📌 1.1. Utilização do Tratamento de Exceções em Java

1.1.1. Visão geral do tratamento de exceções

Uma **exceção** é um evento inesperado que ocorre durante a execução de um programa. Ela interrompe o fluxo normal da aplicação e indica que algo deu errado.

Exemplo prático:
Um programa pede para o usuário informar a idade e tenta converter a entrada para um número inteiro. Se o usuário digitar "vinte", a conversão falha e uma exceção ocorre.

⚠️ Situações possíveis:

1. **Valor válido (ex: "10")**
   ✅ A conversão funciona normalmente e o valor é processado.

2. **Valor inválido (ex: "vinte", "020")**
   ❌ Ocorre uma `NumberFormatException`, pois o valor não pode ser convertido para `int`.

3. **Usuário clica em "Cancelar"**
   ❌ O valor da String se torna `null`. Isso também causará uma exceção ao tentar converter para `int`.

🧠 Detalhe técnico:

No caso de erro, a stack trace mostra:

at java.lang.Integer.parseInt(Unknown Source)

Isso indica que o erro ocorreu dentro do método `parseInt()` — muito útil para depurar.

🧪 Exemplo sem tratamento de exceções:

```
import javax.swing.JOptionPane;

public class TesteExcecoes {
    public static void main(String[] args) {
        String idadeTexto = JOptionPane.showInputDialog("Informe sua idade:");
```

```
        int idade = Integer.parseInt(idadeTexto);
        System.out.println("Sua idade é: " + idade);
    }
}
```

💥 Se o usuário digitar "vinte", ocorre:

Exception in thread "main" java.lang.NumberFormatException: For input string: "vinte"

✅ Como evitar o erro: tratamento com try-catch

```
import javax.swing.JOptionPane;

public class TesteExcecoes {
    public static void main(String[] args) {
        try {
            String idadeTexto = JOptionPane.showInputDialog("Informe sua idade:");
            int idade = Integer.parseInt(idadeTexto);
            System.out.println("Sua idade é: " + idade);
        } catch (NumberFormatException e) {
            System.out.println("Erro: valor inválido para idade.");
        } catch (NullPointerException e) {
            System.out.println("Erro: nenhuma informação fornecida.");
        }
    }
}
```

📘 Conclusão:

- Exceções são comuns e inevitáveis, especialmente em programas que interagem com o usuário ou recursos externos.
- Um bom programa **deve antecipar possíveis falhas** e lidar com elas de forma apropriada, **sem quebrar a aplicação**.
- Usar `try-catch` permite capturar exceções e tomar ações corretivas ou informar o usuário.

1.1.2. Mecanismo de tratamento e captura de exceções: try-catch

O **mecanismo de tratamento de exceções** em Java nos permite **capturar erros durante a execução** e tratá-los adequadamente para evitar que o programa termine de forma inesperada. A solução para isso é o uso de um bloco **`try-catch`**, que envolve o código suscetível a erros (no bloco `try`) e o tratamento de possíveis falhas (no bloco `catch`).

📝 Sintaxe do `try-catch`

A sintaxe básica do bloco `try-catch` é:

```
try {
    // Código que pode gerar exceção
} catch (ClasseDaExceçãoException e) {
    // Código alternativo para tratar a exceção
}
```

Partes principais:

1. **Código desejado** (no `try`): A parte do código que você espera que funcione normalmente, mas que pode gerar exceções.

2. **Classe da Exceção** (no `catch`): A classe da exceção que você deseja capturar. Quando ocorre uma exceção, ela é **instanciada** e passada para o bloco `catch` para que seja tratada.

3. **Código alternativo** (no `catch`): O que o programa faz se uma exceção for capturada. No exemplo, você pode exibir uma mensagem de erro ou tomar outras ações corretivas.

👩‍💻 Exemplo prático com `try-catch`

Vamos reescrever o programa com o **tratamento de exceções** para lidar com uma entrada inválida do usuário:

Código 2: Classe `TesteExcecoes` com tratamento de exceções

```
import javax.swing.JOptionPane;

public class TesteExcecoes {
    public static void main(String[] args) {
```

```
        try {
            String idadeTexto = JOptionPane.showInputDialog("Informe sua idade:");
            int idade = Integer.parseInt(idadeTexto);
            System.out.println("Sua idade é: " + idade);
        } catch (NumberFormatException e) {
            System.out.println("Erro: valor inválido para idade.");
        } catch (NullPointerException e) {
            System.out.println("Erro: nenhuma informação fornecida.");
        }
    }
}
```

Explicação do fluxo:

1. **Bloco `try`**: Aqui, o código tenta converter a entrada do usuário para um número inteiro (`Integer.parseInt(idadeTexto)`).

