UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

AULA 08

LINGUAGEM DE PROGRAMAÇÃO 2
JAVA

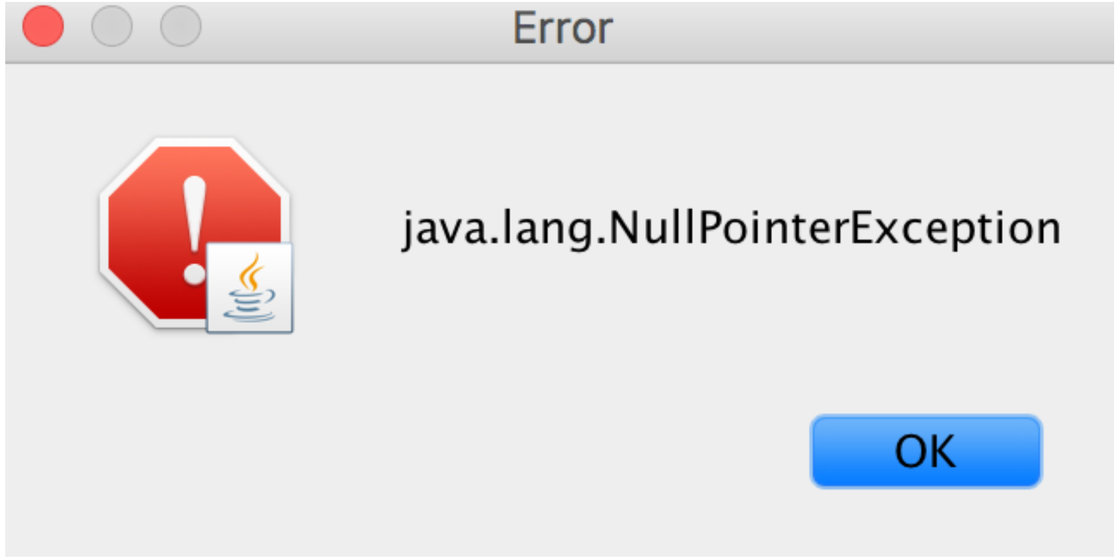
PROF. JANIHERYSON FELIPE

CONTEÚDO DESSA AULA

- CONHECER OS TIPOS DE ERROS EM PROGRAMAÇÃO;
- CONHECER OS MÉTODOS QUE JAVA DISPÕE PARA TRATAR ERROS E EXCEÇÕES;
- DISCUSSÕES E DÚVIDAS GERAIS.

ERROS NA PROGRAMAÇÃO

Erros são parte **inevitável** do processo de programação. Eles podem ocorrer por uma variedade de razões e em diferentes estágios do desenvolvimento de software.



TIPOS DE ERRO

1. Erros de sintaxe:
2. Erros de lógica:
3. Erros de tempo de execução:
4. Erros de semântica:
5. Erros de lógica de negócio:
6. Erros de integração:
7. Erros de configuração:
8. Erros de exceção não tratada:
9. Erros de memória:

ERROS NA PROGRAMAÇÃO

- **Erros de sintaxe:** Esses erros ocorrem quando o código viola as regras gramaticais da linguagem de programação. Por exemplo, esquecer de fechar aspas, parênteses ou chaves, ou usar palavras-chave reservadas de forma incorreta.

```
public static void main(String[] args) {
    System.outi.println(Ola Mundo);
}
```

ERROS NA PROGRAMAÇÃO

- **Erros de lógica:** Esses erros acontecem quando há um problema na lógica do programa. O código pode executar sem erros de sintaxe, mas não produzir o resultado esperado devido a um problema na forma como os comandos foram organizados ou em como as condições foram definidas.

```
public double soma(int a, int b){
    return a - b;
}
```

