Universidade Federal de Uberlândia
Faculdade de Computação
Disciplina: POO2
Prof. Fabiano Azevedo Dorça

Padrões Fábrica

Simple Factory
Factory Method

Padrões Fábrica

- Padrão *Simple Factory*: fornece interfaces para criar objetos sem expor a lógica de criação para o cliente.
- Padrão *Factory Method*: fornece interfaces para criar objetos, mas permite que subclasses determinem qual classe instanciar.
- Padrão *Abstract Factory*: permite criar famílias de objetos relacionados sem expor as suas classes.

Simple Factory

- Simple factory:
  - Visa encapsular a criação de objetos em um método.
  - Isto permite manter em um único local comandos “new ClasseConcreta()”.
  - Desacopla o cliente dos objetos que deseja criar.
  - Permite ao cliente transferir a responsabilidade de instanciação de objetos complexos a uma fábrica.

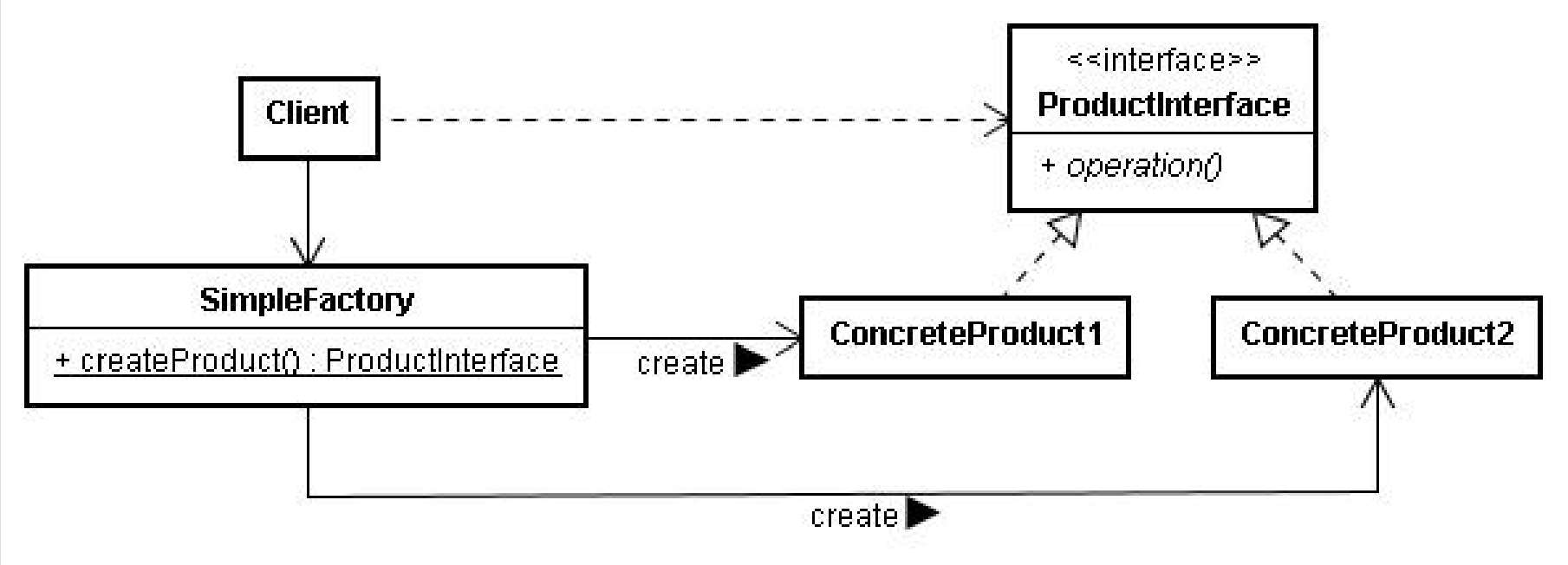
Simple Factory

- Evita modificação de código quando novas classes são criadas
- Evita modificação de código quando a **forma de instanciar objetos é modificada**.
  - Exemplo: novos decoradores são criados, afetando vários pontos do código.
  - Facilita manutenção de código já que a replicação de código de instanciação está em um único local.

