Tratamento de Exceções

EDUARDO HABIB BECHELANE MAIA

HABIB@CEFETMG.BR

Exceções

Uma **exceção** é uma indicação de que um problema que ocorreu

Tratamento de exceções

- programas sejam mais robustos e também tolerantes a falhas
  - Programa trata o erro e continua executando como se nada tivesse acontecido;

Se uma função precisa enviar uma mensagem de erro, ela "**lança**" um objeto representando o erro

try, throw e *catch*

try, *throw* e *catch*

Temos 3 palavras chave:

- *try* (*tentar*);
- *throw* (*lançar*);
- *catch* (*capturar*).

try

A instrução ***try*** é utilizada para definir blocos de código em que exceções possam ocorrer

- O bloco de código é precedido pela palavra ***try*** e é delimitado por **{** e **}**;
- As instruções que podem implicar em exceções e todas as instruções que não podem ser executadas em caso de exceção fazem parte do bloco de código.

throw

A instrução ***throw*** lança uma exceção

- Indica que houve um erro;
- É criado um objeto, que contém a informação sobre o erro
  - Logo, podemos criar uma classe que defina o erro
  - Ou utilizar uma existente.
- Posteriormente, outro trecho de código capturará este objeto e tomará a atitude adequada.

catch

Exceções são tratadas por manipuladores *catch*

- Capturam e processam as exceções.

Pelo menos um *catch* deve seguir um bloco *try*

- O bloco de código é precedido pela palavra ***catch*** seguido de parênteses e é delimitado por **{** e **}**;
- O *catch* cujos parâmetros sejam de tipos iguais ao da exceção será executado;

Normalmente um *catch:*

- imprime o erro,
- termina o programa elegantemente ou
- tenta refazer a operação que lançou a exceção.

try, throw e *catch*

Um engano comum é achar que em todo tratamento de exceções é obrigatório usar pelo menos um *try*, um *throw* e um *catch* juntos

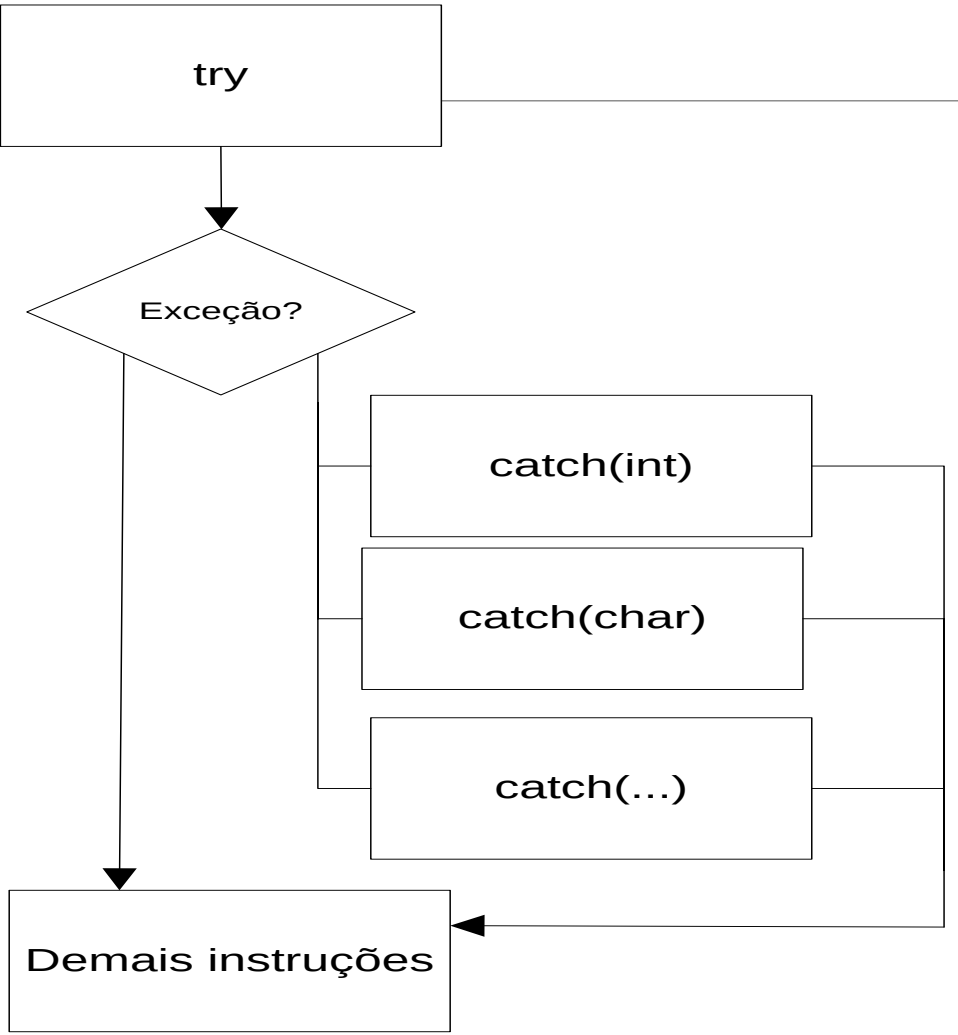
- Recomendável é utilizar juntos o *try* e o *catch* para manipular as exceções;
- O *throw* é utilizado somente para lançar exceções, não para manipulá-las.

