Projeto Detalhado de Software

Padrões de Projetos
(GoF Patterns)

Factory Method, Abstract Factory, Builder

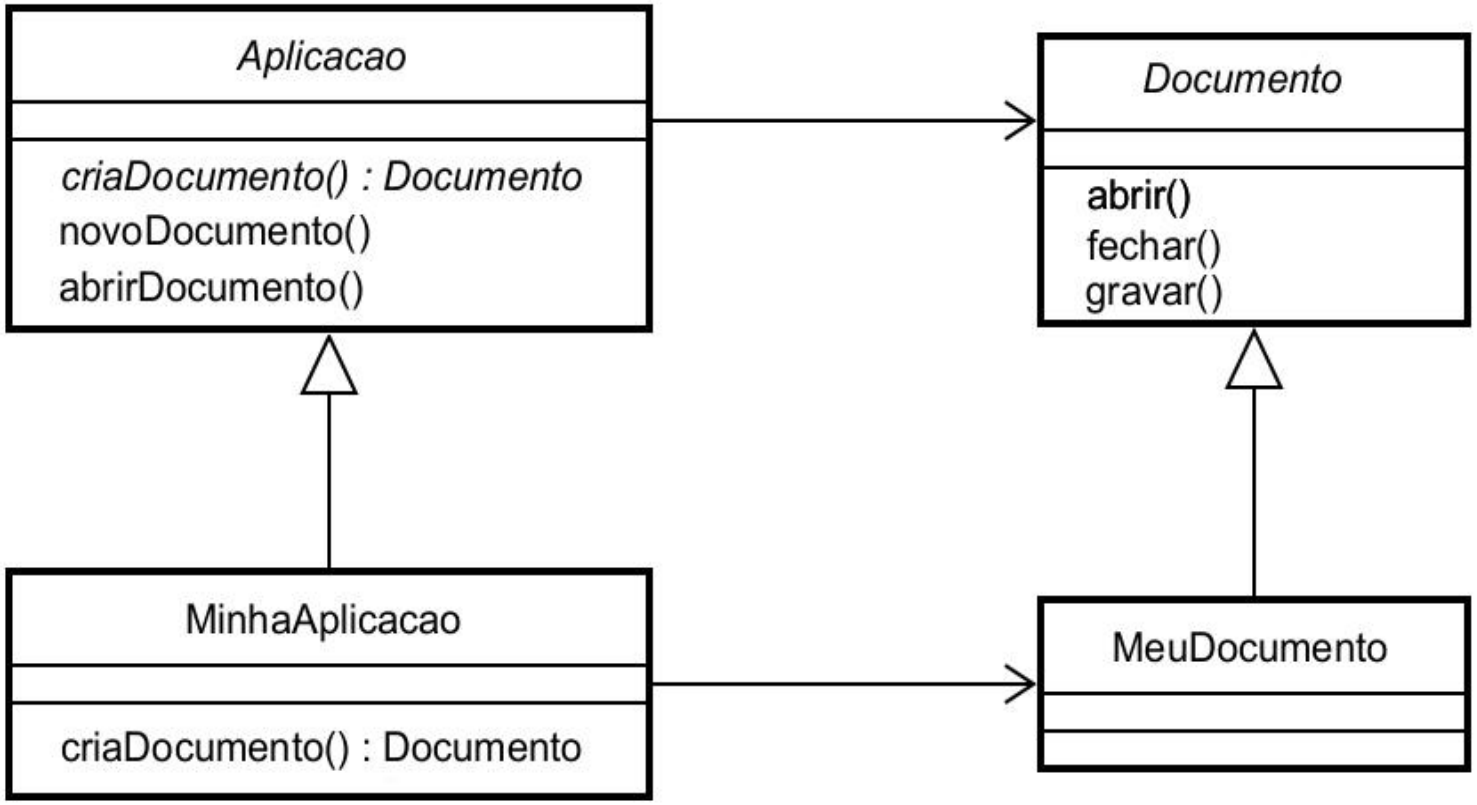
Camilo Almendra | camilo@es.ufc.br | https://sites.google.com/a/es.ufc.br/camilo/

Padrões de Projeto

- Criacionais (Construção)
  - **Factory Method**, **Abstract Factory**, **Builder**
- O que o GRASP fala sobre Criação?
  - A criação dos objetos deve ficar próxima a quem sabe os dados necessários para criação
- Problemas:
  - O processo de criação dos objetos pode envolver algoritmos complexos
  - Pode ser necessário escolher tipos de objetos de acordo com os dados
- Solução:
  - Criar uma classe com a responsabilidade de criação
  - Uma... **invenção pura** para desacoplar **variações** na criação de objetos

Padrão – Factory Method

- **Intenção:** Definir uma interface para criação de objetos, mas permitindo que subclasses decidam quais classes serão instanciadas



