Aula Teórica
Classes Abstratas e Interfaces

Prof. Dirson Santos de Campos

dirson_campos@ufg.br

12/09/2024

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Tópicos

1. **Classes Abstratas**
   Exemplo
2. **Interfaces**
   Exemplo
3. **Interface e Classe Abstratas**
   Exemplo
4. **Comparações**

Classes Abstratas

Classes Abstratas

Vimos em ***Polimorfismo*** que há diversas vantagens de se **referenciar** um objeto a partir de sua *superclasse*.

```
public class Testes {
     public static void main() {
          Animal[] reinoAnimal = new Animal[2];
          reinoAnimal[0] = new Anfíbio("Sapo");
          reinoAnimal[1] = new Ave("Pombo");

          for (int i = 0; i < reinoAnimal.length; i++)
               reinoAnimal[i].mover(10, 8);

          Imóvel algumImóvel = new Apartamento(73, 10000, 4);
          algumImóvel.colocarÀVenda();

          Empresa umaEmpresa = new Empresa();
          Gerente josé = new Gerente("José", umaEmpresa, 4000);
          umaEmpresa.pagarFuncionário(josé);
     }
}
```

Acesso a um vetor de diferentes tipos. Não preciso saber quem é quem, só chamo os métodos que sei que são comuns, por herdarem de Animal.

Acesso a um método polimorficamente. No contexto deste código, não me interessa qual o tipo, só quero colocá-lo à venda.

Parâmetro de função resolvido polimorficamente. Funciona para qualquer funcionário, do Gerente ao Caixa.

Classes Abstratas

Isso mostra como construir uma hierarquia de classes pode ser útil.

Porém, vale pensarmos se faz sentido que possam existir objetos que sejam **instâncias diretas das *superclassses*** que criamos…

```
public class Testes {
     public static void main() {
          Animal ornitorrinco = new Animal("Ornitorrinco");
          / De quê este cara se alimenta? Ele voa?

          Imóvel algumImóvel = new Imóvel(40, 20000);
          / Que tipo de imóvel é esse? Tem quarto e cozinha?

          Empresa umaEmpresa = new Empresa();
          Funcionário joão = new Funcionário("João", umaEmpresa, 2500);
          / Qual o papel deste cara na empresa?
     }
}
```

Classes Abstratas

Uma classe abstrata é uma classe que não pode ser instanciada - não pode ser diretamente utilizada para criar objetos.

O propósito da criação de uma classe abstrata é de fornecer uma superclasse apropriada para que outras classes utilizem como base (herança).

Seus métodos podem ter implementação ou podem ser abstratos também (sem implementação).

Classes Abstratas

Para declararmos uma ***classe abstrata***, usamos mais uma vez um *modificador*:

```
public abstract class
NomeDaClasseAbstrata
```

Com esta simples mudança, a classe deixa de poder ser instanciável, ou seja, não podemos construir objetos dela. Ou seja:

```
NomeDaClasseAbstrata meuObjeto = new NomeDaClasseAbstrata(); // Erro! Não compila!!
```

Podemos, porém, criar objetos de suas subclasses, desde que estas não sejam abstratas também. Chamamos classes não abstratas que herdam de classes abstratas de ***classes concretas***.

Note que mesmo não podendo ser instanciada, uma classe abstrata pode ter um construtor, já que a lógica de inicialização dos atributos herdados ainda pode ser necessária.

Exemplo Classes Abstratas

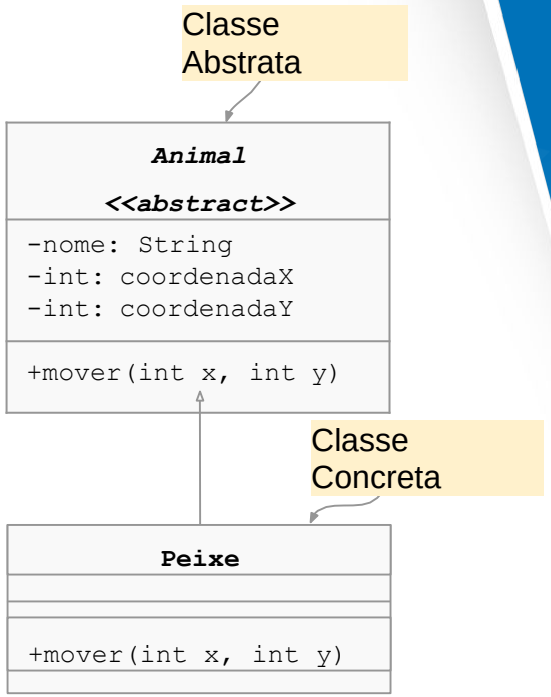
Na representação UML, uma classe abstrata tem seu nome em *itálico*. Para maior clareza, às vezes tem o modificador `<<abstract>>` acompanhando o nome.

