CENTRO UNIVERSITÁRIO SANTO AGOSTINHO

Bacharelado em Engenharia de Software

Disciplina: Design e Arquitetura de Software

Lista de Exercícios

1. (**DRY**) Você está desenvolvendo um sistema de **vendas** para uma loja de eletrônicos. Durante a fase de manutenção, um colega te mostrou a classe responsável pelos **descontos**. Sua tarefa é verificar se esse código está de acordo com boas práticas e propor uma refatoração que **reduza a duplicação de código**.

```
public class CalculadoraDescontos {

    public double calcularDescontoProdutoA(double preco) {
        return preco - (preco * 0.1);
    }

    public double calcularDescontoProdutoB(double preco) {
        return preco - (preco * 0.15);
    }

    public double calcularDescontoProdutoC(double preco) {
        return preco - (preco * 0.2);
    }
}
```

2. (**KISS**) Um estagiário do seu time criou uma **classe para cálculo de fatorial**, e te pediu para revisar o código antes de usar em produção. Aplique o princípio KISS para torná-lo simples.

```
public class Fatorial {
    public int calcularFatorial(int n) {
        if (n == 0) return 1;
        else if (n == 1) return 1;
        else {
            int resultado = 1;
            for (int i = 1; i <= n; i++) {
                resultado = resultado * i;
            }
            return resultado;
        }
    }
}
```

3. (**DRY+KISS**) Seu cliente pediu um **validador de senha** para o formulário de cadastro. Ele precisa garantir que a senha seja segura. Implemente um validador de senha aplicando os princípios de DRY e KISS, com as seguintes regras:

- No mínimo 8 caracteres.
- Pelo menos 1 número.
- Pelo menos 1 letra maiuscula.

4. (**DRY+KISS**)Crie uma classe calculadora. Esta classe deve ser abstrata e implementar as operações básicas (soma, subtração, divisão e multiplicação). Utilizando o conceito de herança, crie uma classe chamada calculadora científica que implementa os seguintes cálculos: raiz quadrada e a potência.

5. (**DRY+KISS**) Crie uma classe em Java chamada **Fatura** para uma loja de suprimentos de informática. A classe deve conter quatro variáveis – o **número** (String), a **descrição** (String), a **quantidade comprada de um item** (int) e o **preço por item** (double). A classe deve ter um **construtor** e um método **get e set** para cada variável de instância. Além disso, deve fornecer um método chamado **getTotalFatura** que calcula o valor da fatura e depois retorna o valor como um double. Se o valor não for positivo, ele deve ser configurado como 0. Se o preço por item não for positivo, ele deve ser configurado como 0.0. Escreva um menu de teste chamado **FaturaTeste** (classe com o main) que demonstra as capacidades da classe Fatura.

6. (**SOLID**)Escreva um programa para armazenar dados de **veículos**, aplicando os conceitos de SOLID, com os seguintes requisitos
    a. Uma classe para **Motor** que contém **NumCilindro** (int) e **Potenci**(int). Inclua um **construtor** sem argumentos que inicialize os dados com zeros e um que inicialize os dados com os valores recebidos como argumento. Acrescente duas funções, uma para a entrada de dados, **Set**(), e uma que imprima os dados, **Print**().

b. Uma classe para **Veiculo** contendo **Peso** em quilos (int), **VelocMax** em Km/h (int) e **Preco** em R$ (float). Inclua um **construtor** sem argumentos que inicialize os dados com os valores recebidos como argumento. Acrescente duas funções, uma para a entrada de dados, **Set**(), e uma que imprima os dados, **Print**().
c. Uma classe para **CarroPasseio** derivada das classes **Motor** e **Veículo** como base. Inclua **Cor** (string) e **Modelo** (string). Inclua um **construtor** sem argumentos que inicialize os dados com zeros e uma que inicialize os dados com os valores recebidos como argumentos. Acrescente duas funções, uma para a entrada de dados, **Set**(), e uma que imprima os dados, **Print**().
d. Uma classe para **Caminhao** derivada das classes **Motor** e **Veículo**. Inclua **Toneladas** (carga máxima), **AlturaMax** (int) e **Comprimento** (int). Inclua um **construtor** sem argumentos que inicialize os dados com zeros e um que inicialize com os valores recebidos como argumento. Acrescente duas funções, uma para a entrada de dados, **Set**(), e uma que imprima os dados, **Print**().
e. **Requisitos Adicionais**:

- Encapsulamento: todas as variáveis devem ser privadas, com acesso controlado via getters e setters.
- Utilize boas práticas de nomenclatura (Java naming conventions).
- Implemente uma classe principal chamada Main para testar os objetos e seus métodos.

- Organize o código em múltiplos arquivos, cada classe em seu próprio .java.

7. (**SOLID**) Uma empresa deseja melhorar a estrutura de seu sistema de Recursos Humanos. Atualmente, existe uma classe chamada **Empregado** que representa um funcionário com **nome, sobrenome e salário mensal**. O sistema deve **calcular o salário anual** dos empregados e permitir a aplicação de **aumentos salariais** de maneira flexível (por percentual, valor fixo, entre outras formas futuras).
   Para facilitar a manutenção, evolução e reutilização do código, o setor de desenvolvimento decidiu aplicar os princípios **SOLID** ao redesenhar essa funcionalidade.
   Requisitos:
   a. Crie a classe Empregado com as seguintes propriedades:

- nome (String)
- sobrenome (String)
- salarioMensal (double)
- Crie o construtor, os métodos get e set, com validação para que o salário não seja negativo.

b. Implemente uma estrutura que utilize os princípios SOLID:

- SRP: Separe a lógica de cálculo de salário da classe Empregado.
- OCP: Permita a adição de novos tipos de aumento sem alterar as classes existentes.
- LSP: Garanta que políticas de aumento diferentes possam ser usadas de forma intercambiável.
- ISP: Use interfaces específicas para cálculo de salário e aplicação de aumento.

- DIP: As classes principais (por exemplo, a classe de teste) devem depender de interfaces, e não de classes concretas.

c. Implemente ao menos duas classes auxiliares:

- Uma para calcular o salário anual.
- Uma para aplicar um aumento percentual de salário.

d. Crie uma classe de teste (EmpregadoTeste) que:

- Crie dois objetos Empregado com dados fictícios.
- Exiba o salário anual de cada um.
- Aplique um aumento de 10%.
- Exiba novamente o salário anual.

Dicas

- Use abstrações (interface) para representar o comportamento esperado, como o cálculo do salário anual ou o tipo de aumento.
- Use boas práticas de nomenclatura, encapsulamento e coesão.
- Mantenha cada classe com uma única responsabilidade clara.
- Comente seu código explicando como os princípios foram aplicados.