

ECM251 - Linguagens I

Teoria - Classes Abstratas e Interfaces

Prof. Murilo Zanini de Carvalho

Prof. Tiago Sanches da Silva

Antes de começar!

Clone seu repositório do Github

- Lembre-se sempre antes de iniciar uma aula, clonar seu repositório remoto e realizar as atividades nele.
- Para cada atividade desenvolvida, criar um novo diretório.



GitHub

Retirado de
(https://miro.medium.com/max/4000/0*MZMI76wKo2FQLqG0.png), em 07/03/2021

Abstração

- Capacidade de representar elementos (físicos com ou não) por suas características e comportamentos.

Encapsulamento

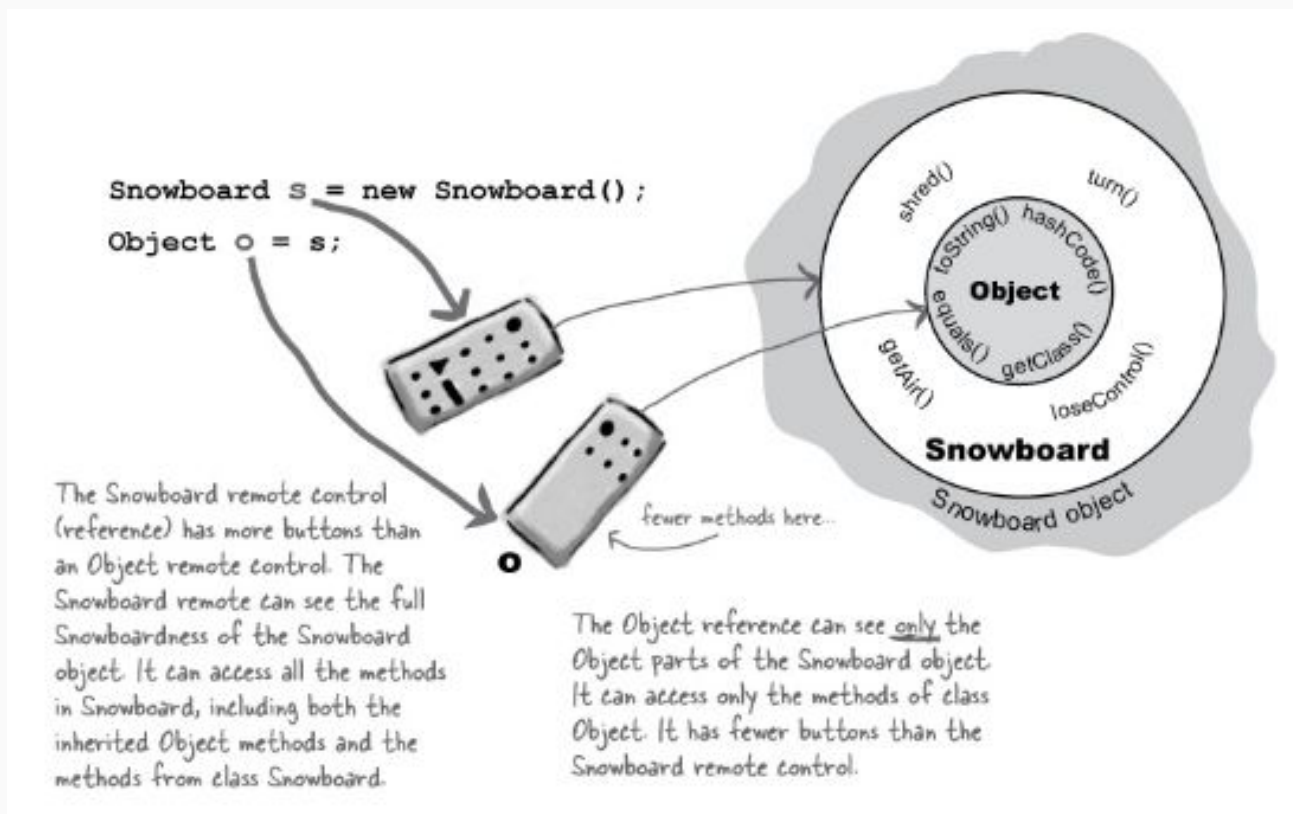
- Isolar a forma como a abstração é realizada, fornecendo uma interface de acesso a ela.