Animal.java

```
public abstract class Animal
      {  private String nome;
      private int coordenadaX =
      0;  private int coordenadaY
      = 0;

      public Animal (String nome) { this.nome = nome; }
      // Método abstrato. Note que não há corpo da função, só o cabeçalho.
      public abstract void mover(int x, int y);
      } //... outros métodos como setCoordenadas
}
```

Peixe.java

```
public class Peixe extends Animal {
      public Peixe (String nome) { super(nome);
      }  public void mover(int x, int y) {
            setCoordenadas(x, y);
            System.out.println("Saí
            nadando!");
      }
}
```



Classes Abstratas

Em **Classes Abstratas** , também podemos declarar **Métodos Abstratos**.

Métodos abstratos são métodos sem implementação definida (sem corpo da função, apenas o cabeçalho) e que exigem uma implementação definida nas subclasses.

A técnica de especificar métodos abstratos permite que o projetista decida **quais** são os comportamentos que as subclasses devem ter ... mas sem determinar **como** tais comportamentos serão implementados … é uma delegação de responsabilidades, já que a classe que herda é obrigada a implementar o método para ser compilada.

Classes Abstratas

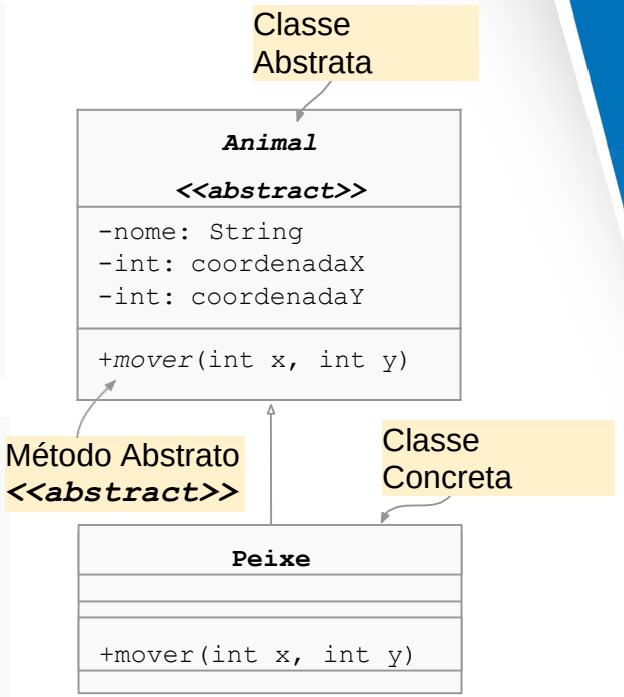
Na representação UML, o método abstrato também tem seu nome em *itálico*.

Animal.java

```
public abstract class Animal {
	private String nome;
	private int coordenadaX = 0;
	private int coordenadaY = 0;

	public Animal (String nome) { this.nome = nome; }

	/ Método abstrato. Note que não há corpo da função, só o cabeçalho.
	public abstract void mover(int x, int y);
}
```

Peixe.java

```
public class Peixe extends Animal {
	public Peixe (String nome) { super(nome); }
	/ Implementação obrigatória do método abstrato!
	public void mover(int x, int y) {
		setCoordenadas(x, y);
		System.out.println("Saí nadando!");
	}
}
```



Interfaces

Interfaces

- Durante a criação de software, é comum que mais de um grupo de programadores trabalhe no mesmo projeto
- É fundamental estabelecer um “contrato” entre os grupos, de forma que os programas possam se comunicar

Interfaces

- Não importa como a implementação será feita
  - O importante é saber a definição do contrato
  - Garante que o software desenvolvido por um grupo se comunica com o outro através deste “contrato”
- Em POO, as *interfaces* fornece esse contrato

Interfaces

Neste contexto uma interface é um contrato que define um conjunto de métodos públicos vazios que devem ser codificados nas subclasses que implementarem a interface. É *como se fosse* uma classe 100% abstrata.

