Universidade Estadual de Campinas - FT
31/03/2025

Técnicas de Revisão

Disciplina: SI304 – Engenharia de Software II
1º semestre de 2025
Prof. Dr. Plínio Roberto Souza Vilela

Alunos:
Líder: Jose Vitor Dutra Antônio - 187174
Laura Rodrigues Russo - 235826
Adriano Baumgarte Bassani Filho - 288824
Thiago Gonçalves Dos Santos - 248799
Marcelo Dos Santos Da Boa Morte - 184524
Roberto Kendy Hassobe - 224141
Vitor Luiz Leal Da Silva - 281459
Maria Clara Souza Muharem - 266864

Sumário

1. Diretrizes de Revisão - Lista de Verificação

1. Conformidade com Requisitos

- O código atende todos os requisitos do projeto ou da tarefa?
- Caso utilize bibliotecas externas ou código de terceiros, ele é bem integrado e seguro?

2. Estrutura do Código

- O código está bem organizado e modularizado?
- Os nomes dos arquivos e pastas seguem um padrão consistente e claro?
- O código é fácil de entender?
- O código é modular e reutilizável?
- Os nomes de variáveis, funções e classes são descritivos?

3. Qualidade do Código

- Há repetição de código?
- O código é bem comentado, explicando partes complexas ou lógicas de negócios?
- O código não tem complexidade excessiva?
- Há comentários desnecessários ou falta de explicações importantes?

4. Desempenho

- O código é eficiente em termos de tempo de execução e uso de memória?
- As tipagens escolhidas são as mais adequadas ao contexto?

5. Tratamento de Erros e Exceções

- As entradas de usuários ou sistemas são validadas corretamente?
- As exceções são tratadas adequadamente, sem ignorar erros importantes?

2. Reunião de Revisão

2.1. Desafio 1

Programa em C que, dado o consumo de uma residência em m³, calcula o valor da conta de água daquela residência.
A linguagem de programação utilizada foi C.

Desenvolvedor: Vitor Luiz Leal Da Silva .
Registrador: Thiago Gonçalves Dos Santos.
Revisores: Maria Clara Souza Muharem, Thiago Gonçalves Dos Santos, Vitor Luiz Leal Da Silva e Adriano Baumgarte Bassani Filho.

Faixa de consumo (m3)	Preço (por m3)
até 10	incluído na franquia
de 11 a 30	R$ 1,00
de 31 a 100	R$ 2,00
de 101 em diante	R$ 5,00

```C/C++
#include <stdio.h>

int ate10 (int m3) { //valor fixo de 7.00 até 10m3
    int valor = 7;
    return valor;
}

int de11ate30 (int m3) {
    int valor = 0;
    for (int i = 10; i < m3; i++){ //calcula o valor acima dos 7.00
        valor = valor + 1;         //para somar o novo valor
    }
    valor = valor + 7;
    return valor;
}

int de31ate100 (int m3) {
    int valor = 0;
    for (int i = 30; i < m3; i++){
        valor = valor + 2;
    }
    valor =valor + 27;
```

```
    return valor;
}

int de101aDiante (int m3) {
    int valor = 0;
    for (int i = 100; i < m3; i++){
        valor = valor + 5;
    }
    valor = valor + 167;
    return valor;
}

int main() {
    int consumo;
    printf("Digite o numero em m3 de consumo total mensal: ");
    scanf("%d", &consumo);
    if (consumo >= 0 && consumo <= 10){
        printf("O valor mensal eh de: %d ", ate10(consumo));
    }
    else if (consumo > 10 && consumo <= 30){
        printf("O valor mensal eh de: %d ", de11ate30(consumo));
    }
    else if ( consumo > 30 && consumo <= 100){
        printf("O valor mensal eh de: %d ", de31ate100(consumo));
    }
    else if (consumo >= 101){
        printf("O valor mensal eh de: %d ", de101aDiante(consumo));
    }
    else{
        printf("Impossivel calcular o valor mensal de consumo");
    }

    return 0;
}
```

1. Conformidade com Requisitos

- O código atende todos os requisitos do projeto ou da tarefa?
  Revisão: Sim, atende todos os requisitos da tarefa.
- Caso utilize bibliotecas externas ou código de terceiros, ele é bem integrado e seguro?
  Revisão: Não utiliza códigos de terceiros ou bibliotecas externas, portanto não se aplica.

