Padrões de Projeto de Software Orientados a Objetos

Tecnologia em Análise e Desenvolvimento de Sistemas

Paulo Mauricio Gonçalves Júnior

Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco

22 de março de 2018

Parte I

Factory Method e Abstract Factory

Introdução I

- Quando usamos `new`, instanciamos classes concretas. Mas ligar nosso código para uma classe concreta pode deixá-lo mais frágil e menos flexível.

```
Duck duck = new MallardDuck();
```

- Se tivermos várias classes concretas, geralmente somos forçados a escrever código assim:

```
Duck duck;
if(picnic) {
  duck = new MallardDuck();
} else if(hunting) {
  duck = new DecoyDuck();
} else if(inBathTub) {
  duck = new RubberDuck();
}
```

Introdução II

- Se precisarmos modificar ou estender, precisaremos modificar o código acima.
- Nosso código então não está fechado para modificações

Exemplo I

- Imagine a implementação de uma pizzaria:

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

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

- Se tivermos mais de uma pizza:

Exemplo II

```
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();
  }

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

- Moveremos a parte que muda para um objeto que só se preocupa com criação de objetos. Esse objeto é chamado de Factory.

Exemplo III

```
public class SimplePizzaFactory {
  public Pizza createPizza(String type) {
    Pizza pizza = null;
    if(type.equals("cheese")) {
      pizza = new CheesePizza();
    } else if(type.equals("pepperoni")) {
      pizza = new PepperoniPizza();
    } else if(type.equals("clam")) {
      pizza = new ClamPizza();
    } else if(type.equals("veggie")) {
      pizza = new VeggiePizza();
    }
    return pizza;
  }
}
```

- Remodelando a classe PizzaStore

Exemplo IV

```
public class PizzaStore {
  SimplePizzaFactory factory;

  public PizzaStore(SimplePizzaFactory factory) {
    this.factory = factory;
  }

  Pizza orderPizza(String type) {
    Pizza pizza = factory.createPizza(type);

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

- Este é chamado de Simple Factory, não sendo um padrão em si.

Exemplo V

- Imagine que a pizzaria cresceu e possui filiais. Todas devem usar o mesmo código. Mas como lidar com diferenças regionais? Cada local possui diferentes estilos de pizza.
- Podemos criar fábricas específicas para cada local.

```
NYPizzaFactory nyFactory = new NYPizzaFactory();
PizzaStore nyStore = new PizzaStore(nyFactory);
nyStore.order("veggie");

ChicagoPizzaFactory chicagoFactory = new ChicagoPizzaFactory();
PizzaStore chicagoStore = new PizzaStore(chicagoFactory);
chicagoStore.order("veggie");
```

Factory Method I

- Para garantir que todas as filiais seguirão os mesmos passos na confecção das pizzas, faremos elas herdarem da classe PizzaStore:

```
public abstract class PizzaStore {

  abstract Pizza createPizza(String item);

  public Pizza orderPizza(String type) {
    Pizza pizza = createPizza(type);
    pizza.prepare();
    pizza.bake();
    pizza.cut();
    pizza.box();
    return pizza;
  }
}
```

- Quem instanciará as pizzas serão as filiais, através das subclasses de PizzaStore.

Factory Method II

- O método `orderPizza` na superclasse não sabe qual pizza será criada.

```
public class NYPizzaStore extends PizzaStore {

  Pizza createPizza(String item) {
    if (item.equals("cheese")) {
      return new NYStyleCheesePizza();
    } else if (item.equals("veggie")) {
      return new NYStyleVeggiePizza();
    } else if (item.equals("clam")) {
      return new NYStyleClamPizza();
    } else if (item.equals("pepperoni")) {
      return new NYStylePepperoniPizza();
    } else return null;
  }
}
```

- Um método de fábrica gerencia a criação de objetos e os encapsula na subclasse. Isso desacopla o código cliente na superclasse do código de criação do objeto na subclasse.

