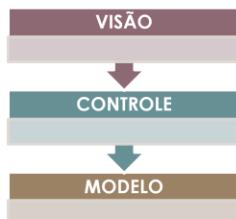
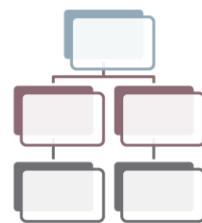


# Arquitetura e Desenho de Software

## AULA 11



Profa. Milene Serrano



# Agenda



Considerações Iniciais

Padrões GoF

\* Categoria: de Criação ou Criacionais

Considerações Finais

## Considerações Iniciais



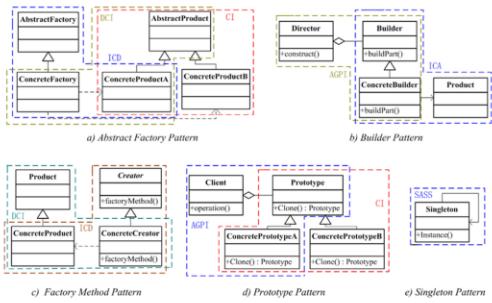
## Padrões GoF



Uma solução consolidada para um problema recorrente no desenvolvimento e manutenção de software orientado a objetos.

- **Referência Relevante:** Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides. Design Patterns: Elements os Reusable Object-Oriented Software. Addison-Wesley, 1995.
- **Gang of Four - GoF**

# GoF Criação



## Padrões de Projeto – GoFs Criacionais



Em um sistema OO, a criação de objetos não é uma tarefa fácil. Destacam-se, nesse contexto, pelo menos dois problemas principais:

- Definir qual classe concreta deve ser utilizada para criar o objeto, e
- Definir como os objetos devem ser criados e como eles se relacionam com outros objetos do sistema.



Seguindo o princípio do encapsulamento, essa complexidade deve ser, preferencialmente, isolada.

## Padrões de Projeto – GoFs Criacionais

### Principais Gofs Criacionais

#### Factory Method

Encapsular a escolha da classe concreta a ser utilizada na criação de objetos de um determinado tipo.

#### Abstract Factory

Encapsular a escolha das classes concretas a serem utilizadas na criação de objetos de diversas famílias.

#### Builder

Separar o processo de construção de um objeto de sua apresentação e permitir a sua criação passo a passo. Diferentes tipos de objetos podem ser criados com implementações distintas de cada passo.

Ao final dessa aula, em **Complementar**, têm detalhes sobre vários desses padrões...

## Padrões de Projeto – GoFs Criacionais

### Principais Gofs Criacionais

Prototype

Singleton

Multiton

Object Pool

Possibilitar a criação de novos objetos a partir da cópia de objetos existentes.

Permitir a criação de uma única instância de uma classe e fornecer um modo de recuperá-la.

Permitir a criação de uma quantidade limitada de instâncias de determinada classe e fornecer um modo para recuperá-las.

Possibilitar o reaproveitamento de objetos.

Alguns padrões são vistos como *anti-patterns*...

Ao final dessa aula, em **Complementar**, têm detalhes sobre vários desses padrões...

## GoF Criacional Factory Method

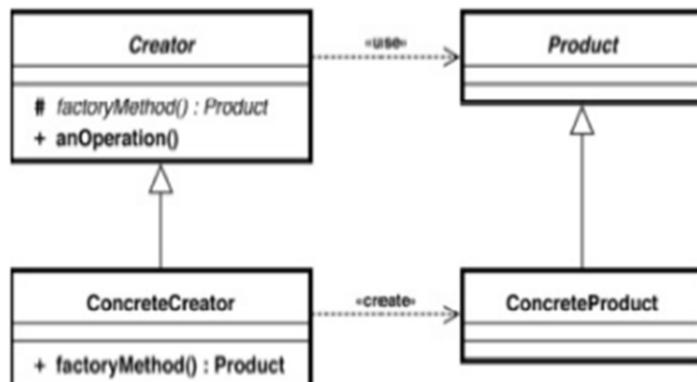


## GoF Criacional – Factory Method

### Objetivo

- Permitir delegar a instanciação às subclasses...