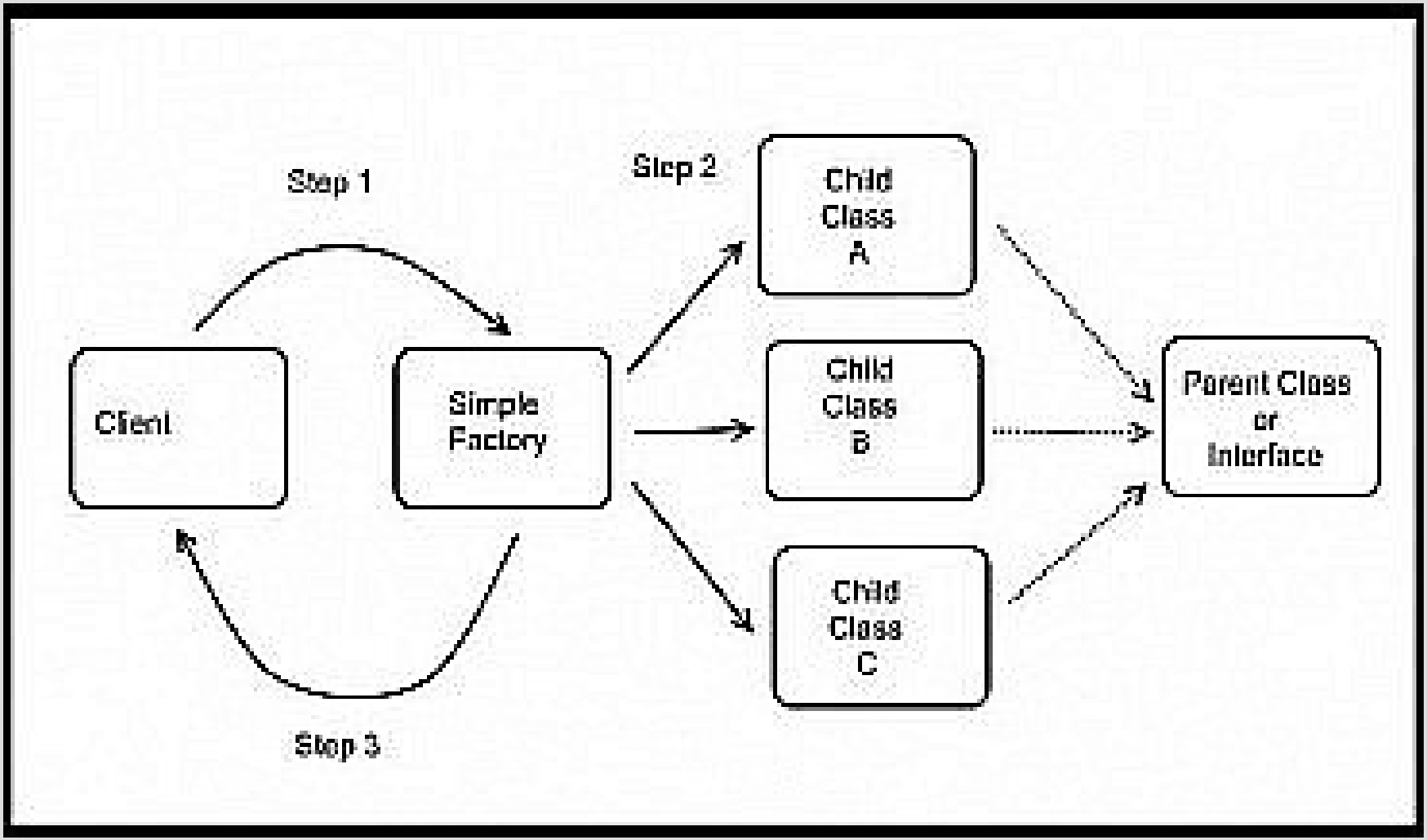
Simple Factory

- Participantes:
  - Client: requisita objetos à fábrica;
  - SimpleFactory: recebe requisiçoes do cliente e devolve objetos instanciados;
  - Product Interface: interface que representa os produtos a serem criados pela fábrica;
  - Concrete Product: classes concretas representando os produtos a serem instanciados pela fábrica.

Simple Factory



Simple Factory



- Fábrica de pizza

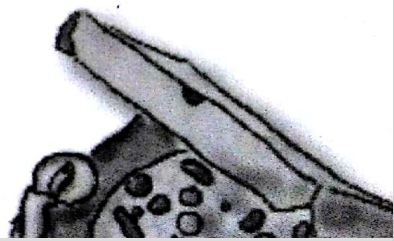
Identificando os aspectos que variam

Imagine que você tem uma pizzaria e, como dono de uma pizzaria moderna em Objectville, pode escrever um código assim:

```
Pizza orderPizza() {
   Pizza pizza = new Pizza();

   pizza.prepare();
   pizza.bake();
   pizza.cut();
   pizza.box();
   return pizza;
}
```

Classe abstrata

Por motivos de flexibilidade, realmente queremos que seja uma classe ou uma interface abstrata, mas não podemos instanciar diretamente nenhuma delas.