Herança

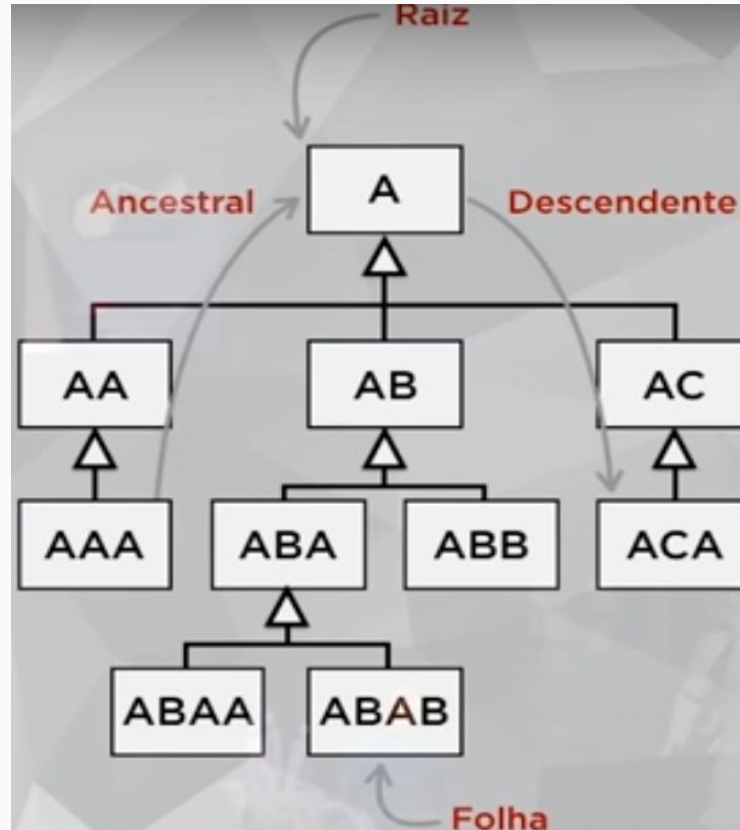
- Transmissão de características de um elemento geral para elementos mais específicos.

Polimorfismo

- Possibilidade de executar o método correto de acordo com a situação/referência utilizada.