## GoF Criacional – Factory Method



Modelagem Genérica

## GoF Criacional – Factory Method

### Participantes

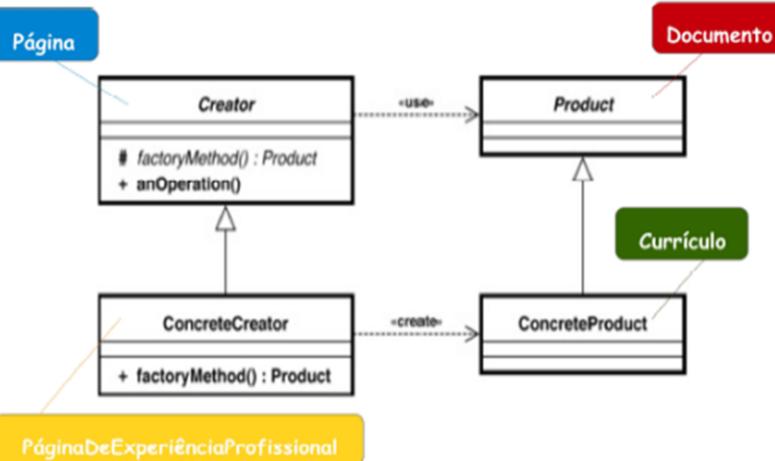
- **Product** (ex. Documento) – define atributos e métodos, esses últimos abstratos ou programados de forma mais genérica, para um conjunto de objetos, os quais serão criados pelo *factory method*.
- **ConcreteProduct** (ex. Relatório, Currículo) – estende *Product*, especializando o que foi definido na superclasse.

## GoF Criacional – Factory Method

### Participantes

- **Creator** (ex. Página) – declara o factory method, o qual retorna um objeto do “tipo” Product. Creator pode também definir uma implementação default do factory method que retorna um objeto ConcreteProduct default. Creator pode ainda chamar um factory method para criar um objeto de Product.
  - **ConcreteCreator** (ex. PáginaDeExperiênciaProfissional, PáginaDeHabilidades, PáginaDeFormaçãoAcadêmica) – sobrescreve o factory method para retornar uma instância de ConcreteProduct.

## GoF Criacional – Factory Method



## GoF Criacional – Factory Method

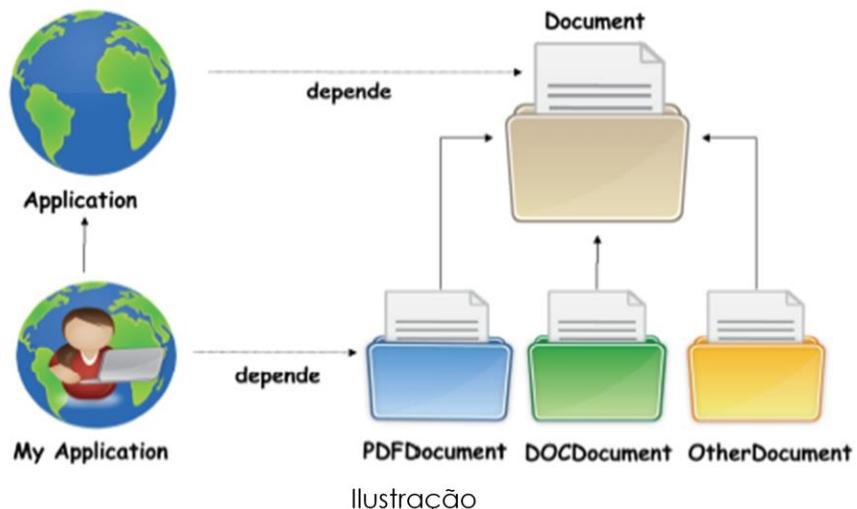
### MOTIVAÇÃO

Considere uma aplicação que precisa realizar operações diversas (**enviarDocumentos**, **criarDocumentos**, **abrirDocumentos** e outras).

Se esse documento for um pdf, existem formas específicas para viabilizar a abertura, o fechamento bem como a gravação desse tipo de documento.

Por outro lado, se for doc, novamente, existem formas específicas para viabilizar a abertura, o fechamento bem como a gravação desse tipo de documento.

## GoF Criacional – Factory Method



## GoF Criacional – Factory Method

SOLUÇÃO

Podemos empregar o padrão de projeto GoF, criacional, **Factory Method**.

Como seria a implementação em Java?