Motivação

Clientes precisam instanciar diferentes tipos de objetos

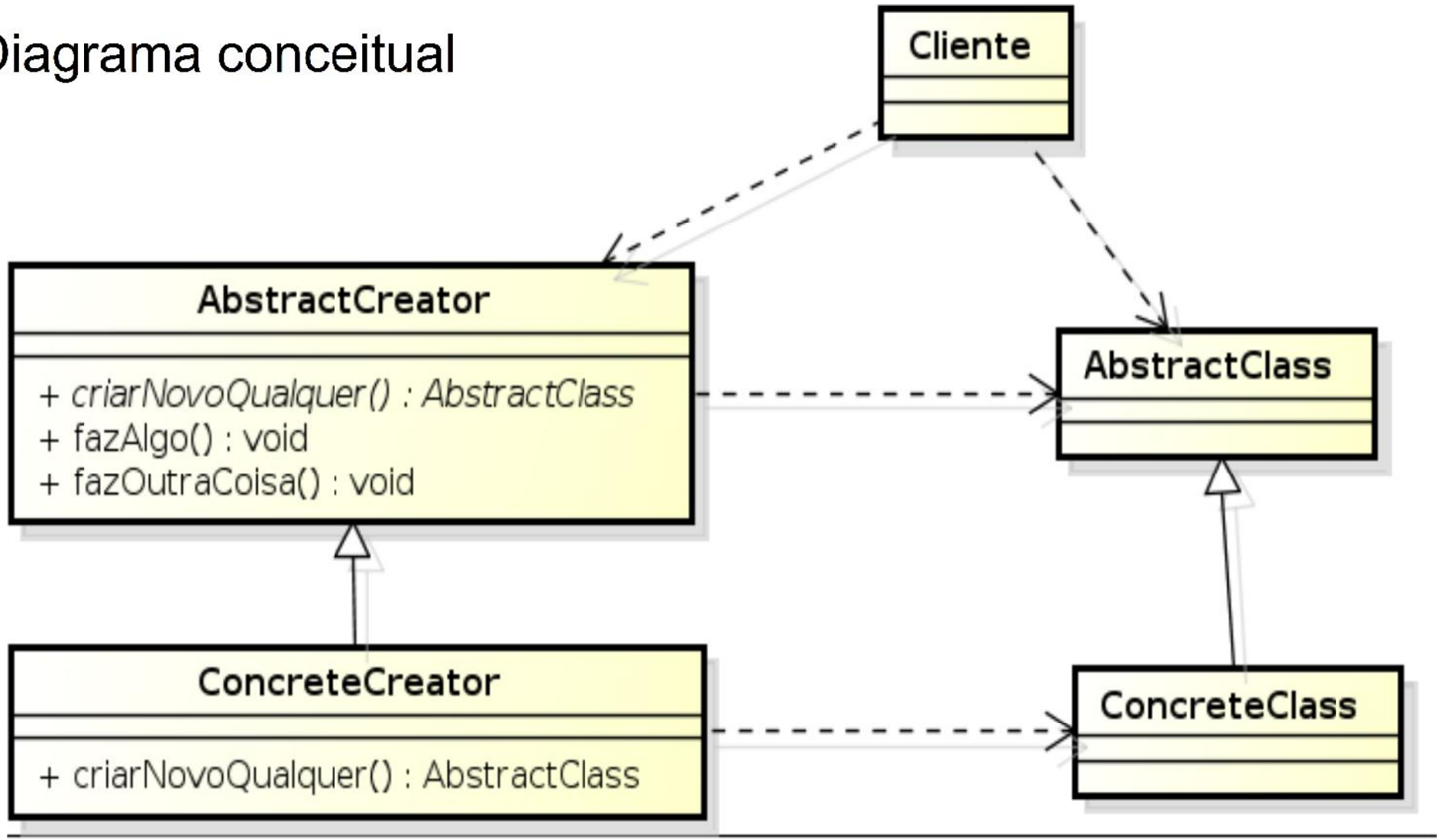
@ wikipedia

Padrão – Factory Method

- **Solução:** Delegar a criação de objetos para fábricas e usar uma interface comum
- **Participantes**
  - *Creator* é a classe que define o método de fabricação que retorna um objeto da *AbstractClass*
  - *Concrete Creator* implementa a interface *Creator* para cada tipo de objeto da hierarquia, retornando objetos da *ConcreteClass*
- **Conseqüência**
  - Isolamento do núcleo do código de classes específicas da aplicação. Nesse caso, “aplicação” é um sistema ou versão que está reutilizando pacotes de classes de negócio

Padrão – Factory Method

- Diagrama conceitual



Padrão – Factory Method

- Exemplo: Tenho 3 tipos de veículos em uma frota (carro,moto e caminhão). Minha lista completa até pode ser de um tipo genérico Veiculo, mas em algum momento eu vou precisar instanciar e criar objetos específicos, certo? Como faríamos isso antes de conhecer padrões?

```
(...)
  Veiculo veiculo;
  if(tipo.equals(“Carro”)){
     veiculo = new Carro();
  } else if(tipo.equals(“Moto”)){
     veiculo = new Moto();
  } else if(tipo.equals(“Caminhao”)){
     veiculo = new Caminhao();
  }
  return veiculo;
```

PROBLEMA 2: para piorar, usaríamos isso em todos os lugares em que criássemos classes específicas.