2. **Bloco `catch`**: Se ocorrer um erro ao tentar converter (como uma entrada inválida como "vinte"), a exceção será capturada e o código no bloco `catch` será executado. No exemplo, ele simplesmente imprime uma mensagem de erro.

⚠️ Exemplo com erro de entrada:

Se o usuário digitar a palavra `"vinte"`, o programa captura a **`NumberFormatException`** e imprime a mensagem:

Erro: valor inválido para idade.

📘 Explicação do mecanismo de captura de exceções:

No código abaixo, o **método `parseInt()`** lança uma exceção **`NumberFormatException`** se a string não puder ser convertida para um número inteiro.

Quadro 2 - Exemplo de tratamento simplificado:

```
public static void main(String[] args) {
    try {
        String idade = "vinte";
        int valorIdade = Integer.parseInt(idade); // Exceção ocorre aqui
    } catch (NumberFormatException e) {
        System.out.println(e.getMessage()); // A mensagem da exceção é exibida
    }
```

}

Neste caso, o método **`parseInt()`** tenta converter a string `"vinte"`, mas não consegue e **lança** a exceção do tipo `NumberFormatException`. A exceção é capturada pelo bloco `catch`, e a mensagem de erro pode ser exibida com **`e.getMessage()`**.

🛠️ Motivos comuns para exceções:

As exceções podem ocorrer por uma série de razões imprevistas, algumas delas são causadas por falhas externas ao código. Aqui estão alguns exemplos comuns:

1. **Tentar abrir um arquivo inexistente**:
    - O arquivo não está presente no local especificado.
2. **Banco de dados indisponível**:
    - A aplicação tenta se conectar a um banco de dados, mas não consegue acessar o servidor.
3. **Permissão de escrita negada**:
    - O código tenta escrever em um arquivo onde o usuário não tem permissão.
4. **Conexão com servidor inexistente**:
    - A aplicação tenta se conectar a um servidor, mas ele não existe ou não está disponível.

🎯 Conclusão:

- **Bloco `try`**: Onde o código que pode gerar uma exceção é colocado.
- **Bloco `catch`**: Onde a exceção é capturada e tratada, evitando que o programa termine de forma abrupta.
- A captura da exceção permite que o programa tome **ações corretivas** ou informe ao usuário o que deu errado, tornando o código **mais robusto e confiável**.

Ótimo! Vamos organizar e explicar de forma clara o conteúdo abordado nos tópicos **1.1.3** e **1.1.4**, com exemplos práticos e explicações diretas.

✅ 1.1.3 – Lançando uma exceção: `throw new`

Nem sempre as exceções vêm da API do Java como vimos no `parseInt()`. Às vezes, **nós mesmos precisamos lançar uma exceção**, quando o programa recebe dados inválidos.

🧠 Exemplo prático: método `exibirMaioridade()`

Vamos supor que você criou um método que recebe uma idade e retorna se a pessoa é **"Maior de Idade"** ou **"Menor de Idade"**. Mas o método pode receber valores inválidos, como idade negativa.

❌ Versão sem tratamento:

```
public class Pessoa {
    public static String exibirMaioridade(int idade) {
        if (idade >= 18) {
            return "Maior de Idade";
        } else {
            return "Menor de Idade";
        }
    }
}
```

Se alguém passar `-10` como idade, o código vai considerar que a pessoa é "Menor de Idade", o que **não faz sentido**.

✅ Versão com `throw new`:

```
public class Pessoa {
    public static String exibirMaioridade(int idade) {
        if (idade < 0) {
            throw new IllegalArgumentException("Idade não pode ser negativa!");
        }

        return (idade >= 18) ? "Maior de Idade" : "Menor de Idade";
    }
}
```

- Aqui usamos `**throw new**` para **lançar uma exceção manualmente**.
- A exceção `IllegalArgumentException` é **ideal** nesse caso, pois estamos recebendo um argumento inválido (idade negativa).

