Abstract Factory

Grupo 12: Filipe Ribeiro
Matheus Silva

Gama, 14 de Setembro de 2016

Introdução

- Padrão Criacional;
- Usado para criação de famílias de objetos
- Uso de classes abstratas ou *interfaces*

“Fornecer uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.”[1]

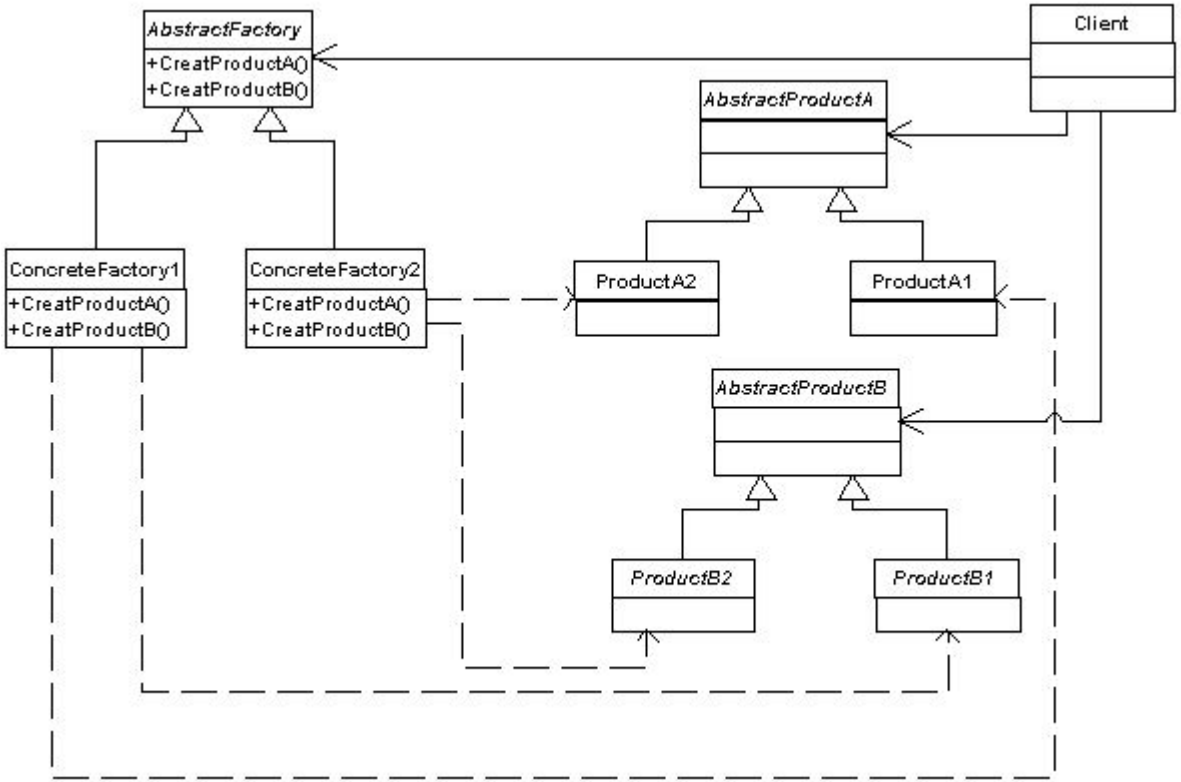
Problemática

- O grande problema que se deseja solucionar, se refere a criação de objetos onde padrões criacionais tendem a diminuir o grande esforço das classes. No contexto do Abstract Factory, isso é feito a partir da criação de classes de “Fábrica”
  - ***Factory Method*** - Criar objetos *on the fly* com alto grau de flexibilidade
  - ***Builder*** - Construir o objeto a partir de uma sequência.
  - ***Prototype*** - Usado quando se espera obter um protótipo inicial do objeto a ser trabalhado e a partir dele iniciar o detalhamento e especificação
  - ***Singleton*** - Usado quando se deseja obter uma instância única de um objeto a ser trabalhado

Principais diferenças com relação ao Factory Method

	Abstract Factory	Factory Method
Padrão Criacional	Sim	Sim
Cria família de objetos	Sim	Não
Grande estrutura de classes	Sim	Sim

Classes Participantes do Padrão



Classes Participantes do Padrão

- **AbstractFactory**: declara uma *interface* para operações de criação de produtos quaisquer
- **ConcreteFactory**: implementa as operações que criam objetos de produtos concretos
- **AbstractProduct**: declara uma *interface* para um tipo de objeto produto
- **Product**: define um objeto a ser criado pela fábrica concreta que implementa a interface em AbstractProduct
- **Client**: usa as *interfaces* declaradas

