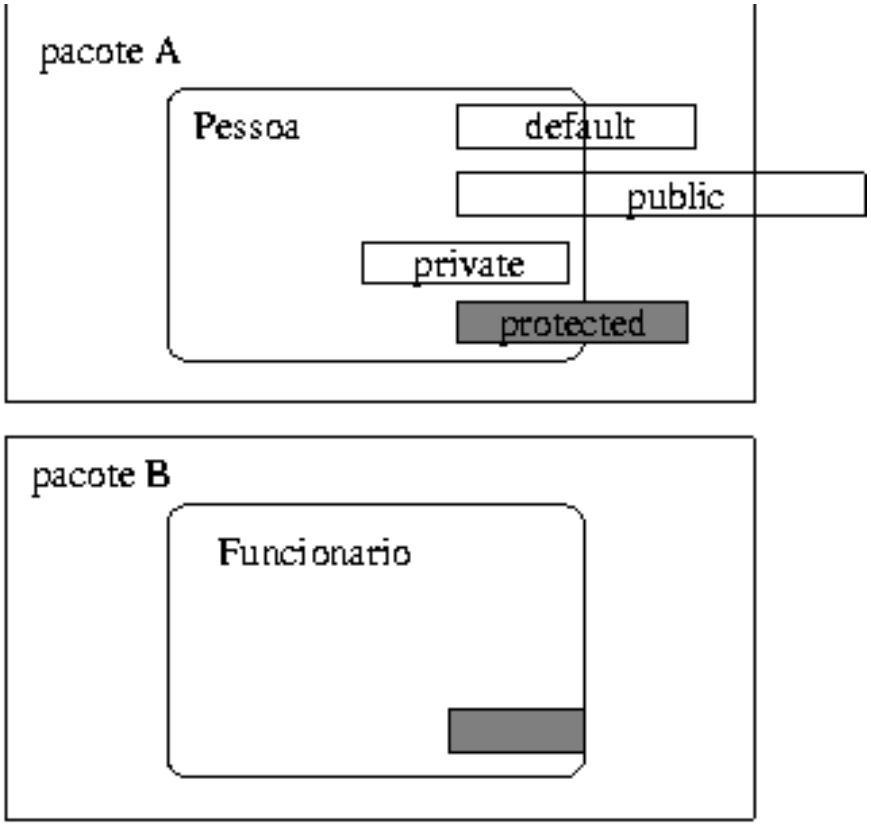
s.f. Limite comum a dois corpos, sistemas, fases ou espaços, que permite sua ação mútua ou intercomunicação ou trocas entre eles: interface produção-distribuição; interface gás-líquido. / Ponto em que interagem coisas diversas. / Informática

Meio físico ou lógico através do qual um ou mais dispositivos ou sistemas incompatíveis conseguem comunicar-se entre si.

Interface


pacote A
Pessoa
default
public
private
protected
pacote B
Funcionario

Interface



- O que é API (*Application Programming Interface*)?
- .... pensem na API do Java....

Interface

- Encapsulamento ???

Interface

- Encapsulamento:
  - Conceito da POO que permite esconder do usuário os atributos e a implementação dos métodos da classe.
  - Conhecendo apenas a *interface* de uma classe, podemos usar objetos da mesma sem conhecer detalhes de implementação.
  - Interface = métodos disponíveis + suas respectivas assinaturas.

Interface

- Interface:
  - Útil também quando queremos ter uma classe, mas não implementá-la.
  - Muito interessante na fase de especificação de um Sistema.
  - Ex: eu poderia definir uma interface de Pessoa (isto é, métodos que uma Pessoa deveria saber executar) sem ter que implementá-la
    - Implementação ficaria por conta das "subclasses"

```
class ComunidadeAcademica
{
        Pessoa [] comunidade = new Pessoa [5000];
        int nrPessoas = 0;
        void inserePessoa(Pessoa p)
        {
                comunidade[nrPessoas] = p;
                nrPessoas++;
        }
        Pessoa buscaPessoa (int nrUsp)
        {
                int i = 0;
                while (i < nrPessoas){
                        if (comunidade[i].obtemNrUsp() == nrUsp)
                                return comunidade[i];
                        i++;
                }
                System.out.println("Pessoa não encontrada");
                return null;
        }
        void imprimeDados(){
                for (int i=0; i<nrPessoas; i++)
                        comunidade[i].imprimeDados();
        }
}
```

Neste exemplo, que métodos Pessoa deveria saber executar?

```
class ComunidadeAcademica
{
        Pessoa [] comunidade = new Pessoa [5000];
        int nrPessoas = 0;
        void inserePessoa(Pessoa p)
        {
                comunidade[nrPessoas] = p;
                nrPessoas++;
        }
        Pessoa buscaPessoa (int nrUsp)
        {
                int i = 0;
                while (i < nrPessoas){
                        if (comunidade[i].obtemNrUsp() == nrUsp)
                                return comunidade[i];
                        i++;
                }
                System.out.println("Pessoa não encontrada");
                return null;
        }
        void imprimeDados(){
                for (int i=0; i<nrPessoas; i++)
                        comunidade[i].imprimeDados();
        }
}
```

Neste exemplo, que métodos Pessoa deveria saber executar?

