Universidade Federal de Ouro Preto

CSI032 - Programação de Computadores II

Padrões de Projeto - Abstract Factory

Professor: Dr. Rafael Frederico Alexandre
Contato: rfalexandre@decea.ufop.br
Colaboradores: Eduardo Matias Rodrigues
Contato: eduardo.matias@aluno.ufop.edu.br

Padrões de Projeto - Abstract Factory

Abstract Factory

Padrões de projeto

Padrões GoF

Criação	Estrutural	Comportamental
Factory Method	Adapter	Interpreter
Abstract Factory	Bridge	Template Method
Builder	Composite	Chain of Responsibility
Prototype	Decorator	Command
Singleton	Façade	Iterator
	Flyweight	Mediator
	Proxy	Memento
		Observer
		State
		Strategy
		Visitor

Abstract Factory
Introdução

Intenção [GAMMA, 1994]

Fornecer uma interface comum para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

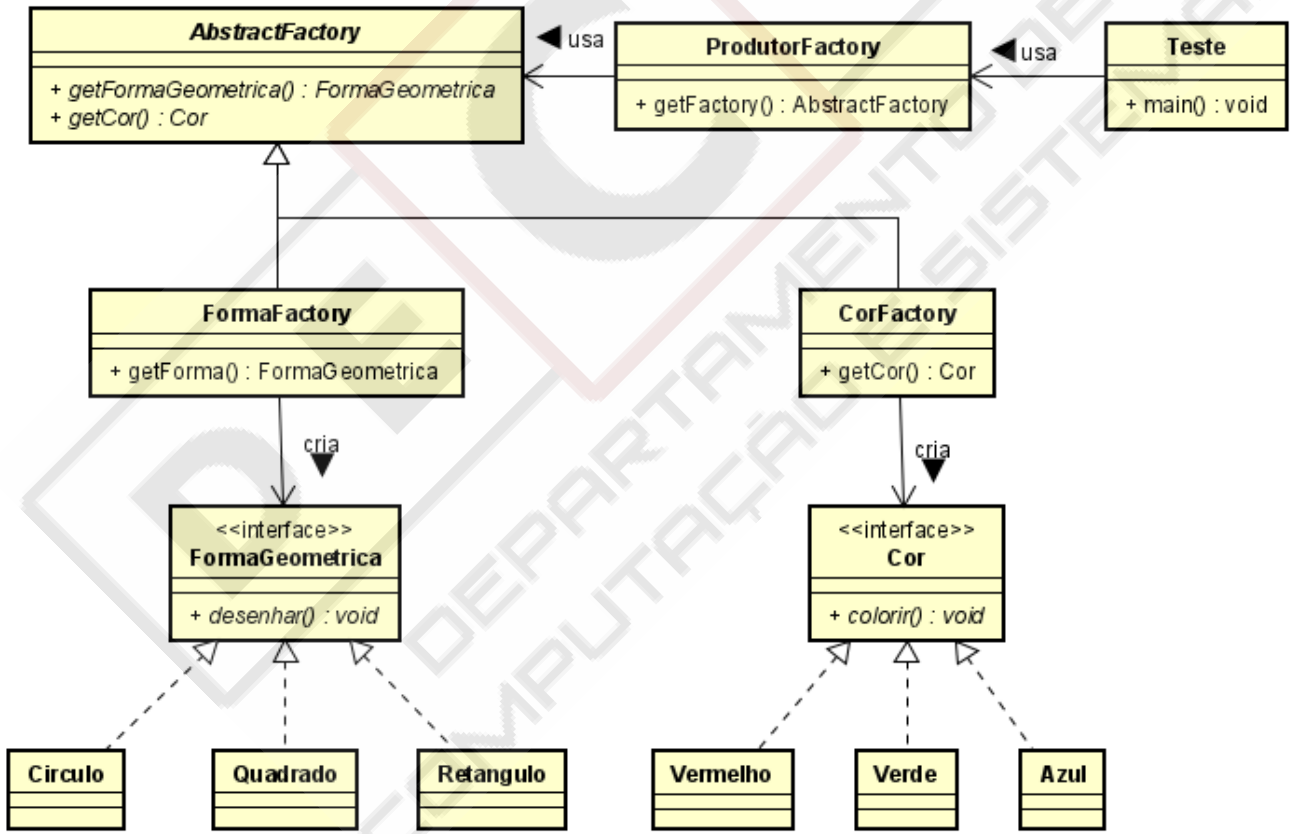
Solução [GAMMA, 1994]

Crie uma interface para representar uma fábrica para cada família de objetos. As subclasses concretas instanciam cada família específica. Baseia-se no uso de uma super-fábrica, responsável por criar outras fábricas.

Abstract Factory

Abstract Factory
Implementação

1. [de Lima, 2010] Segue um exemplo de utilização do padrão Abstract Factory:



Abstract Factory

Interface FormaGeometrica

```
public interface FormaGeometrica {

    void desenhar();

}
```

Abstract Factory

Formas geométricas

```
public class Circulo implements FormaGeometrica {

    @Override
    public void desenhar() {
        System.out.println("Dentro do método desenhar do Círculo");
    }
}

public class Retangulo implements FormaGeometrica {

    @Override
    public void desenhar() {
        System.out.println("Dentro do método desenhar do Retângulo");
    }
}

public class Quadrado implements FormaGeometrica {

    @Override
    public void desenhar() {
        System.out.println("Dentro do método desenhar do Quadrado");
    }
}
```

