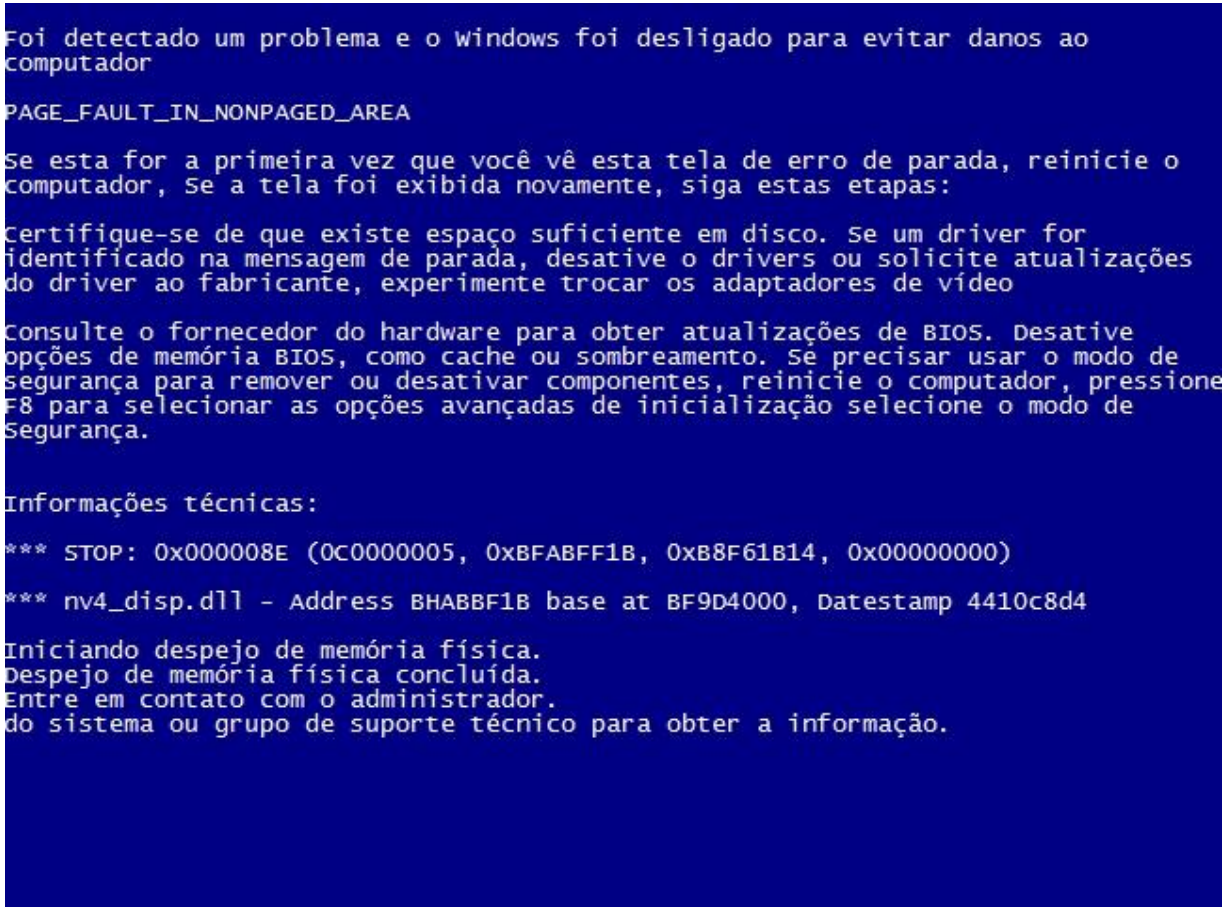
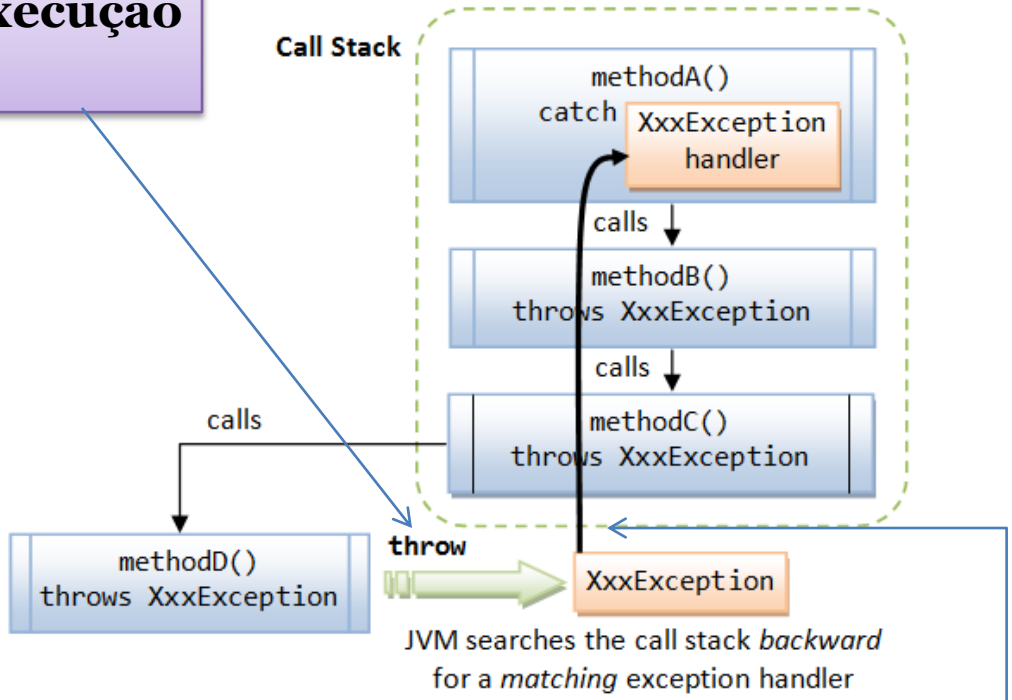
Java Orientado a Objetos
Exceções



