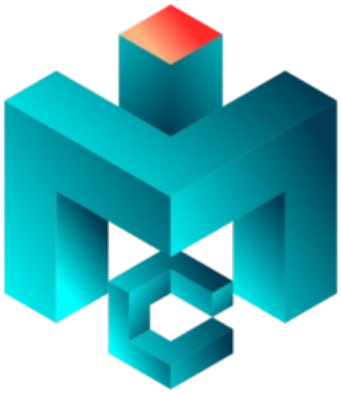


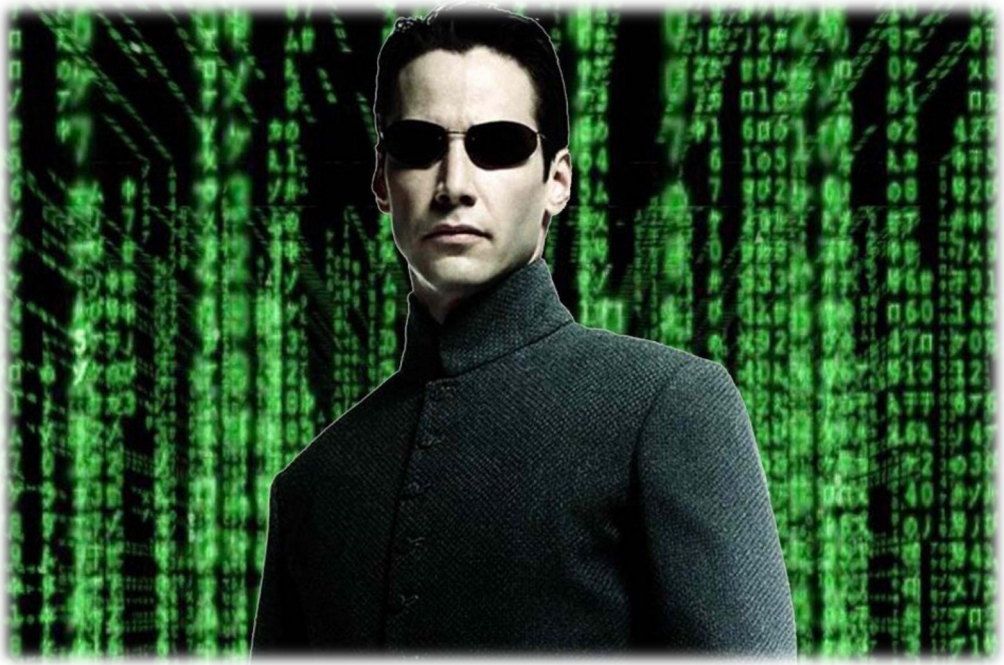
Aula – 7
Padrão Factory

Disciplina: COM221 – Computação Orientada a Objetos II

Prof: Phyllipe Lima
phyllipe@unifei.edu.br

Universidade Federal de Itajubá – UNIFEI
IMC – Instituto de Matemática e Computação

Qual o problema do **new**?



```
Pato pato = new Marreco();
```

Ao usar *new* não estamos lidando com instâncias concretas?

Mas não queremos lidar sempre com *interfaces* ?

Ao ver ***new*** pense em algo concreto

...mas é a única forma de termos objetos

O problema não é o ***new***

O problema é sua localização!

Vamos falar mais sobre construção de objetos!

Quem é invocado ao usarmos o ***new?***

- Construtores permitem a criação de instâncias
- Primeiro recurso usado por programadores para tal finalidade

```
public class Nave {

    public Nave() {

    }

}
```

```
public class Nave {

    public void fazAlgo() {
        Arma a = new Arma();
        a.atirar();
    }

}
```

Limitação dos Construtores

- Precisa ter o nome da classe (perde expressividade no nome)

```
public void criaNaveComPoderes() {}

public void criaNaveDataBase() {}

public void criaNaveXML() {}
```

Limitação dos Construtores

- A única forma de customizar a instância é parametrizando o construtor

```
public Nave() {}

public Nave(String id) {}

public Nave(String id, int opção) {}
```

Limitação dos Construtores

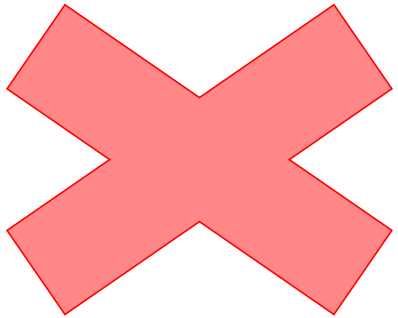
- Sempre cria um objeto novo
- Pode requerer vários parâmetros


```
public Nave(String nome, String tipoArma,
            double vidaInicial, double municao) {}
```

Limitação dos Construtores

Não pode retornar instância da subclasse


```
public Nave(String tipoNave) {
    if(tipoNave=="Fogo")
        return new NaveDeFogo();
}
```



Static Factory Method

- Não é um padrão
- Usa métodos estáticos para criar objetos
- É uma técnica de programação

```
public class Nave {

    public static Nave criaNave() {
        return new Nave();
    }
```

Static Factory Method

Pode retornar subclasse


```
public class Nave {

    public static Nave criaNaveComPoderes() {
        return new NaveDeFogo();
    }
```