## GoF Criacional – Factory Method

Application.java

```
Application.java X
public abstract class Application {
    protected Document document;
    /**
     * Factory Method Abstraction, returning a product
     */
    protected abstract Document createDocument(String codeDoc);
    public void sendDocument(){
        System.out.println("Sending document by e-mail: " + this.document.documentCode);
    }
}
```

Possível Implementação

## GoF Criacional – Factory Method

### MyApplication.java

```
J *MyApplication.java ✘
public class MyApplication extends Application {
    //Factory Method possible implementation.
    protected Document createDocument(String CodeDoc) {
        super.document = new MyPDFDocument(CodeDoc);
        return super.document;
    }
}
```

Possível Implementação

## GoF Criacional – Factory Method

### Document.java

```
Document.java ✘
public abstract class Document {
    String documentCode;
    public Document(String docCode) {
        this.documentCode = docCode;
    }
    public abstract void open();
    public abstract void close();
    public abstract void record();
}
```

Possível Implementação

## GoF Criacional – Factory Method

### MyPDFDocument.java

```
MyPDFDocument.java
public class MyPDFDocument extends Document {
    public MyPDFDocument(String docCode) {
        super(docCode);
    }
    public void open() {
        System.out.println("My PDF Document is opened!");
    }
    public void close() {
        System.out.println("My PDF Document is closed!");
    }
    public void record() {
        System.out.println("My PDF Document is recorded!");
    }
}
```

Possível Implementação

## GoF Criacional – Factory Method

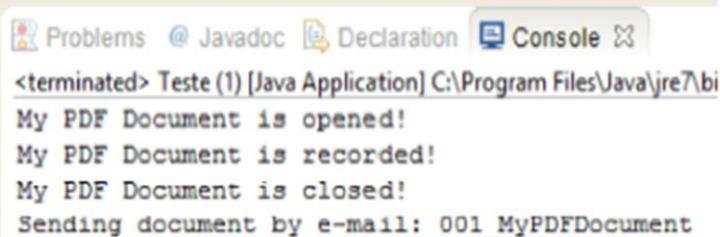
### Teste.java

```
Teste.java ━━
public class Teste {
    public static void main(String[] args) {
        MyApplication myApplication = new MyApplication();
        Document mydocument = myApplication.createDocument("001_MyPDFDocument");
        mydocument.open();
        mydocument.record();
        mydocument.close();
        myApplication.sendDocument();
    }
}
```

Possível Implementação

## GoF Criacional – Factory Method

Executando...

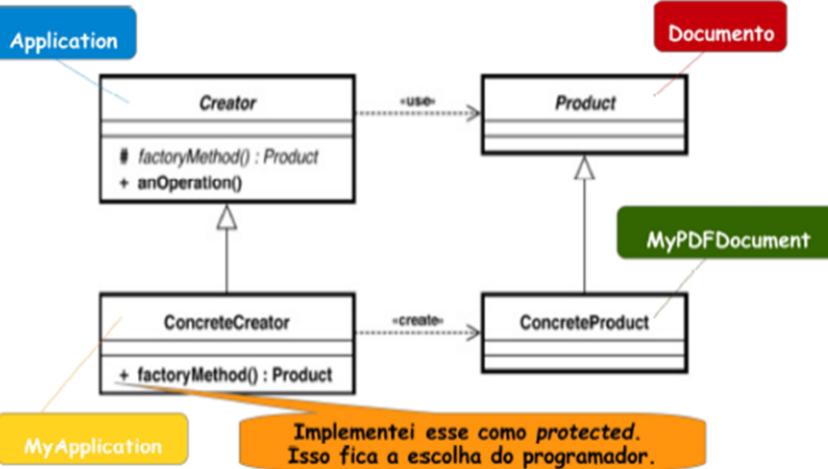


The screenshot shows a Java application window titled "Teste (1) [Java Application]". The "Console" tab is selected, displaying the following output:

```
<terminated> Teste (1) [Java Application] C:\Program Files\Java\jre7\bin  
My PDF Document is opened!  
My PDF Document is recorded!  
My PDF Document is closed!  
Sending document by e-mail: 001_MyPDFDocument
```

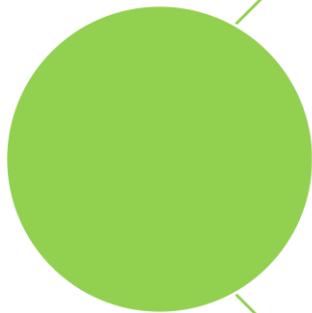
Possível Implementação

## GoF Criacional – Factory Method



Modelagem Aplicada/Instaciada

## GoF Criacional – Factory Method



### Vantagens

Se optarmos por ampliar os documentos aceitos, trabalhando com doc, ppt e outros? Como fica?

- Basta acrescentar mais **ConcreteProduct(s)**, sem alterar as demais classes envolvidas, apenas criando instâncias para esses **ConcreteProduct(s)** no método `createDocument()` de **MyApplication**.

Portanto, ganhamos em reutilização, manutenibilidade, coesão, dentre outros aspectos relevantes para a programação de sistemas OO.

