Interfaces e Classes Abstratas

Prof. Dr. Lucas C. Ribas

Disciplina: Programação Orientada a Objetos

Departamento de Ciências de Computação e Estatística

unesp
UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO

Agenda

- Sumário
- Conceito de interface
- Interfaces em Java
- Atualizações no Java 8
- Classes abstratas
- Classes abstratas *vs* interfaces

unesp

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
- 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

unesp

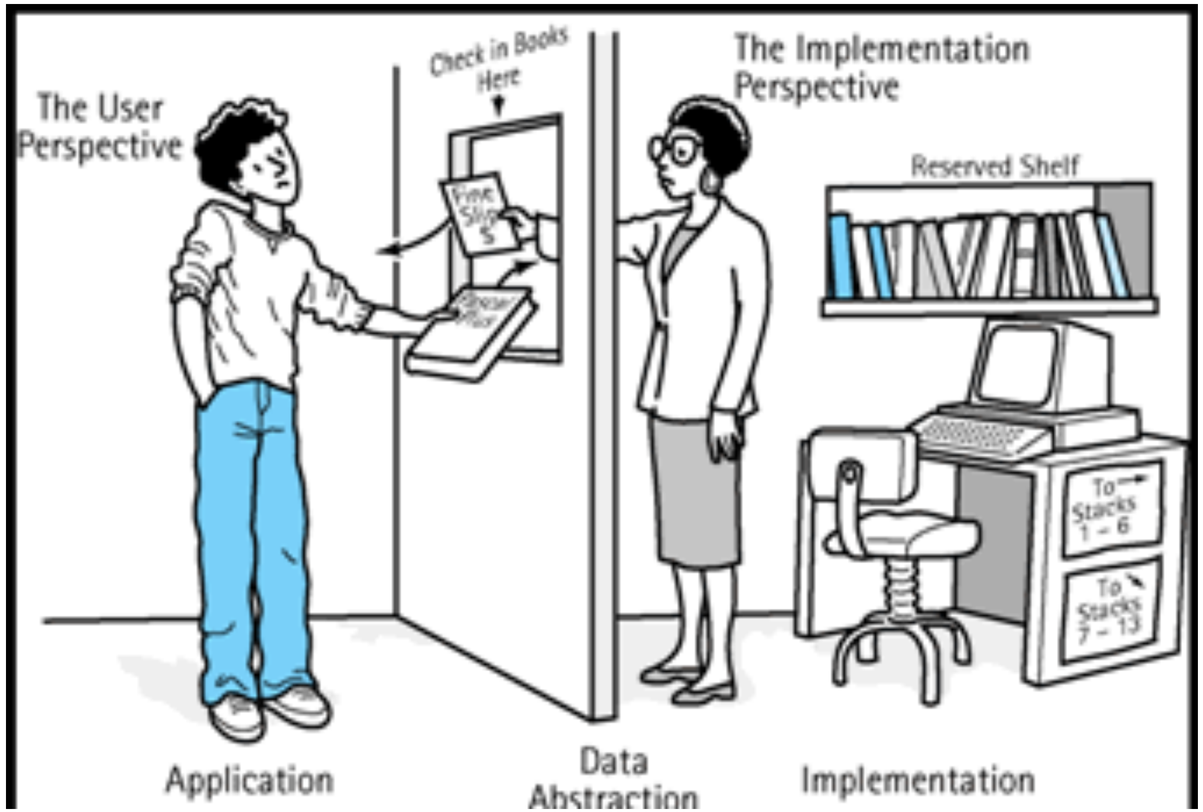
Interfaces

- Imagine que no futuro todos os carros fossem controlados por software (sem motorista)
- As indústrias de carro devem se reunir e definir um contrato, a partir do qual qualquer empresa de software pode criar um controlador de carro
  - Servirá para qualquer carro
  - Não depende dos detalhes de implementação
  - As funcionalidades dos carros podem ser aprimoradas, sem contudo ser preciso mudar os softwares controladores