Try

Se em um bloco *try*, ocorre uma exceção :

- O bloco termina;
- O primeiro *catch* cujo tipo seja igual (ou derivado) da exceção é executado;
- O fluxo de execução passa para a primeira instrução depois do último *catch* relacionado ao *try* que lançou a execução.

Modelo de Terminação



Sintaxe

```
try
{
    // código
}catch( ExceptionName e1 )
{
    // bloco catch
}catch( ExceptionName e2 )
{
 // bloco catch
}catch( ExceptionName eN )
{
    // bloco catch
}
```

Exemplo

Vamos observar um exemplo a partir do projeto que já estávamos fazendo na aula passada.

Tratando uma exceção de divisão por zero em c++

Código em: https://onlinegdb.com/d-YbSDrZO

Criando, Lançando e Capturando Exceções

Nossa classe que representa o erro de divisão por zero é derivada da classe ***runtime_error***

- Definida na ***stdexcept***;
- Derivada da classe ***exception*** da biblioteca padrão C++
  - Classe base de todas as exceções.
- Representa os erros de execução.

Nossa classe simplesmente passa uma *string* de erro ao construtor da classe ***runtime_error***

Criando, Lançando e Capturando Exceções

Ao derivar a classe ***runtime_error***, pode-se utilizar o método virtual What

- Permite que uma mensagem de erro seja exibida.
- No exemplo, utiliza-se um objeto da classe *ExcecaoDivisaoPorZero* para indicar uma tentativa de divisão por zero

Tratando uma exceção de divisão por zero em JAVA

https://onlinegdb.com/H2_cMENiF

Criei uma exceção, mas:

- Em Java, quando uma divisão por zero ocorre durante uma operação aritmética de ponto flutuante, o resultado é:
  - Infinity ou NaN (Not a Number).
- Se a operação de divisão por zero envolver apenas números inteiros:
  - será lançada uma ArithmeticException.
  - É uma exceção padrão no Java que trata tentativas de divisão por zero com inteiros.

finally (em JAVA)

- Usado para garantir que um código seja executado após um bloco try...catch
  - independentemente de uma exceção ser lançada ou não.
- Limpeza de Recursos
  - fechar conexões de banco de dados,
  - arquivos ou
  - liberar recursos de sistema, o que é crucial para evitar vazamentos de recursos.
- Funciona com ou sem Exceções:
  - Executa operações finais mesmo se uma exceção for capturada, não capturada, ou se nenhum erro ocorrer no bloco try.

Finally

O bloco finally é projetado para executar código de limpeza ou finalização,

- não para tratar exceções.
- tratamento de exceções é feito nos blocos catch.
- Se você tem um bloco try sem um catch:
  - qualquer exceção lançada não será tratada ali
  - o bloco finally ainda será executado.

```
import java.io.IOException;

public class Main
{
    public static void main(String[] args) throws IOException {
        System.out.println("Hello World");

        try {
          // Código que pode lançar uma exceção
          throw new IOException("Ocorreu um erro de I/O.");
        } finally {
// Este código será executado independentemente de qualquer forma
         System.out.println("Este é o bloco finally sendo executado.");
        }
    }
}
```

finally

Se, por qualquer motivo, uma nova exceção for lançada dentro do bloco finally:

- irá "mascarar" qualquer exceção que foi lançada no bloco try.
- perigoso, pois a exceção original é perdida.

```
import java.io.IOException;

public class Main
{
   public static void main(String[] args) throws IOException {
            System.out.println("Hello World");

      try {
         // Código que lança uma exceção
         throw new IOException("Ocorreu um erro de I/O.");
      } finally {
         // Lança uma nova exceção, que sobrescreve a IOException
         throw new RuntimeException("Erro no bloco finally.");
      }
      // A RuntimeException do bloco finally é lançada, e a IOException original é perdida.
   }
}
```

Exemplo de finally

```
import java.io.*;

public class FinallyExample {
    public static void main(String[] args) {
        BufferedReader reader = null;
        try {
            reader = new BufferedReader(new FileReader("somefile.txt"));
            String firstLine = reader.readLine();
            System.out.println(firstLine);
        } catch (IOException e) {
            System.out.println("Erro ao ler do arquivo: " + e.getMessage());
        } finally {
            // Bloco finally para garantir que o arquivo seja fechado
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    System.out.println("Erro ao fechar o arquivo: " + e.getMessage());
                }
            }
        }
    }
}
```

Erros Comuns

Não pode haver código entre um bloco ***try*** e um ***catch***

- É um erro de sintaxe.

Cada ***catch*** só pode ter um único parâmetro

- Uma lista de parâmetros é um erro de sintaxe.

É um erro de lógica capturar o mesmo tipo de exceção em dois ***catch*** diferentes;

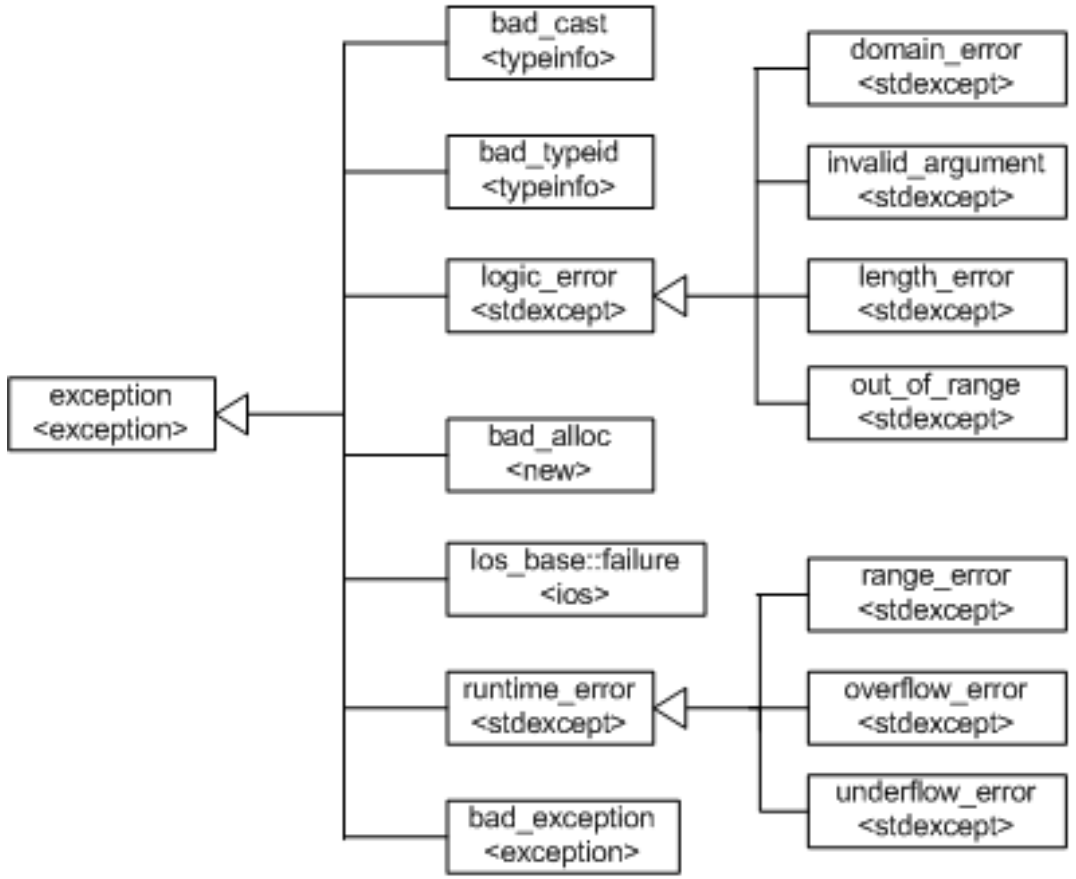
Após o tratamento da exceção, achar que o fluxo de execução volta para o ponto em que a exceção foi lançada.

