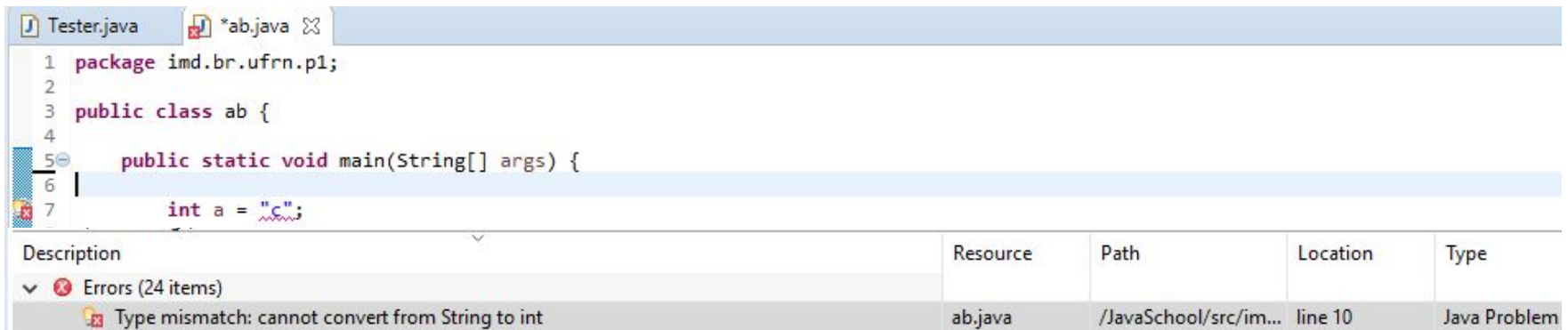


Linguagem de Programação II – IMD0040

Aula 20 – Tratamento de Exceções

Introdução

- ❑ Programas de computador podem conter **erros**.
- ❑ Às vezes, **erros imprevistos**.
- ❑ O que se **deve fazer** na presença de um **erro de sintaxe**?



The screenshot shows an IDE with two tabs: 'Tester.java' and '*ab.java'. The code in 'ab.java' is as follows:

```
1 package imd.br.ufrn.pl;  
2  
3 public class ab {  
4  
5     public static void main(String[] args) {  
6  
7         int a = "c";  
8     }  
9 }
```

Below the code editor, there is a table displaying the error details:

Description	Resource	Path	Location	Type
Errors (24 items)				
Type mismatch: cannot convert from String to int	ab.java	/JavaSchool/src/im...	line 10	Java Problem

Introdução

- ❑ E na presença de um **erro de lógica**?



Introdução

- ❑ Há possibilidade de ocorrer erros imprevistos durante a execução de um **programa em Java**:
 - ❖ Esses erros são conhecidos como **exceções**; e
 - ❖ Podem ser provenientes de erros de lógica; ou
 - ❖ Acesso a dispositivos; ou
 - ❖ Acesso a arquivos externos.

Introdução

❑ **Exceção:**

- ❖ Indicação de um **problema** que ocorre durante a execução de um programa.

❑ **Tratamento de exceções:**

- ❖ **Resolver** exceções que poderiam ocorrer, para que o programa **continue sua execução**, ou **termine** de forma elegante.

- ❑ O tratamento de exceções permite que os programadores criem **programas mais robustos e tolerantes a falhas**.

