+Devs2Blu

Linguagem de programação JAVA

Profª. Heloisa Moura

Padrões de projetos (Design Patterns)

Padrões Criacionais

Abstract Factory.

Padrões de projetos (Design Patterns)

Problema

- Como posso escrever um código onde as classes instanciadas possam variar dentro de uma mesma interface?
- Como garantir que um conjunto de objetos relacionados (ou dependentes) possam ser criados mantendo o contexto único?

Solução Padrões de projetos: Abstract Factory

- Extrair a lógica de criação dos objetos para um abstract factory;
- Criar uma implementação do abstract factory para cada contexto, garantindo que todos os objetos criados estejam relacionados.

Solução Padrões de projetos: Abstract Factory

Definição: A ideia principal do Abstract Factory é definir uma interface que será responsável pela criação de objetos de um grupo relacionado, mas sem determinar a classe concreta dos objetos que serão criados. Assim, cada família de objetos será implementada por uma classe concreta específica.

- Esse padrão é útil quando temos diversas variações de objetos que precisam ser criados de forma organizada, dependendo de um contexto.

Padrões de projetos Abstract Factory

Exemplo Prático: Imagine que estamos desenvolvendo um software para renderizar botões e checkboxes em diferentes temas de interface gráfica (Windows e MacOS).
Precisamos de duas famílias de produtos:

- Botões (Button)
- Checkboxes (Checkbox)

Cada tema (Windows ou MacOS) precisa de sua própria implementação de botões e checkboxes, mas o cliente (o código que usa esses objetos) não deve se preocupar com qual tema está sendo utilizado.

Padrões de projetos Abstract Factory

Estrutura do padrão

1. **Criação das Interfaces dos Produtos**

Produto : **Interface** Button

método: paint()

Produto : **Interface** Checkbox

método: paint()

Padrões de projetos Abstract Factory

2. **Implementação dos Produtos Concretos**

Produto concreto: **Classe concreta** WindowsButton

implementa: Button

método: paint()

Produto concreto: **Classe concreta** MacOSButton

implementa: Button

método: paint()

Padrões de projetos Abstract Factory

3. **Criação da Interface Abstract Factory**

Fabrica do Produtos: **Interface** GUIFactory

método: Button createButton();

método: Checkbox createCheckbox();

Padrões de projetos Abstract Factory

4. Implementação das Fábricas Concretas

Fabrica concreta: **Classe concreta** WindowsFactory

implementa: GUIFactory

método: createButton ()

método: createCheckbox ()

Padrões de projetos Abstract Factory

4. Implementação das Fábricas Concretas

Fabrica concreta: **Classe concreta** MacOSFactory

implementa: GUIFactory

método: createButton ()

método: createCheckbox ()

Padrões de projetos Abstract Factory

5. Implementação do Cliente

Classe concreta ClientApplication

Construtor: Application(GUIFactory factory)
método: paint()