Padrão – Factory Method

- PROBLEMAS:
  - A cada novo tipo de veículo a gente iria precisar mudar esse código para incluir um novo “if” e isso viola a ideia de que as classes devem estar fechadas para modificações
  - Para piorar, usaríamos isso em todos os lugares em que criássemos classes específicas.
- Solução:
  - Definir um único lugar para instanciar classes. Só ela precisa tratar do tipo para instanciar a classe do tipo certo

Padrão – Factory Method

- Como deveria ficar com o padrão fábrica?
  - Fazer uma fábrica de veículos que vai criar veículo para todas as classes que precisarem de um novo veículo

```
public class VeiculoFabrica{
   public Veiculo criaVeiculo(String tipo){
      Veiculo veiculo;
      if(tipo.equals(“Carro”)){
         veiculo = new Carro();
      } else if(tipo.equals(“Moto”)){
         veiculo = new Moto();
     } else if(tipo.equals(“Caminhao”)){
         veiculo = new Caminhao();
     }
     return veiculo;
   }
}
```

Padrão – Factory Method

- Exercício rápido
  - 1 pizzaria produz 3 tipos de pizza: queijo, calabreza e vegetariana
  - Projete o sistema utilizando o padrão fábrica

Padrão – Abstract Factory

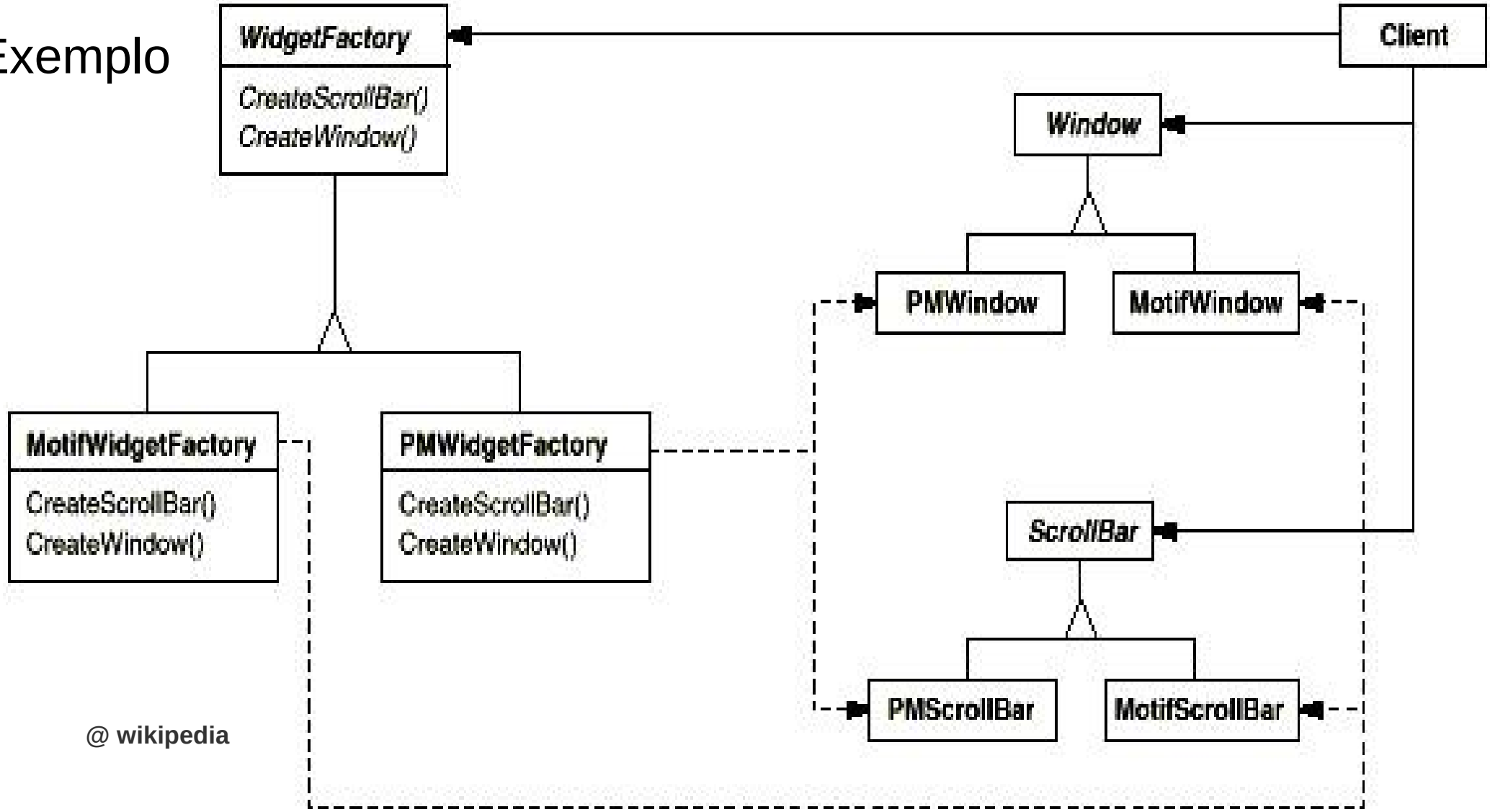
- E se existissem 2 pizzarias na cidades que tivessem os mesmos 3 tipos de pizza? Mesmo sendo do mesmo sabor, as pizzas teriam leve diferença entre elas. Como tornar a fábrica genérica, mas permitir essa variação entre as pizzas do mesmo tipo?