Quando Utilizar Exceções?

Erros síncronos (na execução de uma instrução)

- Índice de vetor fora dos limites;
- *Overflow* aritmético (valor fora dos limites do tipo);
  - **int** varia entre -2147483648 e +2147483647
- Divisão por zero;
- Parâmetros inválidos;
- Alocação de memória.

Hierarquia das Exceções em C++

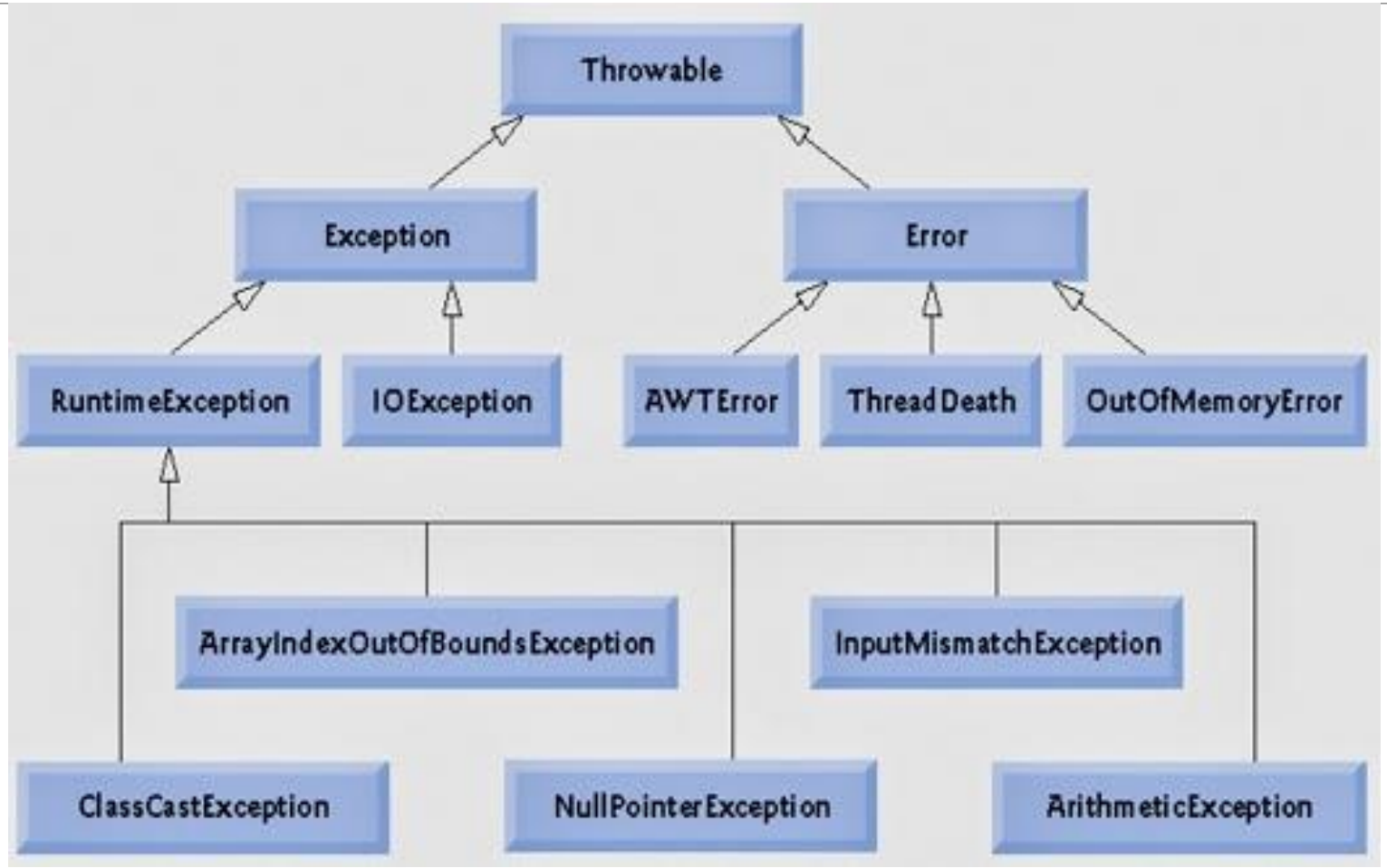


Classes de Exceções da Biblioteca Padrão

Exceção	Descrição
std::exception	Classe base de todas as exceções padrões em c++.
std::bad_alloc	Pode ser lançada pelo new.
std::bad_cast	Pode ser lançada por um dynamic_cast.
std::bad_exception	Útil para tratar exceções inexperadas em C++
std::bad_typeid	Pode ser lançada por um typeid.
std::logic_error	Uma exceção que teoricamente pode se detectada por uma leitura do código.
std::domain_error	Exceção que pode ser lançada quando um domínio matemático incorreto é utilizado.

Exceção	Descrição
std::invalid_argument	Exceção que pode ser lançada na ocorrência de argumentos inválidos.
std::length_error	Exceção lançada se uma String muito grande for criada
std::out_of_range	Exceção lançada quando se acessa uma posição incorreta de um Vector.
std::runtime_error	Uma exceção que ocorre em tempo de execução.
std::overflow_error	Exceção lançada se overflows matemáticos ocorrerem.
std::range_error	Exceção lançada caso se tente armazenar um valor fora do conjunto de valores válidos permitido.
std::underflow_error	Lançada se underflows matemáticos ocorrerem.

Hierarquia de exceções em JAVA



Exceções padrão em JAVA

1. **Throwable:** É a superclasse de todas as exceções e erros. Qualquer coisa que possa ser lançada com a palavra-chave throw e capturada com catch é uma subclasse de Throwable.

2. **Exception:** É a superclasse para todas as exceções que são verificadas pelo compilador (exceto RuntimeException). Elas precisam ser capturadas ou declaradas no método que as lança.

3. **Error:** Indica problemas graves que uma aplicação normalmente não deve tentar capturar, muitas vezes ligados ao ambiente de execução, como problemas na JVM.