unesp

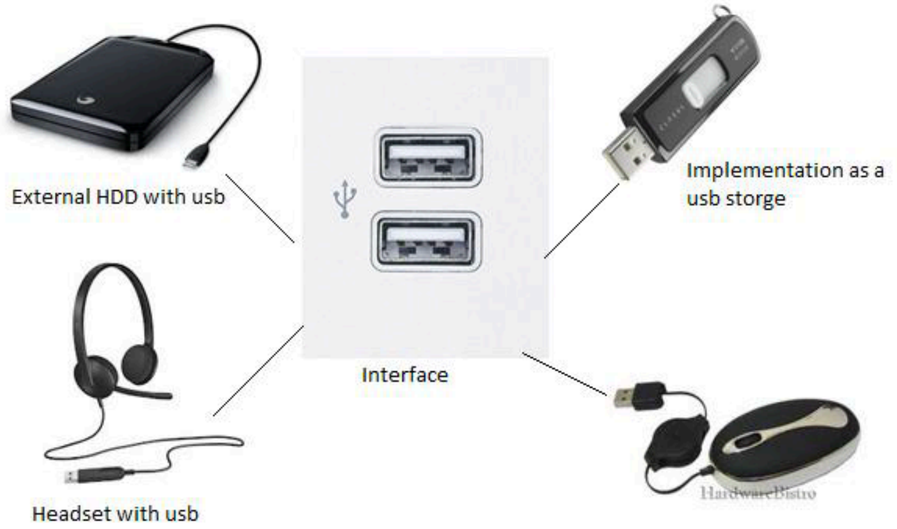
Interfaces

Interfaces


The User Perspective
Check in Books Here
The Implementation Perspective
Reserved Shelf
To Stacks 1 - 6
To Stacks 7 - 13
Application
Data Abstraction
Implementation

Interfaces

- 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 Java, *interfaces* são um tipo especial de referência, parecido com classes
  - NÃO podem ser instanciadas
  - Assim como classes, interfaces podem ser public ou *package-private*
  - Só podem conter campos constantes
    - Implicitamente são public, static e final
  - Os métodos são definidos apenas pela sua assinatura
    - Implicitamente são public e abstract
- Só podem ser **implementadas** por classes
- Podem ser **herdadas** por outra interface

unesp

Interfaces

- Definida em conjunto pelas montadoras de carro

```
public interface OperarCarro {
    // declarações de constantes, se houver
    // assinaturas de métodos
    // Direção é um enum com valores ESQUERDA e DIREITA
    int girar(Direcao direcao, double raio,
    double velocidadeInicial, double velocidadeFinal);

    int mudarFaixa(Direcao direcao, double velocidadeInicial,
    double velocidadeFinal);

    int sinalizarGiro(Direcao direcao, boolean sinalLigado);
    int getRadarFrente(double distanciaAteCarro, double velocidadeDoCarro);
    int getRadarTraseiro(double distanciaAteCarro, double velocidadeDoCarro);
    // mais assinaturas de métodos
}
```

unesp

Interfaces

- Para usar uma interface, ela deve ser implementada
  - Todos os métodos precisam ser implementados na classe (neste caso, pela montadora BMW)

```
public class OperarBMW760i implements OperarCarro { // as assinaturas de métodos de OperarCarro,
com implementação
    // por exemplo:
    int sinalizarGiro(Direcao direcao, boolean sinalLigado) {
        // código para ligar as luzes indicadoras de direção ESQUERDA da BMW
        // código para desligar as luzes indicadoras de direção ESQUERDA da BMW
        // código para ligar as luzes indicadoras de direção DIREITA da BMW
        // código para desligar as luzes indicadoras de direção DIREITA da BMW
    }
    // outros membros, conforme necessário – por exemplo, classes auxiliares
    // não visíveis aos clientes da interface
}
```

unesp

Interfaces

- No exemplo anterior, cada carro deverá implementar a interface **OperarCarro**
- Chevrolet, Toyota, BMW, etc. implementarão ao seu modo esses métodos, de acordo com o carro
- Porém, a interface **OperarCarro** estabelece o contrato entre as empresas de software e a montadora de carros
  - Controladores sabem quais métodos podem utilizar (interface) para controlar um carro

unesp

Definindo Interfaces

- Assim como classes, as interfaces podem ser definidas como public ou *package-private* (ausente)
- Uma interfaces pode estender (herdar) várias interfaces

```
public interface Interface0 extends Interface1, Interface2, Interface3
```