Static Factory Method

Pode retornar objeto já existente

```
public class Nave {

    public static Nave instanciaNave;

    public static Nave criaNave() {
        if(instanciaNave==null)
            return new Nave();
        return instanciaNave;
    }
```

Pst: Singleton

Static Factory Method

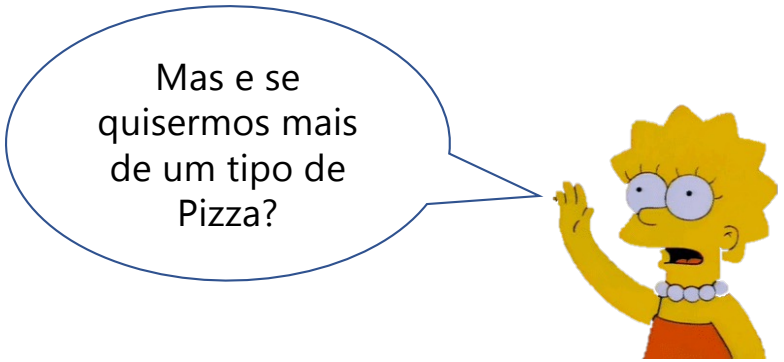
Pode usar expressividade nos nomes

```java
public static Nave criaNaveDataBase(String id) {
    //faz coisas
    return new Nave();
}

public static Nave criaNaveXML(String id) {
    //faz coisas
    return new Nave();
}
```

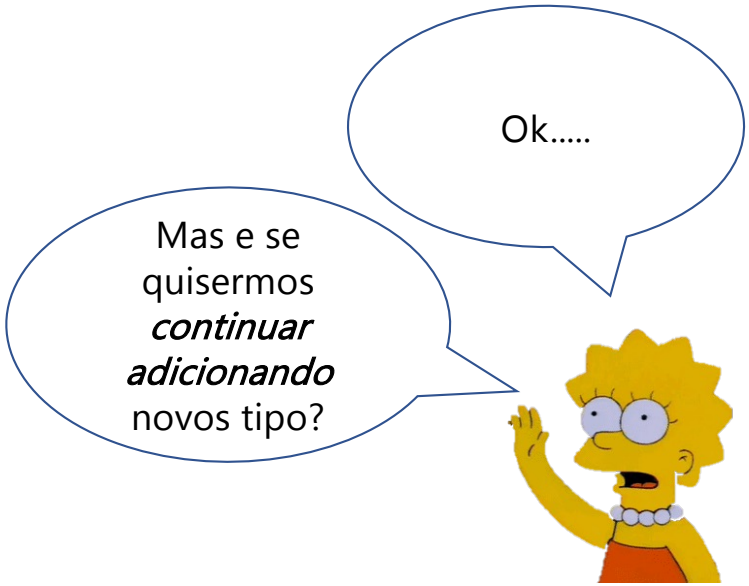
1º DE MAIO

Pizzaria Ponto du Funcionário

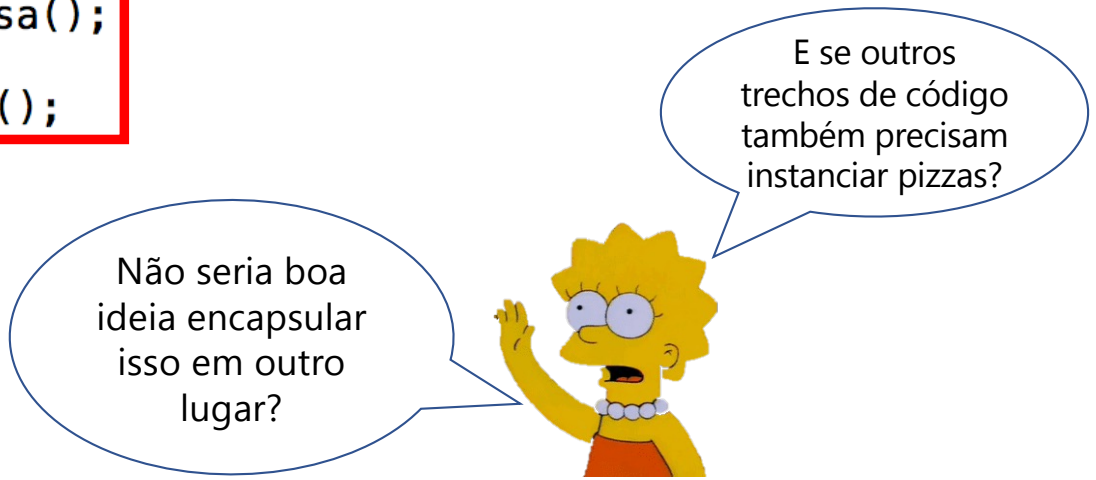
```
public class Pizzaria {

    public Pizza encomendarPizza() {

        Pizza p = new Pizza();

        p.preparar();
        p.assar();
        p.fatiar();
        p.empacotar();

        return p;
    }
```



```
public Pizza encomendarPizza() {

    Pizza p;

    if(tipo == "Calabresa")
        p = new PizzaCalabresa();
    else if(tipo == "Frango")
        p = new PizzaFrango();
    else if(tipo == "Portuguesa")
        p = new PizzaPortuguesa();
    else
        p = new PizzaMozarela();

    p.preparar();
    p.assar();
    p.fatiar();
    p.empacotar();

    return p;
```