Exceções

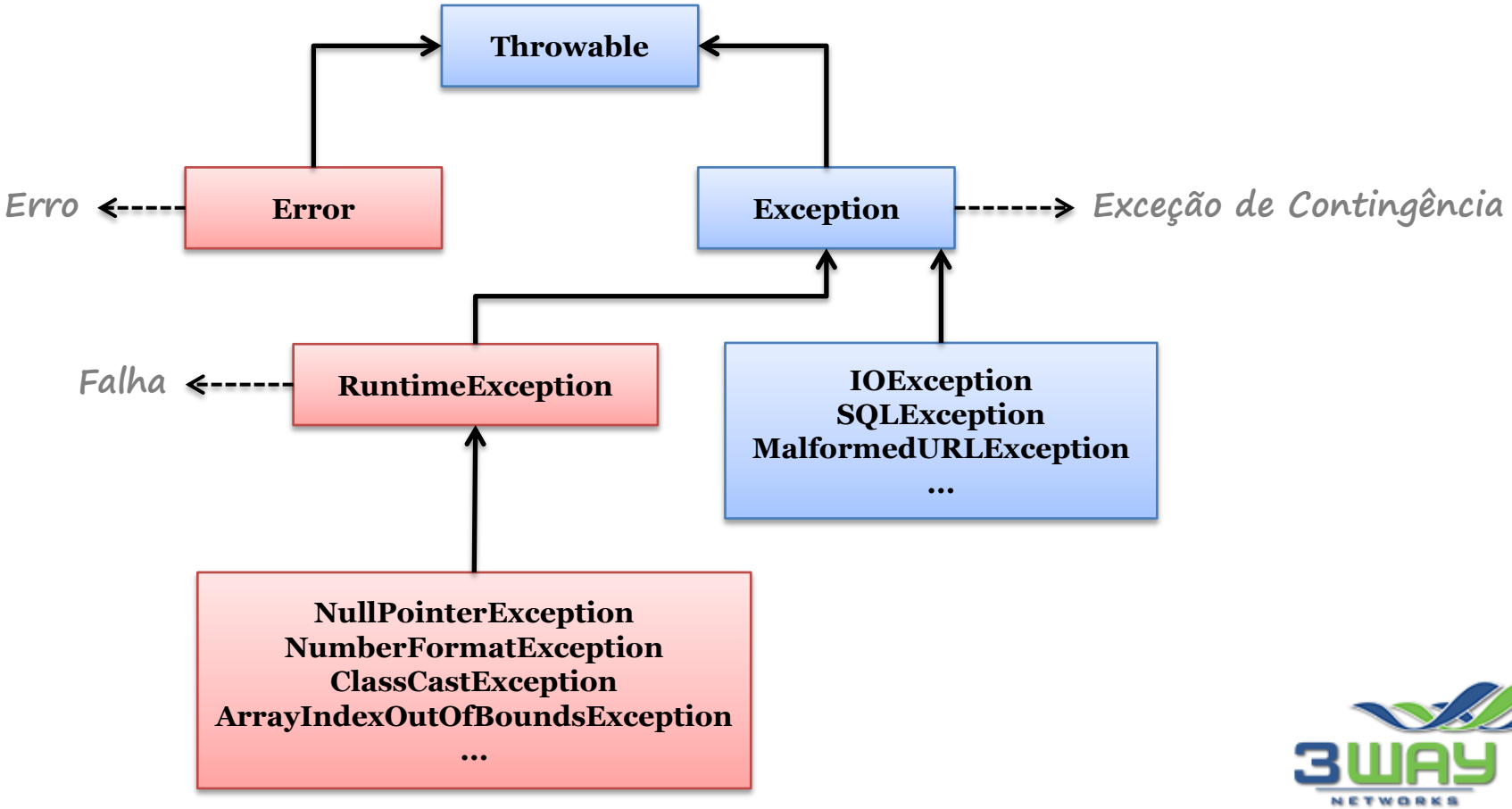
Erros que ocorrem **durante** a **execução** do programa



