Design Patterns / Padrões de Projeto
Exercícios – Padrões de Criação

1. Abstract Factory

1.1. Exercício:

Crie um "Hello, World" que utilize o padrão *Abstract Factory* para escolher dentre duas formas de impressão: (a) na tela ou (b) num arquivo chamado `output.txt`. Seu programa deve escolher dentre as duas fábricas aleatoriamente.

1.2. Exercício:

Considere os seguintes conceitos do mundo real: pizzaria, pizzaiolo, pizza, consumidor. Considere ainda que em uma determinada pizzaria, dois pizzaiolos se alternam. Um deles trabalha segundas, quartas e sextas e só sabe fazer pizza de calabresa (queijo + calabresa + tomate), o outro trabalha terças, quintas e sábados e só sabe fazer pizza de presunto (queijo + presunto + tomate). A pizzaria fecha aos domingos.

Tente mapear os conceitos acima para o padrão Abstract Factory (hierarquia de fábricas, hierarquia de produtos, cliente) e implemente um programa que receba uma data como parâmetro (formato `dd/mm/yyyy`) e imprima os ingredientes da pizza que é feita no dia ou, se a pizzaria estiver fechada, informe isso na tela.

Agora imagine que a pizzaria agora faz também calzones (novamente, de calabresa ou presunto). Complemente a solução com mais este componente.

2. Builder

2.1. Exercício:

Na cadeia de restaurantes *fast-food PatternBurgers* há um padrão para montagem de lanches de crianças. O sanduíche (hambúrguer ou *cheeseburger*), a batata (pequena, média ou grande) e o brinquedo (carrinho ou bonequinha) são colocados dentro de uma caixa e o refrigerante (coca ou guaraná) é entregue fora da caixa. A classe abaixo é dada para representar o pedido de um consumidor:

```
import java.util.*;

public class Pedido {
	private Set<String> dentroDaCaixa = new HashSet<String>();
	private Set<String> foraDaCaixa = new HashSet<String>();

	public void adicionarDentroDaCaixa(String item) {
		dentroDaCaixa.add(item);
	}

	public void adicionarForaDaCaixa(String item) {
		foraDaCaixa.add(item);
	}

	public String toString() {
		StringBuffer buffer = new StringBuffer();
		buffer.append("Seu pedido:\n");
		buffer.append("Dentro da caixa:\n");
		for (String item : dentroDaCaixa) buffer.append("\t" + item + "\n");
		buffer.append("Fora da caixa:\n");
		for (String item : foraDaCaixa) buffer.append("\t" + item + "\n");
		buffer.append("\nTenha um bom dia!\n\n");
		return buffer.toString();
	}
}
```

Neste caso, o padrão *Builder* pode ser usado para separar as tarefas do atendente e do funcionário que monta o pedido. Somente este último sabe como montar os pedidos segundo os padrões da empresa, mas é o atendente quem lhe informa quais itens o consumidor pediu.

Implemente a simulação do restaurante *fast-food* descrita acima utilizando o padrão *Builder* e escreva uma classe cliente que pede um lanche ao atendente, recebe-o do outro funcionário e imprime o pedido.

2.2. Exercício:

Escreva classes para satisfazer os seguintes papéis do padrão *Builder*:

- *Client*: recebe como parâmetros o nome, endereço, telefone e e-mail de uma pessoa, solicita ao *director* que construa informações de contato, recupera a informação do *builder* e imprime;
- *Director*: recebe como parâmetro o *builder* a ser utilizado e os dados de contato. Manda o *builder* construir o contato;
- *Builder*: constrói o contato. Existem três tipos de contato e um *builder* para cada tipo:
  - `ContatoInternet`: armazena nome e e-mail;
  - `ContatoTelefone`: armazena nome e telefone;
  - `ContatoCompleto`: armazena nome, endereço, telefone e e-mail.

A classe que representa o papel *client* deve ter o método `main()` que irá criar um *director* e um *builder* de cada tipo. Em seguida, deve pedir ao *director* que crie um contato de cada tipo e imprimi-los (use o `toString()` da classe que representa a informação de contato).

3. Factory Method

3.1. Exercício:

Construa um programa que receba como parâmetro um ou mais nomes, cada um podendo estar em um dos seguintes formatos:

- `"nome sobrenome"`;
- `"sobrenome, nome"`.

Escreva duas aplicações de construção de nomes, uma para cada formato. Cada uma deve ser responsável por armazenar os nomes criados e imprimi-los quando requisitado. Implemente o padrão *Factory Method* de forma que somente a criação do nome seja delegada às aplicações concretas (subclasses). Seu programa deve criar as duas aplicações e construir objetos da classe `Nome`, que deve ter propriedades `nome` e `sobrenome` para armazenar as informações em separado. Os nomes não precisam ser impressos em ordem.

Ex.:

```
$ java Nomes "McNealy, Scott" "James Gosling" "Naughton, Patrick"
James Gosling
Scott McNealy
Patrick Naughton
```

3.2. Exercício:

Crie dois arquivos texto em um diretório qualquer:

publico.txt

```
Estas são informações públicas sobre
qualquer coisa. Todo mundo pode ver
este arquivo.
```

confidencial.txt

```
Estas são informações confidenciais,
o que significa que você
provavelmente sabe a palavra secreta!
```

Usando o padrão *Factory Method*, crie duas provedoras de informação: uma que retorna informações públicas e outra que retorna informações confidenciais. Utilize o provedor confidencial se o usuário informar a senha "`designpatterns`" como parâmetro para o programa, que deve recuperar a informação e exibi-la na tela.

3.3. Exercício:

Escreva um programa que conte até 10 e envie os números para uma ferramenta de log. Esta ferramenta de log deve ser construída por uma fábrica. Utilize *Factory Method* para permitir a escolha entre dois tipos de log: em arquivo (`log.txt`) ou diretamente no console. A escolha deve ser por um parâmetro passado ao programa ("`arquivo`" ou "`console`").

4. Singleton

4.1. Exercício:

Escreva, compile e execute o programa abaixo. Em seguida, troque sua implementação para que a classe Incremental seja *Singleton*. Execute novamente e veja os resultados.

```
class Incremental {
	private static int count = 0;
	private int numero;

	public Incremental() {
		numero = ++count;
	}

	public String toString() {
		return "Incremental " + numero;
	}
}

public class TesteIncremental {
	public static void main(String[] args) {
		for (int i = 0; i < 10; i++) {
			Incremental inc = new Incremental();
			System.out.println(inc);
		}
	}
}
```