ERROS NA PROGRAMAÇÃO

- **Erros de tempo de execução:** Estes ocorrem durante a execução do programa e geralmente são causados por condições imprevistas que o programador não tratou. Isso inclui divisões por zero, tentativas de acessar índices fora do intervalo em vetores (arrays), entre outros.

```
String[] nomes = new String[2];
for (int i = 0; i <= 3; i++) {
    nomes[i] = "nome" + i;
}
```

ERROS NA PROGRAMAÇÃO

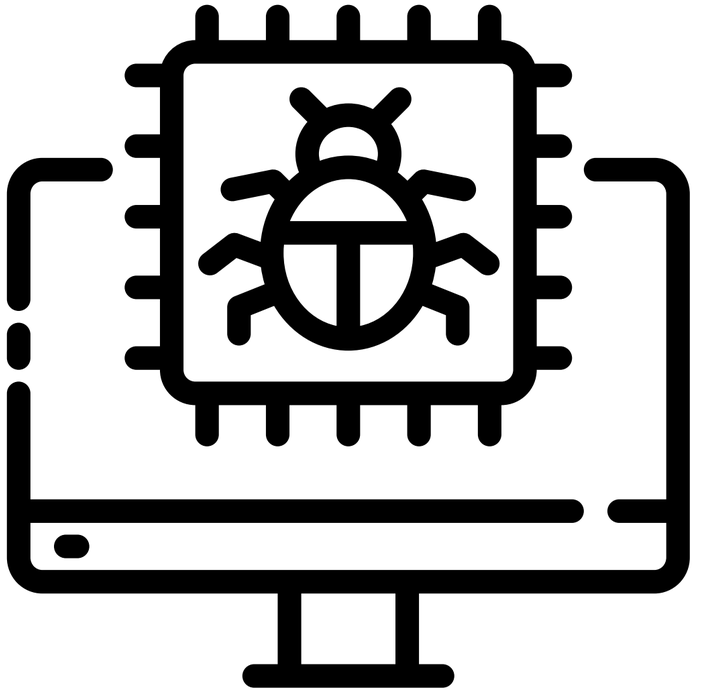
- **Erros de semântica:** Estes são um pouco mais sutis e ocorrem quando o código está corretamente escrito em termos de sintaxe, mas não se comporta conforme o esperado devido a uma compreensão errada da linguagem ou de alguma biblioteca utilizada

```
String[] nomes = new String[2];
for (int i = 0; i < 2; i++) {
    nomes[i] = "nome" + i;
}
System.out.println(nomes);
```