GoF Criacional  
Abstract Factory

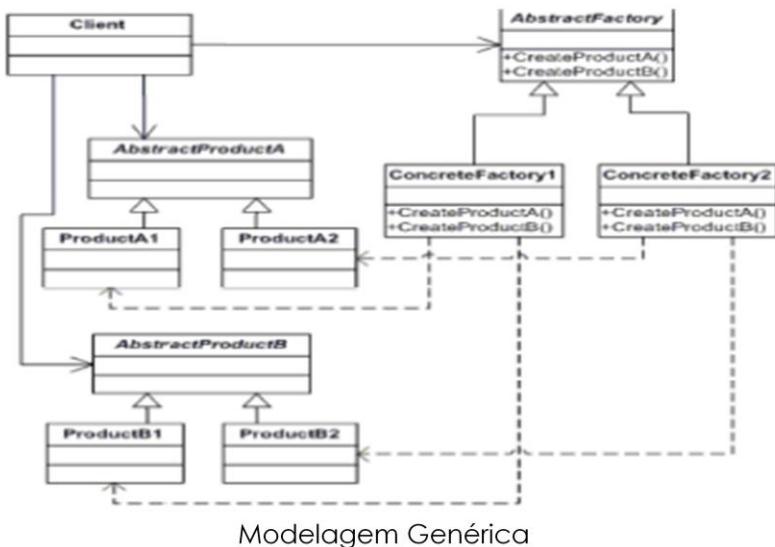


## GoF Criacional – Abstract Factory

### Objetivo

- Permitir a criação de famílias de objetos relacionados ou dependentes por meio de uma única interface e sem que a classe concreta seja especificada.

## GoF Criacional – Abstract Factory



## GoF Criacional – Abstract Factory

### Participantes

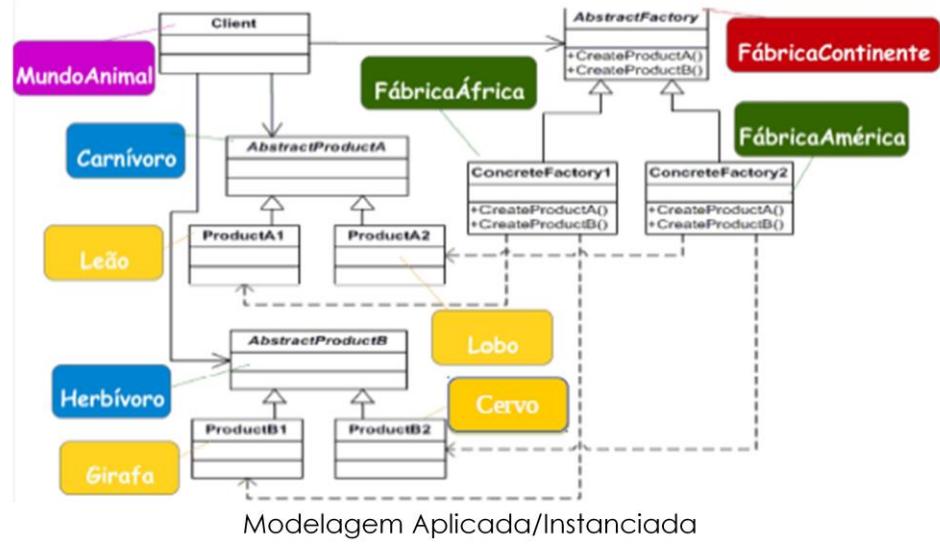
- **AbstractFactory**(ex. FábricaContinente) – declara uma interface para operações de criação de produtos quaisquer.
- **ConcreteFactory** (ex. FábricaÁfrica, FábricaAmérica)  
– implementa as operações que criam objetos de produtos concretos (específicos).

## GoF Criacional – Abstract Factory

### Participantes

- **AbstractProduct** (ex. Herbívoro, Carnívoro) – declara uma interface para um tipo de objeto produto.
- **Product** (ex. Leão, Lobo) – define um objeto produto a ser criado pela fábrica concreta correspondente, a qual implementa a interface declarada em *AbstractProduct*.
- **Client** (ex. MundoAnimal) – usa as interfaces declaradas em *AbstractFactory* e *AbstractProduct*.