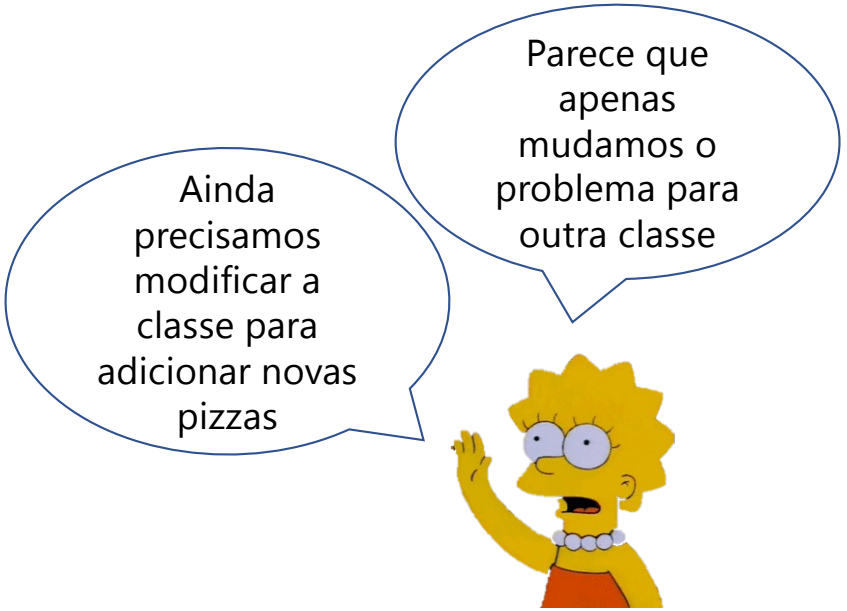


```
public Pizza encomendarPizza() {

    Pizza p;

    if(tipo == "Calabresa")
        p = new PizzaCalabresa();
    else if(tipo == "Frango")
        p = new PizzaFrango();
    else if(tipo == "Portuguesa")
        p = new PizzaPortuguesa();
    else
        p = new PizzaMozarela();

    p.preparar();
    p.assar();
    p.fatiar();
    p.empacotar();

    return p;
```



Encapsulando a instanciação

```java
public class SimplePizzaFactory {

    public Pizza criaPizza(String tipo) {

        Pizza p = null;

        if(tipo == "Calabresa")
            p = new PizzaCalabresa();
        else if(tipo == "Frango")
            p = new PizzaFrango();
        else if(tipo == "Portuguesa")
            p = new PizzaPortuguesa();
        else
            p = new PizzaMozarela();

        return p;
    }

}
```



Verdade! Essa abordagem viola o OCP!
Mas ao menos estamos concentrando mudança em apenas um local!

Nem sempre conseguiremos seguir todos os princípios

Se outros trechos de código precisam de instâncias de pizza, podem chamar de um único local!

```
public Pizzaria(SimplePizzaFactory fabricaPizza) {
    this.fabricaPizza = fabricaPizza;
}

public Pizza encomendarPizza() {

    Pizza p;

    p = fabricaPizza.criaPizza(tipo);

    p.preparar();
    p.assar();
    p.fatiar();
    p.empacotar();

    return p;
}
```

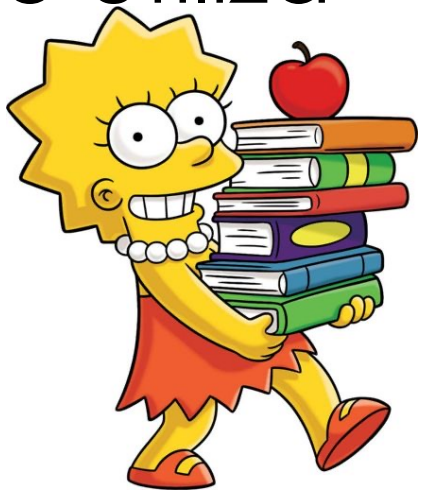
Agora a classe Pizzaria não se preocupa mais em criar instancias!