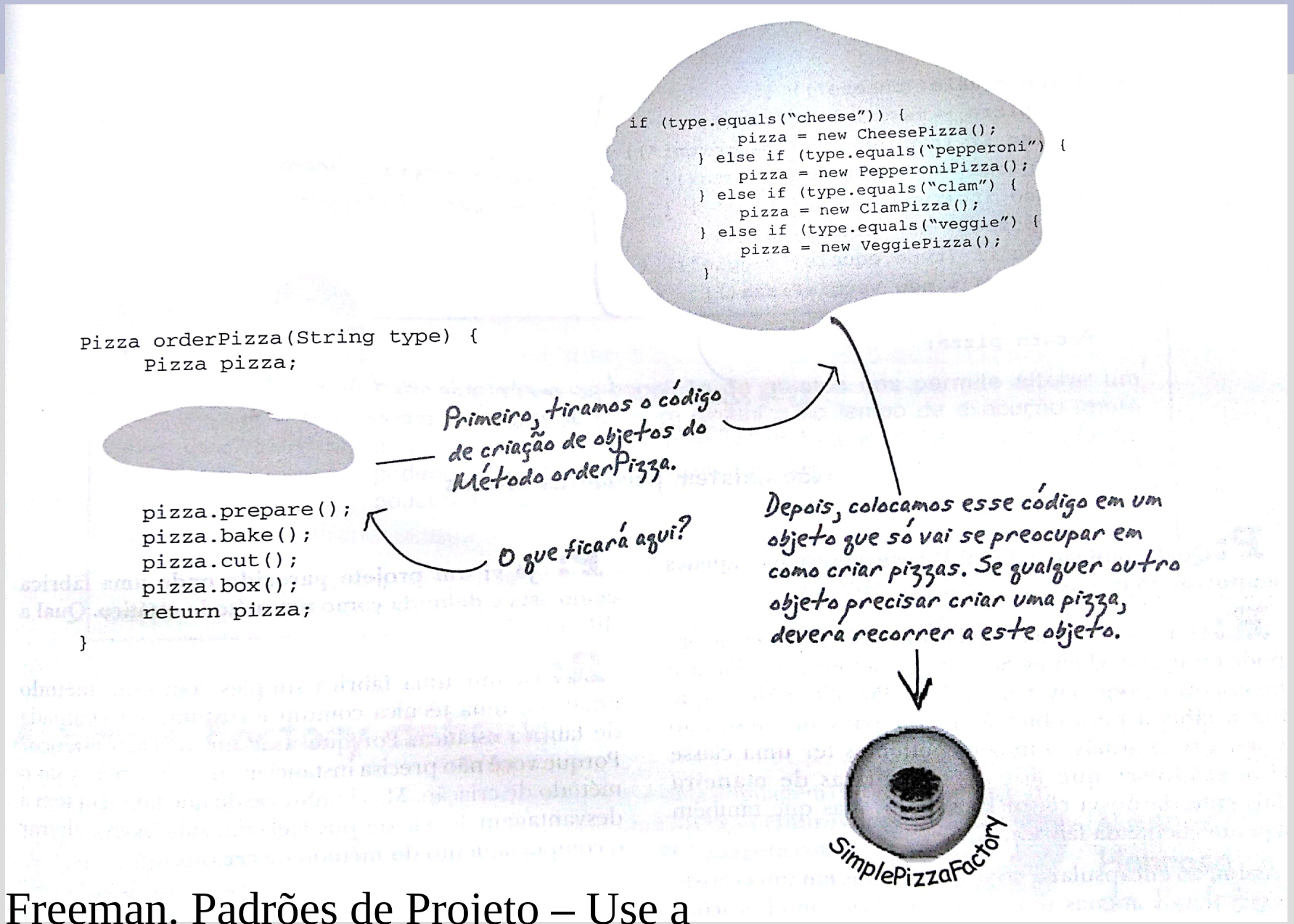
Freeman. Padrões de Projeto – Use a Cabeça.

Este código NÃO está fechado para modificações. Se a pizzaria mudar suas ofertas de pizza, precisamos entrar neste código e modificá-lo.

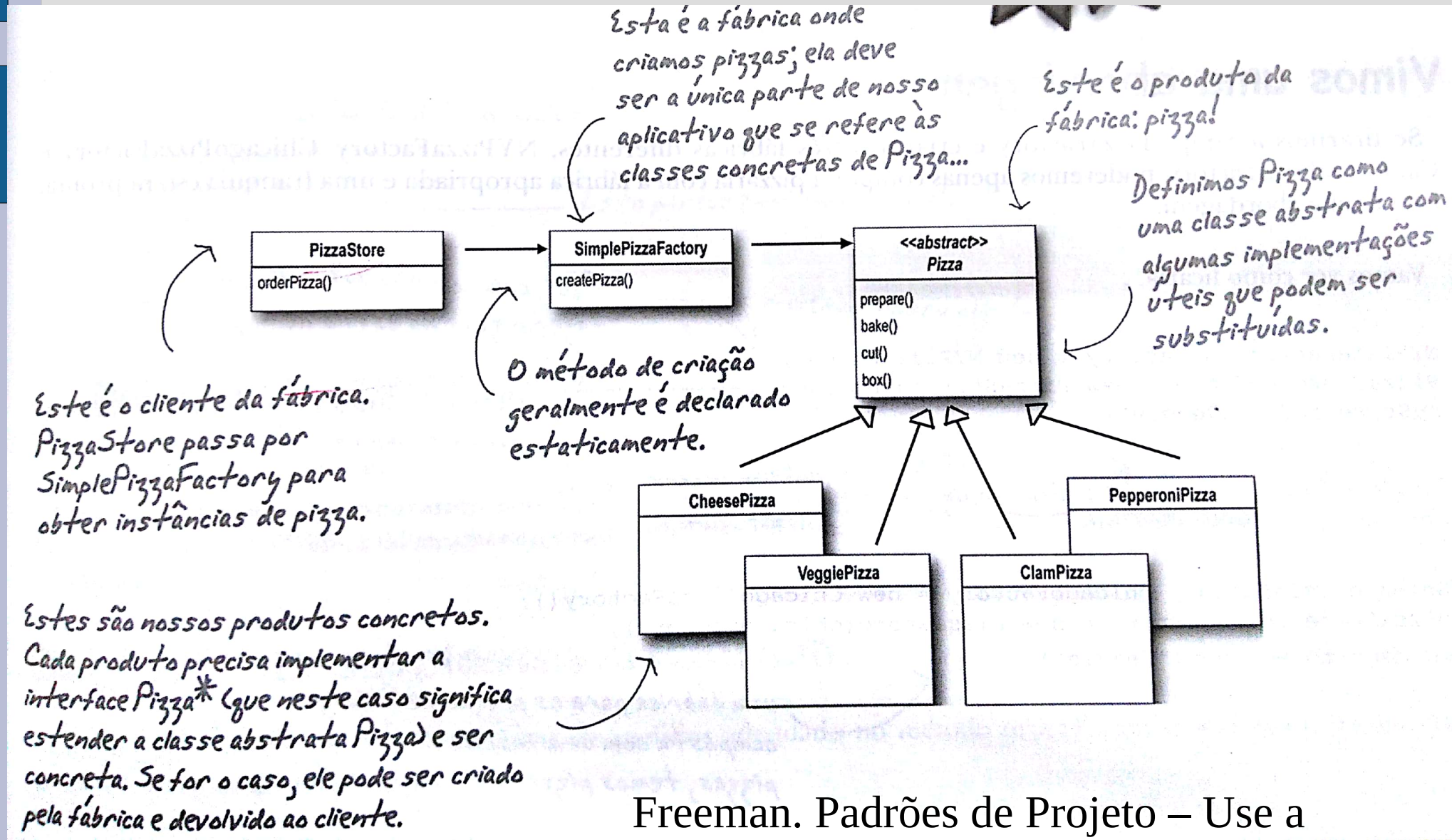
```
Pizza orderPizza(String type) {
    Pizza pizza;

    if (type.equals("cheese")) {
        pizza = new CheesePizza();
    } else if (type.equals("greek") {
        pizza = new GreekPizza();
    } else if (type.equals("pepperoni")
        pizza = new PepperoniPizza();
    } else if (type.equals("clam") {
        pizza = new ClamPizza();
    } else if (type.equals("veggie") {
        pizza = new eggiePizza();
    }

    pizza.prepare();
    pizza.bake();
    pizza.cut();
    pizza.box();
    return pizza;
}
```