## GoF Criacional – Abstract Factory



## GoF Criacional – Abstract Factory

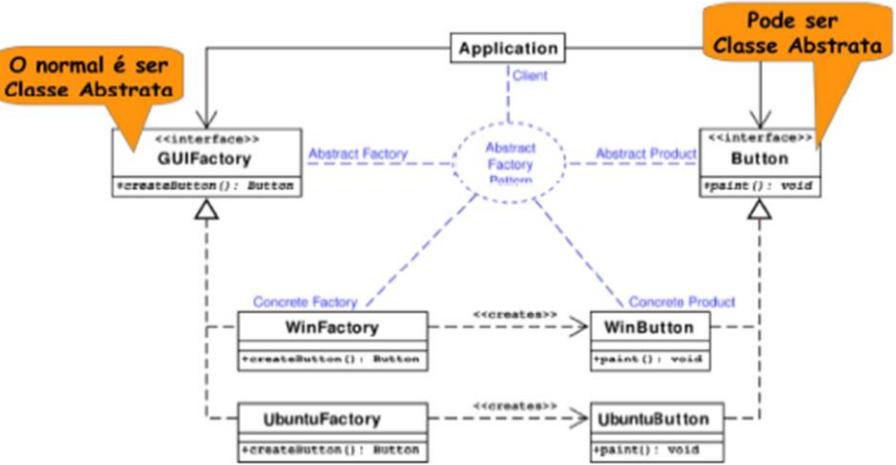
### MOTIVAÇÃO

Considere uma aplicação que precisa trabalhar com interfaces gráficas para diferentes sistemas operacionais bem como cada sistema operacional possui elementos gráficos específicos (botões para Windows e botões para Ubuntu).

Portanto, se o sistema operacional for Windows, a interface gráfica e seus elementos devem ser compatíveis com esse sistema operacional.

O mesmo deve ocorrer, caso o sistema operacional em foco seja o Ubuntu.

## GoF Criacional – Abstract Factory



Ilustração

## GoF Criacional – Abstract Factory

SOLUÇÃO

Podemos empregar o padrão de projeto GoF, criacional, **Abstract Factory**.

Como seria a implementação em Java?



## GoF Criacional – Abstract Factory

GUIFactory.java

```
GUIFactory.java
public abstract class GUIFactory {
    public static GUIFactory defineFactory() {
        if( Configuration.getActualOperationalSystem() == "Windows" ) {
            return new WinFactory();
        }
        else {
            return new UbuntuFactory();
        }
    }
    public abstract Button createButton();
}
```

Simulando  
que obtive o  
sistema operacional  
vigente...

Posso melhorar essa  
comparação com métodos  
específicos para esse fim...

Possível Implementação

## GoF Criacional – Abstract Factory

### Configuration.java

```
Configuration.java ✘  
public class Configuration {  
    public static String getActualOperationalSystem() {  
        return "Ubuntu";  
    }  
}
```

A apenas simulando qual seria a configuração atual, em qual sistema operacional...

Possível Implementação

## GoF Criacional – Abstract Factory

**Button.java**

```
Button.java ✘  
public interface Button {  
    public void paint();  
}
```

Possível Implementação

## GoF Criacional – Abstract Factory

WinButton.java

```
WinButton.java ✘  
public class WinButton implements Button {  
    @Override  
    public void paint() {  
        System.out.println("Render a button in a Windows Operational System's style");  
    }  
}
```

Possível Implementação

## GoF Criacional – Abstract Factory

### UbuntuButton.java

```
*UbuntuButton.java */
public class UbuntuButton implements Button{
    @Override
    public void paint() {
        System.out.println("Render a button in a Linux/Ubuntu Operational System's style");
    }
}
```

Possível Implementação

## GoF Criacional – Abstract Factory

WinFactory.java

```
WinFactory.java
public class WinFactory extends GUIFactory{
    @Override
    public Button createButton() {
        // TODO Auto-generated method stub
        return new WinButton();
    }
}
```

Possível Implementação

## GoF Criacional – Abstract Factory

UbuntuFactory.java

```
UbuntuFactory.java ✘
public class UbuntuFactory extends GUIFactory{
    @Override
    public Button createButton() {
        // TODO Auto-generated method stub
        return new UbuntuButton();
    }
}
```

Possível Implementação

## GoF Criacional – Abstract Factory

### ClientApplication.java

```
J ClientApplication.java 23
public class ClientApplication {

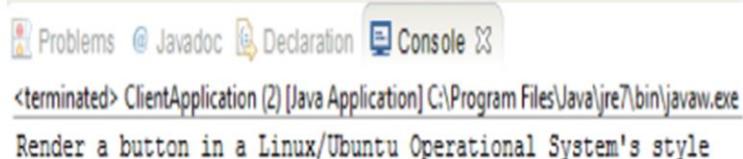
    public static void main(String[] args) {
        GUIFactory factory = GUIFactory.defineFactory();

        Button button = factory.createButton();
        button.paint();
    }
}
```

Possível Implementação

## GoF Criacional – Abstract Factory

Executando...



