Arquitetura de Software

Padrões de Projetos – Padrões Criacionais

Nairon Neri Silva

Padrões de Projeto Criacionais

- Abstraem o processo de instanciação
  - Como seus objetos são criados, compostos e representados

- Importantes à medida que os sistemas evoluem
  - Permitem configurar um sistema com objetos que variam em estrutura e funcionalidade
    - Configuração estática (compilação) ou dinâmica (execução)

Padrões de Projeto Criacionais

1. Abstract Factory
2. **Builder**
3. Factory Method
4. Prototype
5. Singleton

Builder

Padrões Criacionais

Intenção

- Separar a construção de um objeto complexo da sua representação de modo que o mesmo processo de construção possa criar diferentes representações

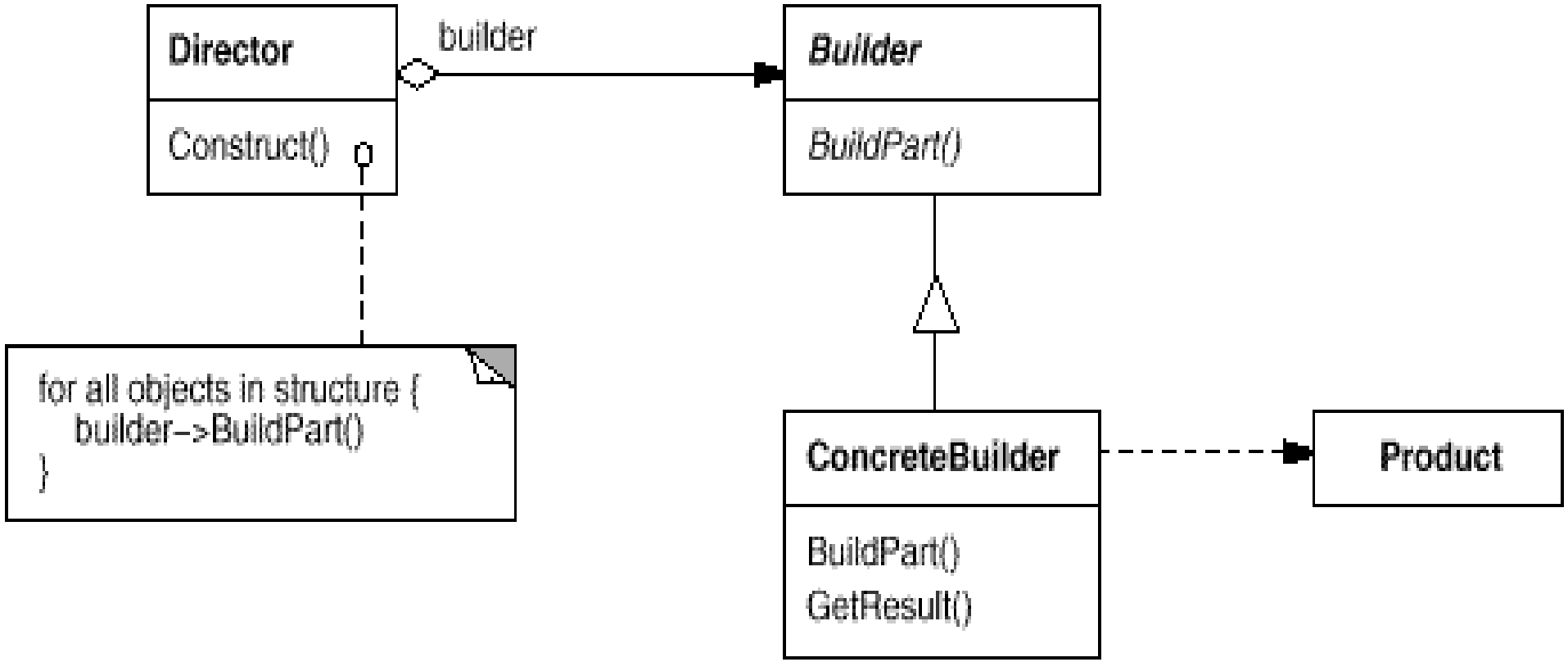
Motivação

- Um leitor de um documento em RTF deveria ser capaz de converter RTF em muitos formatos de texto.
- O leitor poderia converter documentos RTF em ASCII comum ou *widget* de texto, que possa ser editado interativamente.
- O problema, contudo, é que o número de conversões possíveis é aberto.
- Por isso, deve ser fácil acrescentar uma nova conversão sem modificar o leitor.