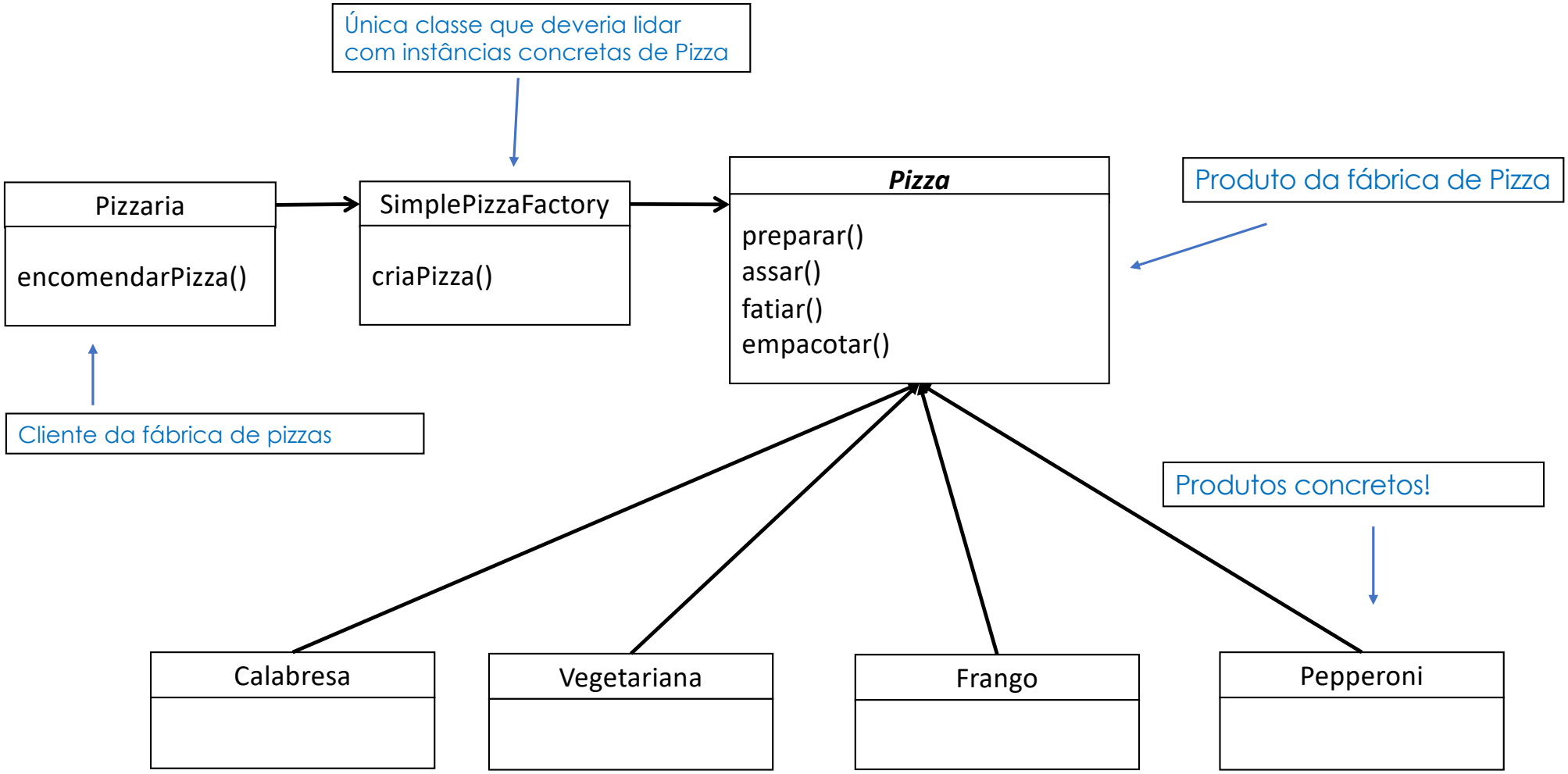
Simple Factory!

Acabamos de utilizar o ***simple factory!***
É um ***design pattern?***

Não!

O Simple Factory é uma variação do Static Factory Method.
A diferença é que ele requer uma instância, e sua variação utiliza métodos estáticos




Única classe que deveria lidar
com instâncias concretas de Pizza
Pizzaria
encomendarPizza()
SimplePizzaFactory
criaPizza()
Pizza
preparar()
assar()
fatiar()
empacotar()
Produto da fábrica de Pizza
Cliente da fábrica de pizzas
Produtos concretos!
Calabresa
Vegetariana
Frango
Pepperoni

Hands-On: Fábrica de Pizzas

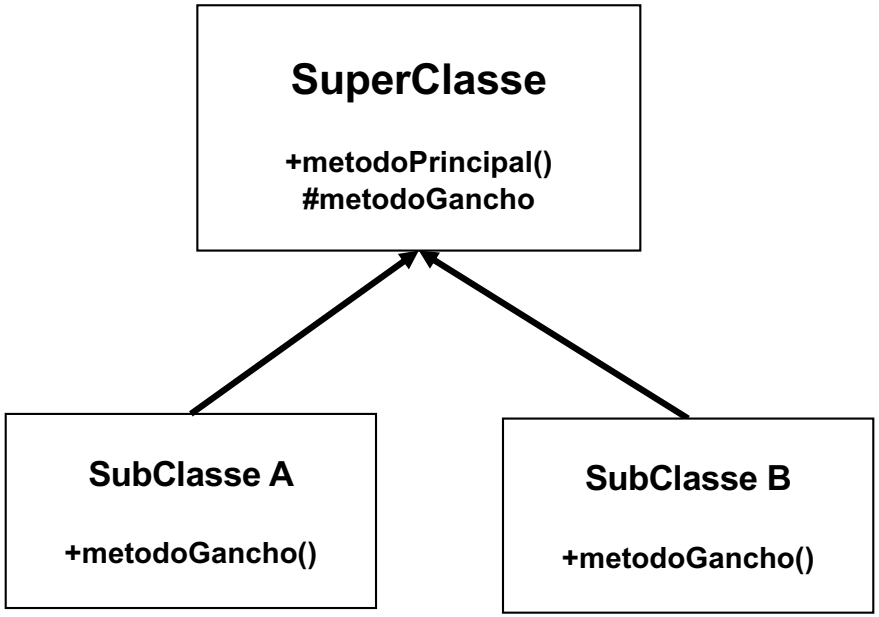
Não confunda o Simple Factory com o ***design pattern*** Factory!