4. **RuntimeException:** São exceções não verificadas e não precisam ser declaradas ou capturadas obrigatoriamente. Erros como divisão por zero e acesso inválido a índices de array caem nessa categoria.

5. **IOException:** É lançada para problemas relacionados a operações de I/O (entrada/saída), como falhas ao ler ou escrever em arquivos.

6. **AWTError:** É lançado para indicar erros sérios que ocorrem na Abstract Window Toolkit (AWT).

Exceções padrão em JAVA

1. **ThreadDeath:** É um erro lançado pela JVM quando a execução de uma thread é parada forçadamente pelo método stop.

2. **OutOfMemoryError:** Indica que a JVM não tem mais memória disponível, geralmente lançada quando a heap da JVM está cheia.

3. **ClassCastException:** Ocorre quando tentamos fazer um cast inválido de um objeto para uma classe da qual ele não é uma instância.

4. **ArrayIndexOutOfBoundsException:** É lançada quando tentamos acessar um índice de array que está fora do intervalo permitido (negativo ou maior que o tamanho do array).

5. **NullPointerException:** É lançada quando tentamos usar um objeto que tem um valor nulo em um contexto que requer um objeto não nulo, como chamar um método em uma referência nula.

6. **InputMismatchException:** É uma exceção não verificada lançada pela classe Scanner para indicar que o próximo token não corresponde ao padrão esperado ou está fora do tipo esperado.

7. **ArithmeticException:** É lançada para erros de condição aritmética, como divisão por zero.

Exemplo 2 – OutOfRange

```
#include <iostream>
#include <vector>
#include <stdexcept> // Necessário para std::out_of_range

int main() {
    try {
        // Suponha que temos um vetor com apenas 5 elementos
        std::vector<int> vetor(5);

        // A tentativa de acessar o elemento no índice 10 vai além do intervalo do vetor
        std::cout << vetor.at(10); // std::vector::at lança std::out_of_range se o índice
está fora do intervalo
    }
    catch (const std::out_of_range& e) {
        std::cerr << "Erro de acesso: " << e.what() << '\n';
    }

    return 0;
}
```