Padrão – Abstract Factory

- **Intenção:** Definir uma interface para criação de famílias de objetos sem especificar suas classes concretas
- **Motivação:** Famílias de objetos relacionadas precisam ser instanciadas

Padrão – Abstract Factory

Exemplo



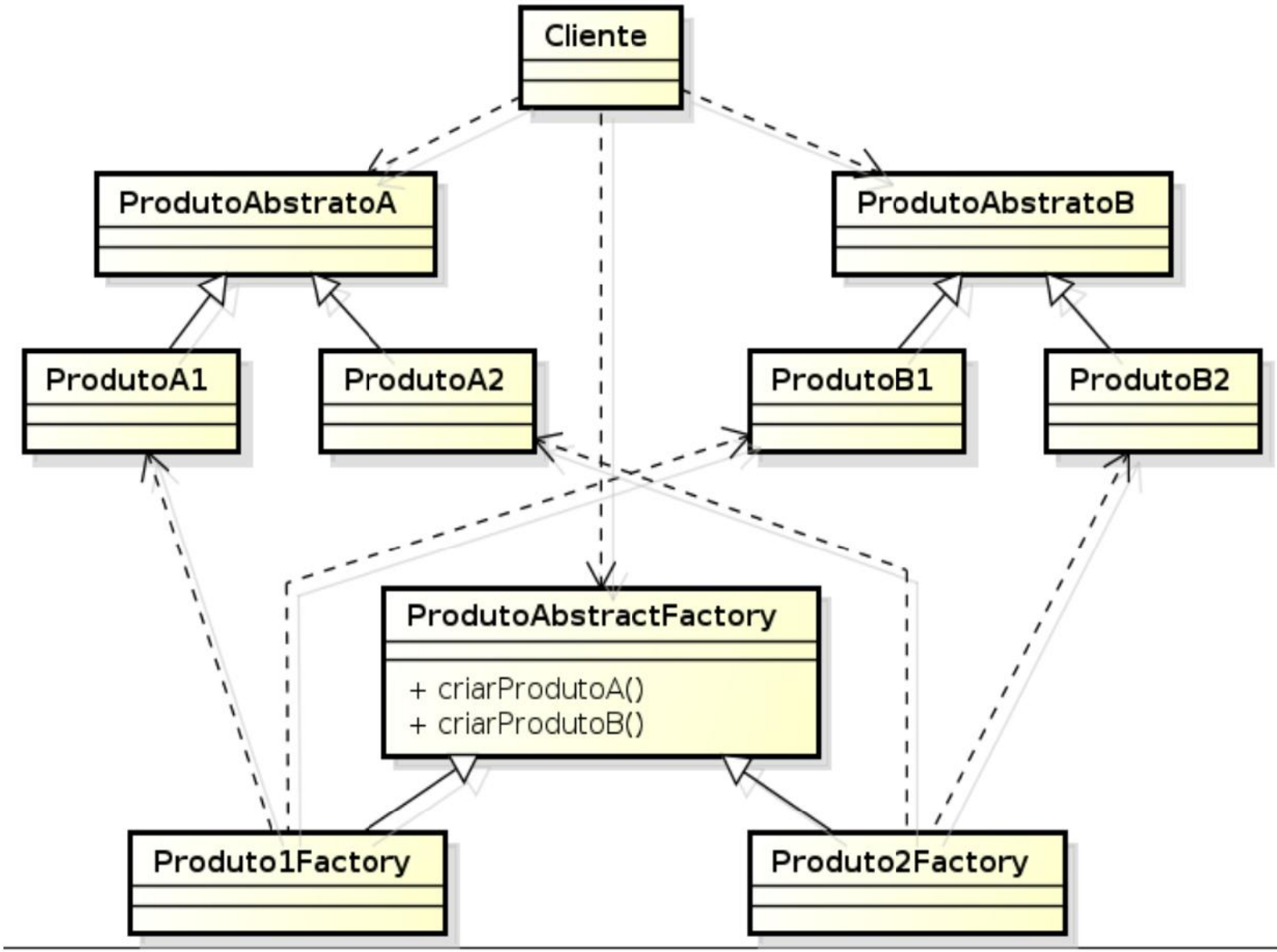
@ wikipedia

Padrão – Abstract Factory

- **Solução:** Coordenar a criação de famílias de objetos
- **Participantes**
  - *Abstract Factory* define a interface para criação de cada tipo da família
  - *Fábricas concretas* são usadas para cada tipo
- **Conseqüência**
  - Isolamento das regras relacionadas a “quais objetos usar” das regras de “como utilizar”

