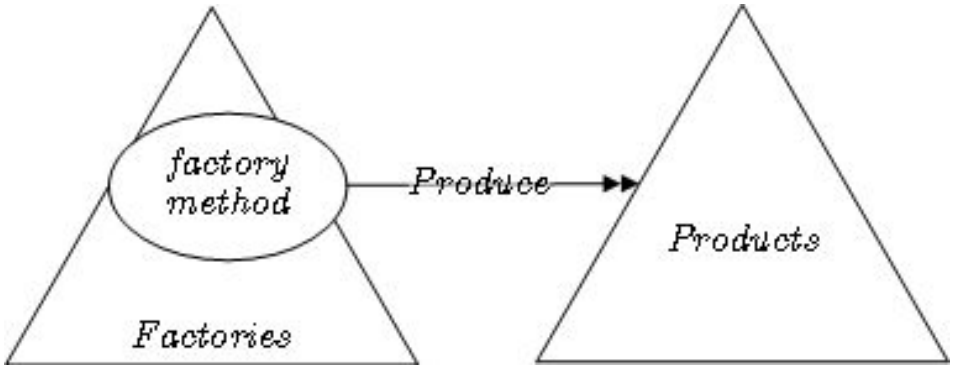
Factory Method
E
Abstract Factory

Renan Cunha & João Almeida

O que é um Factory?

Padrão de projeto de criação por meio de fábricas.
Mecanismo de criação eficiente de objetos

```
def factory():
  return Object()
```

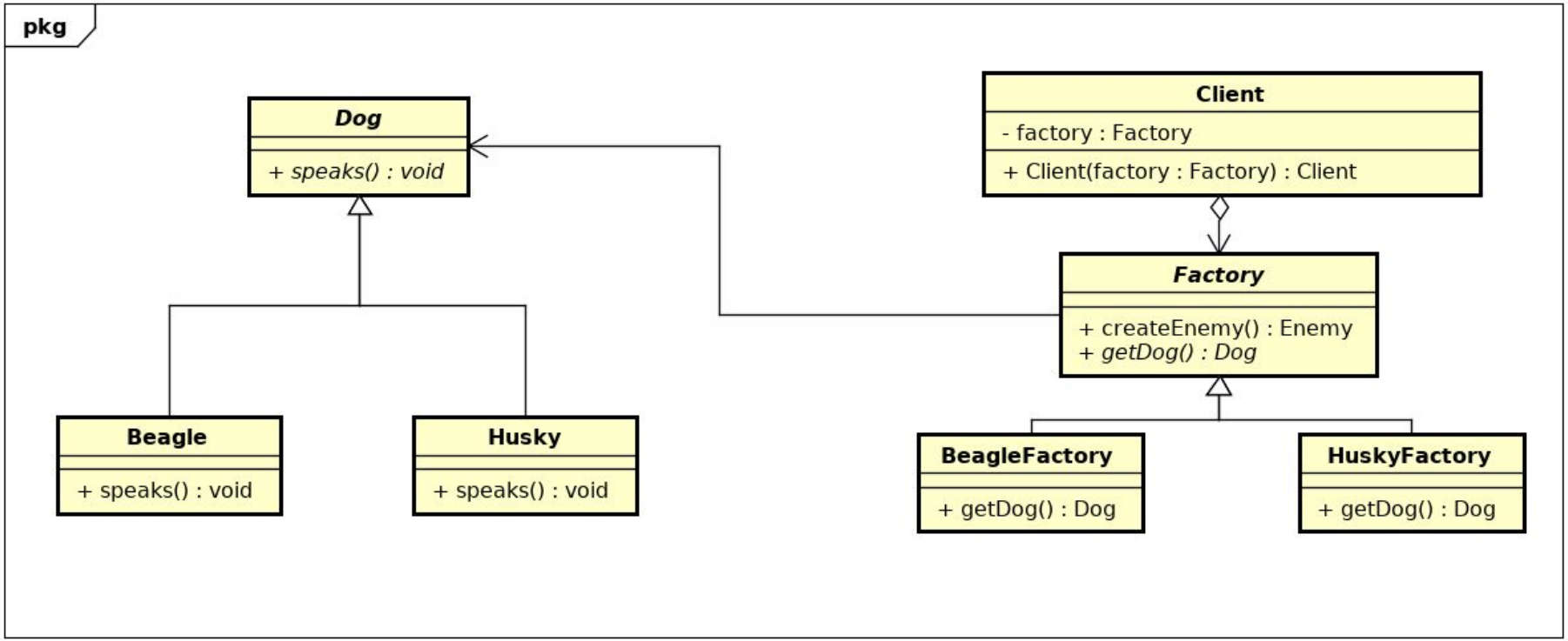


1.Casos de uso

- Criar objetos sem expor a lógica de instanciação ao cliente
- Quando a classe não antecipa o tipo do objeto que será criado
- Permitir que as subclasses decidam qual classe instanciar