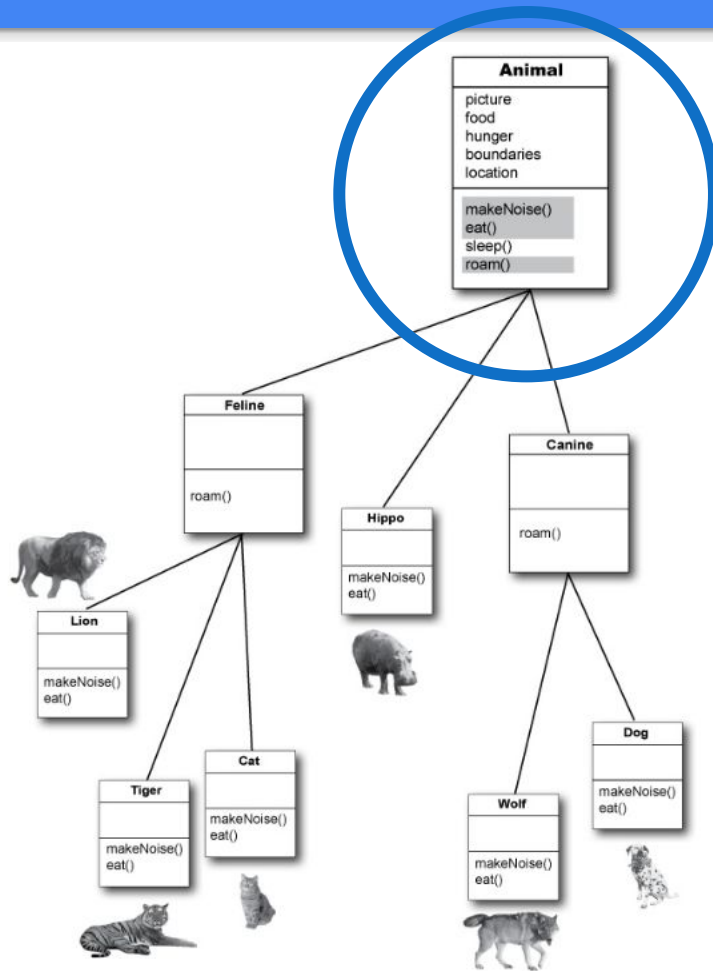


Especialização

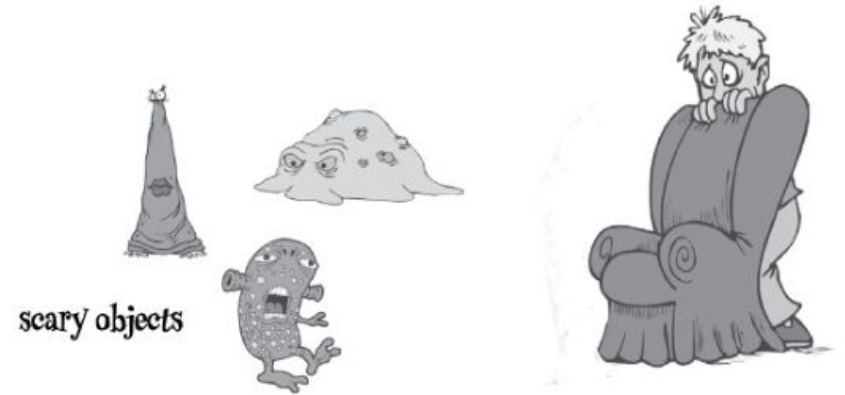


Generalização

Herança



What does a new `Animal()` object *look* like?



Retirado de: "Head First! Java"

Classes Abstratas

Definição

Uma Classe Abstrata é desenvolvida para representar entidades e conceitos abstratos. A classe abstrata é sempre uma superclasse que não possui instâncias (ou melhor, não pode ser instanciada).

Ela define um modelo para determinada funcionalidade e geralmente fornece uma implementação incompleta – a parte genérica – dessa funcionalidade.

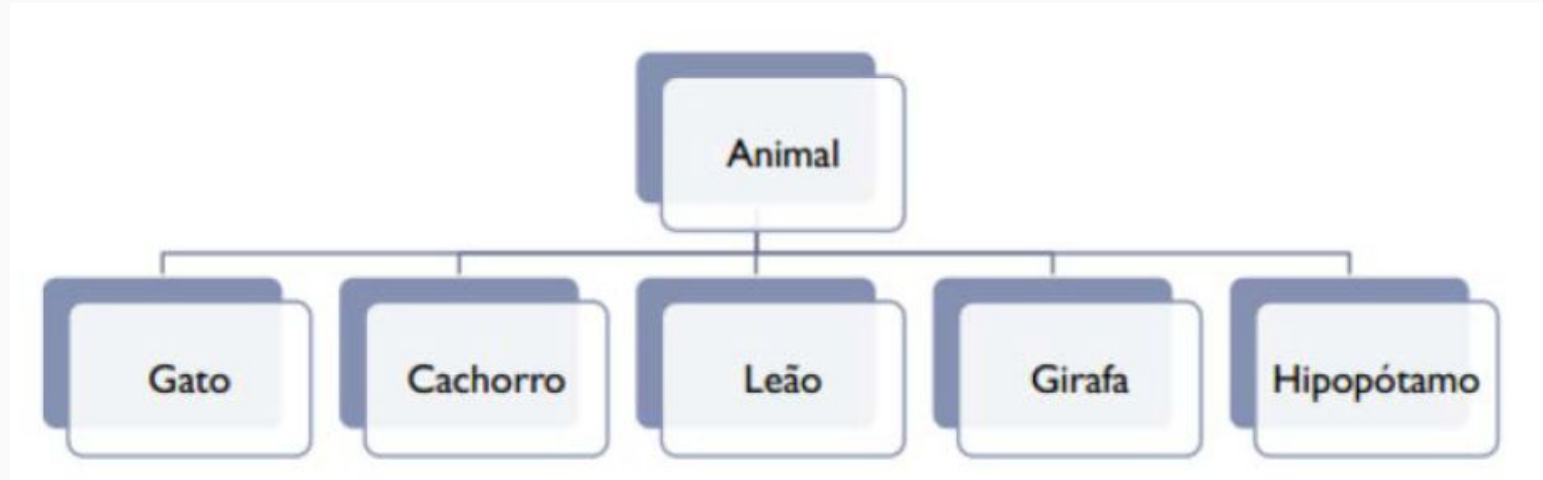
Cada uma das classes derivadas da classe abstrata, completa a funcionalidade dessa classe abstrata, adicionando um comportamento específico (especialização).