Interface

- Um exemplo: criação de um zoológico virtual
  - Vários tipos de animais
  - Mensagens para cada animal:
    - `nasça()`
    - `passeiePelaTela()`
    - `durma()`

Interface

- Um exemplo: criação de um zoológico virtual
  - Mas você não conhece os animais !!!
    - Como um morcego nasce?
    - Como uma zebra dorme?
    - Como um pato anda?

Interface

- Um exemplo: criação de um zoológico virtual
  - Como fazer ???

Interface

- Um exemplo: criação de um zoológico virtual
  - Como fazer ???
  - Pedir para alguém que conhece os animais implementar as classes necessárias. Exemplo: Zebra, Morcego e Pato
  - Só que eu preciso ter um padrão para todos os animais...
  - Então...

Interface

- Um exemplo: criação de um zoológico virtual

```
interface Animal
{
   void nasça();
   void passeiePelaTela();
   void durma();
   double peso();
}
```

Interface

- Um exemplo: criação de um zoológico virtual

```
interface Animal
{
   void nasça();
   void passeiePelaTela();
   void durma();
   double peso();
}
```

Interface

- Um exemplo: criação de um zoológico virtual

```
interface Animal
{
   void nasça();
   void passeiePelaTela();
   void durma();
   double peso();
}
```

O programador que vai implementar a classe Morcego deve declarar explicitamente que vai usar a interface Animal ⇨ palavra chave *implements*

Interface

```
interface Animal
{
   void nasça();
   void passeiePelaTela();
   void durma();
   double peso();
}
```

```
public class Morcego implements Animal
{
   public void nasça()
   {
      System.out.println("Nasceu  um morceguinho...");
   }

   public void passeiePelaTela()
   {
      System.out.println("Voo  rápido  e rasante !!!");
   }

  public void durma()
   {
      System.out.println("Dorme  de cabeça para baixo.");
   }

  public double peso()
   {
      System.out.println("Este  é um morcego gordo!");
      return (4.5);
   }

}
```

Interface

```
interface Animal
{
   void nasça();
   void passeiePelaTela();
   void durma();
   double peso();
}
```

```
public class Morcego implements Animal
{
   public void nasça()
   {
      System.out                ceu  um morceguinho...");
   }
 p
   {
   }
 p
   {
   }
 double peso()
   {
      System.out.println("Este  é um morcego gordo!");
      return (4.5);
   }
}
```

implements – obriga o programador a escrever o código de todos os métodos, considerando suas assinaturas. Todos os métodos da interface devem ser públicos. Por quê?

Interface

```
interface Animal
{
   void nasça();
   void passeiePelaTela();
   void durma();
   double peso();
}
```

Não definimos especificadores de acesso na interface, pelo mesmo motivo.

```
public class Morcego implements Animal
{
   public void nasça()
   {
      System.out                    ceu  um morceguinho...");
   }
 p
   {
   }
 p
   {
   }
double peso()
   {
      System.out.println("Este  é um morcego gordo!");
      return (4.5);
   }

}
```

implements – obriga o programador a escrever o código de todos os métodos, considerando suas assinaturas. Todos os métodos da interface devem ser públicos.

Interface

```
interface Animal
{
   void nasça();
   void passeiePelaTela();
   void durma();
   double peso();
}
```

```
public class Pato implements Animal
{
public double peso;

   Pato(double p)
   {
      peso = p;
   }

   public void nasça()
   {
      System.out.println("Quebra o ovo.");
   }

   public void passeiePelaTela()
   {
      System.out.println("Anda em duas patas. Quá Quá Quá.");
   }

   public void durma()
   {
      System.out.println("Dorme em pé.");
   }

   public double peso()
   {
      return peso;
   }
}
```

Interface

```
interface Animal
{
   void nasça();
   void passeiePelaTela();
   void durma();
   double peso();
}
```

```
public class Zebra implements Animal
{
   double peso;
   int listras;

   Zebra(double p, int l)
   {
      peso = p;
      listras =l;
   }

   public void nasça()
   {
      System.out.println("Zebrinha bebê nascendo...");
   }

   public void passeiePelaTela()
   {
     System.out.println("Zebra galopa pela tela.");
   }

   public void durma()
   {
     System.out.println("Zebra dorme sem roncar.... zzzzzzzzzzzzzzzz...");
   }

   public double peso()
   {
     return peso;
   }

   public void exibeListras()
   {
     System.out.println("Minha zebra tem " + listras + "listras.");
   }

}
```