Padrão – Abstract Factory

- Diagrama conceitual



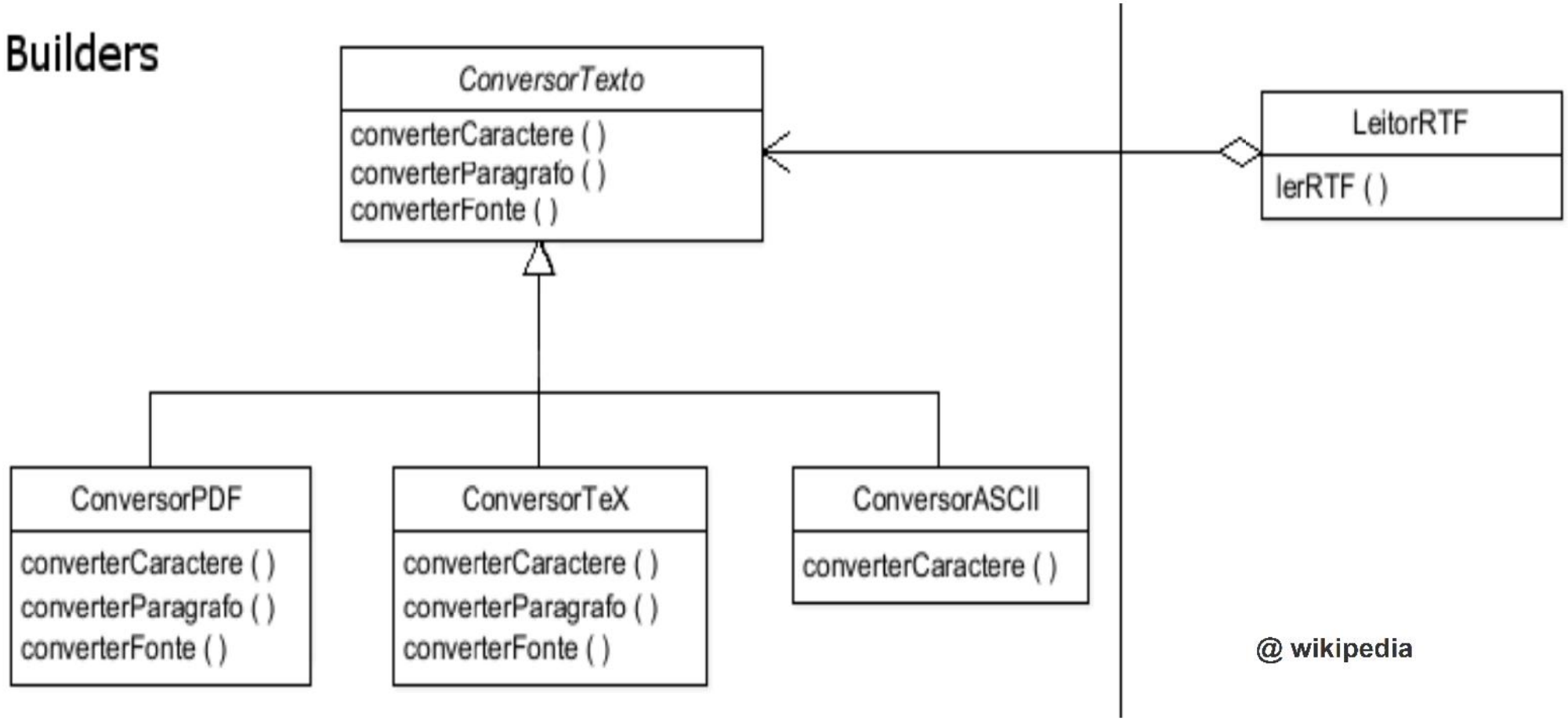
Padrão – Abstract Factory

Factory Method X Abstract Factory

- *Abstract Factory* usa *Factory Method*
- *Factory Method* serve para uma hierarquia de classes
- *Abstract Factory* serve para diferentes hierarquias
- *FM*: Cliente conhece a fábrica concreta, não conhece os objetos
- *AF*: Cliente não conhece nem fábrica nem objetos concretos