É um **evento** que **interrompe** o fluxo normal de **processamento** de uma classe

Categoria de Exceções

Todas as exceções são **subclasses**, direta ou indiretamente, da classe **java.lang.Throwable**



Manipulando Exceções

Utilizando a declaração **try-catch-finally**

```
public static void main(String... args) {
    PrintStream ps = System.out;
    InputStreamReader leitor = new InputStreamReader(System.in);
    int[] array = { 1, 2, 3, 4 };
    try {// IOException
        Character ch = (char) leitor.read();
        // NumberFormatException
        int i = Integer.parseInt(ch.toString());
        // ArrayIndexOutOfBoundsException
        ps.println(array[i]);
    } catch (ArrayIndexOutOfBoundsException e) {
        ps.printf("Indice fora do limite [0..3] : %s\n", e.getMessage());
    } catch (NumberFormatException e) {
        ps.printf("Erro de conversão : %s\n", e.getMessage());
    } catch (IOException e) {
        ps.printf("Erro de entrada/saída : %s\n", e.getMessage());
    } finally {
        ps.println("Sempre passo aqui para fechar todos os recursos");
    }
}
```

*O bloco **catch** recebe um argumento do tipo de exceção que será tratado.*

Tratamento da exceção

Sempre será executado

*Para cada bloco **try**, pode haver **um ou mais** blocos **catch**, mas somente um bloco **finally***

Um catch Múltiplas Exceções

Um catch com Múltiplas classes de Exceção

```
public static void main(String... args) {
    PrintStream ps = System.out;
    InputStreamReader leitor = new InputStreamReader(System.in);
    int[] array = { 1, 2, 3, 4 };
    try {// IOException
        Character ch = (char) leitor.read();
        // NumberFormatException
        int i = Integer.parseInt(ch.toString());
        // ArrayIndexOutOfBoundsException
        ps.println(array[i]);
    } catch (ArrayIndexOutOfBoundsException |
             NumberFormatException |
             IOException e) {
        ps.printf("Um erro aconteceu : %s \n", e);
    } finally {
        ps.println("Sempre passo aqui para fechar todos os recursos");
    }
}
```

O bloco **catch** recebe um argumento de vários tipos de exceção separados pelo operador (|)

Tratamento da exceção

Sempre será executado

JAVA7: Para cada bloco **try**, pode haver **um único catch**, com muitos tipos de Exceção

try-com-recursos

```
InputStreamReader leitor = new InputStreamReader(System.in);
try { // IOException
    Character ch = (char) leitor.read();
} catch (IOException e) {
    ps.printf("Um erro aconteceu : %s \n", e);
} finally {
    if (leitor != null) {
        try { // fecha recurso
            leitor.close();
        } catch (Exception e) {
            ps.println("Sempre fechar o recurso");
        }
    }
}
```

Fechando recursos, tratamento convencional, até Java6, finally explícito e você invoca o método close() do recurso

```
try (InputStreamReader leitor =
        new InputStreamReader(System.in)) {
    // IOException
    Character ch = (char) leitor.read();
} catch (IOException e) {
    ps.printf("Um erro aconteceu : %s \n", e);
}
```

Java 7, o recurso é declarado no **try()** o **finally** é implícito, método close() de **AutoCloseable** é invocado automaticamente

Recursos como arquivos, conexão de banco de dados, socket de rede, etc., que implementam interface **AutoCloseable** o **finally** é implícito

Throw e Throws

Se um método causar uma exceção mas não capturá-la, então deve-se utilizar a palavra-chave **throws**