Interface

```
interface Animal
{
   void nasça();
   void passeiePelaTela();
   void durma();
   double peso();
}
```

```
public class Zebra implements Animal
{
  double peso;
  int listras;

  Zebra(double p, int l)
  {
    peso = p;
    listras =l;
  }

  public void nasça()
  {
    System.out.println("Z
  }

  public void passeiePe
  {
   System.out.println("Z
  }

  public void durma()
  {
   System.out.println("Z                                    zzzzz...");
  }

  public double peso()
  {
   return peso;
  }

  public void exibeListras()
  {
   System.out.println("Minha zebra tem " + listras + "listras.");
  }

}
```

Pode ter métodos extras (específicos da classe)

Interface

- Observação sobre arquivos X classes em Java:
  - normalmente cada classe vai em um arquivo .java de mesmo nome da classe, mas isto não é obrigatório;
  - cada arquivo ".java" pode ter no máximo uma classe pública (do tipo public);
  - caso exista uma classe pública, seu nome deve ser igual ao do arquivo ".java";
  - desta forma, as classes anteriores (Morcego, Pato e Zebra) devem obrigatoriamente estar em arquivos separados;

Interface

- Observação sobre interfaces:
  - não podemos criar objetos a partir de uma interface;
  - no exemplo, não podemos criar objetos diretamente da classe Animal;
  - criamos objetos a partir das classes que implementam a interface (Morcego, Pato e Zebra);
  - no entanto, cada objeto criado, além de ser um objeto da classe respectiva, também é um objeto do tipo Animal (INTERFACE TAMBÉM DEFINE UM TIPO);
  - Vamos usar as classes criadas...

Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50,  105);
    Animal  zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```

O que significa?

Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50,  105);
    Animal  zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```

O que significa? Posso executar os métodos sem ter objetos instanciados

Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50, 105);
    Animal zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```

```
> Zoologico.fazFuncionar()
Zebrinha bebê nascendo...
Zebra galopa pela tela.
Zebra dorme sem roncar....
zzzzzzzzzzzzzzzz...
Zebrinha bebê nascendo...
Zebra galopa pela tela.
Zebra dorme sem roncar....
zzzzzzzzzzzzzzzz...
Nasceu um morceguinho...
Voo rápido e rasante !!!
Dorme de cabeça para baixo.
Quebra o ovo.
Anda em duas patas. Quá Quá Quá
Dorme em pé
```

Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50, 105);
    Animal zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```

Observe este código.
Quais as consequências dessas definições ???

```
interface Animal

{

   void nasça();

   void passeiePelaTela();

   void durma();

   double peso();

}
```

```
public class Zebra implements Animal
{

  double peso;
  int listras;

   Zebra(double p, int l)
   {
      peso = p;
      listras =l;
   }

   public void nasça()
   {
       System.out.println("Zebrinha bebê nascendo...");
   }

   public void passeiePelaTela()
   {
     System.out.println("Zebra galopa pela tela.");
   }

   public void durma()
   {
     System.out.println("Zebra dorme sem roncar.... zzzzzzzzzzzzzzz...");
   }

   public double peso()
   {
     return peso;
   }

   public void exibeListras()
   {
     System.out.println("Minha zebra tem " + listras + "listras.");
   }

}
```

```
interface Animal

{

   void nasça();

   void passeiePelaTela();

   void durma();

   double peso();

}
```

```
public class Zebra implements Animal
{

  double peso;
  int listras;

   Zebra(double p, int l)
   {
      peso = p;
      listras =l;
   }

   public void nasça()
   {
      System.out.println("Zebrinha bebê nascendo...");
   }

   public void passeiePelaTela()
   {
     System.out.println("Zebra galopa pela tela.");
   }

   public void durma()
   {
     System.out.println("Zebra dorme sem roncar.... zzzzzzzzzzzzzzzz...");
   }

   public double peso()
   {
     return peso;
   }

   public void exibeListras()
   {
     System.out.println("Minha zebra tem " + listras + "listras.");
   }

}
```

```
interface Animal

{

   void nasça();

   void passeiePelaTela();

   void durma();

   double peso();

}
```

```
public class Zebra implements Animal
{

  double peso;
  int listras;

   Zebra(double p, int l)
   {
      peso = p;
      listras =l;
   }

   public void nasça()
   {
      System.out.println("Zebrinha bebê nascendo...");
   }

   public void passeiePelaTela()
   {
     System.out.println("Zebra galopa pela tela.");
   }

   public void durma()
   {
     System.out.println("Zebra dorme sem roncar.... zzzzzzzzzzzzzzz...");
   }

   public double peso()
   {
     return peso;
   }

   public void exibeListras()
   {
     System.out.println("Minha zebra tem " + listras + "listras.");
   }

}
```

```
interface Animal

{

   void nasça();

   void passeiePelaTela();

   void durma();

   double peso();

}
```

```
public class Zebra implements Animal
{

  double peso;
  int listras;

   Zebra(double p, int l)
   {
     peso = p;
     listras =l;
   }

   public void nasça()
   {
      System.out.println("Zebrinha bebê nascendo...");
   }

   public void passeiePelaTela()
   {
     System.out.println("Zebra galopa pela tela.");
   }

   public void durma()
   {
     System.out.println("Zebra dorme sem roncar.... zzzzzzzzzzzzzzz...");
   }

   public double peso()
   {
     return peso;
   }

   public void exibeListras()
   {
     System.out.println("Minha zebra tem " + listras + "listras.");
   }

}
```

```
interface Animal

{

   void nasça();

   void passeiePelaTela();

   void durma();

   double peso();

}
```

```
public class Zebra implements Animal
{

  double peso;
  int listras;

   Zebra(double p, int l)
   {
     peso = p;
     listras =l;
   }

   public void nasça()
   {
      System.out.println("Zebrinha bebê nascendo...");
   }

   public void passeiePelaTela
   {
     System.out.println("Zebra
   }

   public void durma()
   {
     System.out.println("Zebra dorme sem roncar.... zzzzzzzzzzzzzzz...");
   }

   public double peso()
   {
     return peso;
   }

   public void exibeListras()
   {
     System.out.println("Minha zebra tem " + listras + "listras.");
   }

}
```

