Design Patterns

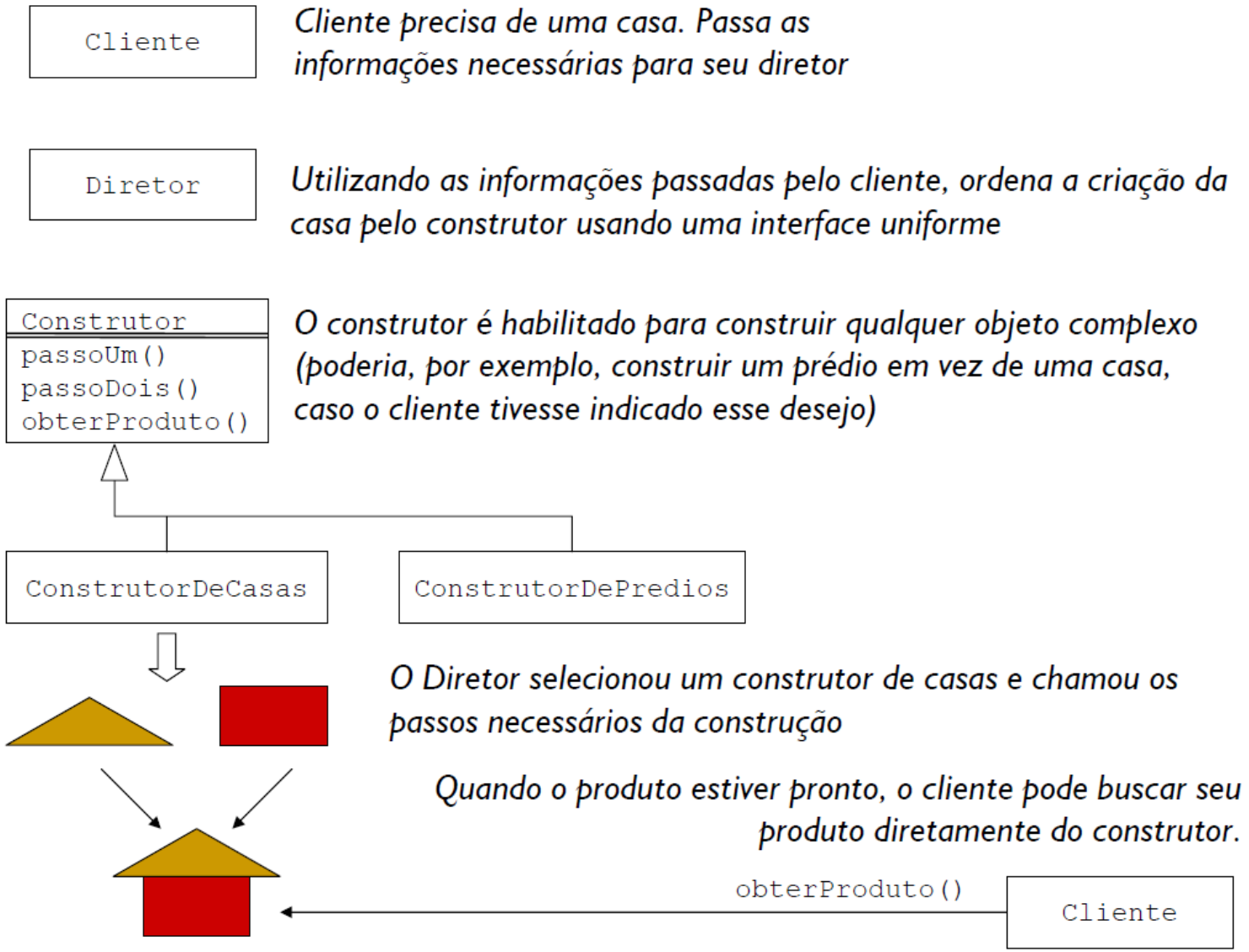
Classificação (GoF)