Abstract Factory

Interface Cor

```
public interface Cor {

    void colorir();

}
```

Abstract Factory

Cores

```
public class Vermelho implements Cor {

    @Override
    public void colorir() {
        System.out.println("Colorir de vermelho");
    }
}

  public class Verde implements Cor {

      @Override
      public void colorir() {
          System.out.println("Colorir de verde");
      }
  }

  public class Azul implements Cor {

      @Override
      public void colorir() {
          System.out.println("Colorir de azul");
      }
  }
```

Abstract Factory
Fábrica Abstrata

```
public abstract class AbstractFactory {

    public abstract Cor getCor(String cor);
    public abstract FormaGeometrica getFormaGeometrica(String forma);

}
```

Abstract Factory

Fábrica de formas geométricas

```
public class FormaFactory extends AbstractFactory {

    @Override
    public FormaGeometrica getFormaGeometrica(String forma) {
        if (forma == null)
            return null;
        if (forma.equalsIgnoreCase("circulo"))
            return new Circulo();
        else if (forma.equalsIgnoreCase("retangulo"))
            return new Retangulo();
        else if (forma.equalsIgnoreCase("quadrado"))
            return new Quadrado();
        return null;
    }

    @Override
    public Cor getCor(String cor) {
        return null;
    }
}
```

Abstract Factory

Fábrica de cores

```
public class CorFactory extends AbstractFactory {

    @Override
    public Cor getCor(String cor) {
        if (cor == null)
            return null;
        if (cor.equalsIgnoreCase("vermelho"))
            return new Vermelho();
        else if (cor.equalsIgnoreCase("verde"))
            return new Verde();
        else if (cor.equalsIgnoreCase("azul"))
            return new Azul();
        return null;
    }

    @Override
    public FormaGeometrica getFormaGeometrica(String forma) {
        return null;
    }
}
```

Abstract Factory

Produtor de fábricas

```
public class ProdutorFactory {

    public static AbstractFactory GetFactory(String fabrica) {
        if (fabrica.equalsIgnoreCase("cor"))
            return new CorFactory();
        else if (fabrica.equalsIgnoreCase("forma"))
            return new FormaFactory();
        return null;
    }
}
```

Abstract Factory

Teste, parte 1

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

    AbstractFactory formaFactory = ProdutorFactory.GetFactory("forma");

    FormaGeometrica forma1 = formaFactory.getFormaGeometrica("circulo");
    forma1.desenhar();

    FormaGeometrica forma2 = formaFactory.getFormaGeometrica("quadrado");
    forma2.desenhar();

    FormaGeometrica forma3 = formaFactory.getFormaGeometrica("retangulo");
    forma3.desenhar();
```

Abstract Factory

Teste, parte 2

```
AbstractFactory corFactory = ProdutorFactory.GetFactory("cor");

Cor cor1 = corFactory.getCor("verde");
cor1.colorir();

Cor cor2 = corFactory.getCor("azul");
cor2.colorir();

Cor cor3 = corFactory.getCor("vermelho");
cor3.colorir();
```

Abstract Factory

Saída do programa

```
run:
Dentro do método desenhar do Circulo
Dentro do método desenhar do Quadrado
Dentro do método desenhar do Retângulo
Colorir de verde
Colorir de azul
Colorir de vermelho
BUILD SUCCESSFUL (total time: 0 seconds)
```

Abstract Factory

Abstract Factory

Consequências [GAMMA, 1994]

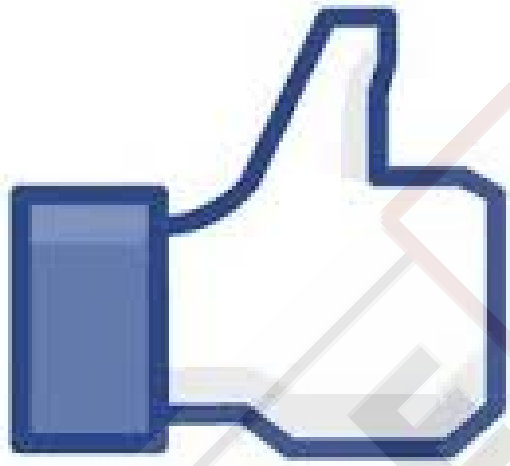
1. Isola as classes concretas: uma vez que as fábricas encapsulam o processo de criação de objetos, os clientes são isolados das classes de implementação;
2. Torna fácil a troca de familia de produtos: se quisermos trocar o tipo de produto criado, basta trocar a fábrica concreta (no nosso exemplo: **FormaFactory** e **CorFactory**);
3. Difícil suportas novos produtos: e se quisermos criar outro produto além de cores e formas geométricas? Teríamos que alterar a classe AbstractFactory e todas as suas filhas, o que é um processo trabalhoso.

Abstract Factory

Abstract Factory

Exercício

1. Utilize o padrão de projeto Abstract Factory para a criação de dois tipos de carros: de luxo ou popular. Para simplificação, considere que nossos carros têm apenas motor e rodas, um carro de luxo possui motor esportivo e rodas de liga leve enquanto que um carro popular possui motor simples e rodas simples;
2. Crie uma classe de teste para sua aplicação. Na classe de teste deve ser informado apenas qual tipo de carro será criado, de luxo ou popular;
3. Adicione os atributos e métodos que julgar necessários;



MUITO OBRIGADO!!!

Referências Bibliográficas I

de Lima, E. S. (2010).
Aula 06 – padrões gof (factory method e abstract factory).
http://edirlei.3dgb.com.br/index.php?option=com$_c$*ontentview* = *articleid* = 103*Itemid* = 123.

GAMMA, Erich. HELM, R. J. R. V. J. (1994).
Padrões de Projeto - Soluções reutilizaveis de software orientado a objetos.
1994 M. C. Escher / Gordon Art - Baam - Holland, São Paulo, Brasil, 1st. edition.