Não faz parte da interface...
E daí?

Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50,  105);
    Animal  zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```

Não posso instanciar objetos da interface (Animal).
Posso instanciar somente das classes que implementam a interface...
Mas posso definir que o tipo do objeto instanciado é Animal

Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50,  105);
    Animal  zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```

Consequência:

> Zebra zebraPequena = new Zebra(50, 105);

> Animal zebraGrande = new Zebra(230,160);

> zebraPequena.exibeListras();

Minha zebra tem 105 listras.

> zebraGrande.exibeListras();

????

Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50,  105);
    Animal  zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```

Consequência:

> Zebra zebraPequena = new Zebra(50, 105);

> Animal zebraGrande = new Zebra(230,160);

> zebraPequena.exibeListras();

Minha zebra tem 105 listras.

> zebraGrande.exibeListras();

Static Error: No method in Animal with name 'exibeListras' accepts arguments ()

Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50, 105);
    Animal zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```

Consequência:

> Zebra zebraPequena = new Zebra(50, 105);

> Animal zebraGrande = new Zebra(230,160);

> zebraPequena.exibeListras();

Minha zebra tem 105 listras.

> zebraGrande.exibeListras();

Static Error: No method in Animal with name 'exibeListras' accepts arguments ()

A associação de qual método executar é dinâmica, mas a interface é definida estaticamente (Java é tipada!)

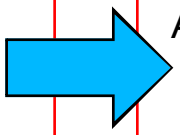
Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50,  105);
    Animal  zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```

Dá para melhorar ???

Interface

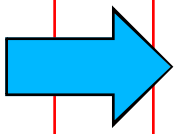
```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50, 105);
    Animal zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```



```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
   Animal bicharada []= new Animal [4];
    bicharada [0] = new Zebra(50, 105);
    bicharada [1] = new Zebra(230,160);
    bicharada [2] = new Morcego();
    bicharada [3] = new Pato (3.2);

    ???
  }
}
```

Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50, 105);
    Animal zebraGrande = new Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```



```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
   Animal bicharada []= new Animal [4];
    bicharada [0] = new Zebra(50, 105);
    bicharada [1] = new Zebra(230,160);
    bicharada [2] = new Morcego();
    bicharada [3] = new Pato (3.2);

    for (int i = 0; i < bicharada.length; i++)
      cicloDeVida(bicharada[i]);
  }
}
```

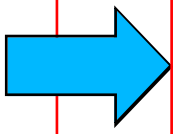
Interface

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
   Animal  bicharada []= new Animal  [4];
    bicharada  [0] = new Zebra(50,  105);
    bicharada  [1] = new Zebra(230,160);
    bicharada  [2] = new Morcego();
    bicharada  [3] = new Pato (3.2);

    for (int i = 0; i < bicharada.length; i++)
      cicloDeVida(bicharada[i]);
   }
}
```

Dá para escrever de outro jeito ???

Interface

```
class Zoologico
{
 static void cicloDeVida (Animal animal)
 {
  animal.nasça();
  animal.passeiePelaTela();
  animal.durma();
 }

 static public void fazFuncionar()
 {
  Animal bicharada []= new Animal [4];
   bicharada [0] = new Zebra(50, 105);
   bicharada [1] = new Zebra(230,160);
   bicharada [2] = new Morcego();
   bicharada [3] = new Pato (3.2);

    for (int i = 0; i < bicharada.length; i++)
      cicloDeVida(bicharada[i]);
  }
}
```



```
class Zoologico
{
 static void cicloDeVida (Animal animal)
 {
  animal.nasça();
  animal.passeiePelaTela();
  animal.durma();
 }

 static public void fazFuncionar()
 {
  Animal bicharada []=
  {new Zebra(50, 105), new Zebra(230,160),
    new Morcego(), new Pato (3.2)}

   for (int i = 0; i < bicharada.length; i++)
      cicloDeVida(bicharada[i]);
  }
}
```