A screenshot of a Java IDE's interface, specifically showing the 'Console' tab. The title bar includes 'Problems', '@ Javadoc', 'Declaration', and 'Console'. The console window shows the output of a Java application named 'ClientApplication'. The output text is: '<terminated> ClientApplication [2] [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe' followed by 'Render a button in a Linux/Ubuntu Operational System's style'.

Possível Implementação

## GoF Criacional – Abstract Factory

### Vantagens

Se optarmos por ampliar os elementos gráficos, com **toolBars**, por exemplo? Como fica?

- Basta acrescentar mais **WinToolBar** e **UbuntuToolBar** como concretas de uma classe abstrata ou interface **ToolBar**. Em **GUIFactory**, acrescentar um método abstrato `createToolBar`, que retorna **ToolBar**. Em **WinFactory**, concretizar esse método, criando uma instância de **WinToolBar**. Em **UbuntuFactory**, concretizar esse método, criando uma instância de **UbuntuToolBar**. Finalmente, basta a **ClientApplication** usar!

# Extra Classe

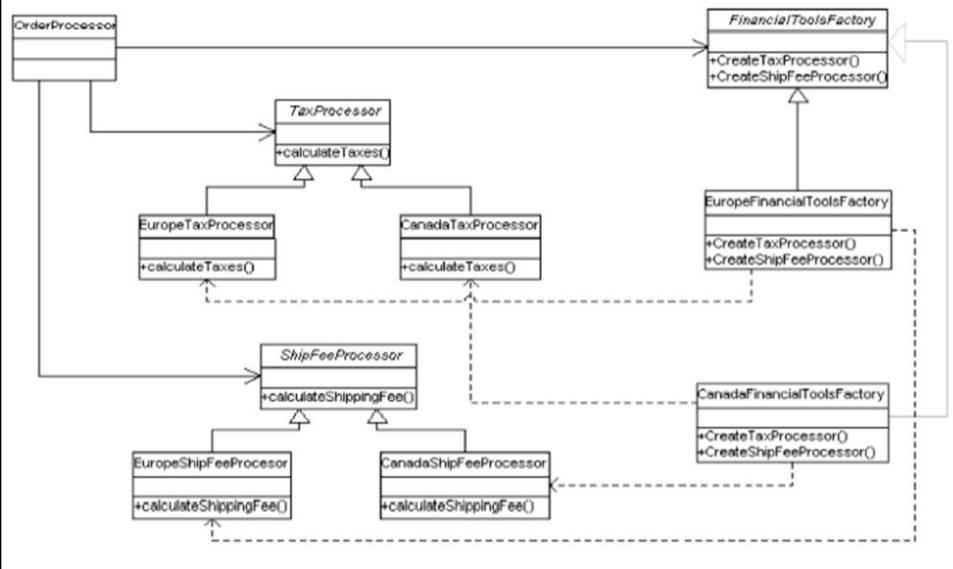


## Extra Classe

Implemente o *Abstract Factory*, de acordo com a modelagem apresentada no próximo slide.

- Cuidado! Não esqueça os participantes envolvidos para adequada implementação do *Abstract Factory*!

## Extra Classe



Exemplo Modelado no Contexto Financeiro...

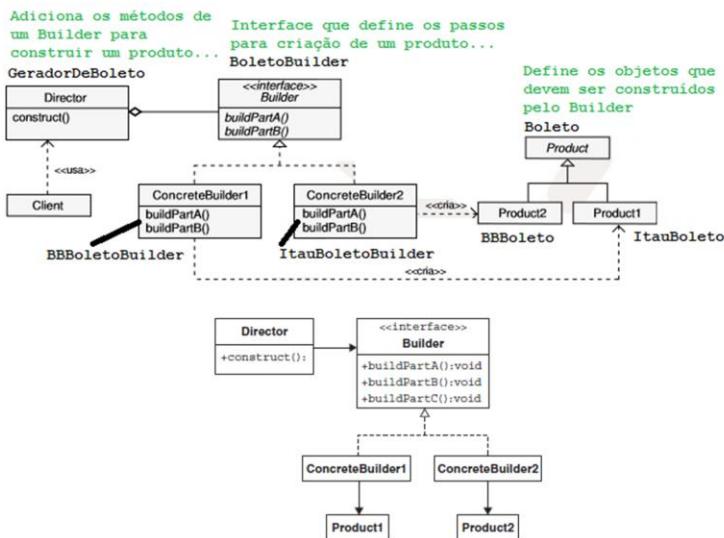
# Complementar



## Complementar

- Seguem *slides* com os demais padrões GoFs criacionais.
- Procurem implementá-los.
- Qualquer dúvida, entrem em contato comigo ou com os nossos monitores.