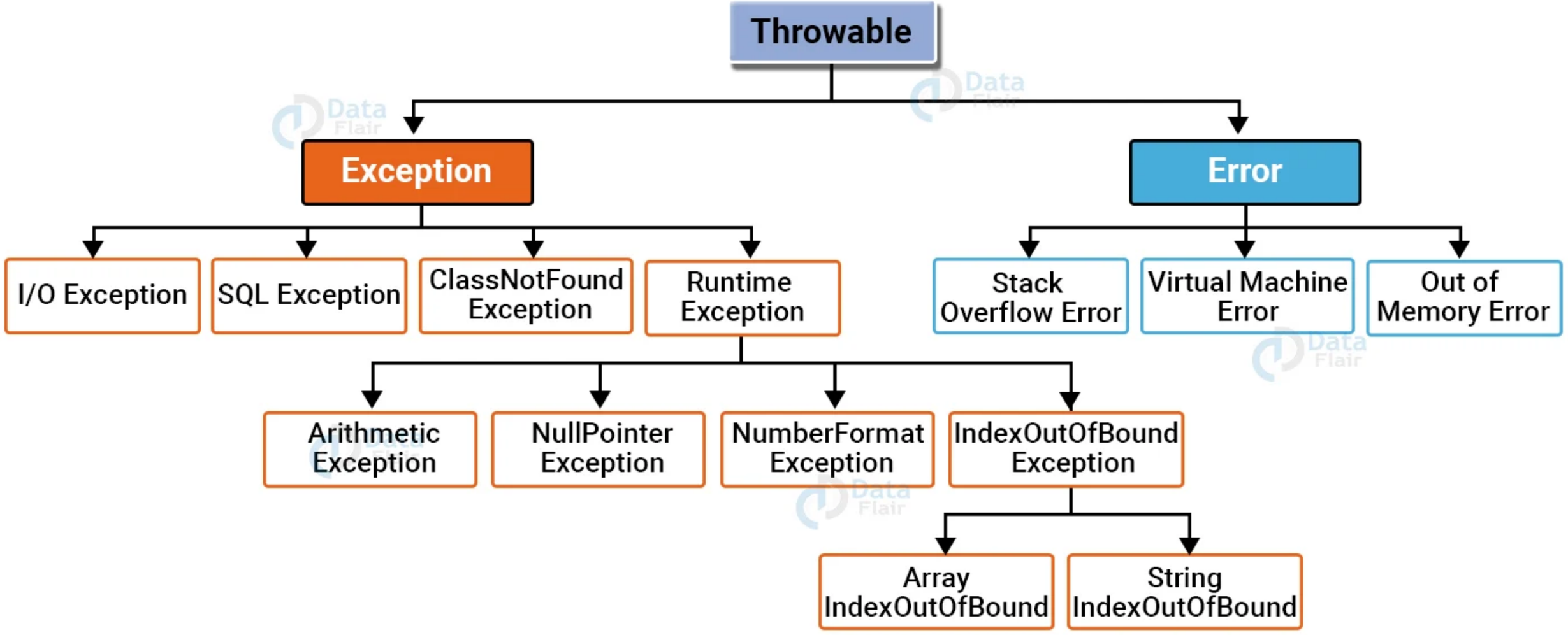
```
[Ljava.lang.String;@5b2133b1
```

PROCESSO DE DEPURAÇÃO

A depuração (**debugging**) é o processo de identificar, isolar e corrigir esses erros em um programa. Isso geralmente envolve o uso de ferramentas de desenvolvimento, como depuradores e registradores de erros, além de técnicas como a impressão de mensagens de depuração e a realização de **testes** para encontrar e corrigir problemas.



Hierarchy of Java Exceptions



PALAVRAS RESERVADAS PARA TRATAMENTO DE EXCEÇÕES

- try
- catch
- throw
- throws
- finally

TRY-CATCH

```java
int[] numeros = new int[4];
System.out.println(numeros[5]);
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 o
ut of bounds for length 4
        at Main.main(Main.java:7)
PS C:\Users\janih\OneDrive\Área de Trabalho\exemploAula>
```

TRY-CATCH

Os trechos de códigos que que podem dar “errado” durante a execução do código devem ser inseridos dentro de um bloco “try”. Esse bloco é seguido imediatamente por um bloco “catch” (capture) que é responsável pela resposta em caso de erro no programa.

```
try{

}catch(Exception e){
    System.out.println(e);
}
```

TRY-CATCH

```
int[] numeros = new int[4];
System.out.println(x:"Antes do erro");
try{
    System.out.println(numeros[5]);
}catch(Exception e){
    System.out.println(x:"Um erro ocorreu");
}
System.out.println(x:"Depois do erro");
```

TRY-CATCH-CATCH...

Caso ocorram situações nas quais mais de um erro pode ocorrer, podemos criar vários blocos “catch” aninhados para tratar cada um dos erros capturados.

```
int[] numeros = { 10, 5, 4, 7, 9, 20, 30 };
int[] divisor = { 2, 0, 5, 0, 5 };
for (int i = 0; i < numeros.length; i++) {
    try {
        System.out.println("Resultado: " + numeros[i] / divisor[i]);
    } catch (ArithmeticException e) {
        System.out.println(x:"Tentou dividir por zero");
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(x:"Tentou acessar uma posição invalida");
    }
}
```