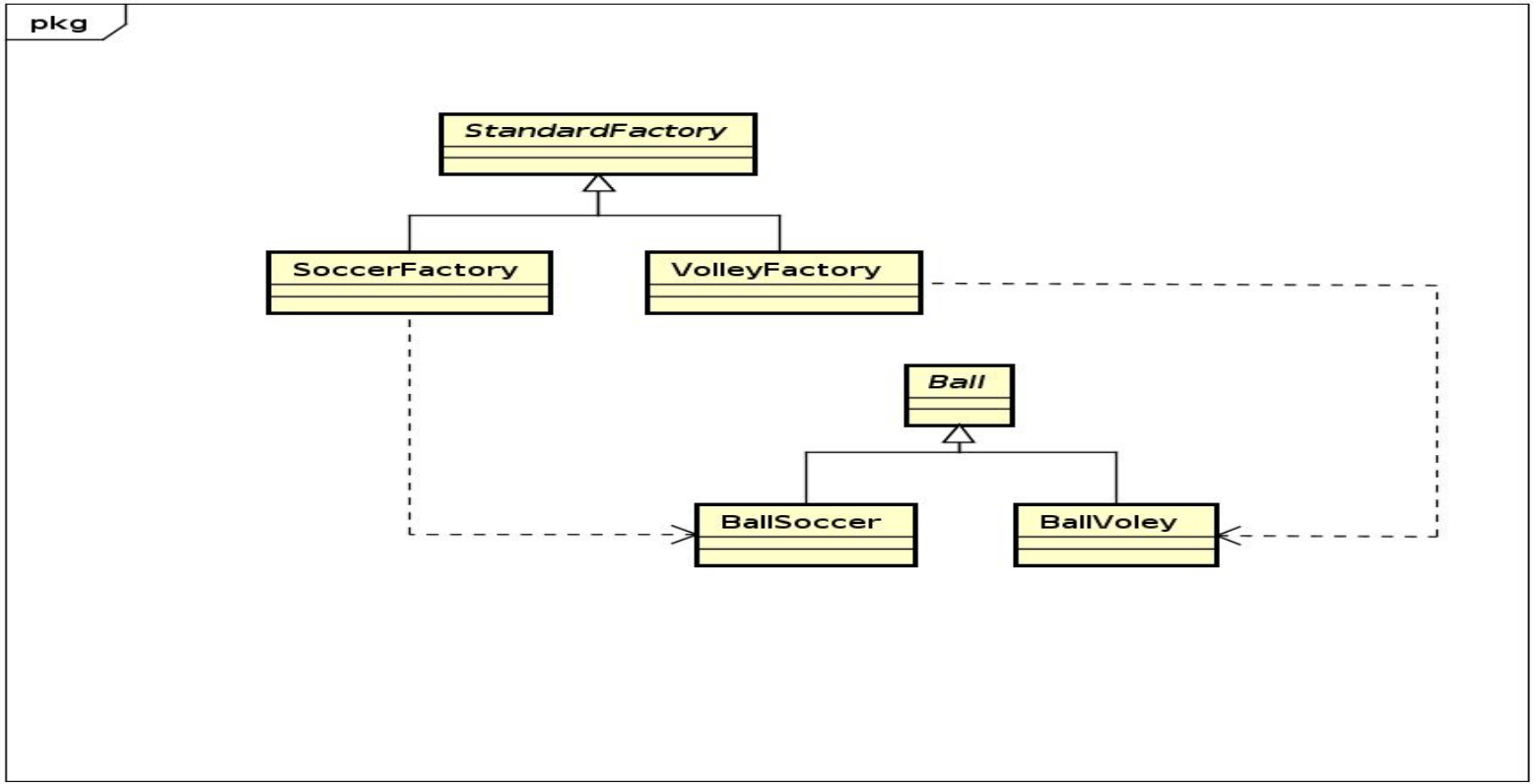
Exemplo

- Considere uma aplicação que precisará trabalhar com a construção de bolas para os mais diversos esportes com as suas mais específicas características;

- Portanto se o esporte a ser praticado for o futebol, a bola deve conter uma estrutura que se adapte há uma partida de futebol;

- O mesmo deve ocorrer caso o esporte a ser praticado seja o basquete, por exemplo

Modelagem



Solução

```python
from abc import ABCMeta

#Abstract Factory
class StandardFactory(object):

    @staticmethod
    def get_factory(factory):
        if factory == 'soccer':
            return SoccerFactory()
        elif factory == 'volley':
            return VolleyFactory()
        raise TypeError('Unknown Factory.')


#Factory
class SoccerFactory(object):
    def get_ball(self):
        return BallSoccer();


class VolleyFactory(object):
    def get_ball(self):
        return BallVolley();
```

```python


# Product Interface
class Ball(object):
    __metaclass__ = ABCMeta
    def play(self):
        pass


# Products
class BallSoccer(object):
    def play(self):
        return 'Bola esta rolando...'


class BallVolley(object):
    def play(self):
        return 'Bola esta voando!'

```

Solução

```
42
43
44  if __name__ =="__main__":
45      factory = StandardFactory.get_factory('volley')
46      ball = factory.get_ball()
47      print ball.play()
48
49      factory = StandardFactory.get_factory('soccer')
50      ball = factory.get_ball()
51      print ball.play()
```

Para pensar

- Em que situações o abstract factory é a melhor solução no contexto de criação de objetos?
- O que a implementação em python modifica na questão do “purismo” do padrão?
- Que tipo de benefícios para o código, no que tange boas práticas de programação, o uso de padrões oferece?

Referências

[1] GAMMA, Erich et al. Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos.

[2] LARMAN, Craig. Utilizando UML e Padrões: Uma Introdução a Análise e ao Projeto Orientado a Objetos. 3ª edição, 2007