Entendendo as exceções

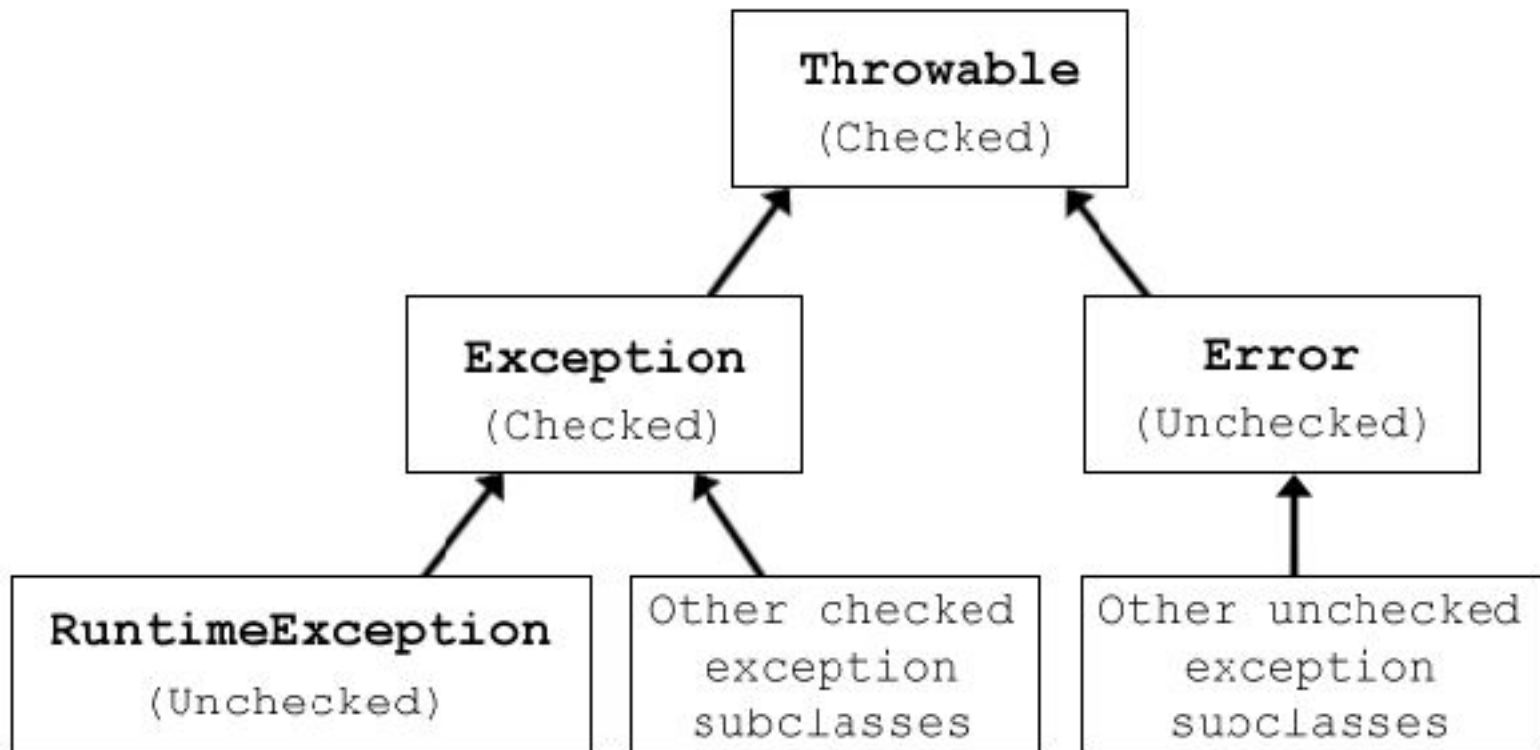
- ❑ As exceções podem ser provenientes de erros de lógica ou acesso a recursos indisponíveis.
- ❑ Alguns possíveis **motivos externos** para ocorrer uma exceção são:
 - ❖ Tentar abrir um arquivo que não existe.
 - ❖ Tentar fazer uma consulta a um banco de dados que não está disponível.
 - ❖ Tentar escrever em um arquivo que não se tem permissão de escrita.
 - ❖ Tentar conectar em servidor inexistente.

Entendendo as exceções

- ❑ Alguns possíveis **erros de lógica** que ocorrem em exceção:
 - ❖ Tentar manipular um **objeto** que está com o **valor nulo**.
 - ❖ Divisão por **zero**.
 - ❖ Tentar manipular um tipo de dado como se fosse outro.
 - ❖ Tentar utilizar um método ou classe não existentes.

Hierarquia de exceção em Java

- ❑ A classe **Throwable** é a superclasse desta hierarquia, e as classes **Exception** e **Error** são as subclasses.