Implementação

https://repl.it/@MarceloFreitas2/FactoryMethod

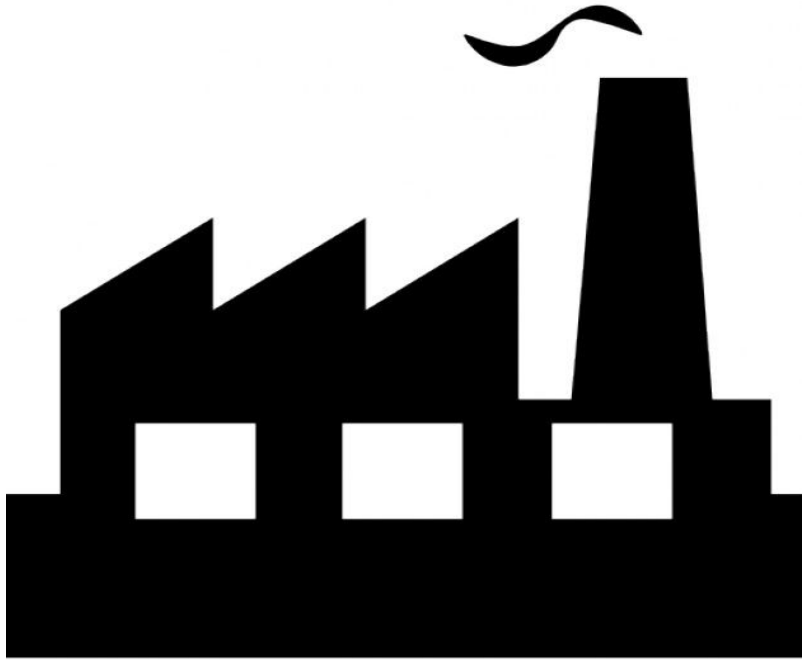


Consequências no uso do FM

- Este padrão permite acréscimo/modificação de código facilitada.
- Omite do cliente o processo de implementação, lógica de negócios e instanciação e de classes.
- Pode deixar o projeto sobrecarregado em número de classes

Abstract Factory

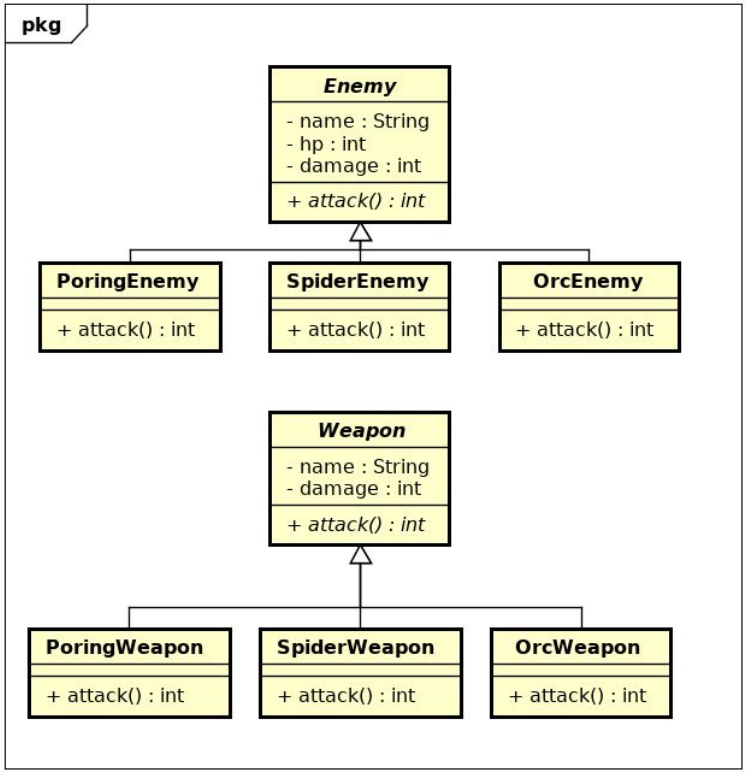
- Objetivo: Fornecer uma interface para criação de famílias de objetos relacionados sem especificar suas classes concretas [GoF]
- Factory Method: Uma Família de Objetos
- Abstract Factory: Mais de uma Família de objetos