Uma classe pode ter uma única superclasse (herança simples), mas pode implementar várias interfaces.

Uma interface não pode definir métodos construtores.

Interfaces

Usando da hierarquia de herança, temos estabelecido relações do tipo "*é um*" (`Peixe` *é um* `Animal`, `Apartamento` *é um* `Imóvel`, `Gerente` *é um* `Funcionário`).

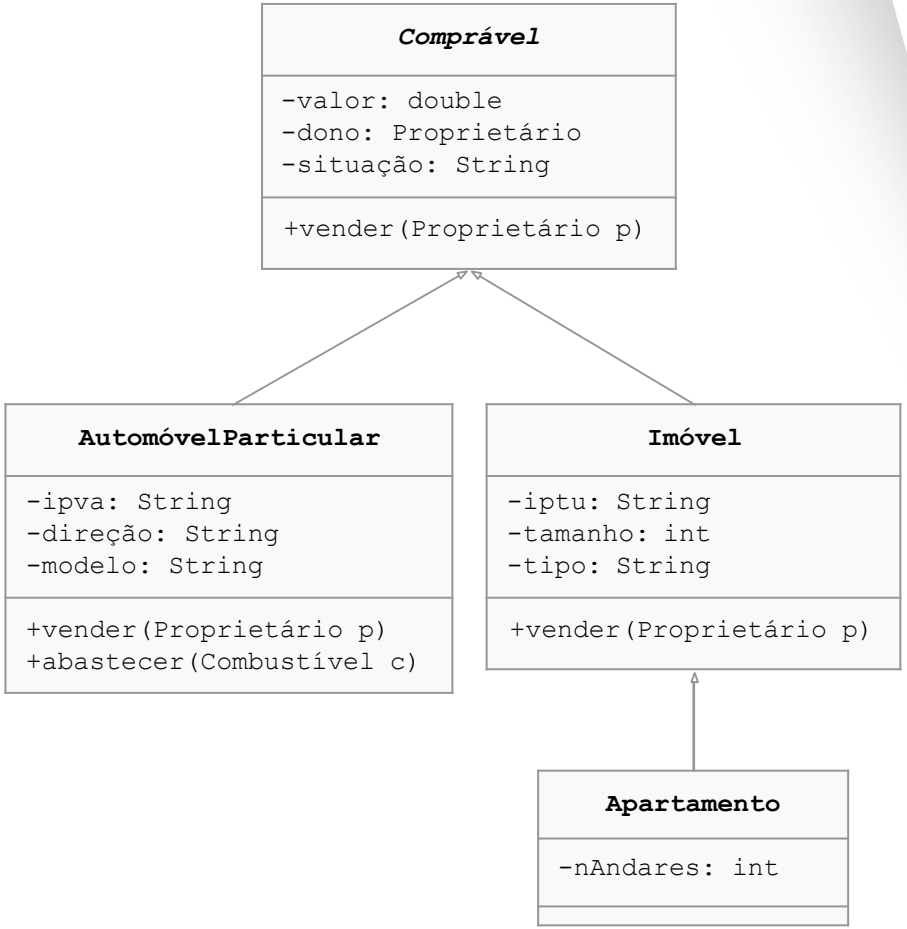
Há um tipo de relação menos forte mas que tem uma similaridade e que também pode ser muito interessante de se determinar: "*faz algo*".

Por exemplo, no mesmo sistema da classe `Imóvel`, podem existir outras coisas que podem ser colocadas *à Venda*, sem necessariamente *serem* Imóveis. Então poderíamos pensar em uma classe mais genérica que "faz" a parte de imóvel que diz respeito à venda. Algo como uma classe "`Comprável`", que tivesse somente os métodos de venda. Deste modo, uma classe `AutomóvelParticular`, também poderia herdar desta classe.