Nomenclatura

Classes Abstratas: Conceitos e ideias, classes que não possuem instâncias diretas, porém suas descendentes podem ter.

Classes Concretas: Classes que possuem uma implementação completa e podem ser instanciadas.

Exemplos de Classes Abstratas



Não existe instância da classe **Animal**, apenas das classes especializadas.

Exemplos de Classes Abstratas



Métodos Abstratos

Métodos Abstratos

Podemos definir um método como abstrato, isso implica, **obrigatoriamente**, que esse método deverá ser implementado nas classes descendentes concretas.

Em Java, podemos declarar um método e uma classe como abstratos utilizando a palavra chave ***abstract***.

Uma classe abstrata pode possuir métodos concretos e abstratos.

Se uma classe possui ao menos um método abstrato, então a classe precisa ser abstrata!

Exemplo Implementação

```
abstract class Animal {  
    public abstract void comer();  
}  
  
class Lobo extends Animal {  
    @Override  
    public void comer() {  
        System.out.println("Eu me alimento como um lobo!");  
    }  
}  
  
class Peixe extends Animal {  
    @Override  
    public void comer() {  
        System.out.println("Eu me alimento como um peixe!");  
    }  
}  
  
public class Teste {  
    public static void main(String[] args) {  
        Animal a = new Lobo();  
        Animal b = new Peixe();  
  
        a.comer();  
        b.comer();  
    }  
}
```

Classes Abstratas

Classe Abstrata

Não pode ser instanciada.
Só pode servir como
progenitora.

Método Abstrato

Declarado, mas não
implementado na
progenitora.

Classe Final

Não pode ser herdada por
outra classe.
Obrigatoriamente folha.

Método Final

Não pode ser sobrescrito
pelas suas sub-classes.
Obrigatoriamente herdado

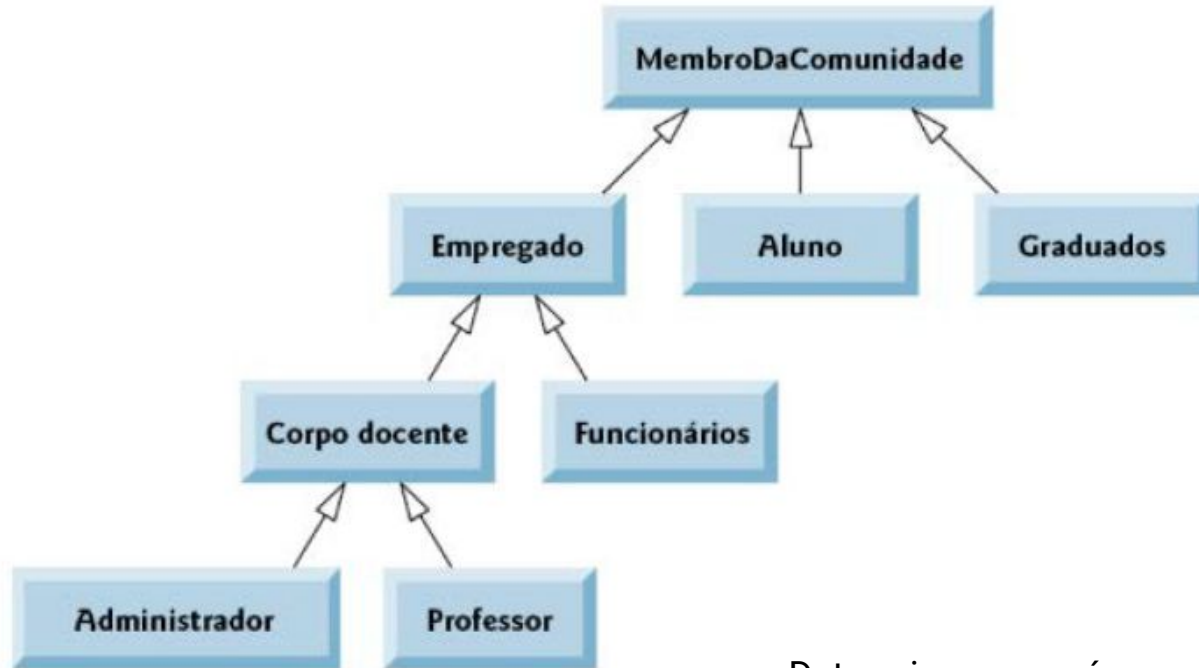
- Toda classe que possui métodos abstratos é automaticamente abstrata e deve ser declarada como tal;
- Uma classe pode ser declarada abstrata mesmo se ela não possuir métodos abstratos;
- Uma classe abstrata não pode ser instanciada;
- Uma classe que é descendente direta de uma classe abstrata pode ser instanciada somente se:
 - I. Sobrescrever e fornecer implementação para cada um dos métodos abstratos de sua superclasse;
- **Se uma subclasse de uma classe abstrata não implementar todos os métodos abstratos que ela herda, então a subclasse também é abstrata.**

Por que não utilizar herança comum apenas?