2. Estrutura do Código

- O código está bem organizado?
  Revisão: Sim, o código está bem organizado, com uma estrutura clara e legível.
- Os nomes dos arquivos e pastas seguem um padrão consistente e claro?
  Revisão: Não se aplica ao código .
- O código é fácil de entender?
  Revisão: Sim, o código é fácil de entender.
- O código é modular e reutilizável?
  Revisão: O código em questão é modular e com alterações pode ser reutilizável.
- Os nomes de variáveis, funções e classes são descritivos?
  Revisão: Sim, tanto os nomes das funções, quanto das variáveis atende ao critério.

3. Qualidade do Código

- Há repetição de código?
  Revisão: Os nomes são similares mas devido ao contexto acabam complementando o código.
- O código é bem comentado, explicando partes complexas ou lógicas de negócios?
  Revisão: O código apresenta o básico nos comentários, porém poderia ser adicionado a lógica do cálculo.
- O código não tem complexidade excessiva?
  Revisão: Não, o código está bem estruturado e não apresenta complexidade excessiva, mantendo a legibilidade e simplicidade.
- Há comentários desnecessários ou falta de explicações importantes?
  Revisão: Não, os comentários se limitam ao necessário.

4. Desempenho

- O código é eficiente em termos de tempo de execução e uso de memória?
  Revisão: Sim, o código roda rápido até com entradas grandes.
- As tipagens escolhidas são as mais adequadas ao contexto?
  Revisão: Sim, a tipagem está correta ao contexto pedido.

5. Tratamento de Erros e Exceções

- As entradas de usuários ou sistemas são validadas corretamente?
  Revisão: Sim , o código recebe as entradas do usuário corretamente .
- As exceções são tratadas adequadamente, sem ignorar erros importantes?
  Revisão: Não, o código não trata entradas inválidas de letras e pontuações corretamente.

Conclusão:
O código atende aos requisitos da tarefa e está bem organizado, com boa legibilidade e modularidade. Contudo, recomenda-se melhorias nos comentários, especialmente na explicação da lógica de cálculo, e a adoção de convenções para nomenclatura de pastas e arquivos em projetos maiores. O código apresenta bom desempenho, mas a inclusão de métricas de complexidade algorítmica seria útil para futuras otimizações.
Quanto ao tratamento de erros, o código não lida adequadamente com entradas inválidas, como letras ou pontuações. É sugerido implementar validações mais rigorosas e tratamento de exceções para melhorar a robustez e segurança do sistema. No geral, o código é eficiente e bem estruturado, com pequenos ajustes necessários para aumentar sua clareza e garantir maior resiliência a erros.

2.2. Desafio 2

Código: O objetivo deste código é analisar a capacidade interna de um elevador. Para isso, recebe, inicialmente, a quantidade de testes realizados e a capacidade máxima. Em seguida, recebe entradas e saídas das pessoas no elevador para cada teste e calcula se ultrapassa a capacidade máxima informada no início. Ao fim, informa se em algum momento ocorreu ultrapassagem de limite.
A linguagem de programação utilizada foi Java.

Desenvolvedor: Roberto Kendy Hassobe
Registrador: Laura Rodrigues Russo
Revisores: Jose Vitor Dutra Antônio, Laura Rodrigues Russo e Roberto Kendy Hassobe

```java
import java.util.Scanner;
import java.util.ArrayList;
```

```java
public class Desafio2 {
      public static void main(String args[]) {
         Scanner input = new Scanner(System.in);

         int teste = input.nextInt();
         int capacidadeMax = input.nextInt();
         boolean capacidadeExcedida = false;

         ArrayList<Integer> listaSair = new ArrayList<>();
         ArrayList<Integer> listaEntrar = new ArrayList<>();

         int saiu = 0;
         int entrou = 0;
         for(int i = 0; i < teste; i++) {
             listaSair.add(input.nextInt());
             listaEntrar.add(input.nextInt());
         }

         for(int i = 0; i < teste; i++) {
             entrou = saiu + listaEntrar.get(i);

             if(i+1 == teste) {
                 break;
             }

             saiu = entrou - listaSair.get(i+1);

             if(entrou > capacidadeMax) {
                 capacidadeExcedida = true;
                 break;
             }
         }

        System.out.printf("Saída: %s", capacidadeExcedida ==
true ? "S" : "N");

    }
}
```