TRY-CATCH-CATCH...

```
int[] numeros = { 10, 5, 4, 7, 9, 20, 30 };
int[] divisor = { 2, 0, 5, 0, 5 };
for (int i = 0; i < numeros.length; i++) {
    try {
        System.out.println("Resultado: " + numeros[i] / divisor[i]);
    } catch (ArithmeticException e) {
        System.out.println(x:"Tentou dividir por zero");
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(x:"Tentou acessar uma posição invalida");
    }catch (Throwable e) {
        System.out.println(x:"Ocorreu um erro genérico");
    }
}
```

TRY-CATCH-FINALLY

O bloco “finally” sempre será executado, independentemento de ter executado o “try” ou o “catch”

```
try {
    System.out.println("Resultado: " + numeros[i] / divisor[i]);
} catch (ArithmeticException e) {
    System.out.println(x:"Tentou dividir por zero");
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println(x:"Tentou acessar uma posição invalida");
}finally{
    System.out.println(x:"Sempre é executada");
}
```

OBTENDO INFORMAÇÕES DO ERRO

Podemos utilizar os métodos getMessage() e printStackTrace() para obter informações sobre o erro que acaba de acontecer.

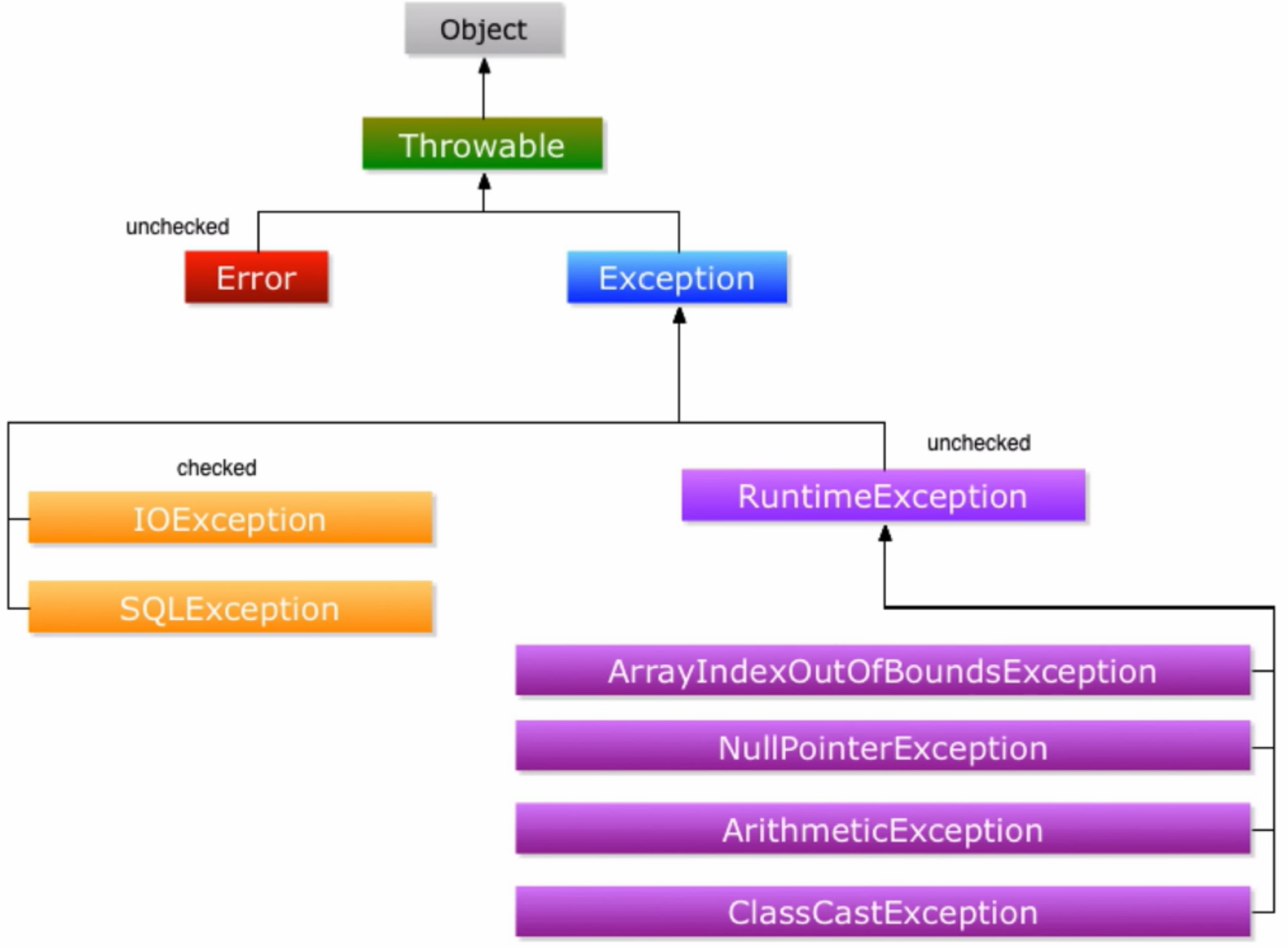
```
catch(Exception e){
    System.out.println(e.getMessage());
    e.printStackTrace();
}
```