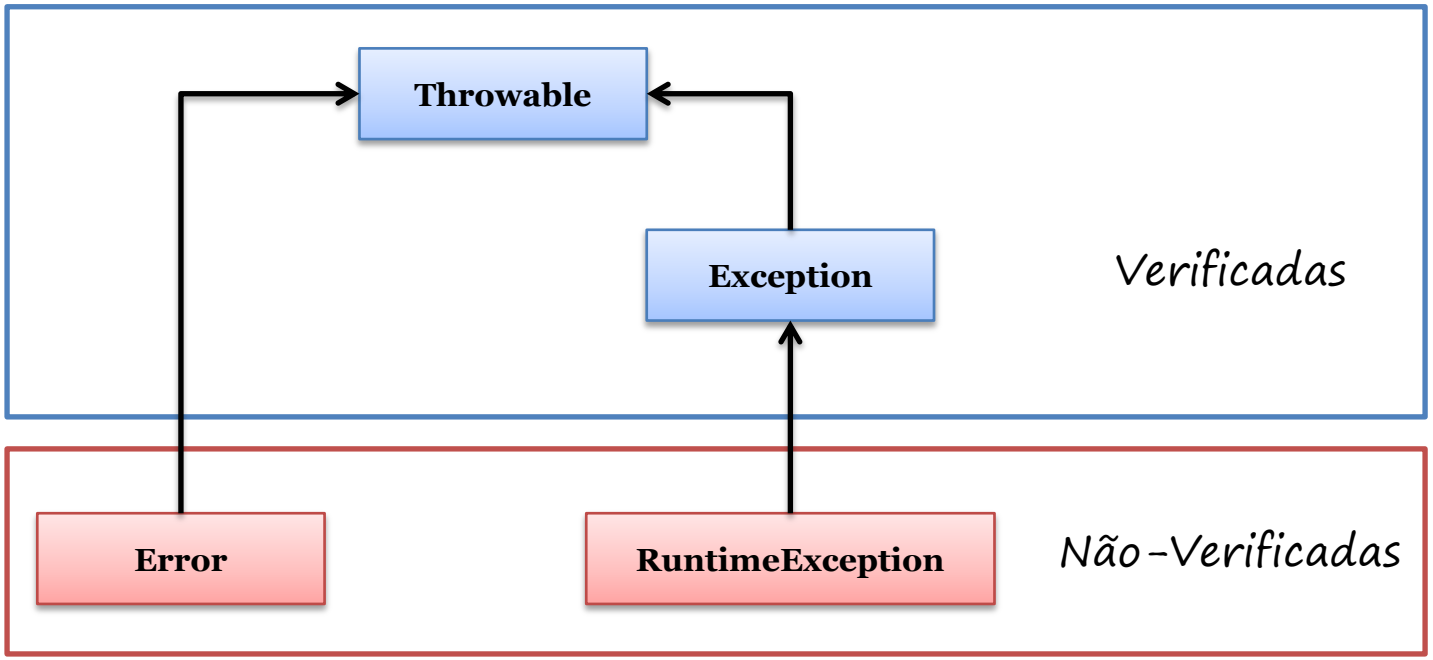
```
public class Calculadora {

    public static void main(String[] args) {

        Double nota1 = 5.0;
        Double nota2 = 3.0;

        try {
            System.out.println(Calculadora.calculaMedia(nota1, nota2));
        } catch (Exception e) {
            System.out.print("Tratamento de erro: ");
            System.out.println(e.getMessage());
        }
    }

    public static Double calculaMedia(Double x, Double y) throws Exception {
        Double media = ( x + y ) / 2;
        if (media < 6) {
            throw new Exception("Criando exceção com throws");
        }
        return media;
    }
}
```

Tratamento da exceção

Desviando a exceção

Lançamento da exceção

Exceções Verificadas e Não-Verificadas



Criando Exceções

```
public class MediaInsuficienteException extends Exception {
    public MediaInsuficienteException() {
        super("Exception criada para média menor que 6.0");
    }
}
```

Atributos de objetos e construtores podem ser adicionados à classe

```
public static void main(String[] args) {
    Double nota1 = 5.0;
    Double nota2 = 3.0;
    try {
        System.out.println(Calculadora.calculaMedia(nota1, nota2));
    } catch (MediaInsuficienteException e) {
        System.out.print("Tratamento de erro: ");
        System.out.println(e.getMessage());
    }
}

public static Double calculaMedia(Double x, Double y) throws MediaInsuficienteException {
    Double media = ( x + y ) / 2;
    if (media < 6) {
        throw new MediaInsuficienteException();
    }
    return media;
}
```

Sobrepondo Métodos e Exceções

```java
public class ClasseA {
    public void metodoA() throws RuntimeException {
        // operações que podem lançar exceção
    }
}
```

Não Pode, a classe Exception é superclasse de RuntimeException

```java
public class ClasseC extends ClasseA {
    @Override
    public void metodoA() throws Exception {
        // operações que podem lançar exceção
    }
}
```

Pode,é subclasse de RuntimeException

```java
public class ClasseB extends ClasseA {
    @Override
    public void metodoA() throws NullPointerException {
        // operações que podem lançar exceção
    }
}
```

Ao sobrepor métodos com **throws**, o método deve lançar a mesma exceção ou um de suas subclasses e **não pode** ser adicionado tipos diferentes.