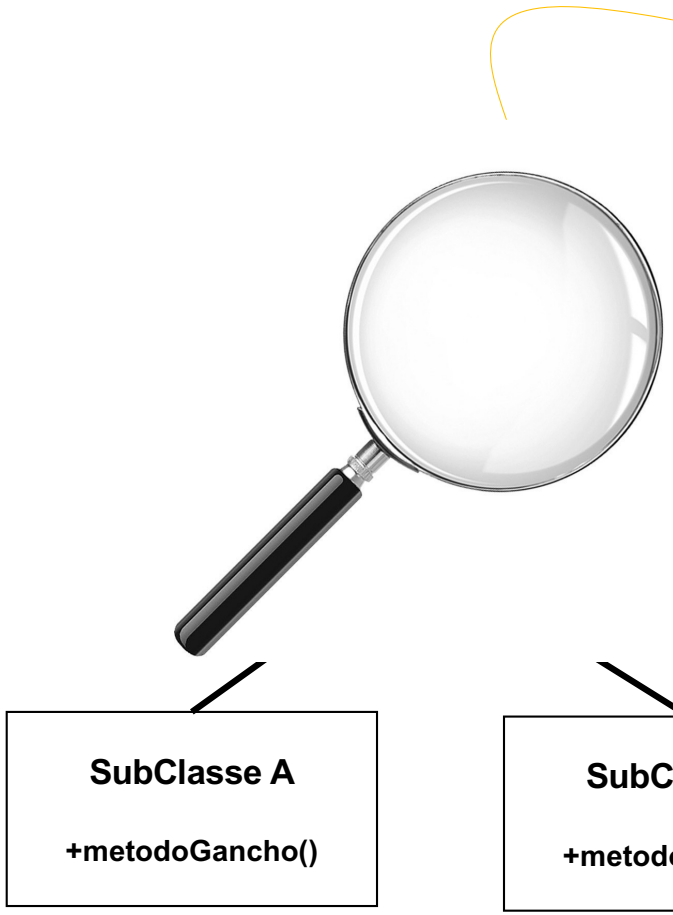
Antes de entendermos o ***factory*** precisamos entender ***hook methods***

Hook Method (Método Gancho)

Hook Method

- Permite a especialização de comportamento
- É um método que só é definido na subclasse
- Funciona como um “Gancho”, no qual uma nova lógica “se pendura”


SuperClasse
+metodoPrincipal()
#metodoGancho
SubClasse A
+metodoGancho()
SubClasse B
+metodoGancho()



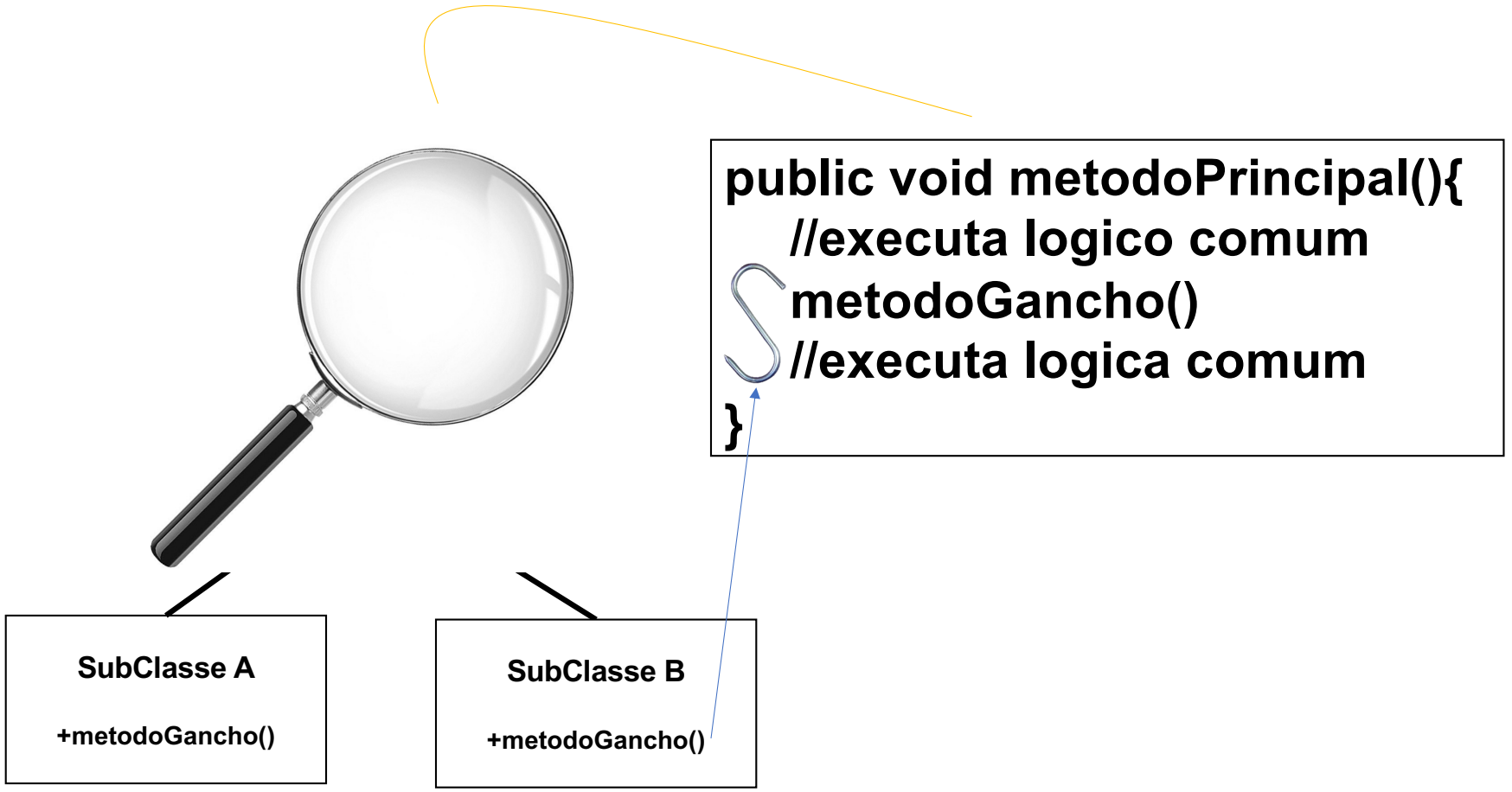
```
public void metodoPrincipal(){
    //executa logico comum
    metodoGancho()
    //executa logica comum
}
```