Factory Method III

Definição

O padrão Factory Method define uma interface para criação de objetos, but delega às subclasses decidir qual classe instanciar. Factory Method permite a uma classe delegar instanciação às subclasses.

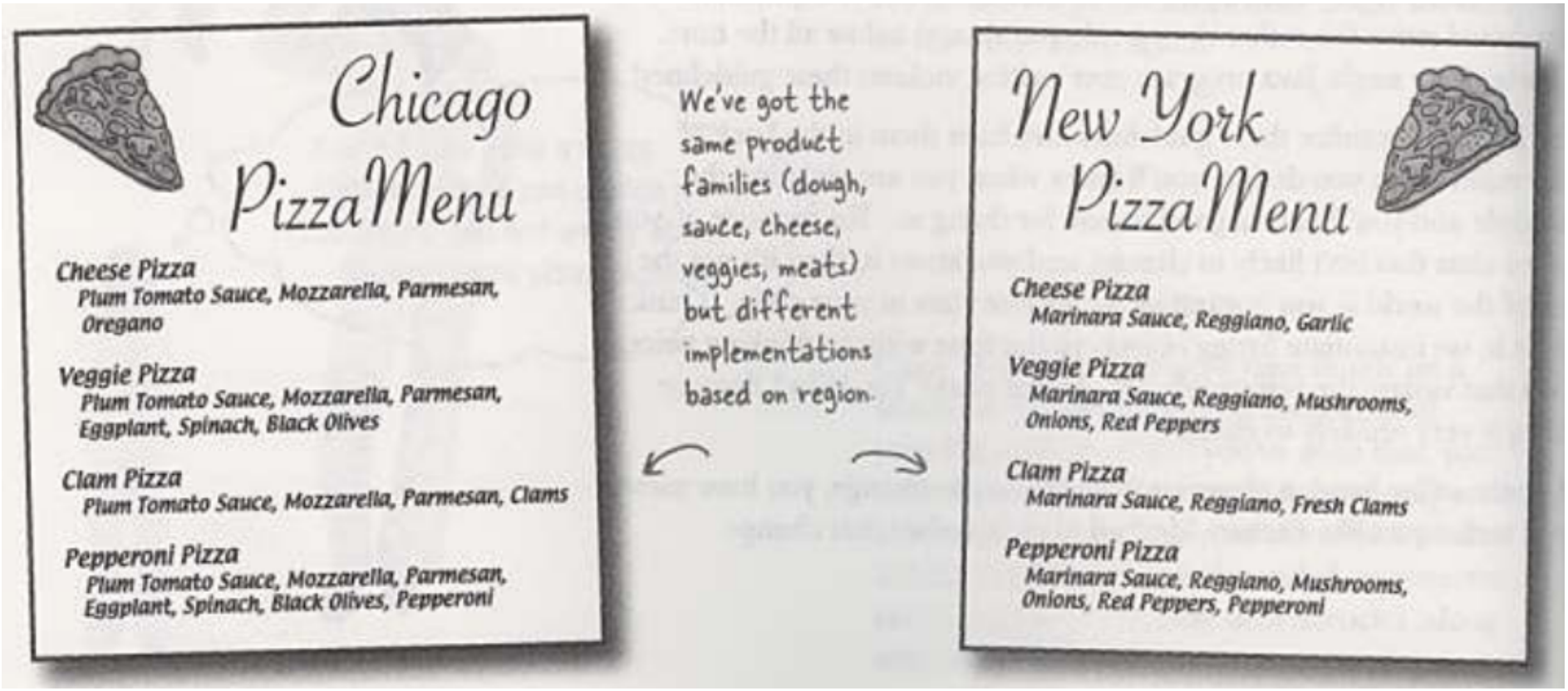
Princípio de Projeto

Princípio da Inversão de Dependências Dependa de abstrações e não de classes concretas.