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Builder

"Separar a construção de um objeto complexo de sua representação para que o mesmo processo de construção possa criar representações diferentes." [GoF]

Problema


Cliente
Cliente precisa de uma casa. Passa as informações necessárias para seu diretor
Diretor
Utilizando as informações passadas pelo cliente, ordena a criação da casa pelo construtor usando uma interface uniforme
Construtor
passoUm()
passoDois()
obterProduto()
O construtor é habilitado para construir qualquer objeto complexo (poderia, por exemplo, construir um prédio em vez de uma casa, caso o cliente tivesse indicado esse desejo)
ConstrutorDeCasas
ConstrutorDePredios
O Diretor selecionou um construtor de casas e chamou os passos necessários da construção
Quando o produto estiver pronto, o cliente pode buscar seu produto diretamente do construtor.
obterProduto()
Cliente

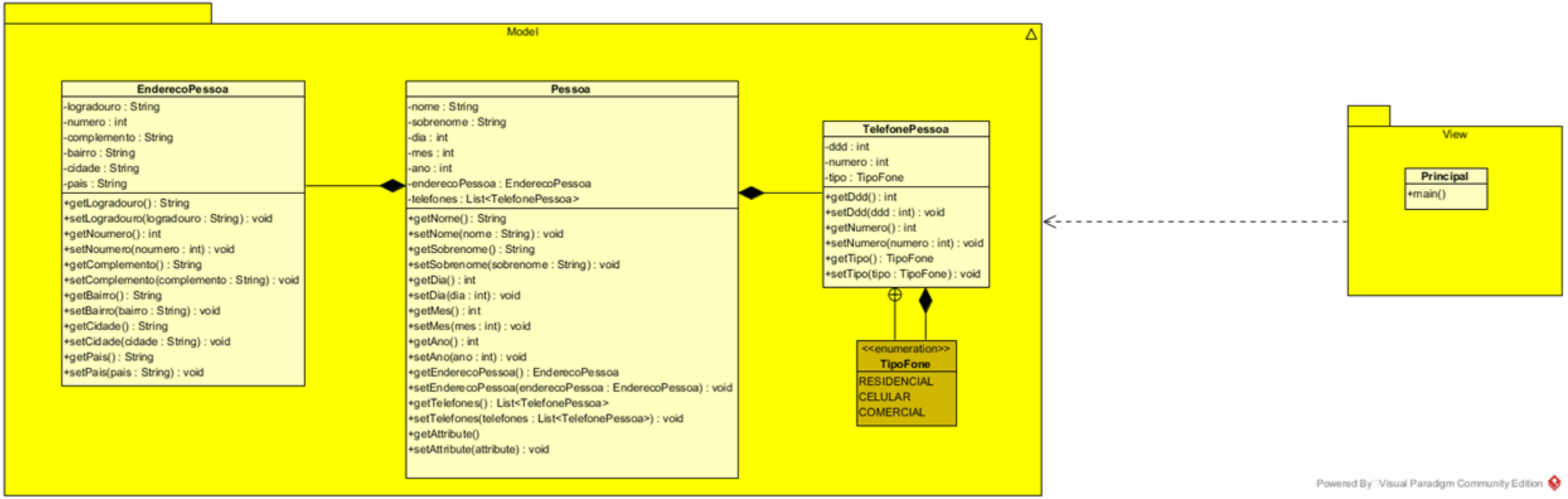
Quando usar ?