SubClasse A

+metodoGancho()

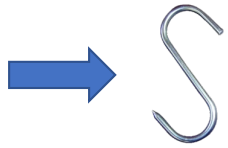
SubClasse B

+metodoGancho()

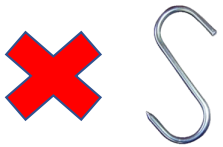

public void metodoPrincipal(){
//executa logico comum
metodoGancho()
//executa logica comum
}
SubClasse A
+metodoGancho()
SubClasse B
+metodoGancho()

Revisando Modificadores de Acesso

- abstract
  - Precisa obrigatoriamente ser implementado
  - Candidato a Hook Method



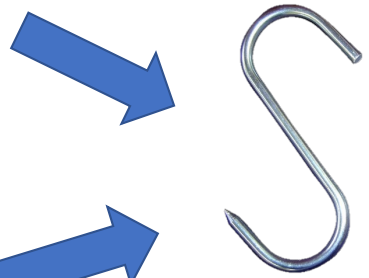
- final
  - Não pode ser sobrescrito
  - Candidato a métodos que invocam os hook methods
  - Esse métodos não podem ter seu funcionamento modificado para manter controle



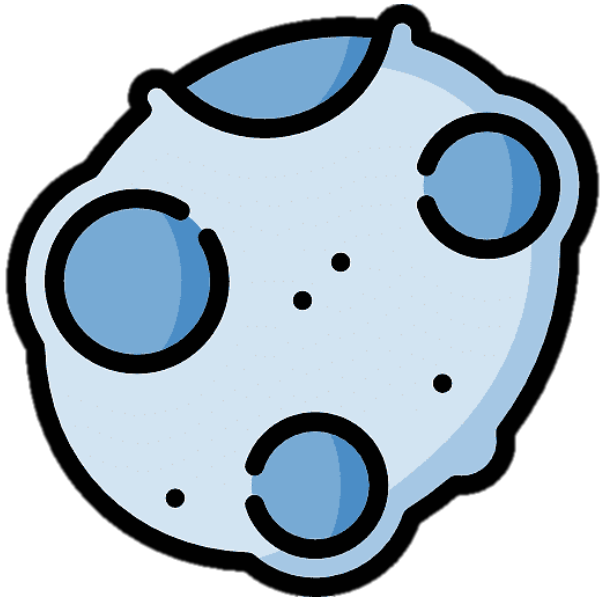
Revisando Modificadores de Acesso

- public
  - Visível por qualquer classe
  - Pode ser sobrescrito

- protected
  - Visível apenas por subclasses
  - Pode ser sobrescrito