## Builder - Modelagem



Possíveis modelagens...

Ilustrado com *GeradorDeBoleto*, *BoletoBuilder*, *BBBoletoBuilder*, *ItauBoletoBuilder*, *Boleto*, *BBBoleto* e *ItauBoleto*.

Poderia ser, conforme colocado na implementação disponível no próximo slide:  
*Waiter*, *PizzaBuilder*, *HawaiianPizzaBuilder*, *SpicyPizzaBuilder* e *Pizza*.

Ou outro uso, em outro domínio, aplicado de forma similar. Ok?

## Builder - Implementação

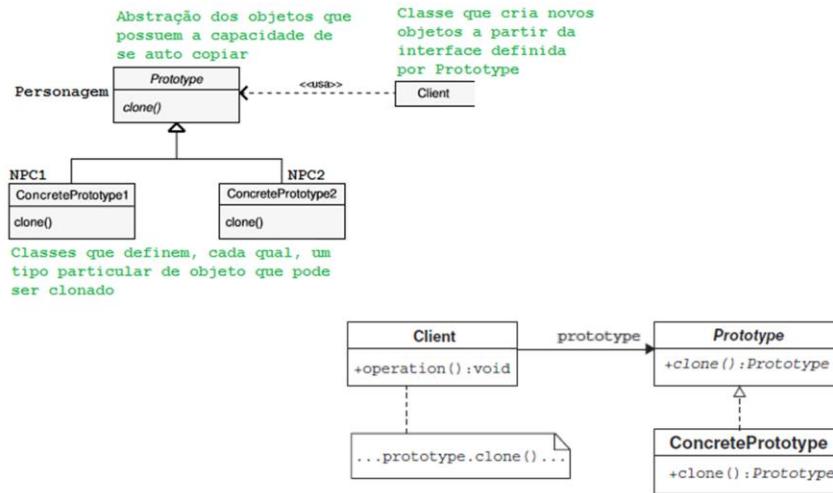
- Por gentileza, consultem:

[https://sourcemaking.com/design\\_patterns/builder/java/2](https://sourcemaking.com/design_patterns/builder/java/2)

- Reparem que têm, ao final do artigo, outros *links* para implementações desse padrão em outras linguagens.

Possível implementação...

# Prototype - Modelagem



Possíveis modelagens...

Ilustrado com *Personagem*, e personagens concretos como: *NPC1* e *NPC2*. Uma entidade *Client* qualquer fazendo uso...

Poderia ser *Figura* (ex. geométrica), com figuras concretas (ex. *quadrado*, *círculo*, dentre outros). Uma entidade *Client* qualquer fazendo uso...

Ou outro uso, em outro domínio, aplicado de forma similar. Ok?

## Prototype - Implementação

```
public Object clone()
{
    try {
        return super.clone();
    } catch (CloneNotSupportedException e) {
        assert false; //this code block should never execute
        return null;
    }
}

public void mouseClicked(MouseEvent e)
{
    Figure newFigure = (Figure) currentFigure.clone();
    newFigure.setCenter(e.getX(), e.getY());
    ((DrawingCanvas) e.getSource()).addFigure(newFigure);
}
```

Possíveis implementações...

1. Ilustrando a implementação para o exemplo da Figura...

Trechos principais de uma possível implementação...

Está implícita a existência de uma classe *Figure* que *implements* uma interface *Cloneable*, a qual contém o método *clone()* ilustrado.

A ação do clique de um *mouse* ilustra uma aplicabilidade desse padrão...

2. Considerem ainda, por gentileza, o material disponível em:

[https://sourcemaking.com/design\\_patterns/prototype/java/2](https://sourcemaking.com/design_patterns/prototype/java/2)

Reparem que existem links para implementações desse padrão em outras linguagens...

3. Ou outro uso, em outro domínio, aplicado de forma similar. Ok?

## Singleton - Modelagem



Classe que permite a criação de uma única instância e fornece um método estático para recuperá-la.



Possíveis modelagens...

Por que anti-pattern?