- *Builder permite que uma classe se preocupe com apenas uma parte da construção de um objeto. É útil em algoritmos de construção complexos*
  - *Use-o quando o algoritmo para criar um objeto complexo precisar ser independente das partes que compõem o objeto e da forma como o objeto é construído*
- *Builder também suporta substituição dos construtores, permitindo que a mesma interface seja usada para construir representações diferentes dos mesmos dados*
  - *Use quando o processo de construção precisar suportar representações diferentes do objeto que está sendo construído*

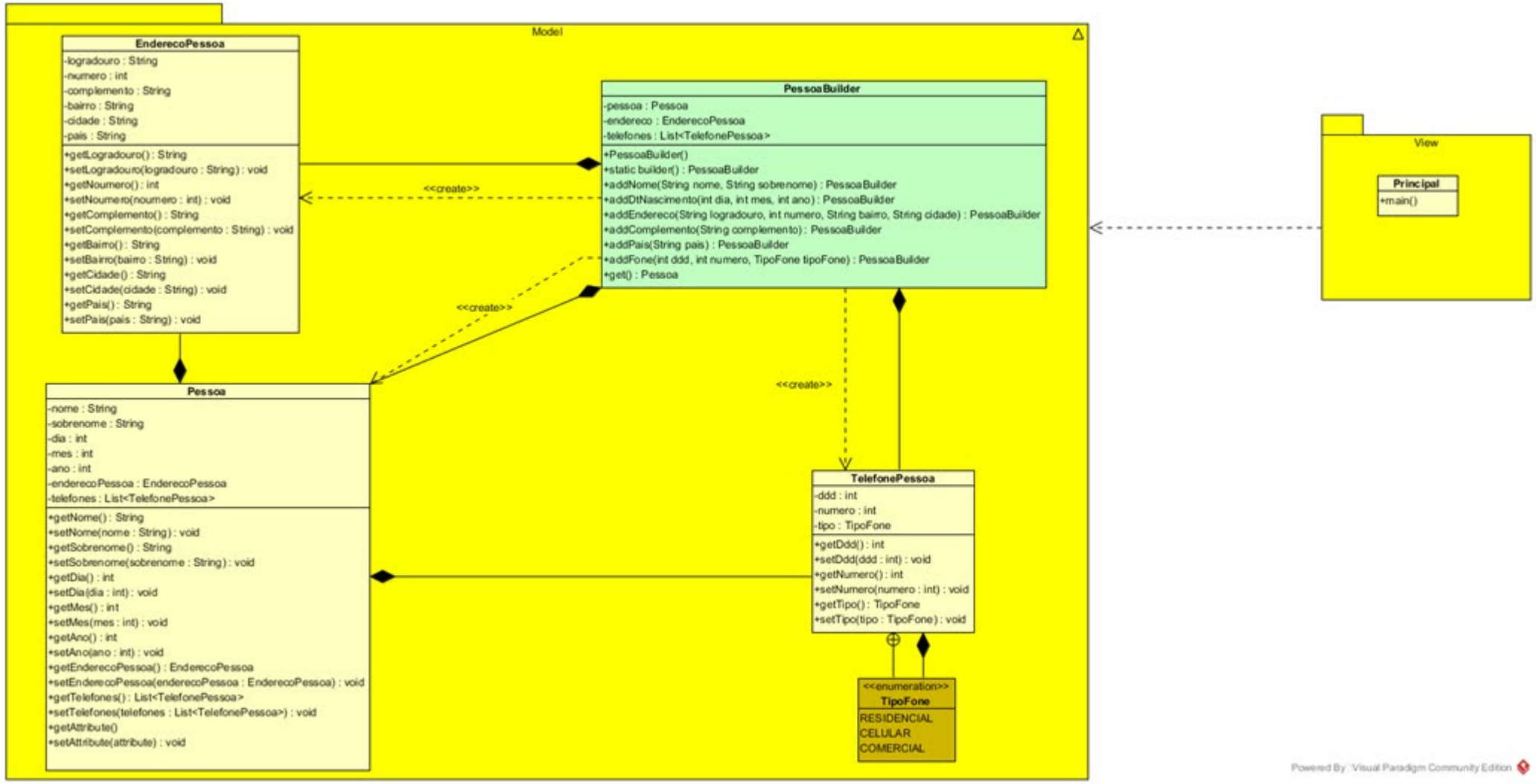
Prática e aplicação em Java

- Considere uma aplicação que deva permitir cadastrar **Pessoas**.
  - Uma pessoa tem diversos atributos, sendo um deles **Endereço**, que tem diversos atributos e outro, uma lista de **Telefones**, que tem vários atributos.
- Alguns dos atributos estão subdivididos, como data de nascimento em dia, mês e ano ou nome completo, dividido em nome e sobrenome.
- Não se deveria permitir uma data de nascimento incompleta ou o nome incompleto
- O diagrama apresenta a implementação