🧪 Testando com try-catch:

```
public class TesteExcecoes02 {
    public static void main(String[] args) {
        try {
            String resultado = Pessoa.exibirMaioridade(-10);
            System.out.println(resultado);
        } catch (IllegalArgumentException e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

🖨️ Saída:

Erro: Idade não pode ser negativa!

Esse tipo de tratamento deixa o programa mais robusto, evitando comportamentos inesperados.

📚 1.1.4 – Classes Java para tratamento de exceções

Java já fornece **diversas classes de exceções** prontas no pacote `java.lang`, que podem ser usadas ou estendidas conforme a necessidade.

🔗 Hierarquia geral:

Todas essas classes derivam da classe base:

```
Throwable
 ├── Error            ← (erros graves que não devem ser tratados)
 └── Exception
     ├── Checked Exceptions
     └── RuntimeException (Unchecked Exceptions)
```

🔹 Algumas exceções mais comuns:

Classe	Quando ocorre
`ArithmeticException`	Divisão por zero
`NullPointerException`	Acessar algo em um objeto nulo
`ArrayIndexOutOfBoundsException`	Acessar posição inválida de um array
`NumberFormatException`	Converter uma string inválida para número
`IllegalArgumentException`	Passar argumento inválido para um método
`IllegalStateException`	Estado incorreto de um objeto ou operação
`ClassNotFoundException`	Classe não encontrada durante carregamento dinâmico
`InterruptedException`	Thread foi interrompida durante a execução
`IndexOutOfBoundsException`	Índice fora dos limites (em listas, arrays, etc.)
`UnsupportedOperationException`	Operação não suportada por determinado método ou estrutura

---

🎯 Dica prática:

Você **não precisa criar sua própria exceção do zero** para cada caso. A API Java já oferece uma variedade que cobre a maioria das situações comuns.

Se mesmo assim for necessário criar uma exceção personalizada, você pode fazer assim:

```
public class IdadeInvalidaException extends RuntimeException {
    public IdadeInvalidaException(String mensagem) {
        super(mensagem);
    }
}
```

E depois lançar:

```
throw new IdadeInvalidaException("Idade não pode ser negativa!");
```

✅ Diferença entre Checked e Unchecked Exceptions

🔹 Checked Exceptions

- São **verificadas em tempo de compilação**.
- O compilador **obriga** você a tratar essas exceções com `try-catch` ou com `throws`.
- Representam situações que **podem ser previstas e recuperadas**, como falhas de I/O ou conexões externas.

🗒️ Exemplos de Checked Exceptions:

Classe	Quando ocorre
`IOException`	Erro de leitura ou escrita em arquivos
`SQLException`	Erro ao acessar um banco de dados
`ClassNotFoundException`	Classe não encontrada durante carregamento
`InterruptedException`	Thread interrompida

🧪 Exemplo de uso:

```
public void lerArquivo() throws IOException {
    BufferedReader br = new BufferedReader(new FileReader("arquivo.txt"));
}
```

Ou:

```
try {
    BufferedReader br = new BufferedReader(new FileReader("arquivo.txt"));
} catch (IOException e) {
    System.out.println("Erro ao ler o arquivo: " + e.getMessage());
}
```

🔸 Unchecked Exceptions (RuntimeException)

- São **verificadas em tempo de execução**.

- **Não obrigam** o uso de `try-catch` ou `throws`.
- Indicam **erros de programação** (bugs) que normalmente devem ser **evitados com validações**.

🗒️ Exemplos de Unchecked Exceptions:

Classe	Quando ocorre
`NullPointerException`	Acessar objeto nulo
`ArithmeticException`	Divisão por zero
`ArrayIndexOutOfBoundsException`	Índice inválido em array
`IllegalArgumentException`	Argumento inválido em um método
`NumberFormatException`	Conversão inválida de string para número

🧪 Exemplo:

```
public void dividir(int a, int b) {
    int resultado = a / b; // Se b = 0, lança ArithmeticException
}
```

🎯 Resumo rápido:

Diferença	Checked Exceptions	Unchecked Exceptions
Verificadas em tempo de...	Compilação	Execução
Tratamento obrigatório?	Sim (`try-catch` ou `throws`)	Não
São subclasses de...?	`Exception`	`RuntimeException`
Representam...	Falhas externas esperadas	Erros de programação