- Uma classe pode implementar várias interfaces
- Supre a falta de herança múltipla

```
public class MyClass implements Interface0, Interface3
```

unesp

Usando Interfaces

- Suponha que queiramos comparar objetos
  - Verificar se um objeto é maior que outro
- Como saber, de forma genérica, que um objeto é maior que outro?
  - Depende do tipo do objeto
  - Também depende de qual atributo queremos comparar
- Qual a melhor forma de fazer isso?
  - Deixar que cada objeto implemente a maneira como deve ser comparado com outro
  - Definir uma interface que estabelece um contrato entre os objetos e quem precisa fazer a comparação

unesp

Usando Interfaces

- Objetos que desejamos comparar devem implementar a interface Comparavel
  - Força a implementação dos métodos da interface (neste caso apenas um)
  - Compara o objeto atual (this) com o passado por parâmetro (other)

```
public interface Comparavel {
    // retorna 1 se este objeto é maior que o outro
    // retorna 0 se este objeto é igual ao outro
    // retorna -1 se este objeto é menor que o outro
    public int eMaiorQue(Comparavel outro);
}
```

unesp

Usando Interfaces

```
public class Retangulo implements Comparavel {
    public int largura = 0;
    public int altura = 0;
    public Point origem;
    // construtores
    // métodos
    public int getArea() {
        return largura * altura;
    }
    public int eMaiorQue(Comparavel outro) {
        Retangulo outroRetangulo = (Retangulo) outro;
        if (this.getArea() < outroRetangulo.getArea())
            return -1;
        else if (this.getArea() > outroRetangulo.getArea())
            return 1;
        else
            return 0;
    }
}
```

unesp

Usando Interfaces

- No exemplo anterior, o método **eMaiorQue** possui um parâmetro do tipo Comparavel
- Ou seja, interfaces podem ser usadas como qualquer outro tipo em Java
- O tipo da interface referencia qualquer objeto que implementa aquela interface
  - Serão visíveis apenas os métodos da interface

unesp

Novos Métodos de Interfaces

- A partir do Java 8, além dos métodos abstratos (sem corpo) também é possível definir dois outros tipos de métodos em uma interface
  - default
  - static
- Esse métodos devem conter uma implementação (corpo) dentro da própria interface
- Se nenhum modificador default ou static for especificado, o método é reconhecido como abstract (só assinatura)

unesp

Novos Métodos de Interfaces

- Métodos **default**

```
public interface Comparavel {
    // retorna 1 se este objeto é maior que o outro
    // retorna 0 se este objeto é igual ao outro
    // retorna -1 se este objeto é menor que o outro
    public default int eMaiorQue(Comparavel outro) {
        ...
    }
}
```

unesp

Novos Métodos de Interfaces

- A vantagem de métodos default surge quando uma interface precisa ser atualizada
- Considere que definimos a interface abaixo e que ela é utilizada em vários programas

```
public interface Fazer {
    void fazerAlgo(int i, double x);
    int fazerOutraCoisa(String s);
}
```

unesp

Novos Métodos de Interfaces

- Se quisermos adicionar um novo método, temos que redefinir a interface

```
public interface Fazer {
    void fazerAlgo(int i, double x);
    int fazerOutraCoisa(String s);
    boolean funcionou(int i, double x, String s);
}
```

- O problema é que todos os programas que usam essa interface vão falhar
  - Eles não implementam corretamente a nova versão da interface (falta o último método)

unesp

Novos Métodos de Interfaces

- Uma possível solução é criar outra interface que estende a antiga
  - Dessa forma, os programas que usam a interface antiga não vão falhar
  - Programadores podem optar por migrar ou não para a nova interface

```
public interface FazerMais extends Fazer {
    boolean funcionou(int i, double x, String s);
}
```

unesp

Novos Métodos de Interfaces

- Uma outra alternativa é criar um método default na interface
  - Como ele já oferece a implementação, não causará falhas nos programas
  - Classe que implementa a interface não tem a obrigação de definir o comportamento do método default
    - Diferentemente dos métodos abstract

```
public interface Fazer {
    void fazerAlgo(int i, double x);
    int fazerOutraCoisa(String s);
    default boolean funcionou(int i, double x, String s) {
        // corpo do método vai aqui
    }
}
```

unesp

Novos Métodos de Interfaces

- Quando uma interface que contém métodos default é estendida, podemos
  - Manter a implementação da interface estendida
    - Basta não mencionar o método na nova interface
  - Redeclarar o método default, tornando-o abstract
  - Redefinir o método default, sobrescrevendo-o