Implementar



Builder


Model
EnderecoPessoa
-logradouro : String
-numero : int
-complemento : String
-bairro : String
-cidade : String
-pais : String
+getLogradouro() : String
+setLogradouro(logradouro : String) : void
+getNoumero() : int
+setNoumero(noumero : int) : void
+getComplemento() : String
+setComplemento(complemento : String) : void
+getBairro() : String
+setBairro(bairro : String) : void
+getCidade() : String
+setCidade(cidade : String) : void
+getPais() : String
+setPais(pais : String) : void
PessoaBuilder
-pessoa : Pessoa
-endereco : EnderecoPessoa
-telefones : List<TelefonePessoa>
+PessoaBuilder()
+static builder() : PessoaBuilder
+addNome(String nome, String sobrenome) : PessoaBuilder
+addDtNascimento(int dia, int mes, int ano) : PessoaBuilder
+addEndereco(String logradouro, int numero, String bairro, String cidade) : PessoaBuilder
+addComplemento(String complemento) : PessoaBuilder
+addPais(String pais) : PessoaBuilder
+addFone(int ddd, int numero, TipoFone tipoFone) : PessoaBuilder
+get() : Pessoa
<<create>>
<<create>>
<<create>>
Pessoa
-nome : String
-sobrenome : String
-dia : int
-mes : int
-ano : int
-enderecoPessoa : EnderecoPessoa
-telefones : List<TelefonePessoa>
+getNome() : String
+setNome(nome : String) : void
+getSobrenome() : String
+setSobrenome(sobrenome : String) : void
+getDia() : int
+setDia(dia : int) : void
+getMes() : int
+setMes(mes : int) : void
+getAno() : int
+setAno(ano : int) : void
+getEnderecoPessoa() : EnderecoPessoa
+setEnderecoPessoa(enderecoPessoa : EnderecoPessoa) : void
+getTelefones() : List<TelefonePessoa>
+setTelefones(telefones : List<TelefonePessoa>) : void
+getAttribute()
+setAttribute(attribute) : void
TelefonePessoa
-ddd : int
-numero : int
-tipo : TipoFone
+getDdd() : int
+setDdd(ddd : int) : void
+getNumero() : int
+setNumero(numero : int) : void
+getTipo() : TipoFone
+setTipo(tipo : TipoFone) : void
<<enumeration>>
TipoFone
RESIDENCIAL
CELULAR
COMERCIAL
View
Principal
+main()
Powered By: Visual Paradigm Community Edition

Exercício

- Considere uma aplicação que permite exibir os diversos produtos de uma loja. A loja vende diversos produtos, como camisetas, calçados, jogos eletrônicos, equipamentos de informática de armazenamento.
  - Para calçados, é necessário saber o tamanho (35 – 44), cor, tipo (tênis ou social) e valor
  - Para camisetas, é necessário saber o tamanho (PP, P, M, G, GG), cor, marca e valor
  - Para jogos eletrônicos, é necessário saber o nome, o videogame (Xbox, Playstation ou Switch) e o valor
  - Para equipamentos de armazenamento, é necessário saber o tipo (SSD, M.S NVMe ou HDD), a capacidade, o fabricante e o valor
- Fazer o diagrama de pacotes e implementar em Java, a solução para que essa aplicação tenha 3 produtos de cada tipo no método main, coloque em um ArrayList e exiba os 12 produtos. Considere usar enums para os casos em que se aplica.