Hierarquia de exceção em Java

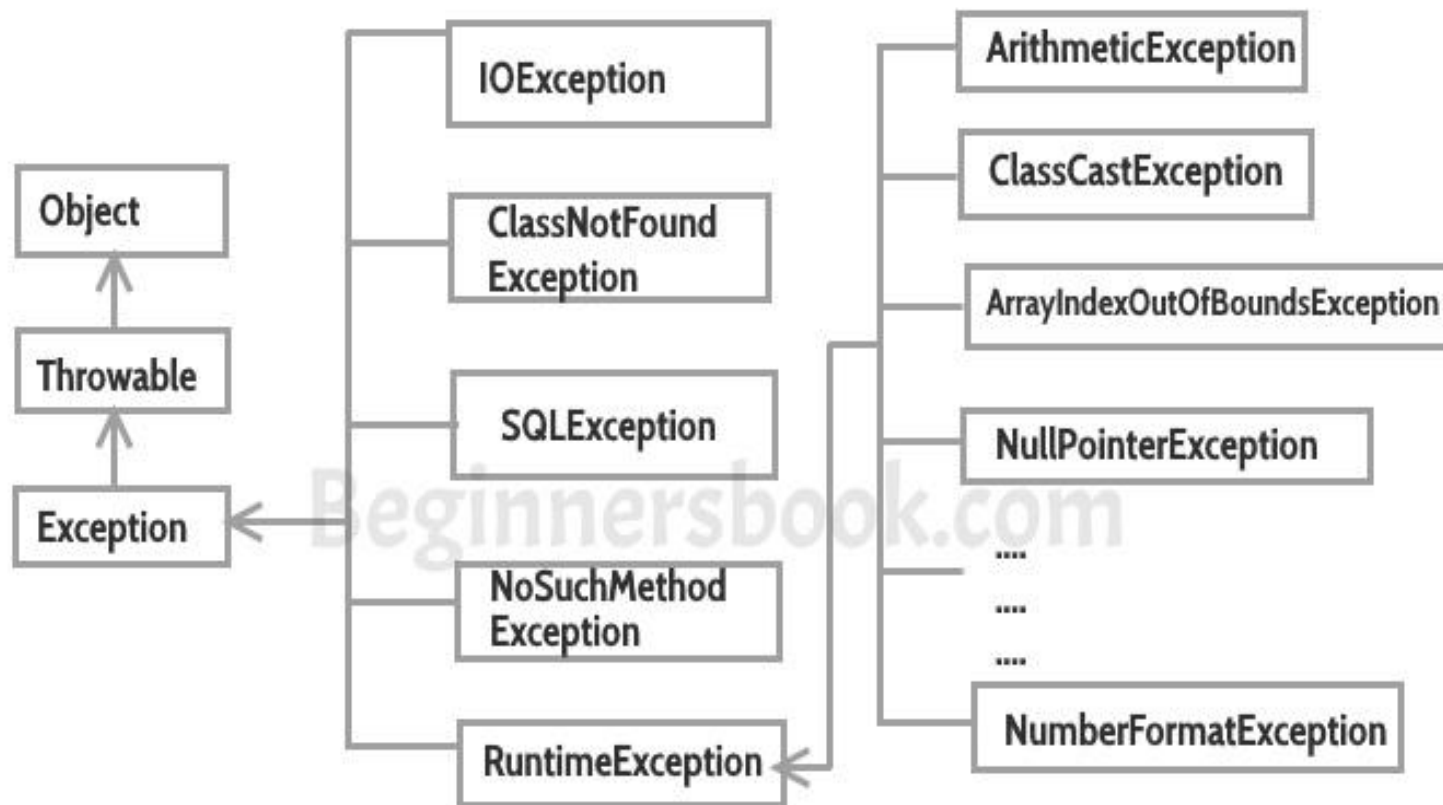
❑ Classe **Throwable**:

- ❖ Somente objetos **Throwable** ou de suas subclasses podem ser **gerados**, **propagados** e **capturados** através do mecanismo de tratamento de exceções.