Interfaces

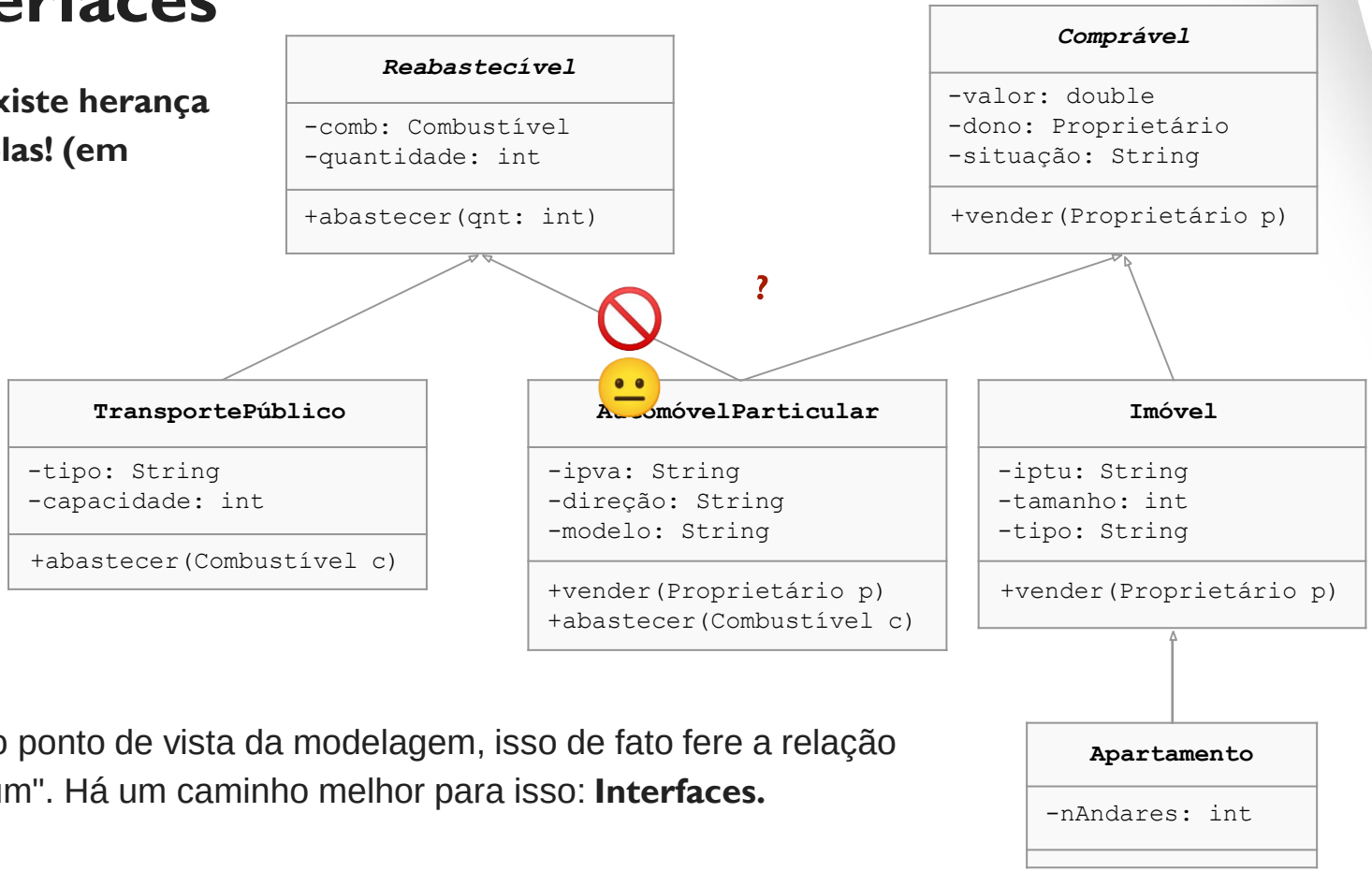
Poderíamos inclusive ter uma classe abstrata, para que cada classe filha ficasse responsável por determinar detalhes internos dessa compra…

Agora, e se neste mesmo sistema, surgisse uma entidade que, assim como o `AutomóvelParticular`, precisa de abastecimento, por exemplo um `Metrô`. Este `Metrô`, porém, não tem como ser comprado, é concessão do estado.



