Design Patterns

REVIEW

UNIVAS

INTENÇÃO

Garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso para a mesma.

UNIVAS

QUANDO USAR?

- Quando for preciso haver apenas uma instância de uma classe, e essa instância tiver que dar acesso aos clientes através de um ponto bem conhecido;

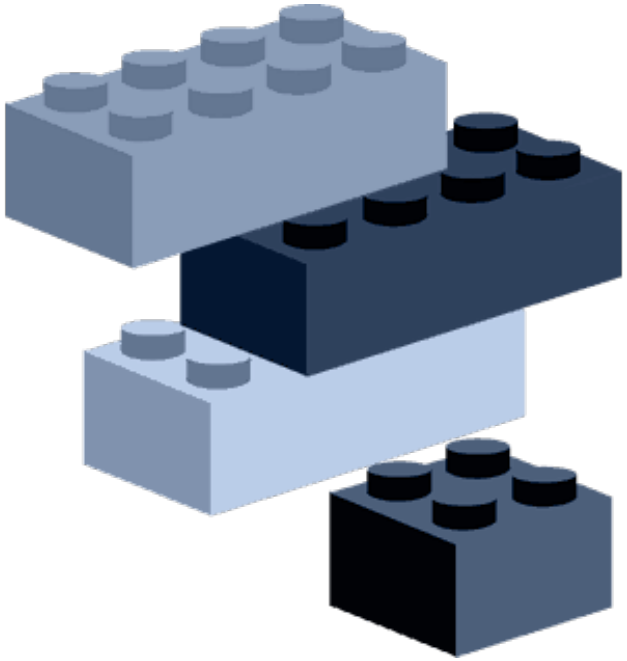
- Quando a única instância tiver de ser extensível através de subclasses, possibilitando aos clientes usar uma instância estendida sem alterar o seu código.

UNIVAS

ESTRUTURA

SingleObject
- Instance : singleton anyData
- <<constructor>> SingleObject() + get() : Singleton anyOperation()

```java
package pp_singleton_ex1;

public class SingleObject {
  //criar o objeto do SingleObject
    private static SingleObject instance = null;

  //tornar o construtor privado para que a classe não possa
  //ser instanciada
    private SingleObject() {}

  //torne o objeto disponível
    public static SingleObject getInstance() {
        if(SingleObject.instance == null) {
            SingleObject.instance = new SingleObject();
        }
        return instance;
    }

    public void showMessage() {
        System.out.println("Hello Singleton!");
    }
}

```



FACTORY METHOD

APRESENTAÇÃO E PRÁTICA DO PP

UNIVAS

ANTES DE MAIS NADA...

No contexto da programação, tudo que possui o nome Factory (fábrica) utiliza operações para a criação de objetos.

UNIVAS

INTRODUÇÃO

O Design Pattern Factory Method atua permitindo adiar a instanciação de um objeto para as subclasses.

Em outras palavras, tem-se uma classe com um método que cria um objeto e força-se as subclasses a implementá-lo.

UNIVAS

CARACTERÍSTICAS

- É um padrão de projeto de criação, ou seja, lida com a criação de objetos;
- Oculta a lógica de instanciação do código cliente, sendo responsável por instanciar as classes desejadas;
- É obtido através de herança, podendo ser criado ou sobrescrito por subclasses;
- Dá flexibilidade ao código cliente, permitindo a criação de novas factories sem alterar as já existentes, garantindo, assim, o Open/Closed Principle;
- Pode utilizar parâmetros para determinar o tipo dos objetos a serem criados ou parâmetros a serem enviados aos objetos.

UNIVAS

MOTIVAÇÃO

Manejar/tratar as possíveis complicações do processo de criação de objeto, tais como:

- ⛔ Duplicação de código;
- ⛔ Informações não acessíveis;
- ⛔ Grau de abstração insuficiente.