A única diferença é que não podemos instanciar a classe abstrata, e se tratando de conceitos abstratos, utilizar classes abstratas adiciona coerência e consistência ao sistema.

A decisão de transformar ou não uma classe em abstrata depende do domínio do seu problema!

Exercício



Determinar o que é concreto e o que é abstrato.

Interface de um objeto

O que é interface de um objeto?

A interface de um objeto é um conjunto de operações públicas que ele pode realizar.

O objeto da classe Lâmpada, por exemplo, tem como interface as operações:

- acender()
- apagar()

Os objetos se comunicam através desses métodos públicos, qualquer outra operação é considerada inválida.

(Entidade) Interface

Interface

A entidade **Interface** então determina métodos obrigatórios para a classe que deseja implementá-la.

Assim como classes abstratas uma interface também define métodos que deverão ser implementados por classes que venham a implementar a interface.

Também como uma classe abstrata, uma interface não pode ser instanciada.

Interface

O nível de acesso de todos os métodos declarados em uma Interface é **public**. Faz sentido?

SIM! Já que Interface (entidade) define parte da interface de um objeto!

Interface

Para criar uma interface basta utilizar a palavra-chave **Interface**, e no corpo definir as assinaturas dos métodos.

Para implementar a **Interface** em um classe, basta utilizar a palavra-chave **implements**.

```
public interface Figura {  
    public abstract double calcularArea( );  
}
```

```
public class Quadrado implements Figura {  
    double lado;  
    public Quadrado(double lado) {  
        this.lado = lado;  
    }  
  
    public double calcularArea( ) {  
        double area = 0;  
        area = lado * lado;  
        return area;  
    }  
}
```



```
public class Circulo implements Figura {  
    double raio;  
  
    public Circulo (double raio) {  
        this.raio = raio;  
    }  
  
    public double calcularArea( ) {  
        double area = 0;  
        area = 3.14 * raio * raio;  
        return area;  
    }  
}
```

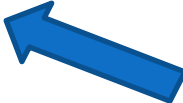
Já que as classes **Quadrado** e **Circulo** implementam a interface **Figura**, logo precisam obrigatoriamente implementar o método **calcularArea()**.

Interface é como um contrato, em que quem assina se responsabiliza por implementar os métodos definidos na Interface (cumprir o contrato).

Uma Interface expõe **o que** o objeto deve fazer, e **não como** ele deve fazer. Como o objeto irá fazer é definido na implementação dessa interface, por parte da classe que a implementará.

Interface

```
public class Main {  
  
    public static void main(String[] args) {  
  
         Figura f1 = new Quadrado(4);  
         Figura f2 = new Circulo(2);  
  
        System.out.println("Área da Figura 1 é: "  
            + f1.calcularArea( ) + "\n"  
            + "Área da Figura 2 é: "  
            + f2.calcularArea( ));  
  
    }  
}
```



```
new Quadrado(4);  
new Circulo(2);
```

Observe que uma **interface** não pode ser **instanciada**, mas é possível um objeto, declarado como sendo do tipo definido por uma interface, **receber** objetos de classes que **implementam tal interface**.

A variável f1 é uma referência para um objeto que implementa Figura, portanto ele só enxergará os métodos descritos nessa Interface.



Interfaces vs Classes Abstratas

Interfaces vs Classes Abstratas

- Classes abstratas **podem conter métodos não abstratos**, isto é, que contêm implementação e que podem ser herdados e utilizados por instancias das subclasses.
- Interfaces não podem conter nenhum método com implementação, **TODOS OS MÉTODOS SÃO IMPLICITAMENTE abstract E public** e não possuem corpo.
 1. Os modificadores public e abstract podem ser omitidos sem problemas.