PALAVRA CHAVE THROWS

Lança a responsabilidade de tratar o erro para quem for utilizar o médoto.

```
public double leNumero() throws Exception {
    Scanner scan = new Scanner(System.in);
    double num = scan.nextDouble();
    return num;
}
```

```
System.out.println("Entre com um número decimal");
try {
    double num = leNumero();
} catch (Exception e) {
    System.out.println("Entrada inválida");
    e.printStackTrace();
```


Object
Throwable
unchecked
Error
Exception
checked
IOException
SQLException
unchecked
RuntimeException
ArrayIndexOutOfBoundsException
NullPointerException
ArithmeticException
ClassCastException

TIPOS DE EXCEÇÕES

- Checked Exceptions – obriga a quem chama o método ou construtor a tratar essa exceção – o compilador checará se ela está sendo devidamente tratada

- Unchecked Exceptions - o compilador não checará e não obrigará a exceção a ser tratada. Código pode ser compilado e executado

EXCEÇÕES PROPRIETÁRIAS

```
int[] numeros = {4, 8, 5, 16, 32, 21, 64, 128};
int[] demon = {2, 0, 4, 8, 0, 2, 4};

for (int i=0; i<numeros.length; i++){
    try{
        if (numeros[i] % 2 != 0){
            //lançar a exception aqui
            throw new Exception("Número ímpar, divisão não exata");
        }
        System.out.println(numeros[i] + "/" + demon[i] + " = " + (numeros[i]/demon[i]));
    }
    catch(ArithmeticException | ArrayIndexOutOfBoundsException  e){
        System.out.println("Aconteceu um erro");
    }
    catch(Exception e){
        System.out.println("Aconteceu um erro");
```

EXCEÇÕES PROPRIETÁRIAS

```
public class NaoInteiroException extends Exception {

    protected int num;
    protected int denom;

    public NaoInteiroException(int num, int denom) {
        super();
        this.num = num;
        this.denom = denom;
    }

    @Override
    public String toString() {
        return "Resultado de " + num + "/" + denom + " não é inteiro";
    }
}
```

EXCEÇÕES PROPRIETÁRIAS

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

    int[] numero = {4, 8, 16, 21, 32, 64, 128};
    int[] denom = {2, 0, 4, 8, 0};

    for (int i=0; i<numero.length; i++){
        try{
            if (numero[i] % 2 != 0){
                throw new NaoInteiroException(numero[i], denom[i]);
            }
            System.out.println(numero[i] + "/" + denom[i] + " = " + (numero[i]/denom[i]));
        }
        catch (ArithmeticException | ArrayIndexOutOfBoundsException | NaoInteiroException e){
            e.printStackTrace();
        }
    }
}
```