**Linguagem de Programação III – Turma TI13 Prof. Anderson Konzen**

**Exercícios de Revisão Prova 1**

1. Os Padrões de Projetos em Orientação a Objeto são arquiteturas utilizadas para desenvolver softwares orientados a objetos flexíveis e sustentáveis assim reduzindo sua complexidade. O modelo Singleton é utilizado para
   1. remover um objeto da memória.
   2. instanciar o objeto quantas vezes for necessário.
   3. garantir que o objeto tenha apenas uma instância.
   4. transformar o objeto em um método.
   5. garantir a associação entre objetos.
2. A empresa CTRL-ALT-DEL foi contratada para desenvolver aplicativos de uma instituição financeira que possam ser executados em diferentes plataformas, como PC, celular e tablet. Para evitar a necessidade de criar diferentes soluções para cada plataforma, a empresa decidiu adotar um padrão de projeto que defina uma família de componentes para cada plataforma e uma implementação que os instancie de acordo com a plataforma-alvo na qual a aplicação estará sendo executada. O padrão adequado a essa implementação é:
   1. Prototype
   2. Adapter
   3. Template Method
   4. Abstract Factory
   5. Strategy
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. Construa um programa que receba como parâmetro um ou mais nomes, cada um podendo estar em um dos seguintes formatos:
   1. "nome sobrenome";
   2. "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.

Exemplo:

* + java Nomes "McNealy, Scott" "James Gosling" "Naughton, Patrick" James Gosling

Scott McNealy Patrick Naughton

1. 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”).
2. Considere:

I. A aplicação cliente interage com uma fábrica genérica de objetos.

II. Os objetos serão gerados efetivamente pela fábrica concreta que estiver sendo utilizada no momento.

III. A aplicação cliente não precisa ser configurada para interagir com cada uma das fábricas concretas.

IV. Novas fábricas concretas podem ser agregadas, alteradas ou retiradas do sistema sem necessidade de alterações na aplicação cliente.

Estas são características e propriedades do padrão de projeto

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

1. Escreva um programa que exiba uma mensagem diferente para cada dia da semana usando o padrão Strategy.
2. O Singleton é um padrão de projeto usado em orientação a objetos que limita a instanciação de uma classe a um único objeto. Analise a seguinte implementação de um Singleton em Java:

**public class** Singleton {

**private static final** Singleton ***instance*** = **new** Singleton();

**private** Singleton() {

}

**public static** Singleton getInstance() {

**return *instance***;

}

}

Considere as três afirmações seguintes acerca da implementação mostrada: I. a declaração do construtor da classe poderia ser omitida sem prejuízo do requisito de unicidade da instância;

1. a declaração do construtor da classe é indispensável, dado o requisito de unicidade da instância, porém o modificador private poderia ser omitido; III. a instância única será construída quer a classe Singleton seja referenciada na aplicação quer não.

Assinale a alternative correta:

* 1. I, II e III são verdadeiras;
  2. I e III são verdadeiras, II é falsa;
  3. I, II e III são falsas;
  4. I e II são verdadeiras, III é falsa;
  5. II e III são verdadeiras, I é falsa.

1. A criação, por meio do Singleton, de várias instâncias de uma mesma classe reduz o desempenho da aplicação. No seu julgamento, essa afirmação está
   1. Correta
   2. Errada
2. O princípio da responsabilidade única estabelece que uma classe deva executar apenas uma tarefa; dessa forma, caso uma classe possua mais uma responsabilidade, deve-se considerar sua decomposição em duas ou mais classes.
   1. Correto
   2. Errado
3. Considere a classe abaixo:

public class Book {

private String name;

private String author;

private String text;

//constructor, getters and setters

* methods that directly relate to the book properties public String replaceWordInText(String word){

return text.replaceAll(word, text);

}

public boolean isWordInText(String word){ return text.contains(word);

}

void printTextToConsole(){

// our code for formatting and printing the text

}

}

Esta classe viola o princípio de SRP, pois além de ter os métodos relacionados ao livro em si, contém um método responsável por imprimir os dados do livro. Apresente uma solução para este caso.

Uma nova classe pode ser criada para lidar apenas com a impressão do livro, tirando assim essa responsabilidade da classe Book.

12. Considere as classe abaixo:

public interface Car {

void turnOnEngine();

void accelerate();

}

public class MotorCar implements Car {

private Engine engine;

//Constructors, getters + setters

public void turnOnEngine() {

//turn on the engine!

engine.on();

}

public void accelerate() {

//move forward!

engine.powerOn(1000);

}

}

public class ElectricCar implements Car {  
 public void turnOnEngine() {

throw new AssertionError("I don't have an engine!");

}

public void accelerate() {

//this acceleration is crazy!

}

}

No caso da classe ElectricCar, estamos violando o princípio de Liskov.

Sugira (com exemplos de código) como poderíamos reverter isso.  
Nesse caso poderíamos refatorar as interfaces para termos alguma relacionada a carros com e sem motor elétrico.

13. Considere as classes abaixo:

public interface Bird {

public void fly();

public void molt();

}

public class Eagle implements Bird {

String currentLocation;

int numberOfFeathers;

public Eagle(int initialFeatherCount) { this.numberOfFeathers = initialFeatherCount;

}

public void fly() {

this.currentLocation = "in the air";

}

public void molt() {

this.numberOfFeathers -= 1;

}

}

public class Penguin implements Bird {

String currentLocation;

int numberOfFeathers;

public Penguin(int initialFeatherCount) { this.numberOfFeathers = initialFeatherCount;

}

public void molt() {

this.numberOfFeathers -= 1;

}  
public void fly() {

throw new UnsupportedOperationException();

}

public void swim() {

this.currentLocation = "in the water";

}

}

Esta implementação fere o princípio da segregação de interface. Sugira mudanças (com código) que façam com que o programa acima respeite este princípio.  
Nesse caso poderíamos ter interfaces mais específicas que tenham ações mais específicas, ao invés de ter uma interface única para Bird cuja tendência é crescer à medida que formos acrescentando novas ações.

14. Observe o código abaixo:

public class BackEndDeveloper {

public void writeJava() {

}

}

public class FrontEndDeveloper {

public void writeJavascript() {

}

}

public class Project {

private BackEndDeveloper backEndDeveloper = new BackEndDeveloper(); private FrontEndDeveloper frontEndDeveloper =

new FrontEndDeveloper();

public void implement() { backEndDeveloper.writeJava(); frontEndDeveloper.writeJavascript();

}

}

Este código fere o princípio da inversão de dependência, pois o Project – módulo de alto nível – depende de módulos de baixo nível como BackEndDeveloper. Note também os métodos writeJava() e writeJavascript() são do contexto das classes correspondentes; no escopo do Project, eles são detalhes pois ambos são formas de desenvolvimento.

Como você melhoraria esse código?  
Poderíamos introduzir uma interface que abstraia a função Developer, e refatorar as classes BackendDeveloper e FrontendDeveloper para usar esta interface. Por final a classe Project seria refatorada para usar esta nova abstração e não depender das classes concretas.