Interface

Primeira versão

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
    Zebra zebraPequena = new Zebra(50,
      105);
    Animal  zebraGrande = new
      Zebra(230,160);
    Morcego m = new Morcego();
    Pato p= new Pato (3.2);

    cicloDeVida(zebraPequena);
    cicloDeVida(zebraGrande);
    cicloDeVida(m);
    cicloDeVida(p);
  }
}
```

Última versão

```
class Zoologico
{
  static void cicloDeVida (Animal animal)
  {
    animal.nasça();
    animal.passeiePelaTela();
    animal.durma();
  }

  static public void fazFuncionar()
  {
     Animal  bicharada []=
     {new Zebra(50,  105),  new Zebra(230,160),
      new Morcego(), new Pato (3.2)}

     for (int i = 0; i < bicharada.length; i++)
       cicloDeVida(bicharada[i]);
   }
}
```

Implementando mais de uma Interface

- No mundo real, os objetos não obedecem uma única interface.
- Exemplo:
  - um indivíduo, pode ser ao mesmo tempo :
    - um homem (ou mulher)
    - um professor
    - um coordenador de curso
    - um sócio de clube
  - cada um desses “papéis” poderia entender mensagens próprias…

Implementando mais de uma Interface

- indivíduo – nasce, cresce, morre:
  - homem – alista-se no exército
  - professor – ministra aula, avalia alunos, publica artigos
  - coordenador de curso – atribui aulas, faz horário
  - sócio de clube – paga mensalidade, usa piscina...

Implementando mais de uma Interface

- Exemplo de implementação:
  - interfaces Voador e Transportador dePessoas

```
interface Voador
{
  void voa();
  void aterrisa();
}
```

```
interface TransportadorDePessoas
{
    void entramPessoas();
    void saemPessoas();
}
```

Implementando mais de uma Interface

```
interface Voador
{
  void voa();
  void aterrisa();
}
```

```
class Ave implements Voador
{
  public void voa()
  {
    System.out.println ("Bate as asas");
  }

  public void aterrisa()
  {
    System.out.println ("Desliza suavemente até tocar o
chão");
  }
}
```

Implementando mais de uma Interface

```
interface TransportadorDePessoas
{
    void entramPessoas();
    void saemPessoas();
}
```

```
class Onibus implements
TransportadorDePessoas
{
  public void entramPessoas()
  {
    System.out.println ("Abre a porta
para as pessoas entrarem");
  }
  public void saemPessoas()
  {
    System.out.println ("Abre a porta
para as pessoas saírem");
  }
}
```

Implementando mais de uma Interface

```
interface Voador
{
  void voa();
  void aterrisa();
}
```

```
interface TransportadorDePessoas
{
    void entramPessoas();
    void saemPessoas();
}
```

```
class Aviao  implements  Voador,  TransportadorDePessoas
{
  public void voa()
  {
    System.out.println  ("Liga  as turbinas  e decola.");
  }

  public void aterrisa()
  {
    System.out.println  ("Abaixa  o trem de pouso e desce -
preferencialmente  no chão e suavemente...  :-)");
  }

  public void entramPessoas()
  {
    System.out.println  ("Procedimento  de embarque  - idosos,
gestantes  e portadores  de deficiência  têm preferência...");
  }

  public void saemPessoas()
  {
    System.out.println  ("Desembarque.  Passageiros  devem
aguardar  até  parada  completa  da aeronave.");
  }
}
```

Implementando mais de uma Interface

```
class TestaInterface
{
  public static void executa()
  {
    TransportadorDePessoas t = new Onibus();
    Voador v = new Ave();
    Aviao a  = new Aviao();

    t.entramPessoas();
    t.saemPessoas();
    v.voa();
    v.aterrisa();
    a.entramPessoas();
    a.voa();
  }
}
```

```
> TestaInterface.executa()
Abre a porta para as pessoas entrarem
Abre a porta para as pessoas saírem
Bate as asas
Desliza suavemente até tocar o chão
Procedimento de embarque - idosos,
gestantes e portadores de deficiência têm
preferência...
Liga as turbinas e decola.
```

Um exemplo pouco mais complexo

- Classe Fruta
  - atributos: nome, peso, preço
  - atributos devem ser carregados no construtor

Um exemplo pouco mais complexo

- Classe Fruta
  - atributos: nome, peso, preço
  - atributos devem ser carregados no construtor

```
class Fruta
{
  String nome;
  double preco, peso;

  Fruta (String n, double p, double pr)
  {
        nome = n;
        peso = p;
        preco = pr;
  }

  void imprime()
  {
        System.out.print("Nome: " + nome);
        System.out.print(“ Peso: " + peso);
        System.out.println(“ Preço: " + preco);
  }
}
```

Um exemplo pouco mais complexo

- Classe Fruta
  - queremos criar um vetor de frutas e:
    - ordenar por nome
    - ordenar por peso
    - como fazer ???

Um exemplo pouco mais complexo

- Classe Fruta
  - queremos criar um vetor de frutas e:
    - ordenar por nome
    - ordenar por peso
    - como fazer ???
      - já vimos métodos de ordenação
      - podemos escolher um método e implementar
      - mudar somente o critério de comparação (preço ou peso)

Um exemplo pouco mais complexo