1. Conformidade com Requisitos

- O código atende todos os requisitos do projeto ou da tarefa?
  Revisão: Sim, o código atende todos os requisitos do desafio

- Caso utilize bibliotecas externas ou código de terceiros, ele é bem integrado e seguro?
  Revisão: Foram usadas bibliotecas, “Scanner” e “ArrayList”, a primeiro para entrada de dados pelo teclado e a segunda fazer lista dinâmica, então está sendo usado de modo integrado e seguro

2. Estrutura do Código

- O código está bem organizado e modularizado?
  Revisão: O código todo está inserido na “main”, poderia ter utilizado alguma função para melhorar organização.

- Os nomes dos arquivos e pastas seguem um padrão consistente e claro?
  Revisão: A classe única foi denominada “Desafio2”, o que não indica o objetivo dela. Poderiam existir duas classes, “main” e a classe “Elevador”, ficando mais ilustrativo.

- O código é fácil de entender?
  Revisão: O código é de fácil entendimento e coerente com a linguagem de programação escolhida (Java).

- O código é reutilizável?
  Revisão: Não, pois há falta de métodos que facilitariam a reutilização.

- Os nomes de variáveis, funções e classes são descritivos?
  Revisão: As variáveis têm nomes claros e descritivos, porém, como citado anteriormente, o nome da classe “Desafio2” precisaria do contexto de um enunciado, seria melhor ser denominada como “Elevador” para ilustrar o objetivo dela.

3. Qualidade do Código

- Há repetição de código?
  Revisão: Não há repetição de código.

- O código é bem comentado, explicando partes complexas ou lógicas de negócios?
  Revisão: Não há comentários.

- O código não tem complexidade excessiva?
  Revisão: Não tem complexidade excessiva.

- Há comentários desnecessários ou falta de explicações importantes?
  Revisão: Há falta de explicações importantes, pois não há comentários.

4. Desempenho

- O código é eficiente em termos de tempo de execução e uso de memória?
  Revisão: Sim, o código é eficiente.

- As tipagens escolhidas são as mais adequadas ao contexto?
  Revisão: Sim, as tipagens são bem escolhidas.

5. Tratamento de Erros e Exceções

- As entradas de usuários ou sistemas são validadas corretamente?
  Revisão: Não há validação, por exemplo, em um “throw”. Se entrar como uma tipagem diferente o código simplesmente quebra.

- As exceções são tratadas adequadamente, sem ignorar erros importantes?
  Revisão: Não há tratamento para número negativo, por exemplo, a possibilidade de a variável “entrou” ser negativa. Não tem como ter quantidade negativa de pessoas no elevador.

Conclusão:

O código atende aos requisitos do desafio e utiliza bibliotecas de forma segura, porém, sua estrutura pode ser aprimorada com a modularização de funções para melhor organização e reutilização. A nomenclatura da classe principal não reflete seu propósito, o que dificulta a compreensão, sendo recomendável dividi-la em duas classes (Main e Elevador). Além disso, o código não possui comentários explicativos e carece de tratamento de erros, permitindo a entrada de valores inválidos, como números negativos. A adição de validações, métodos específicos e uma melhor organização tornaria o código mais claro, reutilizável e robusto.

3. Avaliação dos membros sobre as diretrizes de revisão

A diretriz referente ao treinamento dos membros revisores não foi cumprida devido à falta de tempo. Conforme estabelecido, para que uma revisão seja eficaz, todos os participantes deveriam receber um treinamento formal, abordando tanto os aspectos processuais quanto os psicológicos das revisões. De acordo com Freedman e Weinberg [Fre90], há uma curva de aprendizado estimada para cada grupo de vinte participantes envolvidos nas revisões. No entanto, essa etapa não foi realizada. Apesar disso, todas as demais diretrizes foram seguidas corretamente.

4. Relatório do líder

O aluno Marcelo Dos Santos Da Boa Morte faltou a aula, os demais alunos estavam presentes, e participaram ativamente da atividade.