- Este princípio sugere que os componentes de alto nível não deveriam depender de componentes de baixo nível. Ambos deveriam depender de abstrações.

Abstract Factory I

- Cada região usa uma família de ingredientes diferentes na confecção das pizzas.



Abstract Factory II

- Cada família de ingredientes consiste de um tipo de massa, molho, queijo e toppings.
- Inicialmente criaremos uma fábrica de ingredientes:

```
public interface PizzaIngredientFactory {
  public Dough createDough();
  public Sauce createSauce();
  public Cheese createCheese();
  public Veggies[] createVeggies();
  public Pepperoni createPepperoni();
  public Clams createClam();
}

public class NYPizzaIngredientFactory implements
    PizzaIngredientFactory {
  public Dough createDough() {
    return new ThinCrustDough();
  }
  public Sauce createSauce() {
    return new MarinaraSauce();
```

Abstract Factory III

```
    }
    public Cheese createCheese() {
      return new ReggianoCheese();
    }
    public Veggies[] createVeggies() {
      Veggies veggies[] = { new Garlic(), new Onion(), new Mushroom()
          , new RedPepper() };
      return veggies;
    }
    public Pepperoni createPepperoni() {
      return new SlicedPepperoni();
    }
    public Clams createClam() {
      return new FreshClams();
    }
}
```

- Agora modificaremos as pizzas para apenas usarem ingredientes gerados pela fábrica:

Abstract Factory IV

```
public abstract class Pizza {
  String name;

  Dough dough;
  Sauce sauce;
  Veggies veggies[];
  Cheese cheese;
  Pepperoni pepperoni;
  Clams clam;

  abstract void prepare();

  void bake() {
    System.out.println("Bake for 25 minutes at 350");
  }

  void cut() {
    System.out.println("Cutting the pizza into diagonal slices");
  }

  void box() {
```

Abstract Factory V

```
    System.out.println("Place pizza in official PizzaStore box");
  }

  void setName(String name) {
    this.name = name;
  }

  String getName() {
    return name;
  }

  public String toString() {
    // code to print pizza here
  }
}
```

- Agora criaremos as pizzas usando os ingredientes da fábrica:

Abstract Factory VI

```
public class CheesePizza extends Pizza {
  PizzaIngredientFactory ingredientFactory;

  public CheesePizza(PizzaIngredientFactory ingredientFactory) {
    this.ingredientFactory = ingredientFactory;
  }

  void prepare() {
    System.out.println("Preparing " + name);
    dough = ingredientFactory.createDough();
    sauce = ingredientFactory.createSauce();
    cheese = ingredientFactory.createCheese();
  }
}
```

- Finalmente, precisamos ajustar as lojas para garantir que estão criando as pizzas corretas:

Abstract Factory VII

```
public class NYPizzaStore extends PizzaStore {
  protected Pizza createPizza(String item) {
    Pizza pizza = null;
    PizzaIngredientFactory ingredientFactory = new
        NYPizzaIngredientFactory();
    if (item.equals("cheese")) {
      pizza = new CheesePizza(ingredientFactory);
      pizza.setName("New York Style Cheese Pizza");
    } else if (item.equals("veggie")) {
      pizza = new VeggiePizza(ingredientFactory);
      pizza.setName("New York Style Veggie Pizza");
    } else if (item.equals("clam")) {
      pizza = new ClamPizza(ingredientFactory);
      pizza.setName("New York Style Clam Pizza");
    } else if (item.equals("pepperoni")) {
      pizza = new PepperoniPizza(ingredientFactory);
      pizza.setName("New York Style Pepperoni Pizza");
    }
    return pizza;
  }
}
```

Abstract Factory VIII

Definição

O padrão Abstract Factory provê uma interface para criação de uma família de objetos relacionados ou dependentes sem especificar suas classes concretas.

- Comumente, os métodos de uma fábrica abstrata são implementados como Factory Methods.