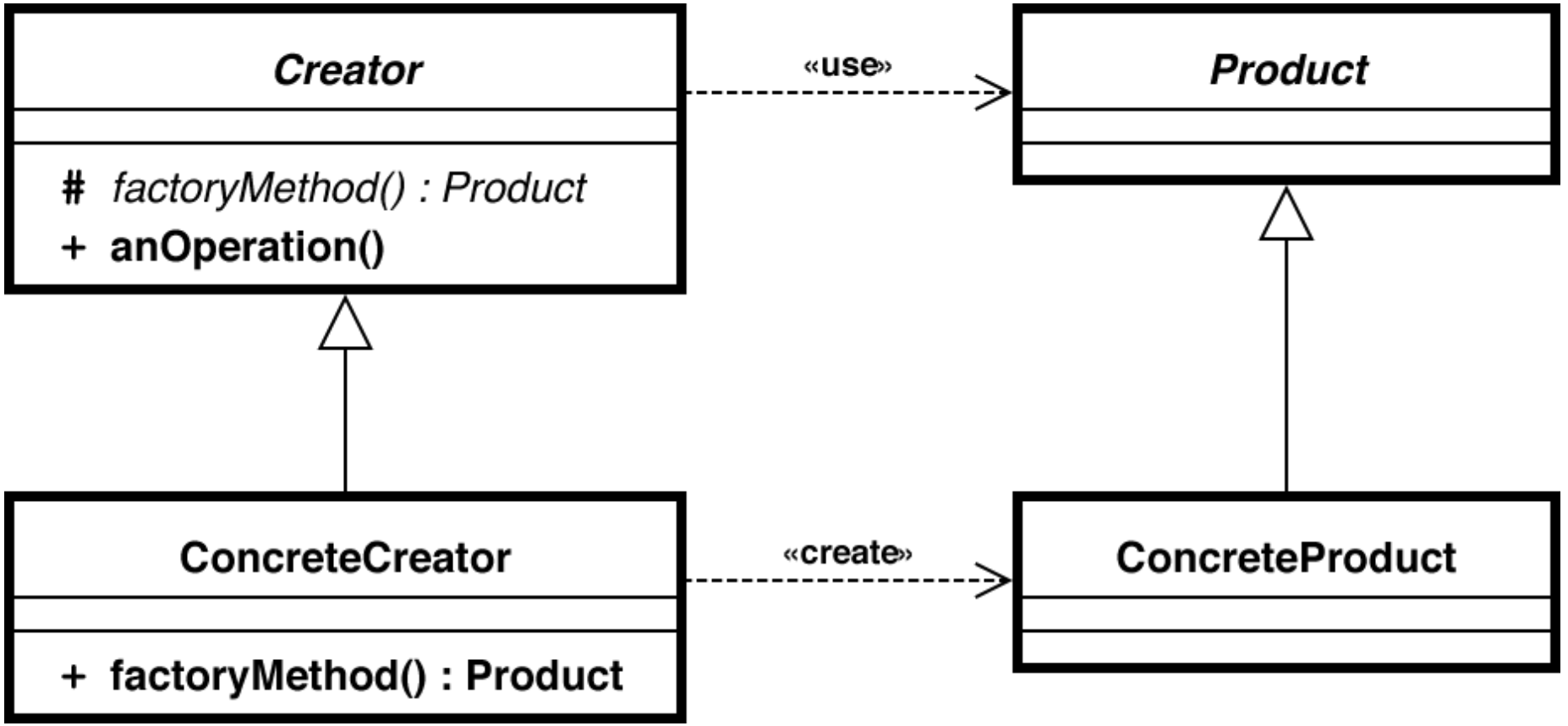
O papel do Factory Method é definir um método separado para a criação dos objetos, no qual as subclasses possam sobrescrever para especificar o “tipo derivado” do produto que será criado.

QUANDO USAR?

Quando não souber com certeza quais os tipos de objeto que o seu código irá precisar;
Permitir a extensão de suas factories para a criação de novos objetos (OCP);
Separar o código que cria do código que utiliza classes (Single Responsability Principle);
Dar um Hook às subclasse para permitir que elas decidam a lógica de criação dos objetos;
Eliminar duplicação de código na criação de objetos.

UNIVAS

ESTRUTURA



ESTRUTURA

Product
Produto abstrado, define uma interface par os objetos criados pelo Factory Method.

ConcreteProduct
Produto concreto, uma implementação para a interface produto.

Creator
Criador abstrato, declara o Factory Method que retorna o objeto da classe Product. Ele também pode definir uma implementação básica que retorne o objeto de uma classe ConcreteProduct.

UNIVAS

ESTRUTURA

ConcreteCreator
Criador concreto, sobrescreve o Factory Method e retorna um objeto da classe ConcreteProduct.

Vejamos o primeiro exemplo:

UNIVAS

Product.java

```java
package pp_factory_method_ex1;

/**
 * Interface de um produto
 */

public interface Product {

    public void Hello();

}
```

```java
package pp_factory_method_ex1;

/**
 * Esta classe concreta contém a implementação
 * de um tipo de produto específico.
 */

public class ConcreteProduct implements Product {

    @Override
    public void Hello() {
        System.out.println("Hello Factory Method!");
    }

}
```

UNIVAS

```java
package pp_factory_method_ex1;

/**
 * Abstração de uma Aplicação capaz de manipular
 * produtos.
 */

public abstract class Creator {

    protected Product prod;

    //Abstração do Factory Method
    abstract Product makeProduct();

    public void createAndShow() {
        this.prod = this.makeProduct();
        System.out.println("Factory Method criou um novo objeto: ");
        System.out.println(prod);
    }

}
```

UNIVAS

```java
package pp_factory_method_ex1;

/**
 * Esta classe concreta contém a implementação
 * de uma aplicação capaz de manipular produtos
 * do tipo ConcreteProduct.
 */

public class ConcreteCreator extends Creator {

    /**
     * Uma implementação do Factory Method. Este método é
     * especializado na criação de produtos do tipo ConcreteProduct
     */
    Product makeProduct() {
        return new ConcreteProduct();
    }

}
```

UNIVAS

```java
package pp_factory_method_ex1;

public class TestProduct {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        ConcreteCreator test = new ConcreteCreator();

        Product product = test.makeProduct();

        product.Hello();
        test.createAndShow();

    }

}
```

UNIVAS

CONSEQUÊNCIAS

Positivas:

Open/Closed Principle, mantendo o código aberto para extensão;

Single Responsability Principle, separando o código que cria do que usa o objeto;

Ajuda o desacoplamento do código.

CONSEQUÊNCIAS

Negativas:

- 🚫 Se usado sem necessidade, haverá uma explosão de subclasses. Será necessário uma classe ConcreteCreator para cada ConcreteProduct.

Neste cenário, avalie outros padrões em substituição ao Factory Method.

UNIVAS

Obrigado!

Professor Gustavo Dias
luizdias@univas.edu.br