Padrão – Builder

- **Intenção:** Encapsular processo de construção de objetos complexos
- **Motivação:** Algoritmo de criação dos diferentes tipos de objetos complexos

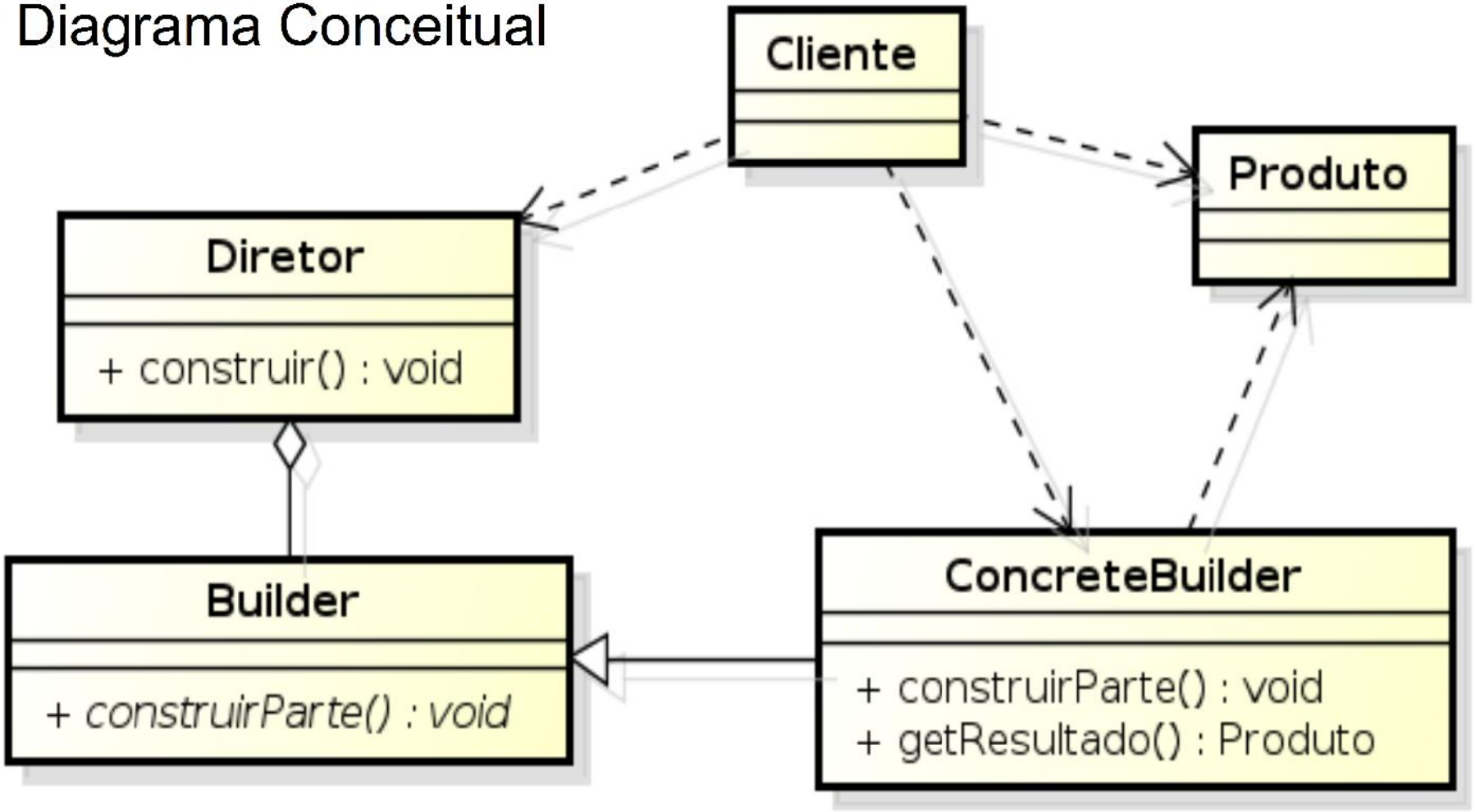


Padrão – Builder

- **Solução:** Simplificar a construção de objetos complexos através da especificação somente do tipo e dos dados. Delegar o processo de criação para o *Builder*
- **Participantes**
  - *Diretor* é responsável pela construção, mas delega a criação e montagem ao *Builder*
  - *Builder* especifica a interface para criação das partes do objeto complexo (*Produto*)
  - ConcreteBuilder cria e monta as partes do *Produto* de acordo com a interface do *Builder*

