Inatel

C206/C06 – Programação Orientada a Objetos com Java

Classe Abstrata

Prof. Christopher Lima

christopher@inatel.br

Objetivos

- Conhecer classes abstratas
- Entender porque existem e quando utilizá-las
- Utilizando métodos abstratos
- Exercícios

Modelando

- Criamos uma superclasse chamada Brownie e outras três subclasses, herdando dela.
  - BrownieNutella
  - BrownieCafe
  - BrownieDoceDeLeite

- Seria possível criarmos instâncias de Brownie?

```java
public static void main(String[] args) {

    Brownie bwCafe = new BrownieCafe( nome: "Brownie de Café",  preco: 10,  sabor: "Café");
    Brownie bwNutella = new BrownieNutella( nome: "Brownie de Nutella",  preco: 70,  sabor: "Nutella");

    bwCafe.addCarrinhoDeCompras();
    bwNutella.addCarrinhoDeCompras();
```

- Sim, é possível! O código compila e executa sem erros
- Mas faz sentido termos instâncias de Brownie?
- Veja que é diferente de termos **referências** do tipo Brownie

```java
public static void main(String[] args) {

    Brownie bwCafe = new BrownieCafe( nome: "Brownie de Café",  preco: 10,  sabor: "Café");
    Brownie bwNutella = new BrownieNutella( nome: "Brownie de Nutella",  preco: 70,  sabor: "Nutella");

    bwCafe.addCarrinhoDeCompras();
    bwNutella.addCarrinhoDeCompras();
```

- No código acima temos um exemplo de uma instância de BrownieNutella e de uma instância de BrownieCafe sendo armazenado como uma **referência** para Brownie!

- Quando pensamos em instâncias, pensamos em objetos concretos. Que realizam comportamentos!
- Quando falamos nos Brownies, nós imaginamos um BrownieNutella, um BrownieCafe e assim por diante.
- Mas “Brownie” parece algo **abstrato** para termos uma instância desse tipo!
- Mas por que então criamos a classe Brownie?

- Em primeiro lugar, para evitar repetir código!
- Criamos classes como BrownieNutella e BrownieCafe, todas elas **herdando** de Brownie e reusando sua estrutura (membros e métodos). Com isso, economizamos bastante código. Observe essas classes

```
public class BrownieNutella extends Brownie {

    public BrownieNutella(String nome, double preco, String sabor) {
        super(nome, preco, sabor);
    }
}
```

```
public class BrownieCafe extends Brownie {

    public BrownieCafe(String nome, double preco, String sabor) {
        super(nome, preco, sabor);
    }

}
```

- E podemos também criar novos tipos de Brownies, como BrownieDoceDeLeite, BrownieDeBacon e etc. Herdando da classe Brownie. Isso favorece a evolução do nosso software.
- E com herança, podemos utilizar o poder do polimorfismo. Criamos métodos genéricos que sabem apenas lidar com a superclasse e os métodos nela presentes.

```java
public class Comprador {

    private String nome;
    private double saldo;


    public Comprador(String nome, double saldo) {
        this.nome = nome;
        this.saldo = saldo;
    }

    public void efetuarCompra(Brownie brownie){
        brownie.addCarrinhoDeCompras();
        brownie.calculaValorTotalCompra();
        System.out.println("Comprador comprou o brownie "+ brownie.getNome());
    }
}
```

- Observe que o método ***efetuarCompra(Brownie brownie)*** recebe instâncias referenciadas ou do tipo Brownie. Ele não precisa conhecer nenhuma classe que herda de Brownie. Isso também favorece a evolução do software.

Classes Abstratas

- Resgatando a questão. Faz sentido ter instâncias do tipo Brownie? Não
- Mas faz todo sentido termos referências do tipo Brownie. Afinal, é assim que o método ***efetuarCompra***(Brownie brownie) funciona. Ele recebe referências para Brownie.
- Concluímos que criamos a classe Brownie apenas para ser referências (variáveis) e não instâncias (objetos na memória).
- Assim, podemos dizer que ela uma classe abstrata.
- No Java e C# temos a palavra chave abstract para esse fim.
- Observe a nova classe abstrata Brownie

```java
public abstract class Brownie {

    protected String nome;
    protected double preco;
    protected String sabor;

    public Brownie(String nome, double preco, String sabor) {
        this.nome = nome;
        this.preco = preco;
        this.sabor = sabor;
    }

    public void addCarrinhoDeCompras() {

        System.out.println("Adicionando no carrinho de compras um: "+ nome);
    }

    public void calculaValorTotalCompra() {
        System.out.println("Calculando valor total da compra de um: "+ nome +": "+preco);
    }

}
```

Classes Abstratas

- Quando fazemos uma classe ***abstract***, estamos passando a seguinte informação.
  - Não desejamos instanciar essa classe.
  - Ela deve ser uma superclasse e suas subclasses serão instanciadas.
  - Ela deve ser usada como referência para permitir o polimorfismo.