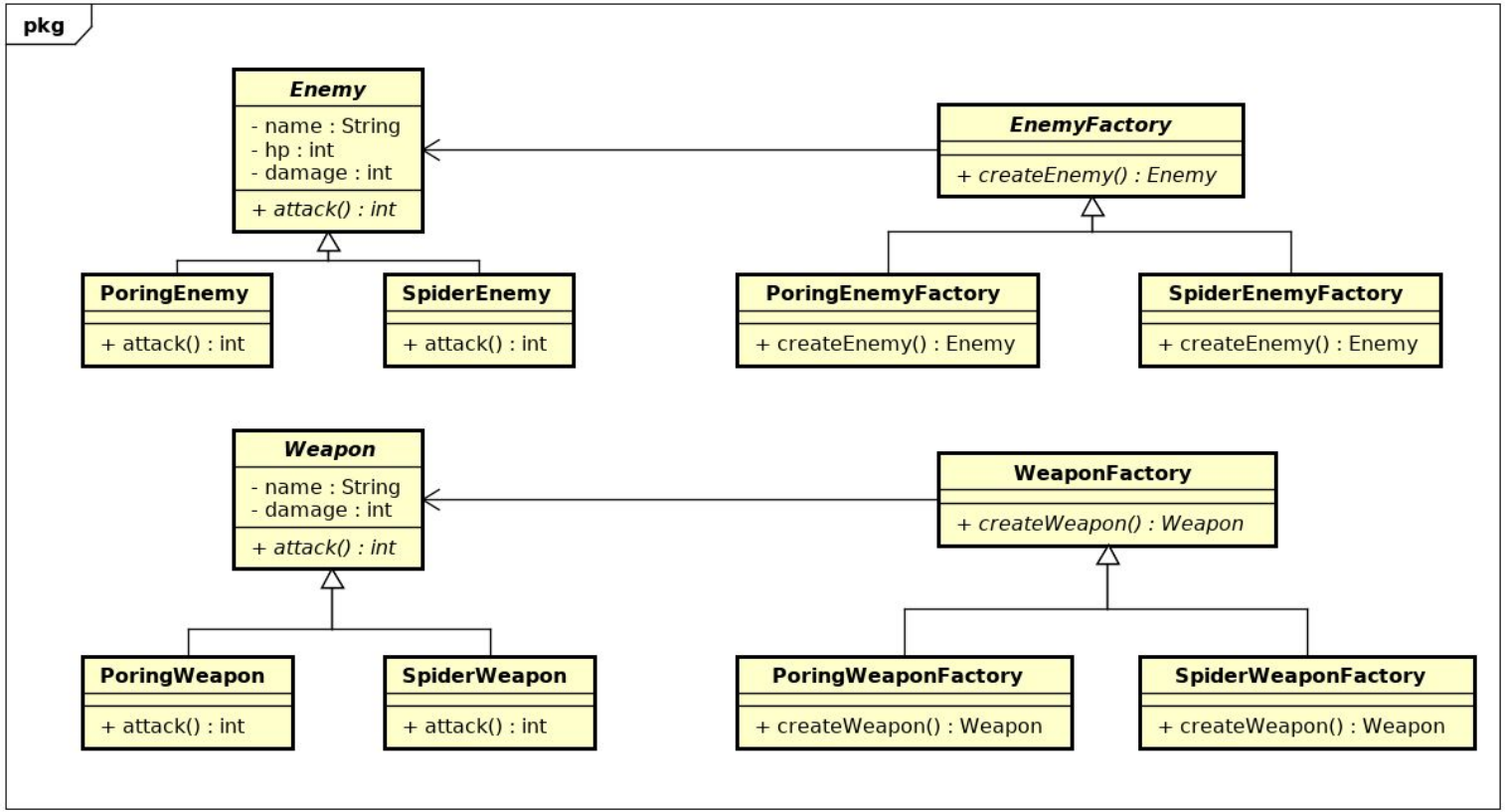
Problema

- Além de vários tipos de inimigos, você também quer criar diferentes tipos de armas que são específicas para cada inimigo.