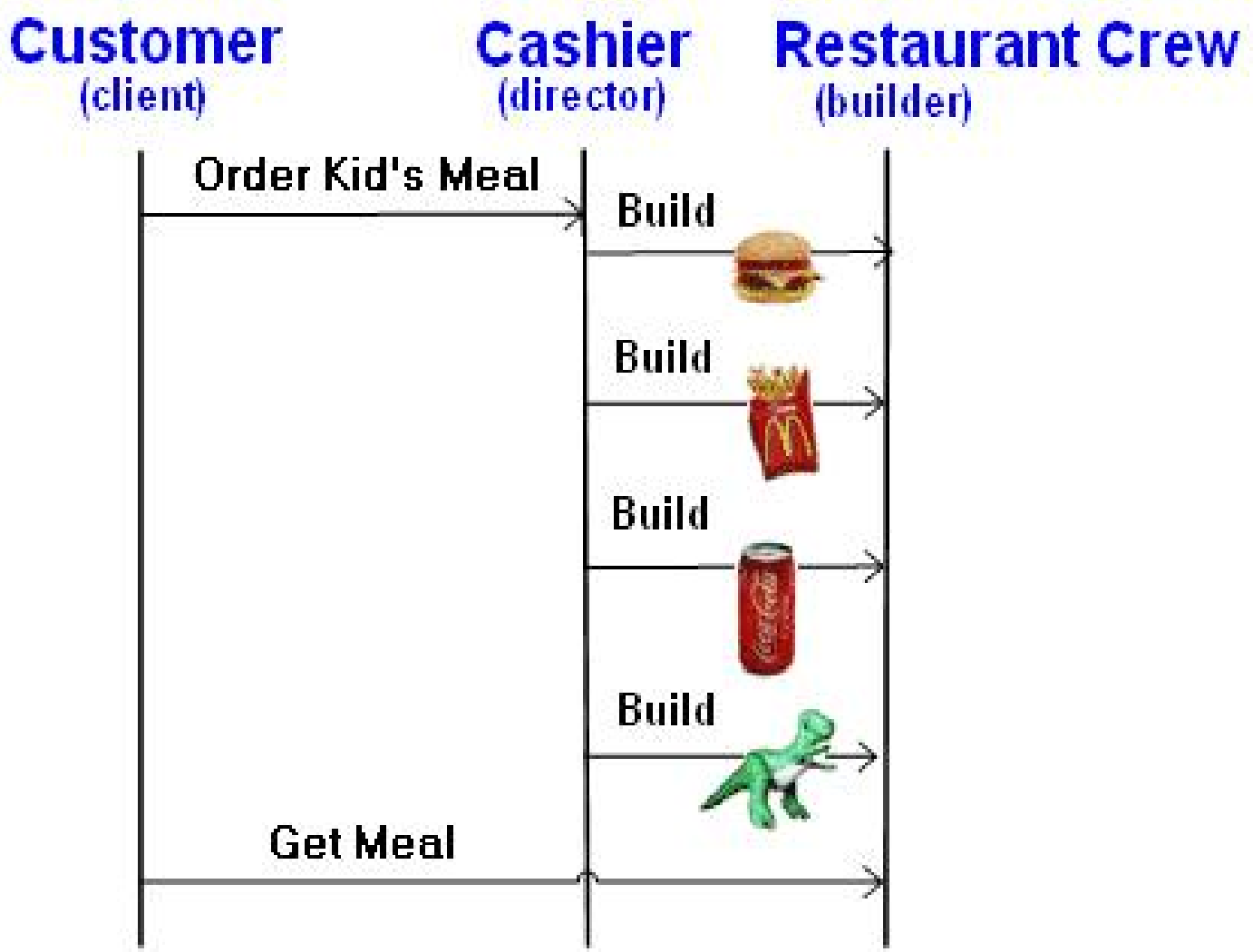
Padrão – Builder

- Diagrama Conceitual



Padrão – Builder

- Exemplo Lúdico



Padrão – Builder

- Exemplo

```
public class CarroProduct {
    double preco;
    String dscMotor;
    int anoDeFabricacao;
    String modelo;
    String montadora;
}
```