```
class Quitanda
{
  // define array e construtor


  // imprime frutas


  // ordena por Preço


  // ordena por Peso


  // método principal que invoca os demais

}
```

```
class Fruta
{
  String nome;
  double preco, peso;

  Fruta (String n, double p, double pr)
  {
        nome = n;
        peso = p;
        preco = pr;
  }

  void imprime()
  {
        System.out.print("Nome:  " + nome);
        System.out.print("  Peso: " + peso);
        System.out.println("  Preço: " + preco);
  }
}
```

Um exemplo pouco mais complexo

```
class Quitanda
{
  // define array e construtor
  ????

  // imprime frutas


  // ordena por Preço


  // ordena por Peso


  // método principal que invoca os demais

}
```

```
class Fruta
{
  String nome;
  double preco, peso;

  Fruta (String n, double p, double pr)
  {
        nome = n;
        peso = p;
        preco = pr;
  }

  void imprime()
  {
        System.out.print("Nome:  " + nome);
        System.out.print("  Peso: " + peso);
        System.out.println("  Preço: " + preco);
  }
}
```

Um exemplo pouco mais complexo

```
class Quitanda
{
  // define array e construtor
  Fruta [] frutas = new Fruta [5];

  public Quitanda()
  {
    frutas [0] = new Fruta ("Laranja", 0.2, 0.5);
    frutas [1] = new Fruta ("Kiwi", 0.1, 2);
    frutas [2] = new Fruta ("Mamão", 1.5, 1.2);
    frutas [3] = new Fruta ("Pera", 0.3, 1.7);
    frutas [4] = new Fruta ("Banana", 0.3, 0.2);
  }


  // imprime frutas
  // ordena por Preço
```

```
class Fruta
{
  String nome;
  double preco, peso;

  Fruta (String n, double p, double pr)
  {
        nome = n;
        peso = p;
        preco = pr;
  }

  void imprime()
  {
        System.out.print("Nome:  " + nome);
        System.out.print("  Peso: " + peso);
        System.out.println("  Preço: " + preco);
  }
}
```

Um exemplo pouco mais complexo

```
class Quitanda
{
  // define array e construtor
  ...
  // imprime frutas

    ????

  // ordena por Preço


  // ordena por Peso


  // método principal que invoca os demais

}
```

```
class Fruta
{
  String nome;
  double preco, peso;

  Fruta (String n, double p, double pr)
  {
        nome = n;
        peso = p;
        preco = pr;
  }

  void imprime()
  {
        System.out.print("Nome:  " + nome);
        System.out.print("  Peso: " + peso);
        System.out.println("  Preço: " + preco);
  }
}
```

Um exemplo pouco mais complexo

```
class Quitanda
{
  // define array e construtor
  ...
  // imprime frutas
        public void imprime()
          {
            for (int i = 0; i < frutas.length; i++)
              frutas [i].imprime();
          }

}
```

```
class Fruta
{
  String nome;
  double preco, peso;

  Fruta (String n, double p, double pr)
  {
        nome = n;
        peso = p;
        preco = pr;
  }

  void imprime()
  {
        System.out.print("Nome:  " + nome);
        System.out.print("  Peso: " + peso);
        System.out.println("  Preço: " + preco);
  }
}
```

Um exemplo pouco mais complexo

```
class Quitanda
{
  // define array e construtor
  // imprime frutas
  // ordena por Preço
  ????

  // ordena por Peso


  // método principal que invoca os demais


}
```

```
class Fruta
{
  String nome;
  double preco, peso;

  Fruta (String n, double p, double pr)
  {
        nome = n;
        peso = p;
        preco = pr;
  }

  void imprime()
  {
        System.out.print("Nome:  " + nome);
        System.out.print("  Peso: " + peso);
        System.out.println("  Preço: " + preco);
  }
}
```

Um exemplo pouco mais complexo

```
class Quitanda
{
 // define array e construtor
 // imprime frutas

 // ordena por Preço – método inserção direta
void ordenaPorPreco ()
{
   int ivet, isubv;
   Fruta frutaAInserir;

   for (ivet=1; ivet < frutas.length; ivet++)
   {

     frutaAInserir = frutas[ivet];
     isubv = ivet;

     while ((isubv > 0) && (frutas[isubv -1].preco > frutaAInserir.
     {
       frutas[isubv] = frutas[isubv - 1];
       isubv--;
     }
     frutas [isubv] = frutaAInserir;
   }
}
```

class Fruta

```
class Fruta
{
  String nome;
  double preco, peso;

  Fruta (String n, double p, double pr)
  {
        nome = n;
        peso = p;
        preco = pr;
  }

  void imprime()
  {
        System.out.print("Nome:  " + nome);
        System.out.print("  Peso: " + peso);
        System.out.println("  Preço: " + preco);
  }
}
```

```
  }
}
```

Um exemplo pouco mais complexo

```
class Quitanda
{
  // define array e construtor
  // imprime frutas
  // ordena por Preço

 // ordena por Peso
????