Isto é o que varia. Como a seleção de pizzas muda com o tempo, este código terá que ser modificado muitas vezes.

Isto é o que esperamos que permaneça igual. Para a maioria, preparar, assar e encaixotar uma pizza é a mesma coisa há anos. Então, não esperamos que esse código mude, apenas as pizzas nas quais ele opera.



Freeman. Padrões de Projeto – Use a



Freeman. Padrões de Projeto – Use a

Simple Factory

Exemplo: Uma fábrica de personagens

```
public class SimplePersonagemFactory{
    public [static] Personagem createPersonagem(int tipo){
        Personagem p = null;
        if (tipo == 1)
            p = new Mago();
        else if (tipo ==2)
            p = new Feiticeiro();
        else if (tipo == 3)
            p = new Campones();
        else
            p = new Lanceiro;
        return p;
    }
```

Simple Factory

Exemplo: Uma fábrica de personagens com ataques decorados com armas especificas

```
public class SimplePersonagemFactory{

    public [static] Personagem createPersonagem(int tipo){
        Personagem p = new Personagem();
        if (tipo == 1) //mago
            p.setAtaque(new Magia(new Ataque1()));
        else if (tipo ==2) //feiticeiro
            p.setAtaque(new Feitico(new Ataque2()));
        else if (tipo == 3) //campones
            p.setAtaque(new Facao(new Ataque3()));
        else
            p = new Lanceiro;
        return p;
    }
```

Simple Factory

Exemplo: Uma fábrica de personagens com diversos métodos de criação

```
public class SimplePersonagemFactory{

    public [static] Personagem createMago(){
        return new Mago();
    }
    public [static] Personagem createFeiticeiro(){
        return new Feiticeiro();
    }
    public [static] Personagem createCampones(){
        return new Campones();
    }
}
```

Simple Factory

Exemplo: Uma fábrica de personagens com estratégias

```
public class SimplePersonagemFactory{

    public [static] Personagem createPersonagem(int tipo){
        Personagem p = new Personagem();
        if (tipo == 1) {
            p.setAtaque(new AtaqueFraco());
            p.setCorrida(new CorridaRapida());
         } else if (tipo ==2) {
            p.setAtaque(new AtaqueForte());
            p.setCorrida(new CorridaLenta());
          }
        .....
        return p;
    }
```

Simple Factory

- Algumas possibilidades incluem:
  - a geração automática randomica dos personagens.
  - a modificação da forma de instanciação de um objeto sem alteração do restante do código

Simple Factory

- Conclusões
  - Além de evitar problemas de manutenção futuros, o padrão simple factory abre novas possibilidades relacionadas à instanciação de objetos.
  - Facilita a implementação quando a classe concreta a ser instanciada é definida em tempo de execução.
  - Evita problemas quando o código que instancia objetos é uma área de mudanças frequentes.
  - Evita duplicação de código de instanciação e fornece um local único para fazer a manutenção.
  - Encapsula a instanciação de objetos complexos, como cadeias, decorators, composites, etc.

Factory Method

- **Factory Method**
  - * Define uma interface para instanciação de objetos, mas deixa as **subclasses** decidirem que classes instanciar.
  - * Permite a uma classe **postergar** a instanciação às subclasses.
  - * Regra: “Criar objetos numa operação separada de modo que subclasses possam redefinir a maneira como eles são criados.”
  - * Essa regra permite que projetistas de subclasses possam mudar a classe de objetos que a classe ancestral instancia.

Factory Method

- Aplicabilidade
  - Use o padrão factory method quando:
    - * Uma classe não pode antecipar a classe de objetos que deve criar;
    - * Uma classe quer que suas subclasses especifiquem os objetos que criam.

Factory Method

Participantes:

As classes e/ou objetos que participam do padrão são:

Product: Define uma interface de objetos que o método-fábrica cria.