❑ Subclasses:

- ❖ A classe **Exception** e suas subclasses representam **situações excepcionais**, que podem ocorrer em um programa Java e que podem ser capturadas pelo aplicativo.
- ❖ A classe **Error** e suas subclasses representam situações anormais que podem acontecer na JVM, e **normalmente não é possível recuperá-las**.

Categorias de exceções



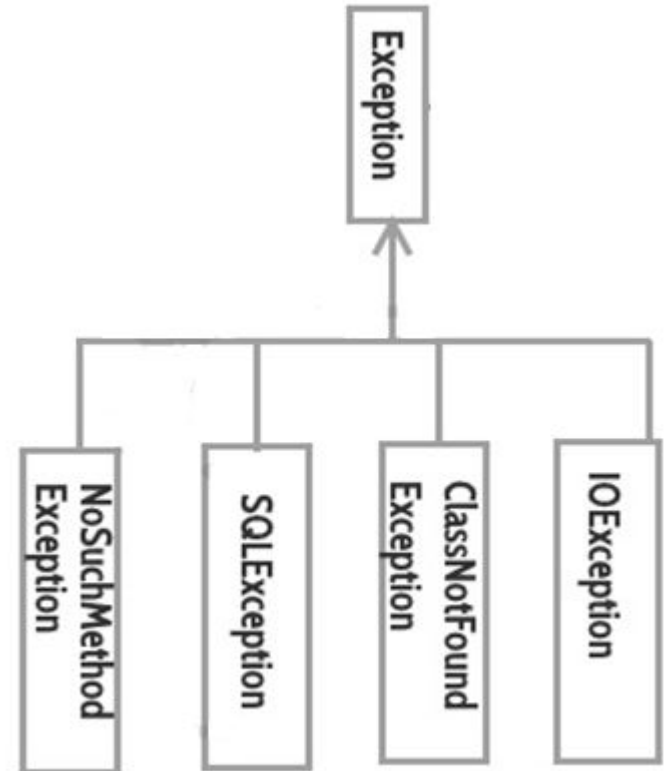
Categorias de exceções

❑ Exceções verificadas (checked):

❖ Representam condições inválidas em áreas fora do controle imediato do programa.

❖ São subclasses da classe **Exception**.

- Exemplo: **IOException** (Problemas de entradas inválidas).