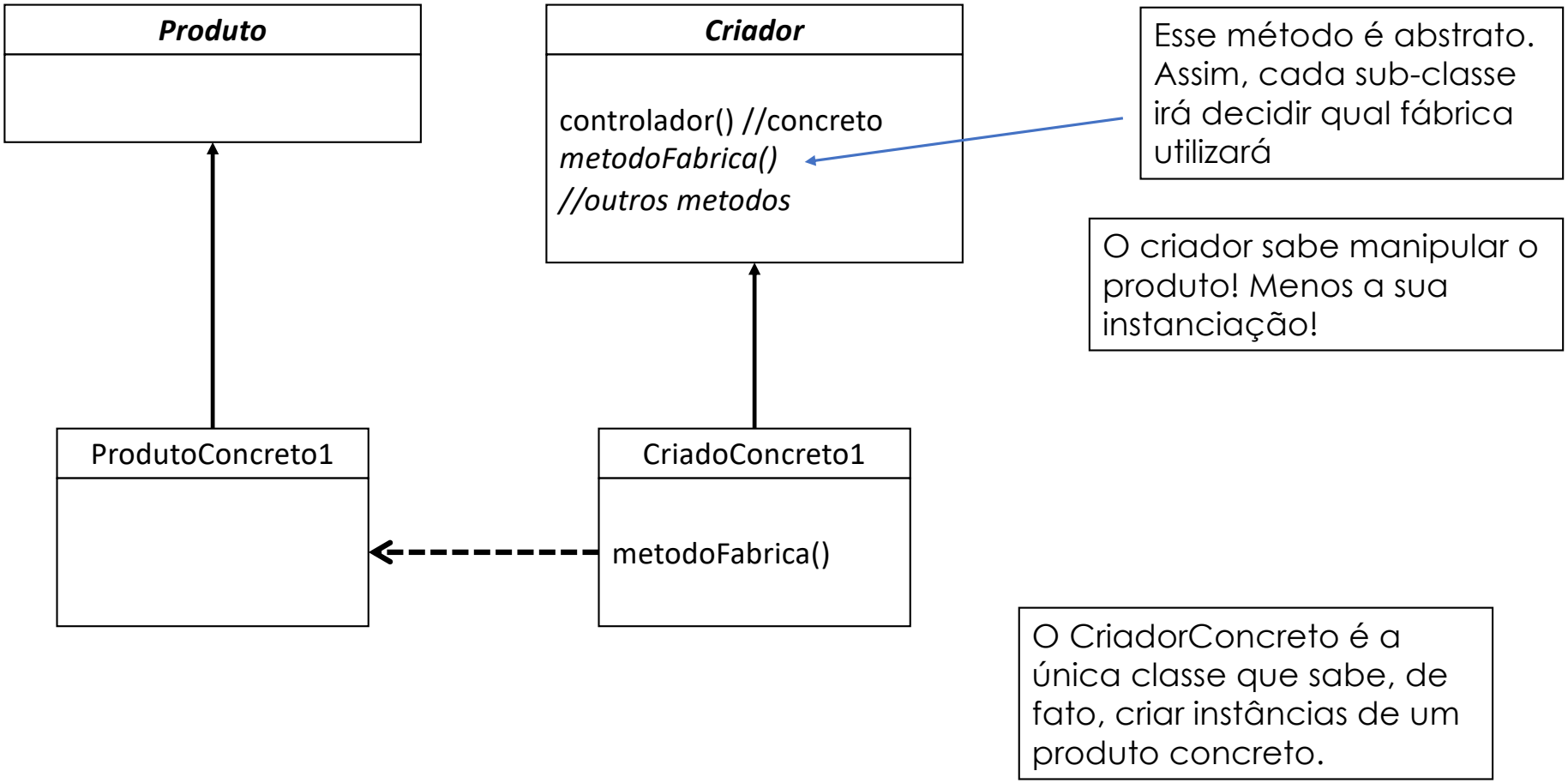
Space Shooter

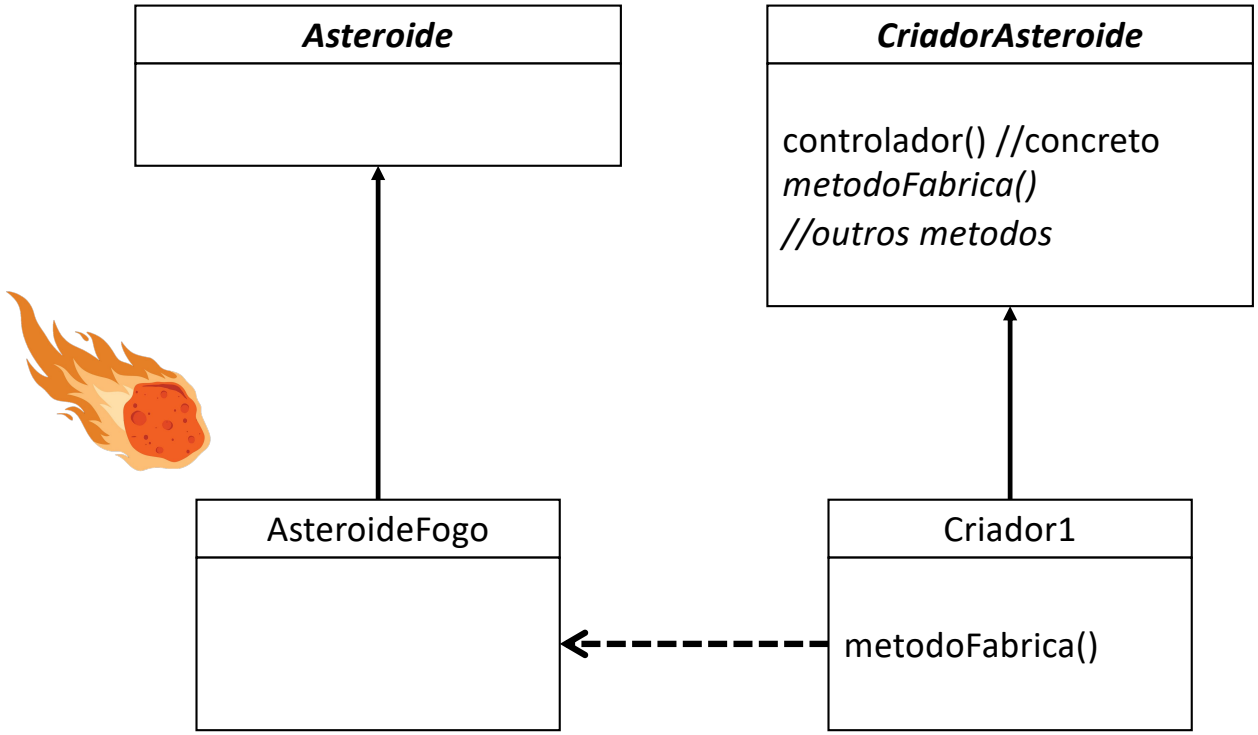


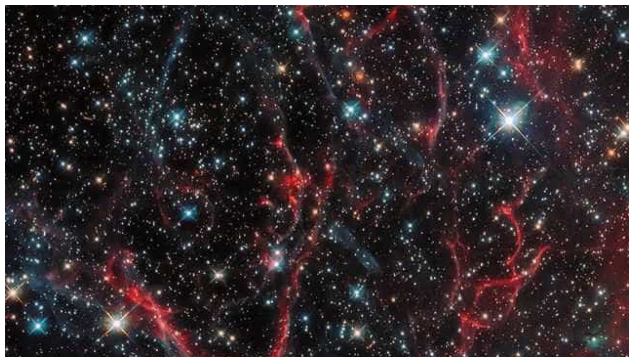
O objetivo é destruir os asteroides. Mas existem vários tipos, para cenários diferentes!

Factory:

Define uma interface (super-tipo) para criar um objeto, mas as sub-classes que irão decidir quais instâncias utilizarão.


Produto
Criador
controlador() //concreto
metodoFabrica()
//outros metodos
Esse método é abstrato. Assim, cada sub-classe irá decidir qual fábrica utilizará
O criador sabe manipular o produto! Menos a sua instanciação!
ProdutoConcreto1
CriadoConcreto1
metodoFabrica()
O CriadorConcreto é a única classe que sabe, de fato, criar instâncias de um produto concreto.


Asteroide
CriadorAsteroide
controlador() //concreto
metodoFabrica()
//outros metodos
AsteroideFogo
Criador1