Interfaces

Não existe herança múltiplas! (em Java)



Mas do ponto de vista da modelagem, isso de fato fere a relação de "é um". Há um caminho melhor para isso: **Interfaces.**

Interfaces

Considere então estes dois exemplos de interfaces. Note como, diferentemente de classes, não há atributos nem construtor. Apenas assinaturas indicando os métodos que devem ser implementados:

Comprável.java

```
public interface Comprável {
     public void vender(Proprietário
     novoDono);  public void colocarÀVenda();
}
```

Reabastecível.java

```
public interface Reabastecível {
     public void abastecer(int quantidade);
}
```

Interfaces

Uma versão da classe `Imóvel` que implementa esta interface:

Imóvel.java

```
public class Imóvel implements Comprável {
      //... Atributos da classe ...
      //... Atributos da classe ...

      public Imóvel(/*... Parâmetros ... */) { /*... Lógica do construtor ... */}

      public void vender(Proprietário novoDono) {
            / Lógica para vender imóveis, como checar se o iptu está em dia…
      }
      public void colocarÀVenda() {
            / Coloca atributo situação desta classe como "à venda", dentre outras
            coisas…
      }

      / … Todos os outros métodos da classe
}
```

Interfaces

Uma versão da classe `AutomóvelParticular` que implementa duas interfaces:

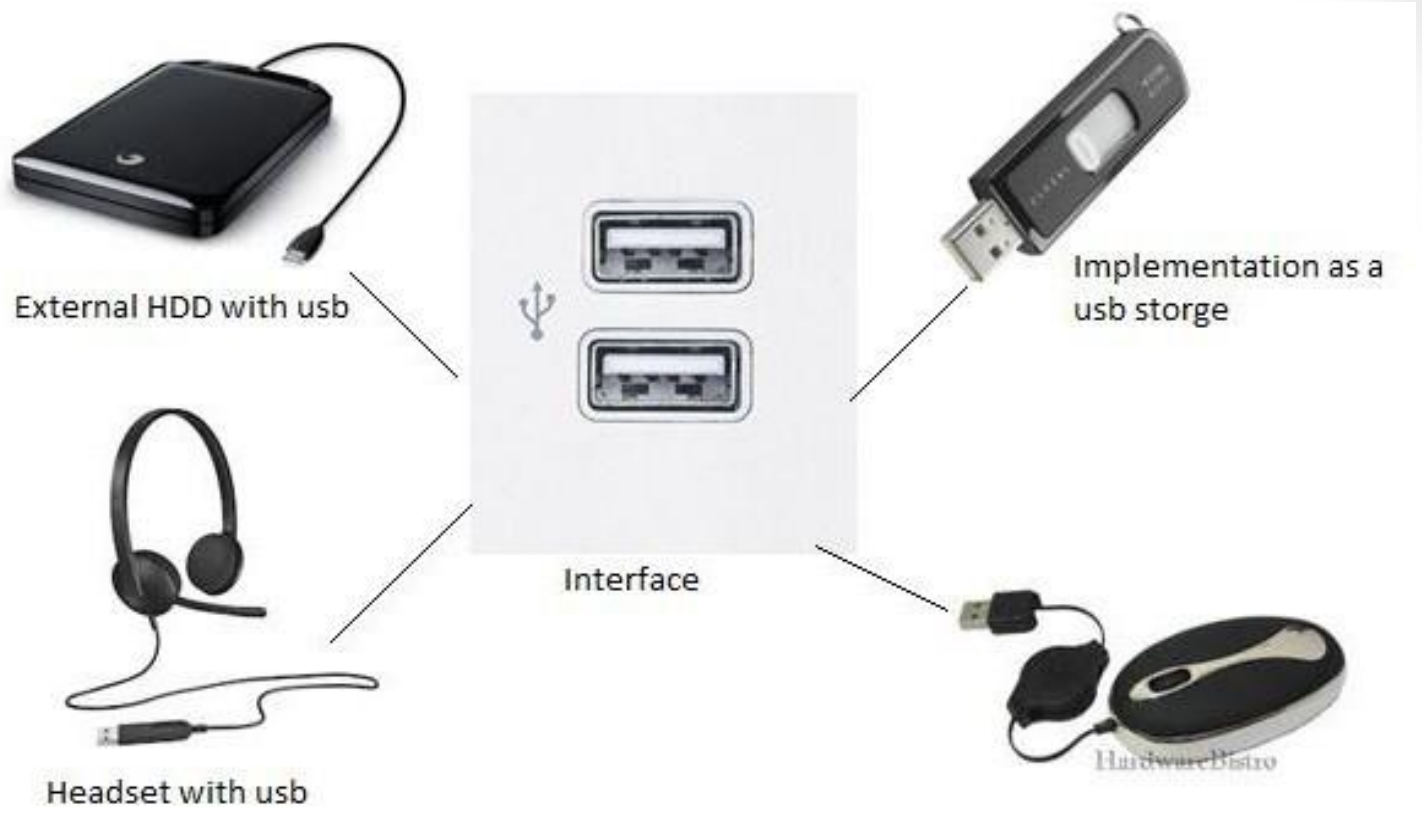
Imóvel.java

```
public class AutomóvelParticular implements Comprável, Reabastecível {
      //... Atributos da classe ...
      //... Atributos da classe ...

      public AutomóvelParticular(/*... Parâmetros ... */) { /*... Lógica do construtor ...

      */}  public void vender(Proprietário novoDono) {
            / Lógica para vender automóvel, como checar se o ipva está em dia…
      }
      public void colocarÀVenda() {
            / Coloca atributo situação desta classe como "à venda", dentre outras coisas…
      }
      public void abastecer(Combustível c) {
            / Lógica para abastecer o automóvel
      }
}     / … Todos os outros métodos da classe
```