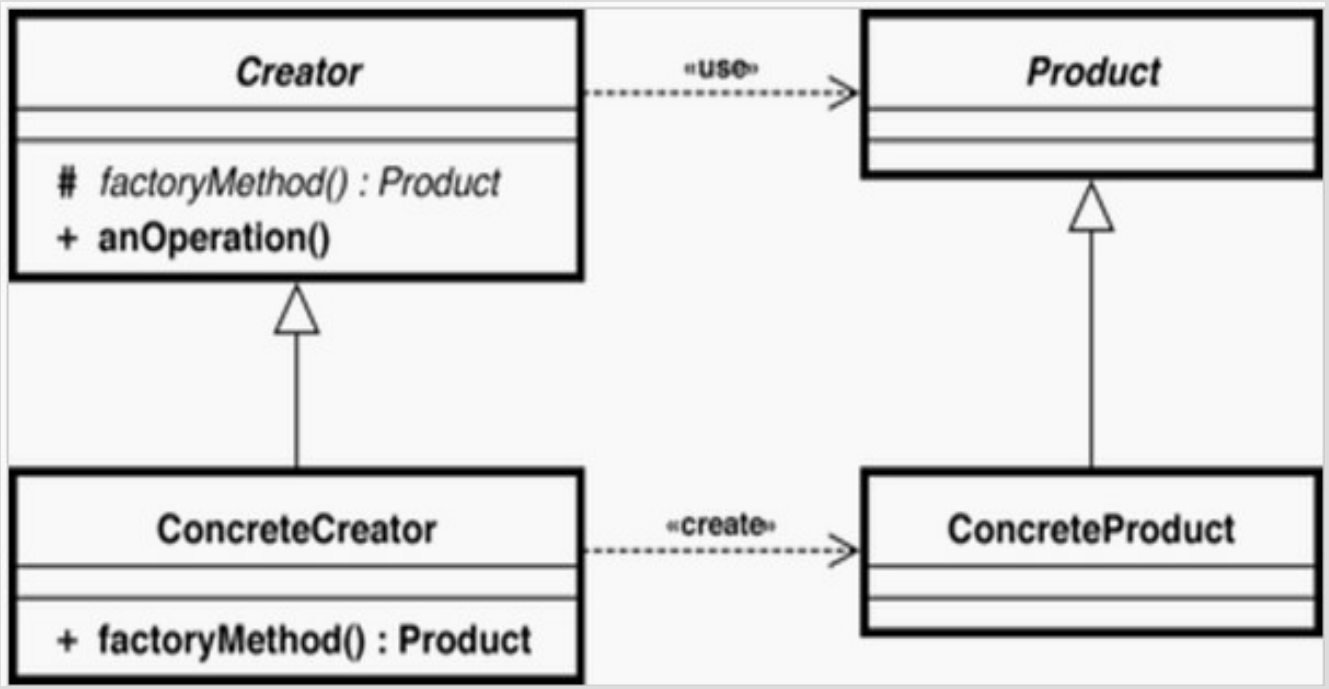
ConcreteProduct: Implementa a interface Product, criando um produto concreto.

Creator: Declara o método-fábrica, que retorna um objeto do tipo Product.

ConcreteCreator: Sobrescreve o método-fábrica para retornar uma instância de ConcreteProduct.