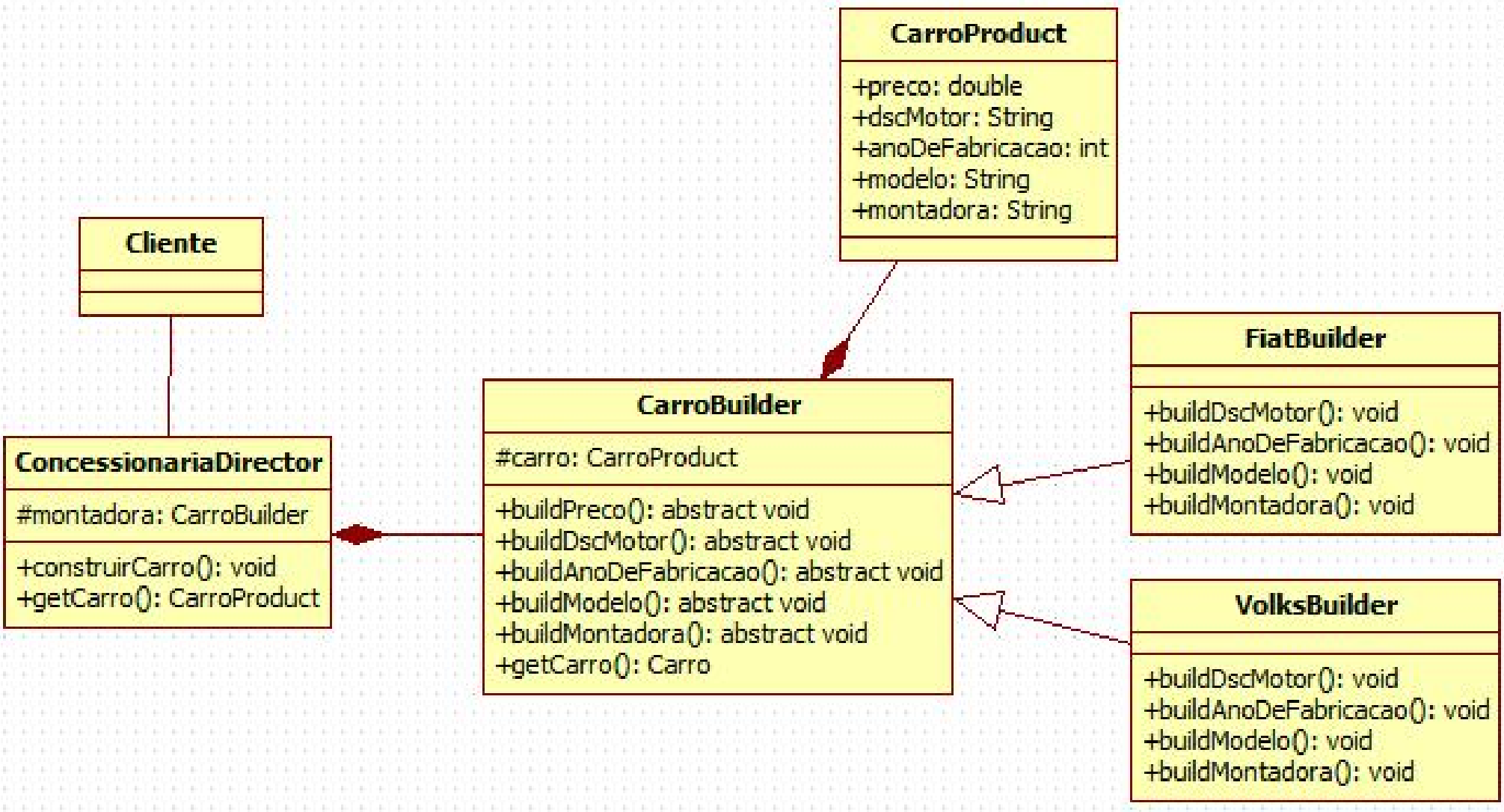
```
public abstract class CarroBuilder {
    protected CarroProduct carro;
    public CarroBuilder() {
        carro = new CarroProduct();
    }
    public abstract void buildPreco();
    public abstract void buildDscMotor();
    public abstract void buildAnoDeFabricacao();
    public abstract void buildModelo();
    public abstract void buildMontadora();
    public CarroProduct getCarro() {
        return carro;
    }
}
```

Padrão – Builder

```
public class FiatBuilder extends CarroBuilder {
    public void buildPreco() {
        carro.preco = 25000.00;
    }
    public void buildDscMotor() {
        carro.dscMotor = "Fire Flex 1.0";
    }
    public void buildAnoDeFabricacao() {
        carro.anoDeFabricacao = 2011;
    }
    public void buildModelo() {
        carro.modelo = "Palio";
    }
   public void buildMontadora() {
        carro.montadora = "Fiat";
    }
}
```

```
public class ConcessionariaDirector {
    protected CarroBuilder montadora;
     public ConcessionariaDirector(CarroBuilder
      montadora) {
        this.montadora = montadora;
    }
     public void construirCarro() {
        montadora.buildPreco();
        montadora.buildAnoDeFabricacao();
        montadora.buildDscMotor();
        montadora.buildModelo();
        montadora.buildMontadora();
    }
     public CarroProduct getCarro() {
        return montadora.getCarro();
    }
}
```

Padrão - Builder



Relacionamentos

- Composite
  - Os objetos construídos com *Builder* normalmente são *Composites*
- Fábricas
  - Builder é um tipo mais específico de Factory
  - Factory podem ser usadas para instanciar objetos *Builder*

Conclusão

- Padrões de Projetos são

LINGUAJAR BÁSICO DOS PROJETISTAS

CATALOGADOS A PARTIR DE EXPERIÊNCIAS

- Builder

ENCAPSULA A CRIAÇÃO DE UM OBJETO

COMPLEXO, EM GERAL COM MUITAS PARTES

- Factory Method / Abstract Factory

ENCAPSULA A COMPLEXIDADE DA CRIAÇÃO DE OBJETOS