Interfaces em *Hardware - USB*

- O padrão USB é um exemplo
- Está presente em diferentes dispositivos
- As empresas que criam dispositivos que se conectam através de USB, só precisam conhecer o protocolo (mensagens trocadas) de uma conexão USB

Interfaces em *Hardware - USB*



Interfaces implementadas em bibliotecas da Linguagem Java

Há diversas Interfaces implementadas pela java.io:

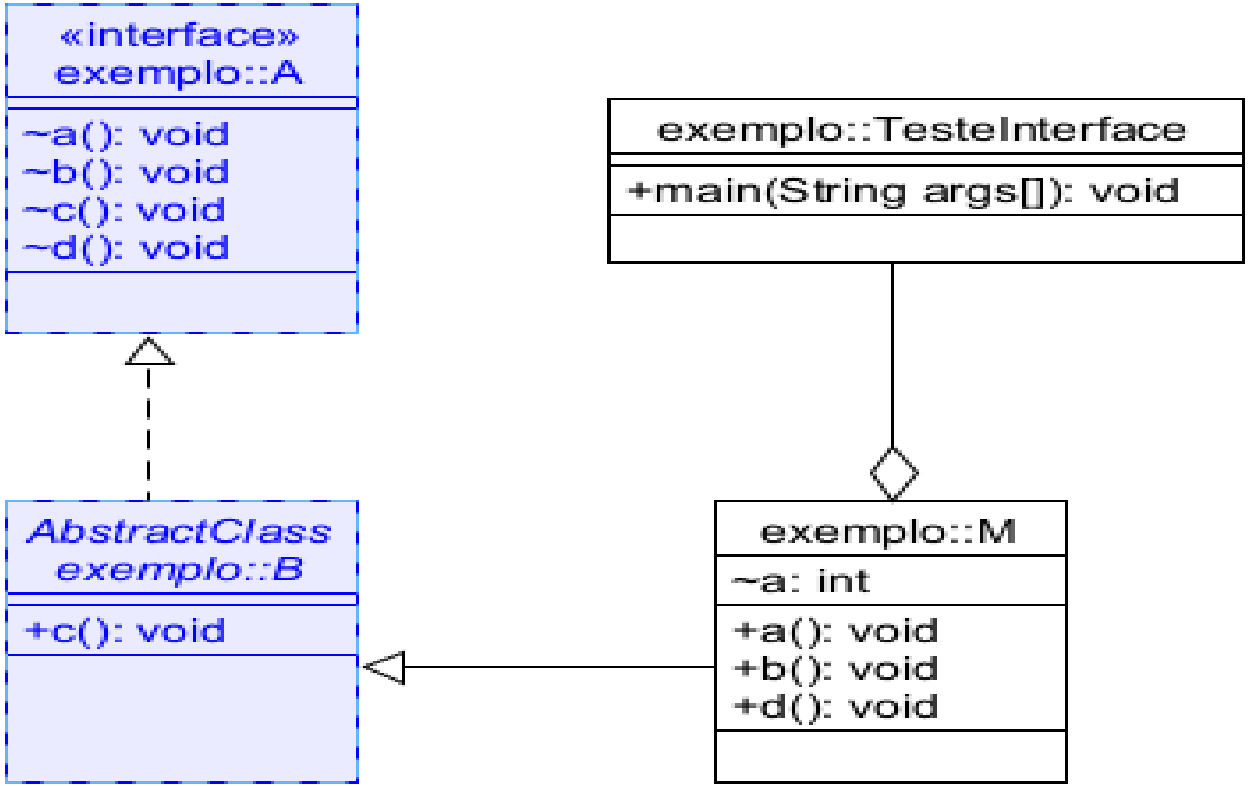
- Comparable;
- Serializable;
- Runnable;
- ActionListener, KeyListener, MouseListener;