Factory Method

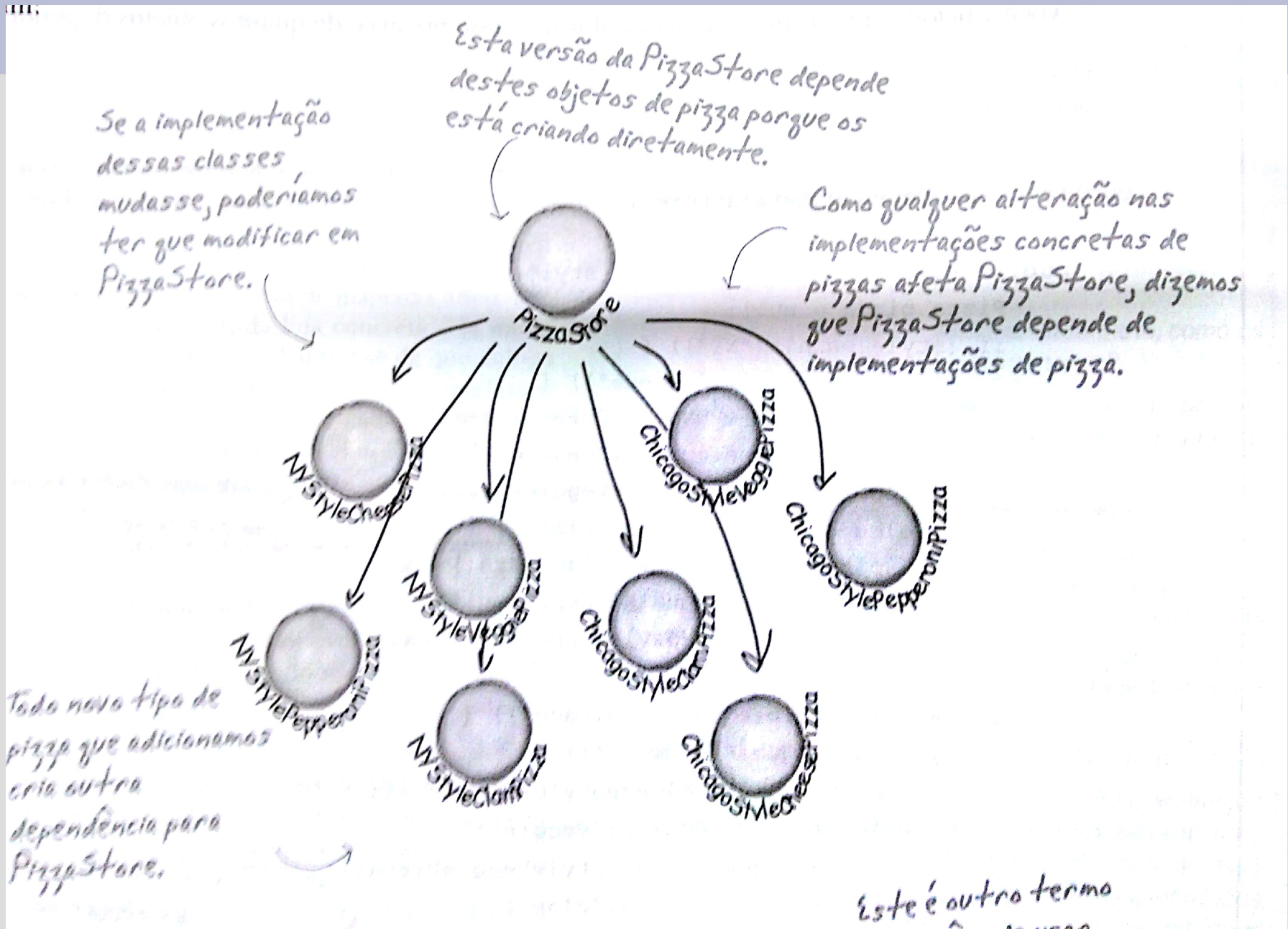
- Diagrama



Factory Method

- Na construção de frameworks...
  - * Os factory methods eliminam a necessidade de anexar classes específicas das aplicações cliente no código do framework.
  - * O código lida somente com interface Product.
  - * Portanto ele pode trabalhar com quaisquer classes ConcreteProduct definidas pelo usuário.

Factory Method



Factory Method

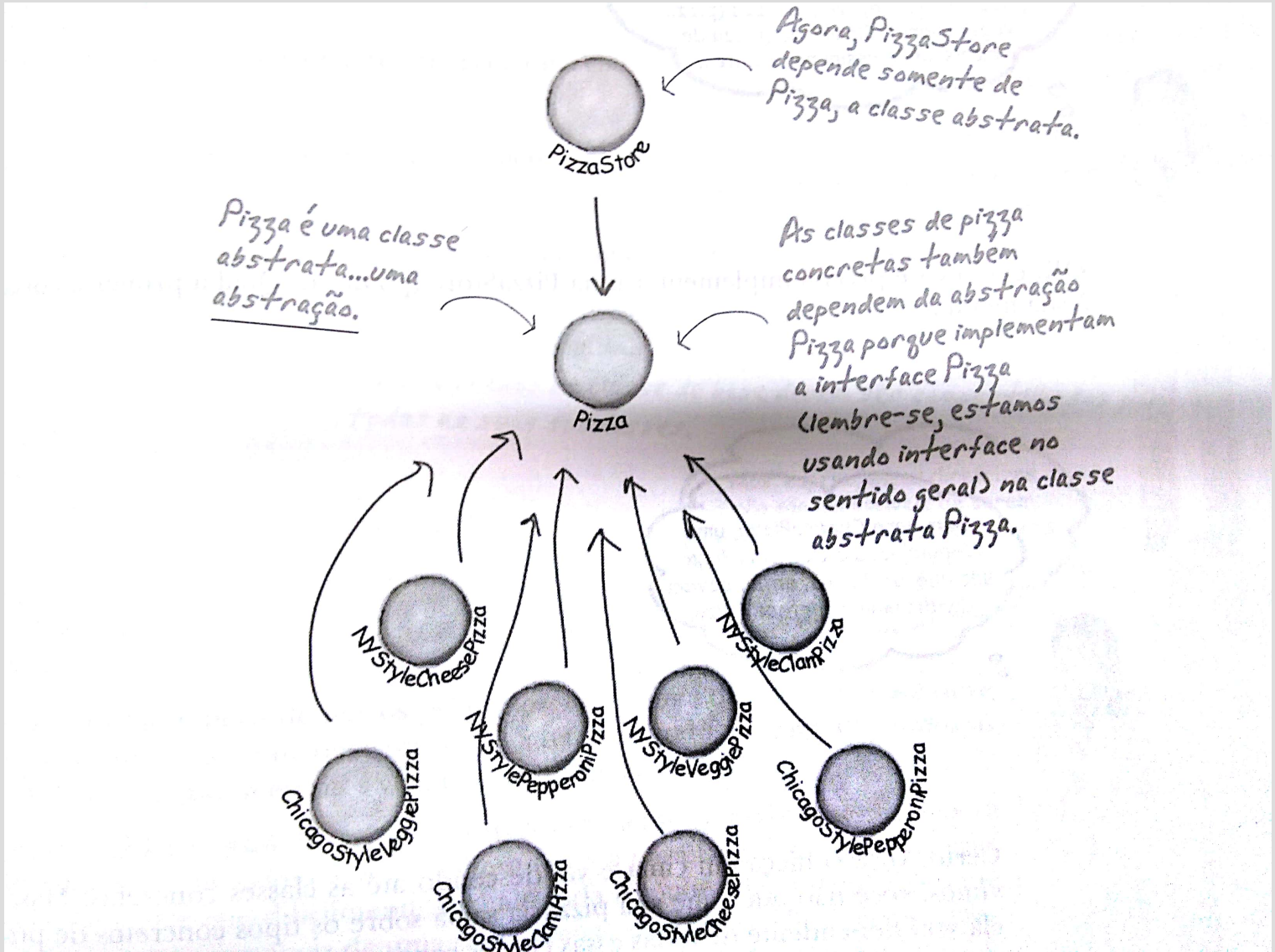
- Princípio da inversão da dependência
  - Componentes de alto nível não dependam de componentes de nível inferior; os dois devem depender de abstrações
  - Considerando que **PizzaStore** é nosso componente de alto nível, e as implementações de pizzas são componentes de baixo nível, então **PizzaStore** é dependente das classes concretas de pizza...