Interfaces vs Classes Abstratas

- Um diferença essencial entre classes abstratas e interfaces em Java é que uma subclasse somente pode herdar de uma única classe (abstrata ou não) enquanto qualquer classe **pode implementar várias interfaces** simultaneamente.
- Interfaces são, portanto, um mecanismo simplificado de implementação de “herança múltipla” em Java, que possibilita que mais de uma interface determine os métodos que uma classe herdeira deve implementar.

Múltiplas Interfaces

Uma classe pode implementar mais de uma interface, assumindo assim vários comportamentos.

```
public interface Impressora {  
    public void imprime(Documento d);  
}
```

```
public interface Fax {  
    public void transmite(Documento d);  
}
```

```
public class FaxImpressora implements Impressora, Fax {  
    public void imprime(Documento d) {  
        ...  
    }  
    public void transmite(Documento d) {  
        ...  
    }  
}
```

Múltiplas Interfaces

```
public interface Radio {  
  
    public void liga();  
  
    public void desliga();  
  
    public void trocaEstacao(  
        int frequencia);  
  
}
```

```
public interface Relogio {  
  
    public String getHoras();  
  
}
```

```
public class RadioRelogio  
    implements Radio, Relogio {  
  
    public void liga() {  
        // Implementação  
    }  
  
    public void desliga() {  
        // Implementação  
    }  
  
    public void trocaEstacao  
        (int frequencia) {  
        // Implementação  
    }  
  
    public String getHoras() {  
        // Implementação  
        return null;  
    }  
  
}
```


IMPORTANTE!

Consideração Importante

Aprender a sintaxe de como cria uma Interface e como implementa através de uma classe é a parte fácil.

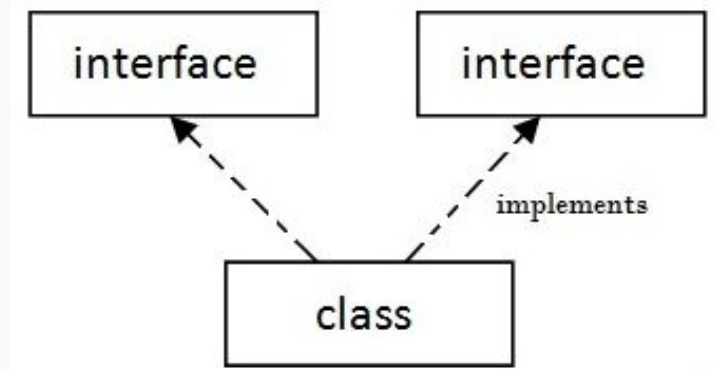
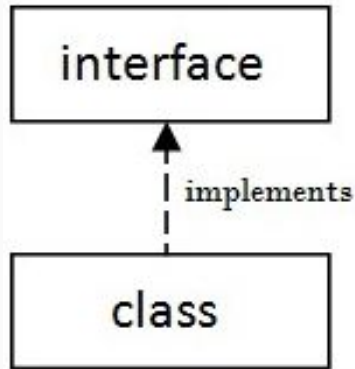
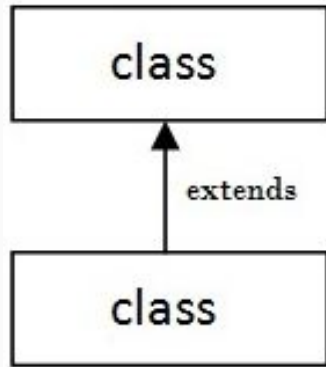
Muitas vezes o difícil é entender o conceito e as vantagens de sua utilização.

Uma interface estabelece uma espécie de **contrato** que é obedecido pelas classes que a implementam.

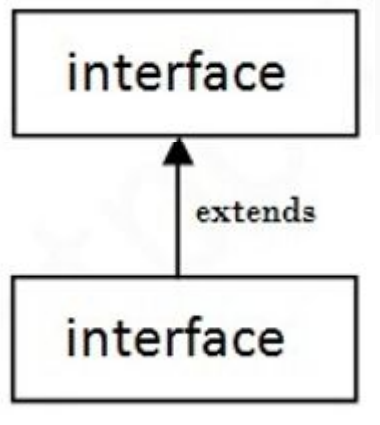
Sendo assim, quando uma classe implementa uma interface, garante-se que todas as funcionalidades especificadas pela interface serão oferecidas pela classe.