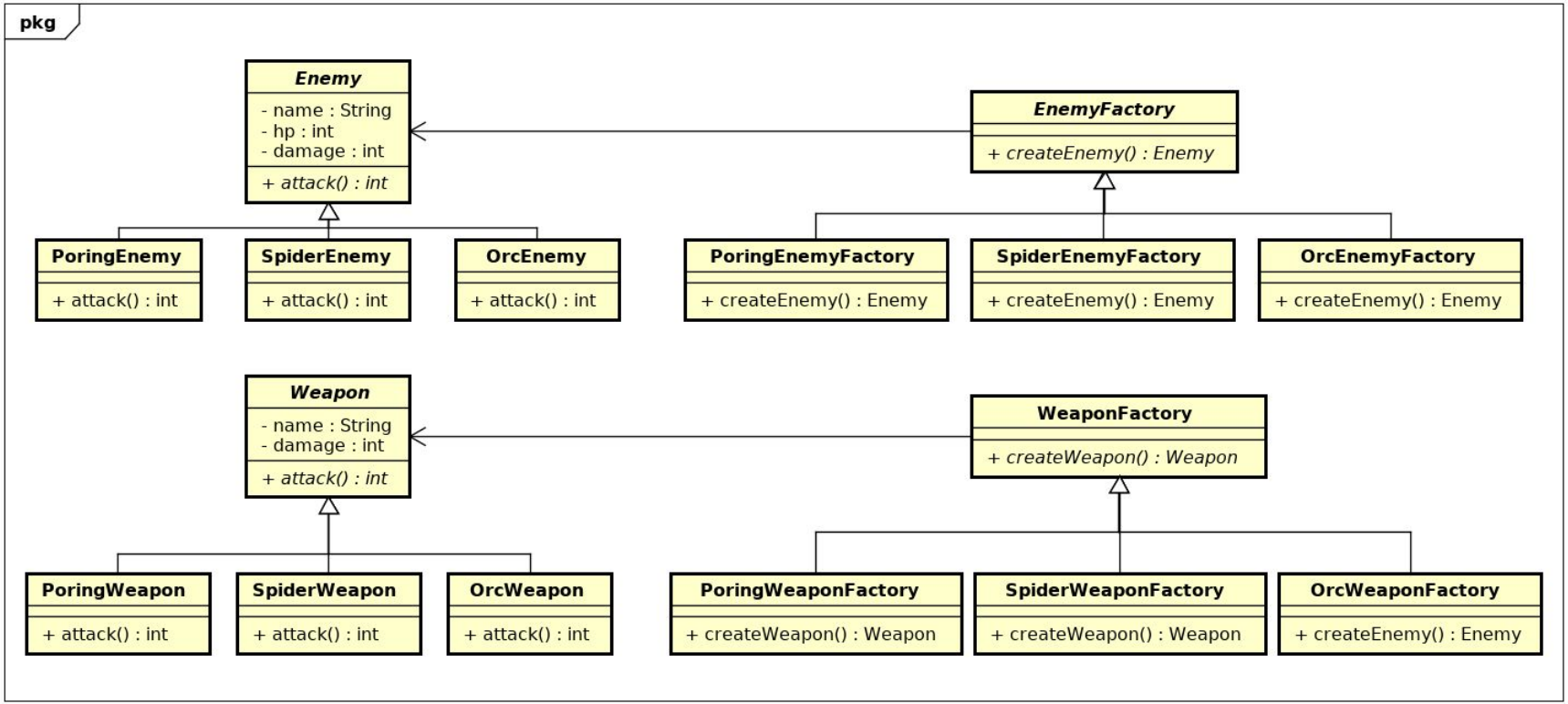
- Ex.: Você tem os inimigos Poring e Aranha, e quer criar armas para Porings e armas para Aranhas