Dentre as principais razões encontradas na literatura, destacam-se:

- Qualidade da implementação depende da linguagem;
- Difícil de testar (simulações dependem de instância extra);
- Uso equivocado do padrão como um substituto para variáveis globais;
- Inicialização "preguiçosa" é complicada em ambiente *multithreaded*, e
- Difícil ou impossível de implementar em ambiente distribuído, sendo necessário garantir que cópias serializadas refiram-se ao mesmo objeto.

## Singleton - Implementação

```
public class Singleton
{
    private static Singleton instance = null;

    private Singleton() {}

    public static synchronized Singleton instance()
    {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

    ...any other methods...
}
```

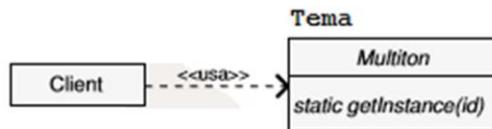
Possível implementação...

Considerem ainda, por gentileza, o material disponível em:

[https://sourcemaking.com/design\\_patterns/singleton/java/2](https://sourcemaking.com/design_patterns/singleton/java/2)

Reparem que existem links para implementações desse padrão em outras linguagens...

## Multiton - Modelagem



Classe que permite a criação de uma quantidade limitada de instâncias e fornece um método estático para recuperá-las.

Possível modelagem...

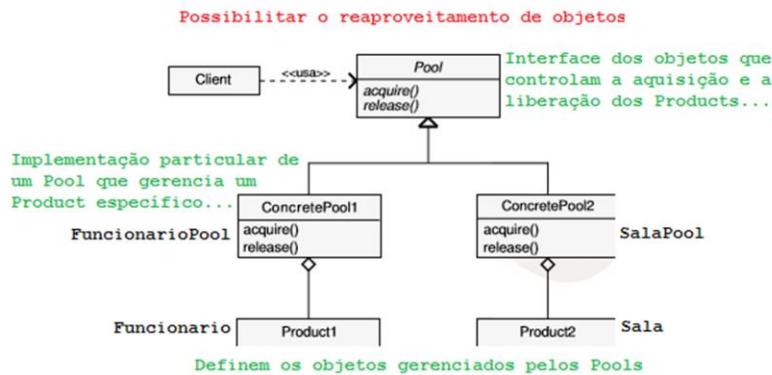
## Multiton- Implementação

- Por gentileza, consultem:

<http://www.blackwasp.co.uk/multiton.aspx>

Possível implementação...

## Object Pool - Modelagem



Possível modelagem...

Aqui, ilustrado com *FuncionarioPool* e *SalaPool*.

Mas, poderia ser um *JDBCConnectionPool*, conforme ilustrado na implementação disponível no próximo slide.

Ou outro uso, em outro domínio, aplicado de forma similar. Ok?

## Object Pool - Implementação

- Por gentileza, consultem:

[https://sourcemaking.com/design\\_patterns/object\\_pool/java](https://sourcemaking.com/design_patterns/object_pool/java)

- Reparem que têm, ao final do artigo, outros *links* para implementações desse padrão em outras linguagens.

Possível implementação...

## Considerações Finais



## Considerações Finais

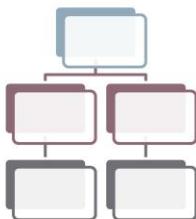
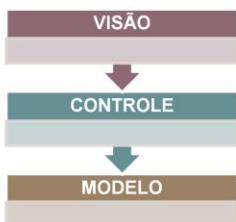
- Nessa aula, conhecemos os padrões GoFs criacionais.
- Continuem os estudos! Só se aprende praticando!
- Nas referências, têm vários materiais complementares! : )

# Referências



## Referências

- LARMAN, Craig. Utilizando UML e Padrões: Uma Introdução a Análise e ao Projeto
- Orientado a Objetos. 3a. edição. Bookman, 2007.
- COCKBURN, Alistair. Escrevendo Casos de Uso Eficazes. Bookman, 2005.
- SILVA, Ricardo Pereira. UML 2 em Modelagem Orientada a Objetos. Visual Books, 2007.
- PRESSMAN, Roger S. Engenharia de Software. 6a. edição . McGraw-Hill, 2006.
- IEEE. SWEBOK-Guide to the Software Engineering Body of Knowledge, 2004.
- SOMMERSVILLE, Ian. Engenharia de Software. 8a. edição. Pearson, 2007.



FIM

Dúvidas?

CONTATO:  
[mileneserrano@unb.br](mailto:mileneserrano@unb.br)  
ou  
[mileneserrano@gmail.com](mailto:mileneserrano@gmail.com)

Sugestões?