unesp

Novos Métodos de Interfaces

- Colisão de interfaces
  - Se uma classe implementa duas interfaces que possuem métodos default com a mesma assinatura, o método deve ser sobrescrito

```java
public class ClasseExemplo implements InterfaceA, InterfaceB {
    @Override
    public void metodoDefault() {
        // Podemos escolher qual implementação padrão utilizar
        // Por exemplo, usamos a implementação da InterfaceA
        InterfaceA.super.metodoDefault();

        // Ou, alternativamente, executar alguma lógica específica da classe
        // System.out.println("Método default personalizado da ClasseExemplo");
    }
}
```

unesp

Novos Métodos de Interfaces

- Precedência de superclasses
  - Se uma superclasse provê uma implementação de um método que tem a mesma assinatura de um método default de uma interface, a implementação da superclasse tem precedência

```
public interface InterfaceA {
    default void metodoDefault() {
        System.out.println("Método default da InterfaceA");
    }
}

public class SuperClasse {
    public void metodoDefault() {
        System.out.println("Método default da SuperClasse");
    }
}

public class ClasseExemplo extends SuperClasse implements InterfaceA {
    // Não precisamos sobrescrever o metodoDefault() aqui,
    // pois a implementação da SuperClasse tem precedência sobre
a InterfaceA
}

public class Main {
    public static void main(String[] args) {
        ClasseExemplo exemplo = new ClasseExemplo();
        exemplo.metodoDefault(); // Chama a implementação da SuperClasse
    }
}
```

Novos Métodos de Interfaces

- Métodos static
  - Similar aos métodos default, devem conter uma implementação
  - Foram permitidos para organizar melhor alguns métodos que estão estritamente relacionados a uma interfaces e não a uma classe
  - Os métodos static em interfaces não podem ser herdados nem sobrescritos pelas classes que implementam a interface