Solução Imediatista: Vários Factory Methods



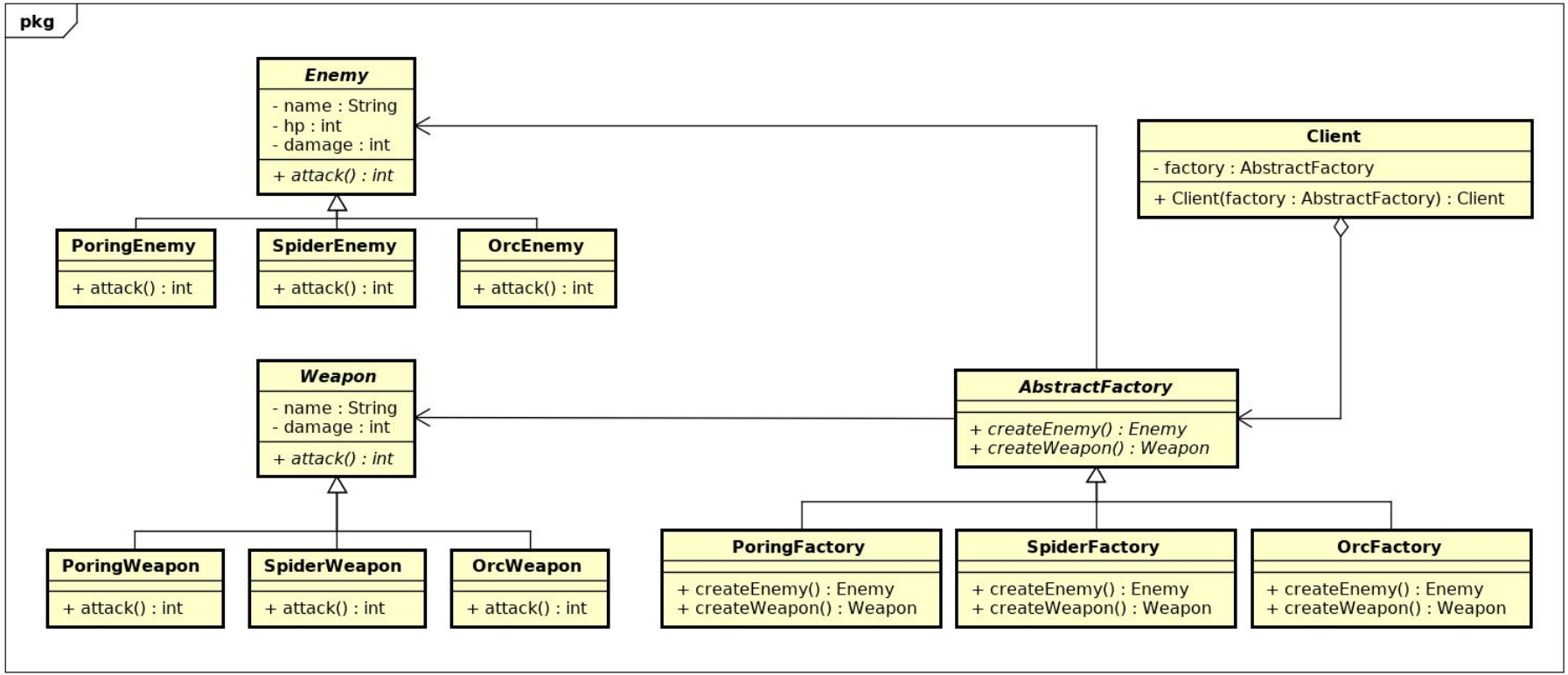
E se precisássemos de mais Tipos de Objetos?



Problemas Encontrados

- Solução não escalável
- Impossibilita Reuso de código
- Não segue o princípio Open/Closed
- Não segue o princípio da Responsabilidade Única
- Será que há uma solução melhor?

Solução com Abstract Factory



Exemplo Factories

```
public class PoringFactory extends AbstractFactory {

  @Override
  public Enemy createEnemy() {
    return new PoringEnemy();
  }

  @Override
  public Weapon createWeapon() {
    return new PoringWeapon();
  }
}
```

```
public class SpiderFactory extends AbstractFactory {

@Override
  public Enemy createEnemy() {
    return new SpiderEnemy();
  }

  @Override
  public Weapon createWeapon() {
    return new SpiderWeapon();
  }
}
```

Exemplo Abstract Factory

```
public abstract class AbstractFactory {
  private static final PoringFactory PORING_FACTORY = new PoringFactory();
  private static final SpiderFactory SPIDER_FACTORY = new SpiderFactory();

  static AbstractFactory getFactory(Type type) {
          //mostrado ao lado
 }


  public abstract Enemy createEnemy();
  public abstract Weapon createWeapon();
}
```

```
static AbstractFactory getFactory(Type type) {
    AbstractFactory factory = null;
    switch (type) {
      case SPIDER:
        factory = SPIDER_FACTORY;
        break;
      case PORING:
        factory = PORING_FACTORY;
        break;
      default:
        System.out.println("Erro");
        System.exit(1);
        break;
    }
    return factory;
  }
```

Uso Pelo Cliente

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

    AbstractFactory factory = AbstractFactory.getFactory(EntityType.PORING);
    Enemy poring = factory.createEnemy();
    Weapon poringWeapon = factory.createWeapon();

    factory = AbstractFactory.getFactory(EntityType.SPIDER);
    Enemy spider = factory.createEnemy();
    Weapon spiderWeapon = factory.createWeapon();

    //uso dos objetos

}
```

Diagrama de Classes Padrão Abstract Factory