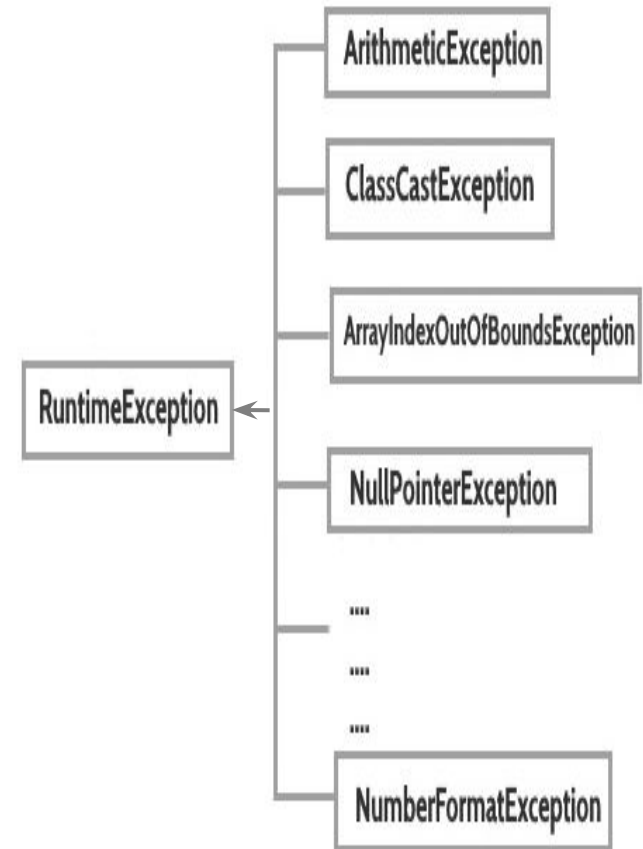


Categorias de exceções



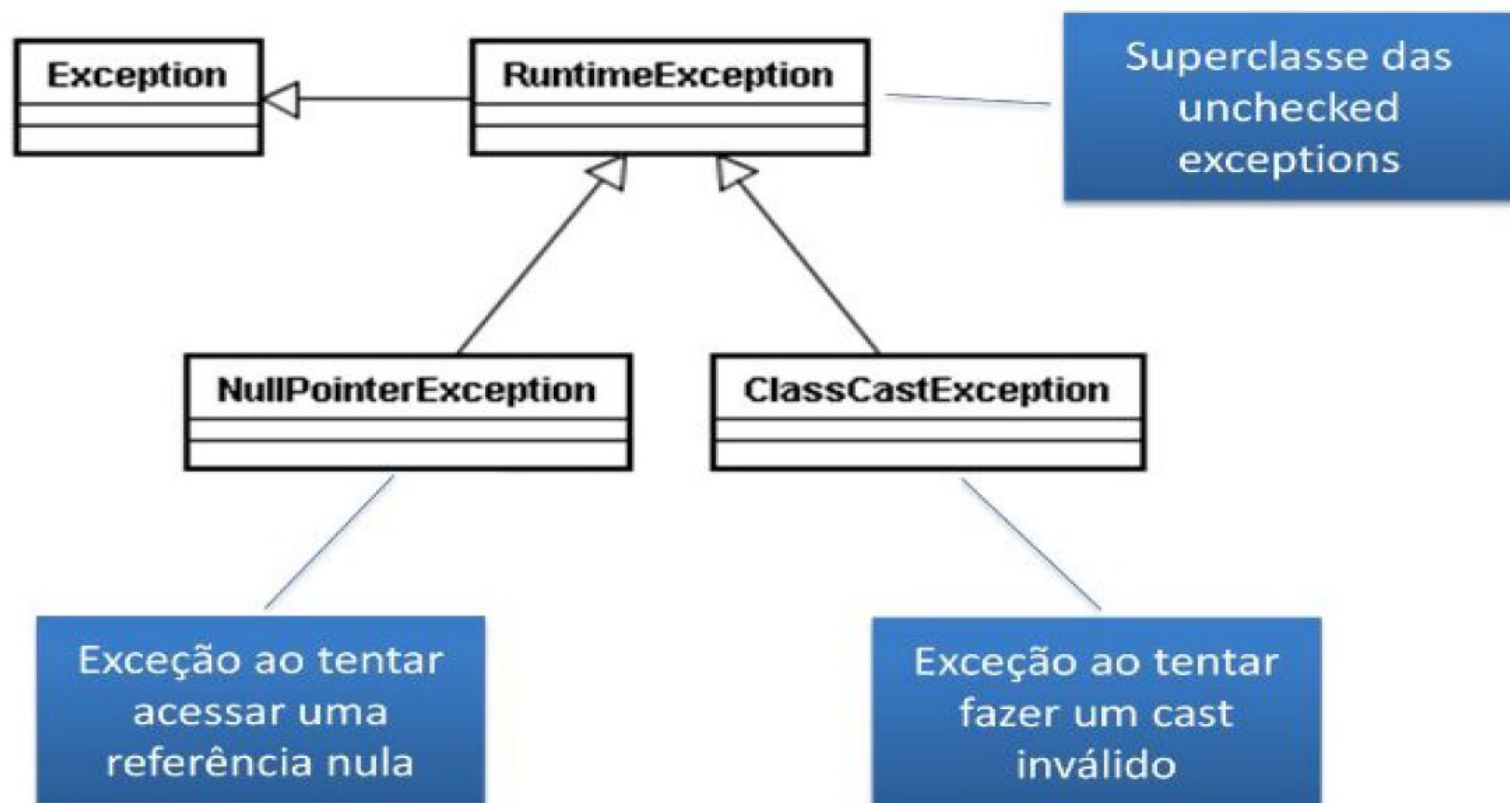
Exceções não-verificadas (unchecked):

- ❖ Representam defeitos no programa (**bugs**).
- ❖ Exceções que herdam da classe **RuntimeException**.
- ❖ Se uma exceção não-verificada ocorrer, o programa **terminará** ou executará com **resultados inesperados**.