Java: ArrayIndexOutOfBoundException

```
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        try {
            // Suponha que temos uma lista com apenas 5 elementos
            List<Integer> lista = new ArrayList<>(5);

            // Preenchendo a lista com números de exemplo
            for (int i = 0; i < 5; i++) {
                lista.add(i);
            }

            // A tentativa de acessar o elemento no índice 10 vai além do intervalo
da lista
            System.out.println(lista.get(10)); // get() pode lançar
IndexOutOfBoundsException se o índice está fora do intervalo
        } catch (IndexOutOfBoundsException e) {
            System.err.println("Erro de acesso: " + e.getMessage());
        }
    }
}
```

Exemplo 2 – Range Error

```
#include <iostream>
#include <stdexcept> // Necessário para std::range_error
#include <math.h>

double calcularRaiz(double numero) {
    if (numero < 0) {
        throw std::range_error("Não pode calcular a raiz quadrada de um número
negativo.");
    }
    return sqrt(numero);
}

int main() {
    try {
        double numero = -5;
        double raiz = calcularRaiz(numero);
        std::cout << "A raiz quadrada de " << numero << " é " << raiz << std::endl;
    }
    catch (const std::range_error& e) {
        std::cerr << "Erro: " << e.what() << '\n';
    }

    return 0;
}
```

Java: IllegalArgument Exception

```
public class Main {
    public static double calcularRaiz(double numero) throws IllegalArgumentException
{
        if (numero < 0) {
            throw new IllegalArgumentException("Não pode calcular a raiz quadrada de
um número negativo.");
        }
        return Math.sqrt(numero);
    }

    public static void main(String[] args) {
        try {
            double numero = -5;
            double raiz = calcularRaiz(numero);
            System.out.println("A raiz quadrada de " + numero + " é " + raiz);
        } catch (IllegalArgumentException e) {
            System.err.println("Erro: " + e.getMessage());
        }
    }
}
```

Exemplo de tratamento de entrada com cin

```
#include <iostream>
#include <limits>

int main() {
   double num;

   while (true) {
      std::cout << "Por favor, insira um número em ponto flutuante: ";
      std::cin >> num;

      if (std::cin.fail()) {

         std::cin.clear(); // Limpa o estado de falha de cin

         std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
         // Ignora a entrada até o próximo newline
         std::cout << "Entrada inválida. Tente novamente.\n";


      } else {
         std::cout << "O número inserido é: " << num << std::endl;
         break; // Sai do loop se a entrada for válida
      }
   }

   return 0;
}
```

Exemplo 2 de tratamento de entrada com cin

```
#include <iostream>
#include <limits>


int main() {
    int nota;

    std::cout << "Digite a nota do aluno (0 a 20): ";
    std::cin >> nota;
    if ((!std::cin >> nota) || (nota < 0) || (nota > 20)) {
        std::cout << "Inseriu nota inválida" << std::endl;
        std::cin.clear(); // Limpa o estado de falha
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // Descarta a
entrada até o próximo newline
    } else {
        if (nota < 10)
            std::cout << "O aluno foi reprovado" << std::endl;
        else
            std::cout << "O aluno foi aprovado" << std::endl;
    }

    return 0;
}
```

Tratamento entrada em java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double num;

        while (true) {
            System.out.print("Por favor, insira um número em ponto flutuante: ");

            if (!scanner.hasNextDouble()) {
                System.out.println("Entrada inválida. Tente novamente.");
                scanner.next(); // Consome a entrada inválida
            } else {
                num = scanner.nextDouble();
                System.out.println("O número inserido é: " + num);
                break; // Sai do loop se a entrada for válida
            }
        }

        scanner.close();
    }
}
```

Exemplo 3 – invalid_argument

// O exemplo abaixo foi feito apenas para mostrar como lançar a exceção invalid_argument

```
int comparar(int a, int b) {
    if (a < 0 || b < 0) {
        throw std::invalid_argument("a or b negative");
    }
}

int main() {
    try {
        comparar(-1, 0);
    } catch (const std::invalid_argument& e) {
        cout << "Erro: "<< e.what();    }
}
```

Exercício

Criar um programa de cadastro de funcionários que possua métodos para cadastrar funcionários, adicionar bônus e aumentar o salário, em que você realize o tratamento de exceção, de forma a garantir que:

1. O Bônus seja maior do que 0
2. O Aumento de Salário seja maior do que 0
3. O salário seja maior do que R$ 1340 (valor do salário mínimo).
4. Estude o funcionamento da exceção invalid_argument e crie uma função que lance essa exceção.

Resposta em Java

https://onlinegdb.com/bibA2-3dj