Interfaces e Classes Abstratas - Exemplo

Classe Abstrata Implementa Interface

- Uma classe que implementa uma interface deve, necessariamente, implementar todos os métodos abstratos
- Se a classe que implementa a interface for abstrata, essa exigência desaparece
  - Alguns métodos podem ser implementados e outros não
- Os métodos que ainda não foram definidos na classe abstrata deve ser definido na subclasse desta classe abstrata

Exemplo de Uso de Interface e Classe Abstrata



Exemplo de Interface e Classe Abstrata

```
package exemplo;

public interface A {
	void a();
	void b();
	void c();
	void d();
}
```

Exemplo de Interface e Classe Abastrata


```
package exemplo;

public abstract class B implements A {
public void c(){
      System.out.println("Eu sou um C");
   }
}

}
```

Exemplo de Interface e Classe Abastrata

```
package exemplo;

public class M extends B {
  public void a(){
        System.out.println("Eu sou um a");
  }
  public void b(){
        System.out.println("Eu sou um b");
  }
  public void d(){
        System.out.println("Eu sou um d");
  }
}
```

Exemplo de Interface e Classe Abstrata

```
package exemplo;

public class TesteInterface {

public static void main(String args[]){
        A a = new M();
        a.a();
        a.b();
        a.c();
        a.d();
        }
}
}
```

Comparações

Classes Abstratas x Interfaces

- Ambos são projetos de classes que não podem ser instanciados;
- Ambas exigem a implementação dos métodos abstratos nas subclasses;
- Ambas podem conter métodos com ou sem implementação
    - Antes do Java 8, interfaces não podiam conter métodos com implementação (default ou static)

Classes Abstratas x Interfaces

Classes Abstratas	Interfaces
Pode ter qualquer tipo de instância ou variáveis estáticas, mutáveis ou imutáveis.	Só pode ter variáveis estáticas finais. Uma interface nunca pode alterar seu próprio estado.
Uma classe pode herdar (palavra-chave **extends**) apenas uma classe abstrata.	Uma classe pode implementar múltiplas interfaces.
Somente pode ser herdada (palavra-chave **extends**).	Pode ser implementada (palavra-chave **implements**.) Uma interface também pode fazer **extends** de interfaces.

Classes Abstratas x Interfaces

Classes Abstratas	Interfaces
Pode ter método construtor.	Não pode ter método construtor.
Pode ter qualquer tipo de método.	Pode ter métodos abstratos. Pode ter métodos padrão e estáticos **(introduzidos a partir do Java 8).** Pode ter métodos privados com a implementação **(introduzido a partir do Java 9).**

Classes Abstratas x Interfaces

- **Qual utilizar?**
  - Depende da aplicação
  - Em geral, interfaces são utilizadas por classes que não tem relação entre si
    - Serializable, Clonable, Comparable
    - Não existe uma relação forte (herança) entre as classes
  - Se há a necessidade de oferecer atributos, interfaces não serão úteis
    - Com herança, os atributos serão herdados
    - Naturalmente existe uma dependência maior entre as classes