  // método principal que invoca os demais


}
```

```
class Fruta
{
  String nome;
  double preco, peso;

  Fruta (String n, double p, double pr)
  {
        nome = n;
        peso = p;
        preco = pr;
  }

  void imprime()
  {
        System.out.print("Nome:  " + nome);
        System.out.print("  Peso: " + peso);
        System.out.println("  Preço: " + preco);
  }
}
```

Um exemplo pouco mais complexo

```
class Quitanda
{
  // define array e construtor
  // imprime frutas
  // ordena por Preço

 // ordena por Peso
void ordenaPorPreco ()
{
    int ivet, isubv;
    Fruta frutaAInserir;

    for (ivet=1; ivet < frutas.length; ivet++)
    {

      frutaAInserir = frutas[ivet];
      isubv = ivet;

      while ((isubv > 0) && (frutas [isubv -1].peso > frutaAInserir.peso))
      {
        frutas[isubv] = frutas[isubv - 1];
        isubv--;
      }
      frutas [isubv] = frutaAInserir;
    }
}
```

```
class Fruta
{
  String nome;
  double preco, peso;

  Fruta (String n, double p, double
pr)
  {
        nome = n;
        peso = p;
        preco = pr;
  }

  void imprime()
  {
        System.out.print("Nome: " +
nome);
        System.out.print(“ Peso: " +
peso);
        System.out.println(“ Preço: " +
preco);
  }
}
```

Um exemplo pouco mais complexo

```
class Quitanda
{
  // define array e construtor
  // imprime frutas
  // ordena por Preço
  // ordena por Peso

  // método principal que invoca os demais
  public static void main (String [] args)
  {
    Quitanda xepa = new Quitanda();

    System.out.println("Desordenado");
    xepa.imprime();
    System.out.println("Em ordem de preço");
    xepa.ordenaPorPreco();
    xepa.imprime();
    System.out.println("Em ordem de peso");
    xepa.ordenaPorPeso();
    xepa.imprime();
  }

}
```

```
C:\Fatima\each-usp\Disciplinas\ACH2001-2009\DrJava>java Quitanda
Desordenado
Nome: Laranja Peso: 0.2 Preto: 0.5
Nome: Kiwi Peso: 0.1 Preto: 2.0
Nome: Mamão Peso: 1.5 Preto: 1.2
Nome: Pera Peso: 0.3 Preto: 1.7
Nome: Banana Peso: 0.3 Preto: 0.2
Em ordem de preto
Nome: Banana Peso: 0.3 Preto: 0.2
Nome: Laranja Peso: 0.2 Preto: 0.5
Nome: Mamão Peso: 1.5 Preto: 1.2
Nome: Pera Peso: 0.3 Preto: 1.7
Nome: Kiwi Peso: 0.1 Preto: 2.0
Em ordem de peso
Nome: Kiwi Peso: 0.1 Preto: 2.0
Nome: Laranja Peso: 0.2 Preto: 0.5
Nome: Banana Peso: 0.3 Preto: 0.2
Nome: Pera Peso: 0.3 Preto: 1.7
Nome: Mamão Peso: 1.5 Preto: 1.2
```

Um exemplo pouco mais complexo

- Qual é o problema do código apresentado?

Um exemplo pouco mais complexo

- Qual é o problema do código apresentado?
  - Código duplicado para ordenação.
- E se tivesse que ordenar por outros critérios?

Um exemplo pouco mais complexo

- Qual é o problema do código apresentado?
  - Código duplicado para ordenação.
- E se tivesse que ordenar por outros critérios?
  - Problema … mais código repetido!

Um exemplo pouco mais complexo

- Qual é o problema do código apresentado?
  - Código duplicado para ordenação.
- E se tivesse que ordenar por outros critérios?
  - Problema … mais código repetido!
- Solução: interface!

Um exemplo pouco mais complexo

- Qual é o problema do código apresentado?
  - Código duplicado para ordenação.
- E se tivesse que ordenar por outros critérios?
  - Problema … mais código repetido!
- Solução: interface!
  - manter o código de comparação em um único método
  - manter em métodos diferentes somente os trechos diferentes.

Um exemplo pouco mais complexo

```
interface ComparadorDeFrutas
{
  boolean ehMenor (Fruta a, Fruta b);
}
```

```
class ComparaPreco implements ComparadorDeFrutas
{
  public boolean ehMenor (Fruta a, Fruta b) {return (a.preco < b.preco);}
}
```

```
class ComparaPeso implements ComparadorDeFrutas
{
  public boolean ehMenor (Fruta a, Fruta b) {return (a.peso < b.peso);}
}
```

```
class ComparaNome implements ComparadorDeFrutas
{
  public boolean ehMenor (Fruta a, Fruta b) {return (a.nome.compareTo(b.nome)< 0);}
}
```

Um exemplo pouco mais complexo

```
class Quitanda2
{
  // define array e construtor


  // imprime frutas


  // ordena – somente um método para ordenar!!!