Relações Possíveis Entre Classes e Interfaces

Relações Possíveis Entre Classes e Interfaces



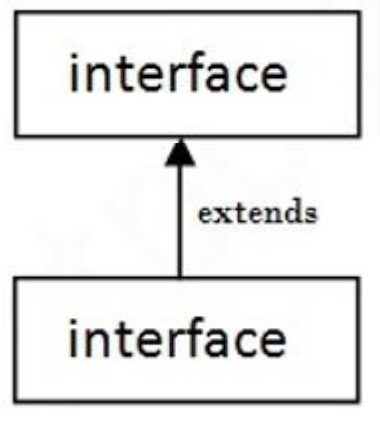
Relações Possíveis Entre Classes e Interfaces



É possível que uma interface herde outra interface, mas no que isso implica?

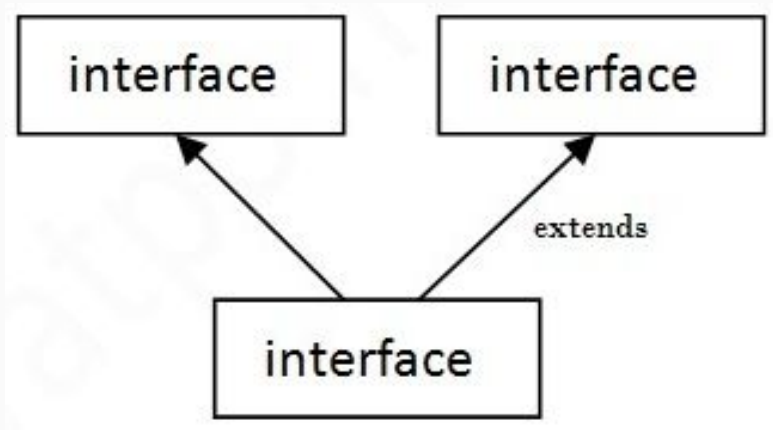
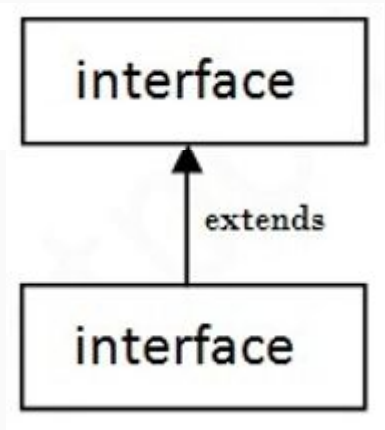
Basicamente, que para uma classe implementar a interface herdeira, deve implementar também a Interface pai.

Relações Possíveis Entre Classes e Interfaces



```
interface Printable{  
    void print();  
}  
  
interface Showable extends Printable{  
    void show();  
}  
  
class TestInterface4 implements Showable{  
    public void print(){System.out.println("Hello");}  
    public void show(){System.out.println("Welcome");}
```

Relações Possíveis Entre Classes e Interfaces



Interfaces no Java 8

Desde Java 8, podemos ter corpo de método na interface. Mas precisamos torná-lo método padrão.

```
interface Drawable{  
    void draw();  
    default void msg(){System.out.println("default method");}  
}  
  
class Rectangle implements Drawable{  
    public void draw(){System.out.println("drawing rectangle");}  
}  
  
class TestInterfaceDefault{  
    public static void main(String args[]){  
        Drawable d=new Rectangle();  
        d.draw();  
        d.msg();  
    }}  

```


Recapitulando

Recapitulando (Adaptado de *Head First*):

- Fazer uma **CLASSE** quando ela não estender ninguém (não herdar de ninguém a não ser de *Object*) e quando a classe não passar no teste É-UM para nenhuma outra classe.
- Fazer uma **SUBCLASSE** (herdar de outra classe) quando você precisar fazer uma versão mais específica dela, sobrescrevendo ou adicionando novos métodos.
- Usar **CLASSES ABSTRATAS** quando for definir um modelo para um grupo de subclasses e você possui alguma implementação de código comum que todas as subclasses devem possuir. Torne uma classe abstrata quando desejar garantir que ninguém vai criar instâncias (criar objetos) dela.
- Use uma **INTERFACE** quando você quer definir uma funcionalidade que algumas de suas classes podem assumir, mesmo sem implementar aquele árvore de herança.

Perguntas?



Retirado de
(<https://cdn-icons-png.flaticon.com/512/1268/1268705.png>), em 02/03/2022