unesp

Classes Abstratas

unesp

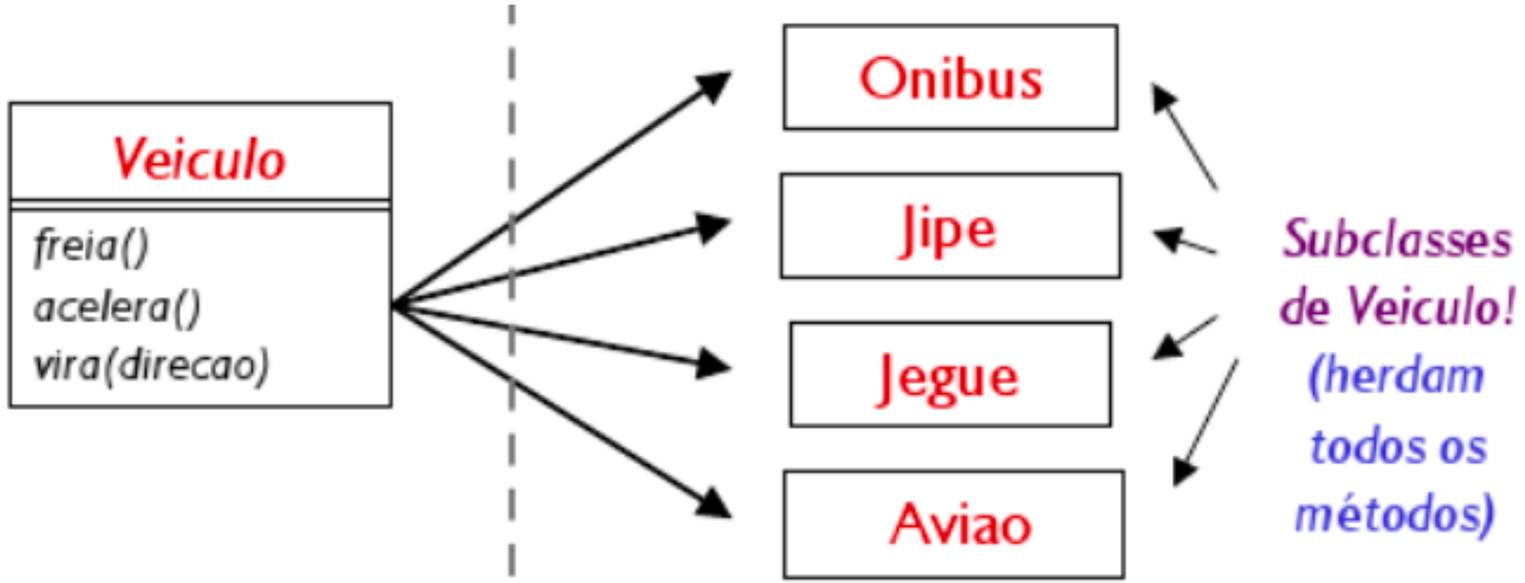
Classes Abstratas

- Em muitos casos, desejamos definir uma classe geral, que representa objetos de maneira genérica, mas que não faz sentido possuir uma instância
- **Exemplo:** Classes Animal, Vaca, Gato, Ovelha
  - No mundo real, todo animal é de algum subtipo
  - Não faz sentido que exista um objeto Animal
  - Porém, a definição da classe Animal é vantajosa, pois permite compartilhar as características comuns de todos os animais
- Neste caso, faz mais sentido que a classe Animal seja **abstrata**
  - As outras classes são ditas **concretas**

unesp

Classes Abstratas

- Todo veículo será sempre de um dos subtipos
- Definimos, neste problema, que não faz sentido existir instâncias da classe Veiculo



Classes Abstratas

- Vimos que os métodos de uma interface são implicitamente abstratos
  - Não possuem corpo (implementação)
  - Estabelecem o contrato mas não o comportamento
  - Obrigam as classes a implementarem
- Métodos abstratos também podem ser definidos em uma classe, **desde que a classe seja abstrata**

unesp

Classes Abstratas

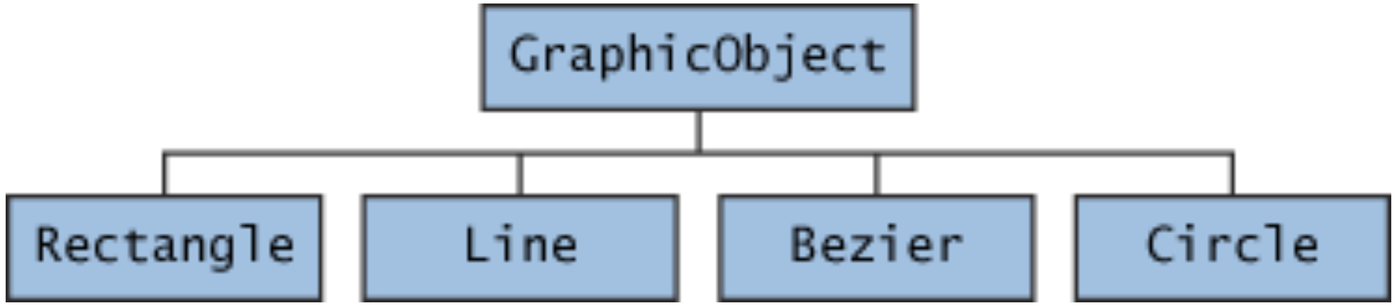
- Uma classe abstrata
  - Pode conter métodos abstratos e não abstratos
  - Pode conter campos como qualquer outra classe
  - Não pode ser instanciada (new)
  - Pode ser herdada
- Quando classes abstratas são herdadas, métodos abstratos devem ser implementados
  - Ou a nova classe deve ser declarada abstrata

```
public abstract class ObjetoGrafico {
    // declarar campos
    // declarar métodos não abstratos
    abstract void desenhar();
}
```

Classes Abstratas

- **Exemplo.** Imagine uma aplicação para desenhar diferentes formas geométricas
  - Linhas, Círculos, Curvas Bezier e Retângulos
- Todas essas formas possuem estados e comportamentos
  - **Estados:** posição, orientação, cor da linha, cor de fundo
  - **Comportamentos:** moveTo, rotate, resize, draw
- Alguns atributos e comportamentos são iguais para todos os métodos
  - **Ex:** position, cor de fundo e moveTo

Classes Abstratas



- Esta é uma situação perfeita para uma superclasse abstrata
- Nela, os membros definem os estados e comportamentos compartilhados por todos os subtipos