Factory Method

- Esse princípio nos diz que devemos escrever nosso código para que dependamos de abstrações, e não de classes concretas.
- Isso serve tanto para nossos módulos de alto nível quanto para os de baixo nível.

Factory Method

Aplicando o princípio...

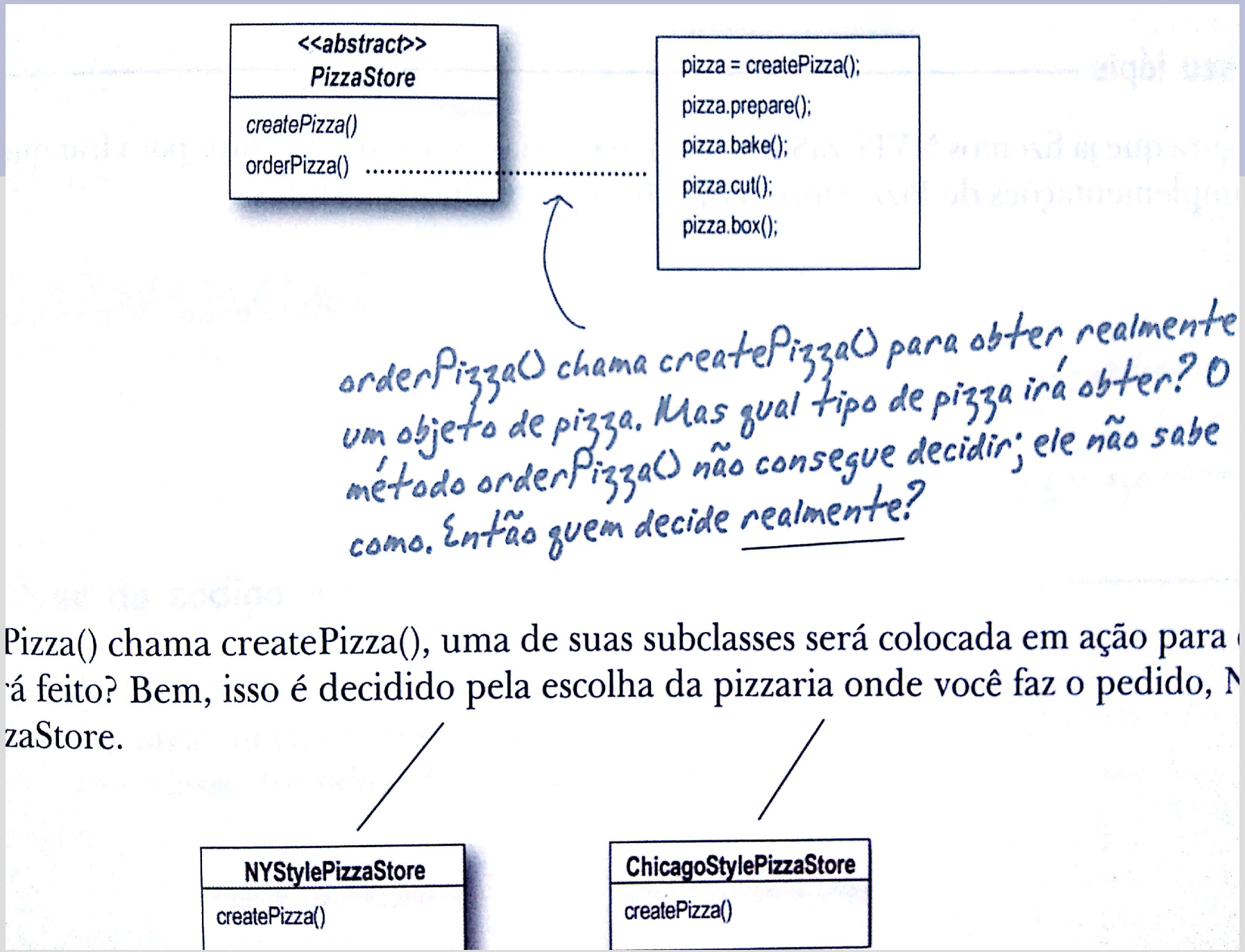


Factory Method

- Aplicando o princípio, notamos que nosso componente de alto nível (PizzaStore) e nossos componentes de baixo nível (as pizzas) dependem da abstração Pizza