metodoFabrica()



Hands-On: Space Shooter

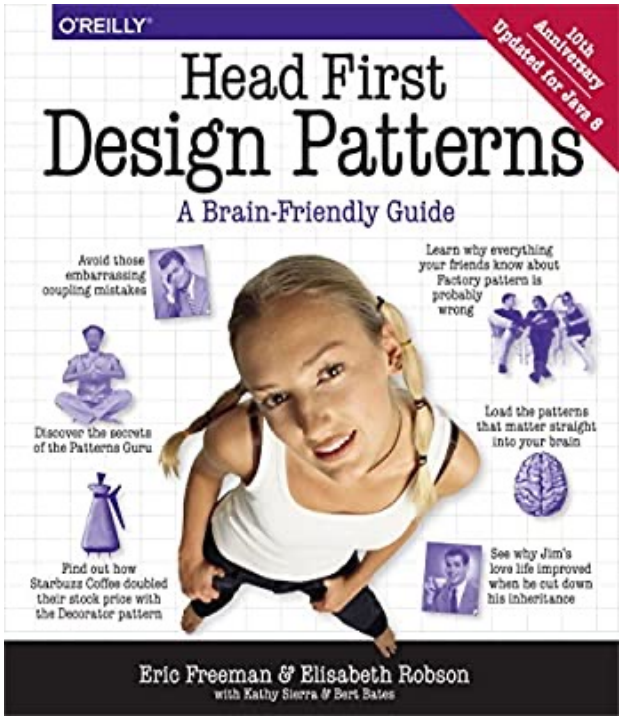
Referência

[1] This was once revealed to me in a dream.

- Capítulo 6 do livro Engenharia de Software Moderna
  - Padrões de Projeto
  - https://engsoftmoderna.info/cap6.html

Referência - Complementar

[1] This was once revealed to me in a dream.



- Head First Design Patterns
- Cap 4

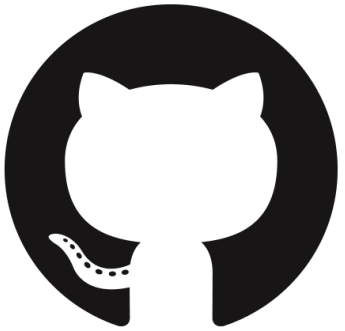
Referência - Complementar

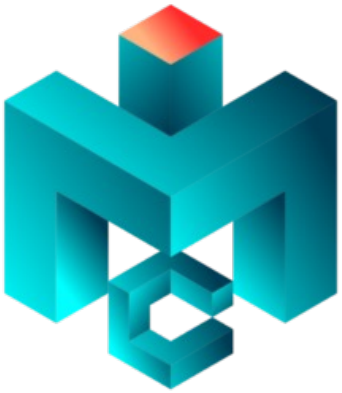
[1] This was once revealed to me in a dream.

- Design Patterns com Java
- Cap 2

Implementações

- https://github.com/phillima-classroom/COM221





Aula – 7
Padrão Factory

Disciplina: COM221 – Computação Orientada a Objetos II

Prof: Phyllipe Lima
phyllipe@unifei.edu.br

Universidade Federal de Itajubá – UNIFEI
IMC – Instituto de Matemática e Computação