Aplicabilidade

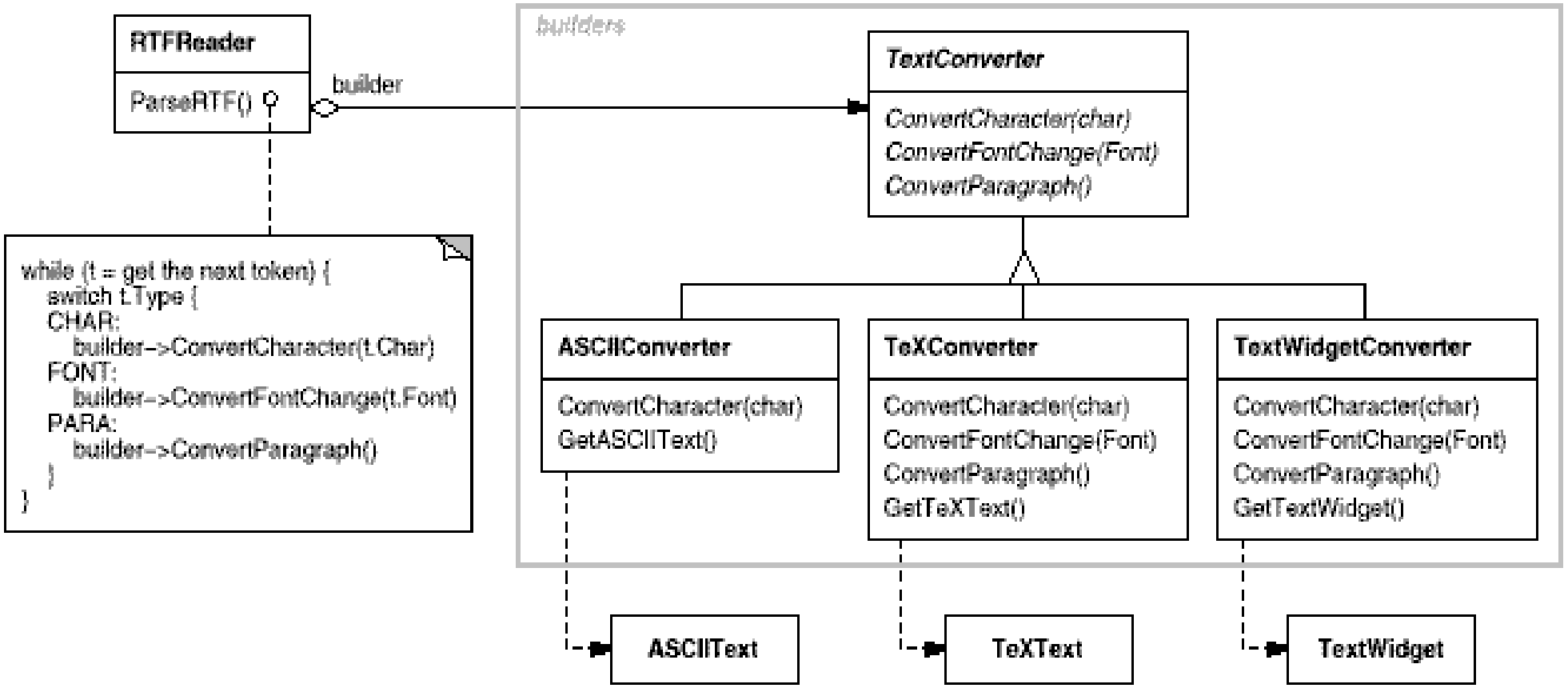
1. Quando o algoritmo para a criação de um objeto complexo deve ser independente das partes que compõem o objeto e como elas são montadas

2. Quando o processo de construção deve permitir diferentes representações para o objeto que é construído


Director
builder
Builder
Construct()
BuildPart()
for all objects in structure {
builder->BuildPart()
}
ConcreteBuilder
Product
BuildPart()
GetResult()

Participantes

- **Builder** (TextConverter)
  - Interface abstrata para criação de partes de um produto
- **ConcreteBuilder** (ASCIIConverter, TeXConverter, TextWidgetConverter)
  - Constrói e monta partes de um produto
- **Director** (RTFReader)
  - Constrói um objeto usando a interface de Builder
- **Product** (ASCIIText, TeXText, TextWidget)
  - Objeto complexo em construção e partes constituintes


RTFReader
ParseRTF()
builder
builders
TextConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
while (t = get the next token) {
switch t.Type {
CHAR:
builder->ConvertCharacter(t.Char)
FONT:
builder->ConvertFontChange(t.Font)
PARA:
builder->ConvertParagraph()
}
}
ASCIIConverter
ConvertCharacter(char)
GetASCIIText()
TeXConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTeXText()
TextWidgetConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTextWidget()
ASCIIText
TeXText
TextWidget

Consequências

1. Permite variar a representação interna de um produto
2. Isola o código para construção e representação
3. Oferece um controle mais fino sobre o processo de construção

Padrões Relacionados

- Abstract Factory é semelhante ao Builder no sentido de que também constrói objetos complexos
  - A diferença principal é que o padrão Builder focaliza a construção de um objeto complexo passo a passo
  - A ênfase do Abstract Factory é sobre famílias de objetos

- Um Composite é o que frequentemente o Builder constrói

Saiba mais...

- https://www.youtube.com/watch?v=2VwLvwsIu-8
- https://www.youtube.com/watch?v=W-96z2EjoJ0
- https://www.youtube.com/watch?v=dbw_BMHEgkY
- https://refactoring.guru/pt-br/design-patterns/builder

Exemplo de Código em C++: https://gist.github.com/pazdera/1121152

Atividade

1) Explique o que é um padrão de projeto;
2) Explique o que são padrões criacionais e qual o objetivo;
3) Explique de forma suscinta o objetivo do padrão Abstract Factory;
4) Explique de forma suscinta o objetivo do padrão Builder;
5) Pesquise, cite e explique pelo menos um situação real de aplicação de cada um dos padrões (diferentes dos citados até o momento) vistos até agora (Abstract Factory e Builder);