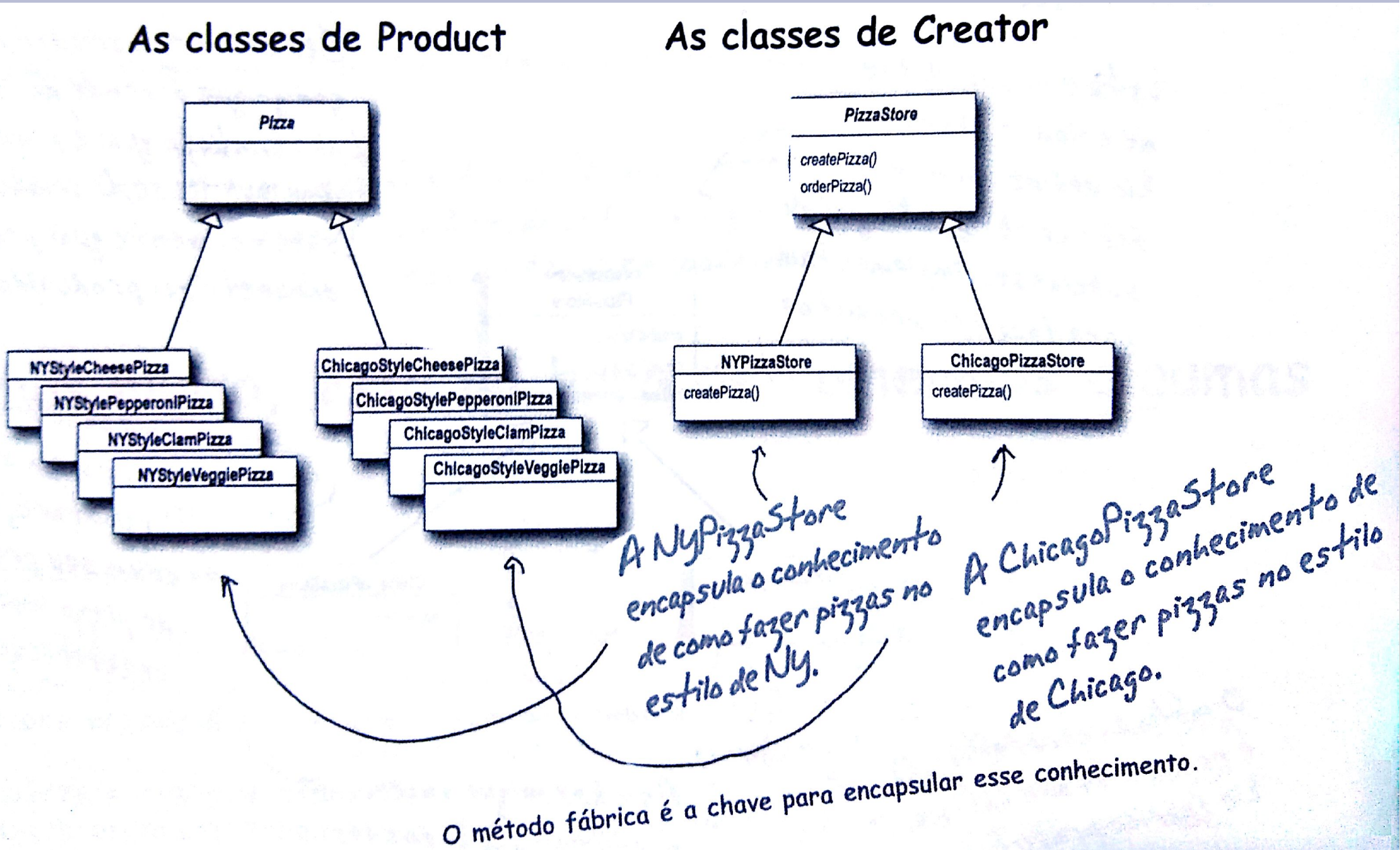
- O Factory Method é uma das técnicas mais eficazes para aplicar o princípio da inversão de dependência.

Factory Method



Freeman. Padrões de Projeto – Use a Cabeça.

Factory Method



Freeman. Padrões de Projeto – Use a Cabeça.

Factory Method

- Desta forma, os componentes de baixo nível e os de alto nível passam a depender de uma abstração

Factory Method

- Consequências importantes:
  - Fornece ganchos para subclasses.
    - * Criar objetos dentro de uma classe com um método fábrica é sempre **mais flexível** do que criar um objeto diretamente.
    - * Factory method dá às classes um **gancho** para que se possa implementar diferentes versões da fábrica.

Factory Method

- Implementação:

- Três variações principais:
  - (1) o caso em que a classe Creator é uma classe abstrata e não fornece uma implementação para o método fábrica que ele declara.
  - (2) o caso em que o Creator é uma classe concreta e fornece uma implementação default para o método fábrica.
  - (3) o caso em que o Creator é uma classe abstrata que define uma implementação default para o método fábrica.

Factory Method

- Exemplo:

```
public abstract class Game{
        public abstract Personagem createPersonagem(int tipo);

        public void Jogar(){
            //logica de execução (controller)
            Personagem p1 = createPersonagem(1);
            Personagem p2 = createPersonagem(2);
            p1.atarcar(p2);
            p1.correr();
            p2.atacar(p1);
        ...
}
```

Factory Method

```
public class TradicionalGame extends Game{

        public Personagem createPersonagem(int tipo){
         Personagem p = new Personagem();
         if (tipo == 1) {
            p.setAtaque(new AtaqueFraco());
            p.setCorrida(new CorridaRapida());
          } else if (tipo ==2) {
            p.setAtaque(new AtaqueForte());
            p.setCorrida(new CorridaLenta());
           }
         .....
         return p;
}
```

Factory Method

```
public class AdvancedGame extends Game{

        public Personagem createPersonagem(int tipo){
          Personagem p = new Personagem();
          if (tipo == 1) {
              p.setAtaque(new PoderFogo(new AtaqueFraco()));
              p.setCorrida(new CorridaRapida());
           } else if (tipo ==2) {
              p.setAtaque(new PoderVento(AtaqueForte()));
              p.setCorrida(new CorridaLenta());
            }
          .....
          return p;
}
```

Factory Method

O Padrão Factory Method define uma interface para criar um objeto, mas permite às classes decidir qual classe instanciar. O Factory Method permite a uma classe deferir a instanciação para subclasses.

Freeman. Padrões de Projeto – Use a cabeça.

Factory Method

- Conclusão
  - * Todos os padrões factory encapsulam a criação de objetos.
  - * O padrão factory method encapsula a criação de objetos deixando as subclasses decidirem quais objetos concretos criar.

Factory Method

- Fim!