Lista de Exercícios - Padrões de Projeto

1. Imagine um leopardo em uma caçada. Quando encontra outro animal, ele toma duas ações: uma inicial e outra final. Essas ações podem variar de estratégia de acordo com as características do animal a sua frente, conforme a tabela abaixo. Aplique o padrão Strategy para criar um simulador de caçadas do leopardo.

Animal Avistado	Estratégia	Ação inicial	Ação final
Gazela	Atacar sozinho	Aproximar furtivamente.	Atacar, pular em cima, morder o pescoço.
Bisão	Atacar em bando	Agrupar companheiros. Separar um bisão dos demais.	Atacar junto, derrubar, morder o pescoço.
Leão	Prezar pela vida	Rezar para não ter sido visto.	Fugir como se sua vida dependesse disso (pois ela depende).

2. Você foi contratado por uma empresa que desenvolve aplicações para edição e manipulação de imagens. Seu chefe apresentou para você as seguintes classes:

```
public class Visualizador{

        public void Visualizar(){
            Imagem img = new Imagem();
            img.carregar();
            img.exibir();
            img.fechar();
        }
}

public class Imagem{

    public void carregar() {
           System.out.println("Imagem BMP:");
           System.out.println("Carregando imagem BMP...");
           System.out.print("...");
           System.out.print("...");
           System.out.print("");
    }

    public void exibir() {
           System.out.println("Exibindo imagem por 20 segundos.");
    }

    public void fechar() {
           System.out.println("Fechando imagem.");
    }
}
```

Essas classes são responsáveis por carregar imagens do tipo BMP. Seu chefe mandou que você alterasse o código de forma que a criação de novos visualizadores de imagem ficasse mais flexível. Crie as seguintes classes: VisualizadorJPG – que visualiza imagens do tipo JPG e ImagemJPG – que trata de imagens JPG. O código deve ser desenvolvido de forma que a criação de qualquer outro visualizador seja rápida e flexível. Para resolver esse problema altere o código acima usando o padrão Factory Method.

3. 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.

4. Na biblioteca existe a lista de espera de determinado livro. Existe uma única lista de espera para cada livro e quando o livro retorna para a biblioteca os alunos inscritos na lista recebem a notificação de que o livro está disponível. Crie um programa a partir do contexto apresentado, considerando que o padrão de projeto Observer.

5. Crie uma classe NumeroUm que tem um método imprimir() que imprime o número “1” na tela. Implemente decoradores para colocar parênteses, colchetes e chaves ao redor do número (ex.: “{1}”). Combine-os de diversas formas.

6. A seguir estão os códigos fonte de um cliente, uma interface para um somador que ele espera utilizar e uma classe concreta que implementa uma soma, mas não da maneira esperada pelo cliente. Como você pode ver abaixo, o cliente espera usar uma classe que soma inteiros em um vetor, mas a classe pronta soma inteiros em uma lista. Crie um adaptador para resolver esta situação.

```
public class Cliente {

	private SomadorEsperado somador;

	private Cliente(SomadorEsperado somador) {
		this.somador = somador;
	}

	public void executar() {
		int[] vetor = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
		int soma = somador.somaVetor(vetor);
		System.out.println("Resultado: " + soma);
	}
}

public interface SomadorEsperado {
	int somaVetor(int[] vetor);
}

public class SomadorExistente {

	public int somaLista(List<Integer> lista) {
		int resultado = 0;
		for (int i : lista) resultado += i;
			return resultado;
	}

}
```