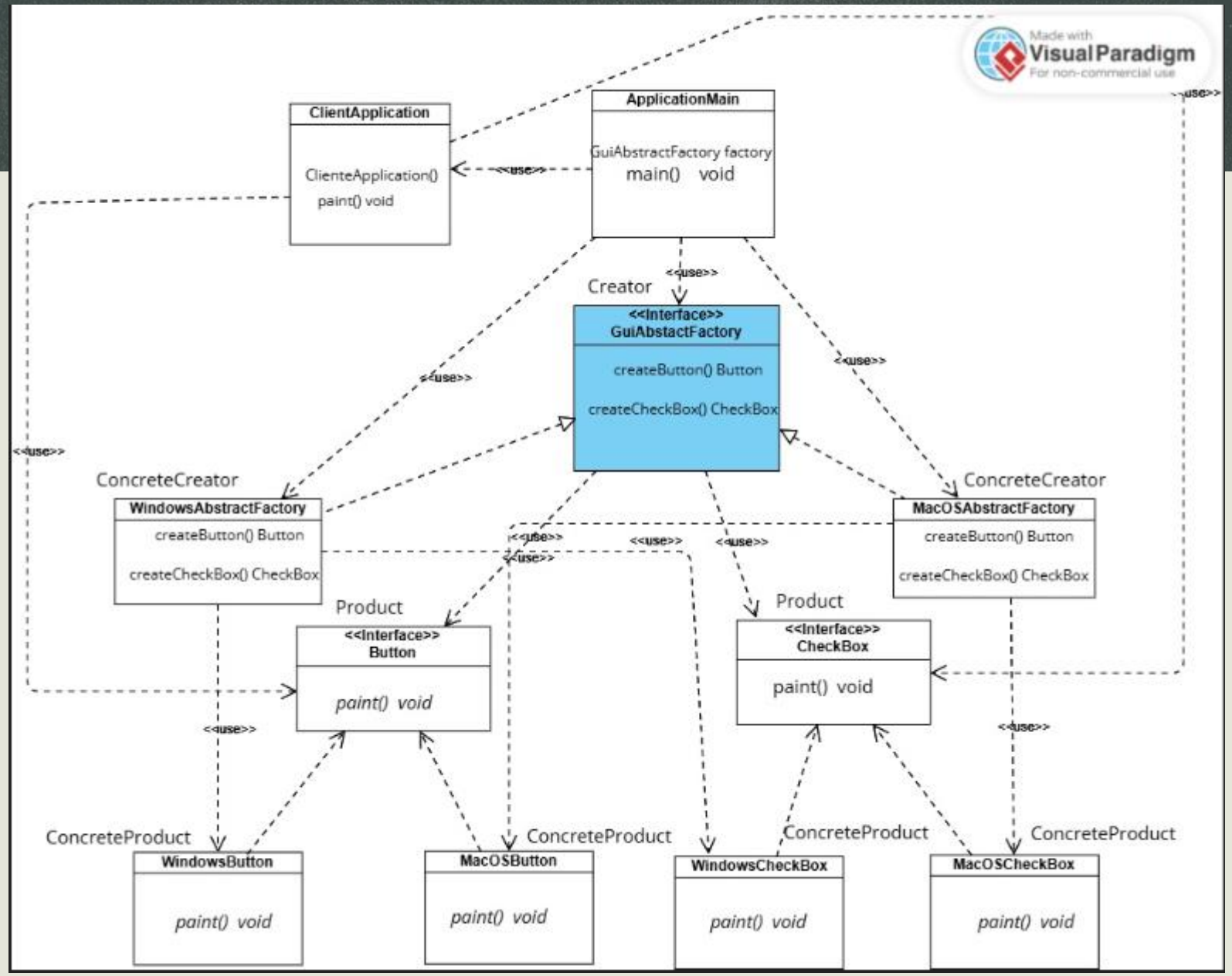
6. Configuração da Fábrica a ser Usada

Utilização da Fabrica: **Classe** ApplicationMain

+Devs2Blu

Fundamentos avançados OOP

Diagrama de classes



A implementação em Java

```java
//Interface do produto Button
public interface Button {
 void paint();
}
```

```java
//Interface do produto Checkbox
public interface CheckBox {
 void paint();
}
```

```java
//Implementação do produto botão para Windows
public class WindowsButton implements Button {
 @Override
 public void paint() {
     System.out.println("Botão estilo Windows");
 }
}
```

```java
//Implementação do checkbox para Windows
public class WindowsCheckBox implements CheckBox {
 @Override
 public void paint() {
     System.out.println("Checkbox estilo Windows");
 }
}
```

```java
//Implementação do produto botão para MacOS
public class MacOSButton implements Button {
 @Override
 public void paint() {
     System.out.println("Botão estilo MacOS");
 }
}
```

```java
//Implementação do produto checkbox para MacOS
public class MacOSCheckBox implements CheckBox {
 @Override
 public void paint() {
     System.out.println("Checkbox estilo MacOS");
 }
}
```

A implementação em Java

```java
// abstract factory ou criador
public interface GUIAbstractFactory {
    Button createButton();
    CheckBox createCheckbox();
}
```

```java
//Fábrica concreta para Windows criador concreto
public class WindowsAbstractFactory implements GUIAbstractFactory {
 @Override
 public Button createButton() {
     return new WindowsButton();
 }

 @Override
 public CheckBox createCheckbox() {
     return new WindowsCheckBox();
 }
}
```

```java
//Fábrica concreta para MacOS criador concreto
public class MacOSAbstractFactory implements GUIAbstractFactory {
 @Override
 public Button createButton() {
     return new MacOSButton();
 }

 @Override
 public CheckBox createCheckbox() {
     return new MacOSCheckBox();
 }
}
```

A implementação em Java

```
//Implementação do Cliente
public class ClientApplication {
    private Button button;
    private CheckBox checkbox;

    public ClientApplication(GUIFactory factory) {
        button = factory.createButton();
        checkbox = factory.createCheckbox();
    }

    public void paint() {
        button.paint();
        checkbox.paint();
    }
}
```

A implementação em Java

```
// Configuração da Fábrica a Ser Usada
// Em um ambiente real, a fábrica a ser usada seria selecionada
// dinamicamente com base no ambiente.
public class ApplicationMain {
    public static void main(String[] args) {
        GUIFactory factory;

        // Escolha do sistema operacional
        String osName = System.getProperty("os.name").toLowerCase();

        if (osName.contains("win")) {
            factory = new WindowsFactory();
        } else if (osName.contains("mac")) {
            factory = new MacOSFactory();
        } else {
            throw new UnsupportedOperationException("Sistema operacional não suportado.");
        }

        ClientApplication app = new ClientApplication(factory);
        app.paint();
    }
}
```

Vamos ver na IDE

Padrões de projetos Abstract Factory

Quando Usar Cada Padrão

- **Factory Method**:
  - Ideal para quando há uma única hierarquia de objetos e queremos delegar a responsabilidade de criação para subclasses.
  - É adequado para quando o tipo de produto pode variar, mas ainda é um único produto.
  - O foco está na criação de um único produto, onde subclasses especificam o objeto a ser criado.

- **Abstract Factory**:
  - É útil para quando precisamos criar famílias de objetos relacionados ou dependentes, de modo que esses objetos possam trabalhar juntos.
  - Ideal para aplicações que precisam de grupos de produtos relacionados, como em uma interface de usuário que altera o tema.
  - É mais robusto e organizado em casos que exigem múltiplos produtos relacionados.

Diferenças entre os padrões Factory

Característica	Factory Method	Abstract Factory
Criação de Objetos	Cria um único produto por vez	Cria famílias de produtos relacionados
Objetivo Principal	Delegar a criação de um único objeto às subclasses	Garantir que famílias de produtos compatíveis sejam criadas juntas
Complexidade	Menos complexo	Mais complexo
Exemplo de Uso	Documentos (Word, PDF, etc.)	Interface gráfica (Windows, MacOS com botões, checkboxes, etc.)

Resumo

Em resumo, o Factory Method é mais simples e serve para criar um único produto, enquanto o Abstract Factory é mais avançado e permite a criação de famílias de produtos que precisam funcionar bem juntos.

Padrões de projetos Abstract Factory

Exercícios