- O compilador Java garante que ela não será instanciada. Mas pode ser referenciada normalmente.
- Apenas suas subclasses poderão ser instanciadas

Classes Abstratas

```
public class Main {

    public static void main(String[] args) {

        Brownie brownie = new brownie("Jeferson", 100, "Nutella");

    }
}
```

Classes Abstratas

- Seria correto dizer que toda superclasse deve ser abstrata?
- Não!!!!!
- Depende do escopo do seu projeto e de suas abstrações. Não é obrigatório fazer toda superclasse abstrata
- Poderíamos fazer BrownieNutella ser uma superclasse também. E criarmos novos tipos de BrownieNutella a partir dela.

Classes Abstratas

- Repare que podemos ter várias camadas (gerações) de Herança.
- Mas não podemos ter uma mesma classe herdando de mais de uma classe em uma única declaração.

```
public class BrownieCafe extends Brownie {

    public BrownieCafe(String nome, double preco, String sabor) {
        super(nome,preco,sabor);
    }

    public void adicionaCafe() {
        System.out.println(super.nome +" adicionando mais café");
    }
}
```

Classes Abstratas

- O Código abaixo não compila, pois estamos tentando, na mesma declaração fazer uma classe herdar de outras duas!

```
public class BrownieCafe extends Brownie, BrownieNutella {

    public BrownieCafe(String nome, double preco, String sabor) {
        super(nome,preco,sabor);
    }

    public void adicionaCafe() {
        System.out.println(super.nome +" adicionando mais café");
    }
}
```

Método Abstrato

- Vamos pensar no método addCarrinhoDeCompras() na superclasse Brownie.

```
public void addCarrinhoDeCompras() {

    System.out.println("Adicionando no carrinho de compras um: "+ nome);
}
```

Método Abstrato

- Imagine que as subclasses resolvessem não implementar métodos específicos.
- Nesse caso, cada subclasse utilizaria o comportamento da superclasse.
- E se quiséssemos forçar que cada subclasse sobrescreva o método addCarrinhoDeCompras()? Afim de garantir comportamento específico?
- Quando temos uma classe abstrata, podemos ter também um método abstrato. Isto é, não possui implementação na superclasse, e toda subclasse é obrigada a implementar.
- Vamos deixar esse método abstrato, na classe abstrata Brownie.

Método Abstrato

```
// Não compila - Método abstrato não pode ter implementação
public abstract void addCarrinhoDeCompras() {

    System.out.println("Adicionando no carrinho de compras um: "+ nome);
}
```

- Remova a implementação:

```
// Agora sim
public abstract void addCarrinhoDeCompras();
```

Método Abstrato

- Vamos na classe BrownieNutella remover o método addCarrinhoDeCompras()
- Perceba que o código não irá compilar

```
// Não compila pois é necessário implementar, obrigatoriamete
// o método abstrato presente na superclasse
public class BrownieNutella extends Brownie {

    public BrownieNutella(String nome, double preco, String sabor) {
        super(nome,preco,sabor);
    }

    public void adicionaNutella() {

        System.out.println(super.nome +" adicionando mais nutella");
    }

}
```

Método Abstrato

- Perceba a mensagem de erro gerada pelo IntelliJ

```
public class BrownieNutella extends Brownie {

    public Br
```

Class 'BrownieNutella' must either be declared abstract or implement abstract method 'addCarrinhoDeCompras()' in 'Brownie'

Implement methods Alt+Shift+Enter More actions... Alt+Enter

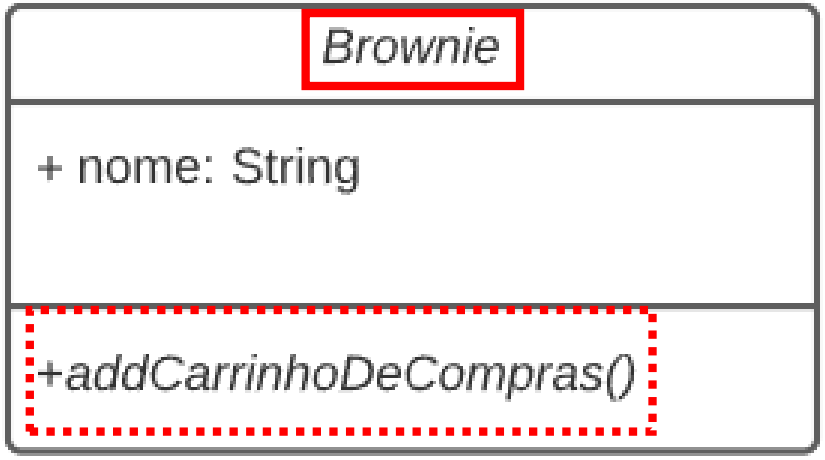
- Se clicarmos em “Implement methods”, o IntelliJ já colocará o corpo desses métodos na classe BrownieNutella

Método Abstrato

```java
public class BrownieNutella extends Brownie {

    public BrownieNutella(String nome, double preco, String sabor) {
        super(nome,preco,sabor);
    }

    @Override
    public void addCarrinhoDeCompras() {
        //Agora é só fazer a implementação
    }

}
```