Classes Abstratas

```
abstract class ObjetoGrafico {
    int x, y; // posição
    ...
    void moverPara(int novoX, int novoY) {
        x = novoX;
        y = novoY;
    }
    abstract void desenhar();
    abstract void redimensionar();
}
```

unesp

Classes Abstratas

```
class Circulo extends ObjetoGrafico {
    void desenhar() {
        ...
    }
    void redimensionar() {
        ...
    }
}
```

```
class Retangulo extends ObjetoGrafico {
    void desenhar() {
        ...
    }
    void redimensionar() {
        ...
    }
}
```

unesp

Classes Abstratas 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

unesp

Classes Abstratas implementa Interface

```
abstract class ClasseAbstrata implements Interface1 {
    // implementa todos os métodos da Interface1 exceto um
}



class ClasseConcreta extends ClasseAbstrata {
    // implementa o método restante da Interface1
}
```

unesp

Classes Abstratas vs Interfaces

- Semelhanças
  - Ambas não podem ser instanciadas
  - Ambas podem conter métodos com ou sem implementação
    - Antes do Java 8, interfaces não podiam conter métodos com implementação (default ou static)
- Diferenças
  - Classes abstratas podem conter campos que não são public static final (constantes)
  - Métodos concretos (não abstratos) em classes abstratas podem ter definido seu modificador de acesso
    - Em interfaces, qualquer método é sempre público

unesp

Classes Abstratas vs 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

unesp

Classes Abstratas vs Interfaces

	Objetos	Herança	Métodos	Atributos	Construtor
Interface	Não pode ter instâncias	Uma classe pode **implementar** várias	Métodos abstratos, default e static	Somente constantes	Não pode ter
Classe Abstrata	Não pode ter instâncias	Uma classe pode **estender** apenas uma	Métodos concretos e abstratos	Constantes e atributos	Pode ter

unesp

Agenda

- Sumário
- Conceito de interface
- Interfaces em Java
- Atualizações no Java 8
- Classes abstratas
- Classes abstratas *vs* interfaces

unesp

Exercício

- Escreva uma classe abstrata chamada *Produto* que implementa a interface *Comparable*
  - Veja a API do Java para informações sobre essa interface
  - A comparação entre produtos deverá ser implementada considerando seu custo-benefício
- Um Produto deve conter
  - Um nome
  - Um preço
  - Métodos que achar necessário

unesp

Exercício

- Escreva as classes *Shampoo*, *Biscoito* e *Leite*, filhas de Produto
  - Shampoo contem um campo que indica a irritabilidade do shampoo para peles normais (**int**)
  - Biscoito contem um campo que indica quantidade de componentes cancerígenos em sua fórmula (**int**)
  - Leite contem um campo que indica quantos dias o leite dura após ser embalado (**int**)
  - **Cada instância terá um valor para esses campos**
  - Crie uma fórmula para cada produto que combine seu preço e as características individuais de cada um para calcular o custo-benefício. Essa fórmula deve ser usada na implementação do método **compareTo** de cada classe.

Exercício

- Escreva uma classe concreta chamada *Supermercado*
  - Essa classe não precisa ter atributos
  - Contem o método main()
  - No método main
    - Crie um array para cada tipo específico de produto
      - Shapoo[], Biscoito[], Leite[]
    - Crie algumas instâncias e coloque dentro dos arrays
    - Compare todos os produtos de um mesmo tipo entre si, indicando quem tem maior custo-benefício
      - compareTo

unesp

Exercício - Sobre a interface *Comparable*

- A interface *Comparable* da API do Java utiliza o conceito de *Generics*, do qual trataremos mais adiante
- Se o tipo *generic* não for definido quando utilizamos a interface, o método **compareTo** dessa interface terá como parâmetro um tipo Object
  - int compareTo (Object obj)
- Para utilizar o recurso *generic*, basta colocar entre <> o tipo de classe que se deseja comparar
  - public class MyClass implements Comparable<MyClass>
  - Isso faz com que o método **compareTo** tenha a assinatura abaixo
  - int compareTo (MyClass obj)

5

unesp

Bibliografia

- DEITEL, H. M. & DEITEL, P.J. "Java : como programar", Bookman, 2017.
- Material baseado nos slides:
  - Luiz E. Virgilio da Silva. Notas de Aula de Programação Orientada a Objetos (ICMC/USP).