  // método principal que invoca os demais


}
```

Um exemplo pouco mais complexo

```
class Quitanda2
{
  // define array e construtor
  // imprime frutas
  // ordena – somente um método para ordenar!!!
 void ordena (ComparadorDeFrutas c)
 {
   int ivet, isubv;
   Fruta frutaAInserir;

   for (ivet=1; ivet < frutas.length; ivet++)
   {

     frutaAInserir = frutas[ivet];
     isubv = ivet;

     while ((isubv > 0) && (c.ehMenor(frutaAInserir, frutas [isubv -1]) ))
     {
       frutas[isubv] = frutas[isubv - 1];
       isubv--;
     }
     frutas [isubv] = frutaAInserir;

   }
 }
```

Um exemplo pouco mais complexo

```
class Quitanda2
{
  // métodos anteriores
  // método principal que invoca os demais
public static void main (String [] args)
  {
    Quitanda2 xepa = new Quitanda2();

    System.out.println("Frutas desordenadas");
    xepa.imprime();

    System.out.println("Em ordem de preço");
    ComparadorDeFrutas cpr = new ComparaPreco();
    xepa.ordena(cpr);
    xepa.imprime();

    System.out.println("Em ordem de peso");
    ComparadorDeFrutas cp = new ComparaPeso();
    xepa.ordena(cp);
    xepa.imprime();

    System.out.println("Em ordem alfabética de nome");
    xepa.ordena(new ComparaNome ()); // forma super contraída de chamar o método ordena
    xepa.imprime();
  }

}
```

Um exemplo pouco mais complexo

```
class Quitanda2
{
  // métodos anteriores
  // método principal que invoca os demais
public static void main (String [] args)
  {
    Quitanda2  xepa = new Quitanda2();

    System.out.println("Frutas  desordenadas");
    xepa.imprime();

    System.out.println("Em  ordem  de preço");
    ComparadorDeFrutas  cpr = new ComparaPr
    xepa.ordena(cpr);
    xepa.imprime();

    System.out.println("Em  ordem  de peso");
    ComparadorDeFrutas  cp = new ComparaPeso();
    xepa.ordena(cp);
    xepa.imprime();

    System.out.println("Em  ordem  alfabética  de nome");
    xepa.ordena(new  ComparaNome  ()); // forma super contraída  de chamar o método ordena
    xepa.imprime();
  }

}
```

```
C:\Fatima\each-usp\Disciplinas\ACH2001-2009\DrJava>java Quitanda2
Frutas desordenadas
Nome: Laranja Peso: 0.2 Preto: 0.5
Nome: Kiwi Peso: 0.1 Preto: 2.0
Nome: Mamão Peso: 1.5 Preto: 1.2
Nome: Pera Peso: 0.3 Preto: 1.7
Nome: Banana Peso: 0.3 Preto: 0.2
Em ordem de preto
Nome: Banana Peso: 0.3 Preto: 0.2
Nome: Laranja Peso: 0.2 Preto: 0.5
Nome: Mamão Peso: 1.5 Preto: 1.2
Nome: Pera Peso: 0.3 Preto: 1.7
Nome: Kiwi Peso: 0.1 Preto: 2.0
Em ordem de peso
Nome: Kiwi Peso: 0.1 Preto: 2.0
Nome: Laranja Peso: 0.2 Preto: 0.5
Nome: Banana Peso: 0.3 Preto: 0.2
Nome: Pera Peso: 0.3 Preto: 1.7
Nome: Mamão Peso: 1.5 Preto: 1.2
Em ordem alfabética de nome
Nome: Banana Peso: 0.3 Preto: 0.2
Nome: Kiwi Peso: 0.1 Preto: 2.0
Nome: Laranja Peso: 0.2 Preto: 0.5
Nome: Mamão Peso: 1.5 Preto: 1.2
Nome: Pera Peso: 0.3 Preto: 1.7
```

Por que interfaces é importante???

- Se objetos fazem referência a interfaces e não a classes:
  - mais fácil alterar as classes de um sistema sem ter que alterar aquelas que as utilizam (se a assinatura do método se mantém igual);
  - fácil implementar *polimorfismo de comportamento*: classes que mudam de comportamento – chaveamento de comportamento pode ser feito durante compilação ou durante execução;
  - desenvolvimento de sistemas grandes, envolvendo muitos programadores – todos obedecem as interfaces – integração posterior mais fácil;
  - eliminação de código repetido.

Programe para as interfaces,

não para as implementações!!!

Observação

- Interfaces Java também podem definir constantes de classe (atributos static final)
  - Na interface você NÃO precisa colocar o static final
- Ex:

```
interface Cores{
    int branco = 0;
    int preto = 255;
}
```

Exercícios

- Exercício 1 da lista 2.

Interfaces

Professoras:
Ariane Machado Lima
Fátima L. S. Nunes