UML

- No diagrama UML, classes e métodos abstratos aparecem com a fonte itálica



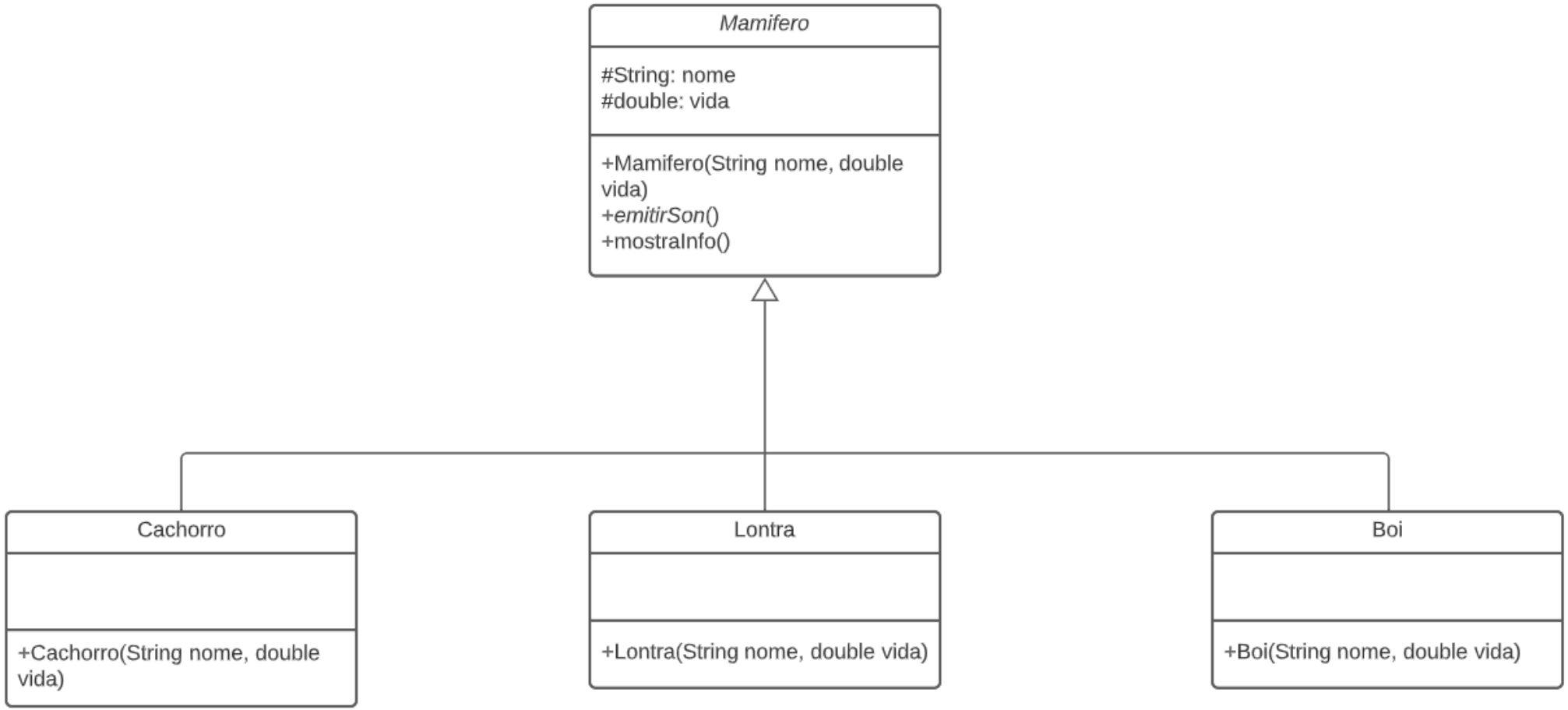
Outros Exemplos

- Seguem outros exemplos que ***pode*** fazer sentido ser uma classe abstrata
  - Pessoa
  - Funcionário
  - Mamífero
  - Veículo
  - Animal
  - Bolo
  - Doce
  - …

Exercício

- Crie classes Java para modelar o UML no próximo slide
- Crie uma classe Main, com método main() para fazer o teste!
- Cada classe que herda de Mamifero, deve implementar o seu método *emitirSon()*. Pode apenas imprimir mensagens.
- Esse exercício está separado do exemplo apresentado nessa aula.
- A classe Mamifero e o método *emitirSom*() são abstratos.

Exercício


Mamifero
#String: nome
#double: vida
+Mamifero(String nome, double vida)
+emitirSon()
+mostraInfo()
Cachorro
+Cachorro(String nome, double vida)
Lontra
+Lontra(String nome, double vida)
Boi
+Boi(String nome, double vida)

Resolução dos Exercícios

https://github.com/chrislima-inatel/C206_C125

Material Complementar

- Capítulo 10 da apostila FJ-11
  - Classes Abstratas