Categorias de exceções

❑ Exceções não-verificadas:



Lançando Exceções

❑ Cláusula **throws**:

- ❖ Especifica as exceções que um método pode lançar;
- ❖ Aparece depois da lista de parâmetros do método e antes do corpo do método;
- ❖ Contém uma lista separada por vírgulas das exceções;
- ❖ As exceções podem ser lançadas pelas instruções no corpo do próprio método, ou por outros métodos chamados pelo primeiro método;
- ❖ As exceções podem ser dos tipos listados na cláusula **throws** ou de subtipos.

Lançando Exceções

- ❑ O lançamento de exceções é feito através da palavra **throw**:

```
public void fazerAlgo() throws Exception {  
    throw new Exception();  
}
```

O *throw* é usado para
lançar a exceção

O *throws* indica que o método
pode lançar a exceção

Lançando Exceções

- É possível também lançar subclasses* da exceção declarada pelo **throws**:

```
public void fazerAlgo() throws Exception {  
    throw new IOException();  
}
```

IOException é uma
subclasse de *Exception*

A declaração *throws Exception*
está de acordo com a exceção
lançada pelo método

*É necessário cuidado ao tomar esta decisão. Pode gerar outros problemas

Tratando Exceções

- ❑ Bloco **try** contém o código que pode lançar (**throw**) uma exceção.
- ❑ Consiste na palavra-chave **try** seguida por um bloco de código entre chaves.
- ❑ Se ocorrer uma **exceção** em algum ponto, o restante do código contido no bloco **try** **não será executado**.

Capturando exceções

- ❑ Um bloco **catch**:
 - ❖ Captura, isto é, recebe e trata uma exceção.
 - ❖ Começa com a palavra-chave **catch**.
 - ❖ O parâmetro de exceção identifica o **tipo de exceção** e permite que o bloco **catch** interaja com o objeto da exceção capturada.
 - ❖ Bloco do código entre chaves será executado quando uma **exceção do tipo** adequado ocorrer.

Capturando exceções

- ❑ Bloco **catch** correspondente:
 - ❖ Ao tipo do parâmetro de **exceção** corresponde exatamente ao **tipo de exceção lançado** ou é uma superclasse dele.
- ❑ Exceção **não-capturada**:
 - ❖ Uma exceção que ocorre e não há nenhum bloco **catch** correspondente.
 - ❖ Faz com que o programa termine, se tiver somente uma **thread**;
 - ❖ Do contrário apenas a **thread** atual é terminada.

Tratando Exceções

```
public void m1() throws Exception {  
    throw new Exception();  
}
```

```
public void m2() {  
    try {  
        m1();  
    } catch (Exception e) {  
        ...  
    }  
    ...  
}
```

Se uma *Exception* acontecer,
o fluxo é desviado para o
bloco *catch*

Ao fim do bloco *catch*, a
execução continua após o bloco

Tratando Múltiplas Exceções

```
public void m1() throws IOException, SQLException {  
    ...  
}
```

```
public void m2() {  
    try {  
        m1();  
    } catch (IOException e) {  
        ...  
    } catch (SQLException e) {  
        ...  
    }  
    ...  
}
```

Dependendo da exceção, o bloco *catch* correspondente é executado

No máximo um bloco *catch* é executado

Refinando o tratamento

❑ Bloco **finally**:

- ❖ Consiste na palavra-chave **finally** seguida por um bloco do código entre chaves;
- ❖ É opcional em uma instrução **try**;
- ❖ Normalmente, é colocado depois do último bloco **catch**;
- ❖ Executa se uma exceção for lançada no bloco **try** correspondente, ou qualquer um dos seus blocos **catch** correspondentes;
- ❖ Em geral, contém código de liberação de recursos.

Refinando o tratamento

□ Exemplo:

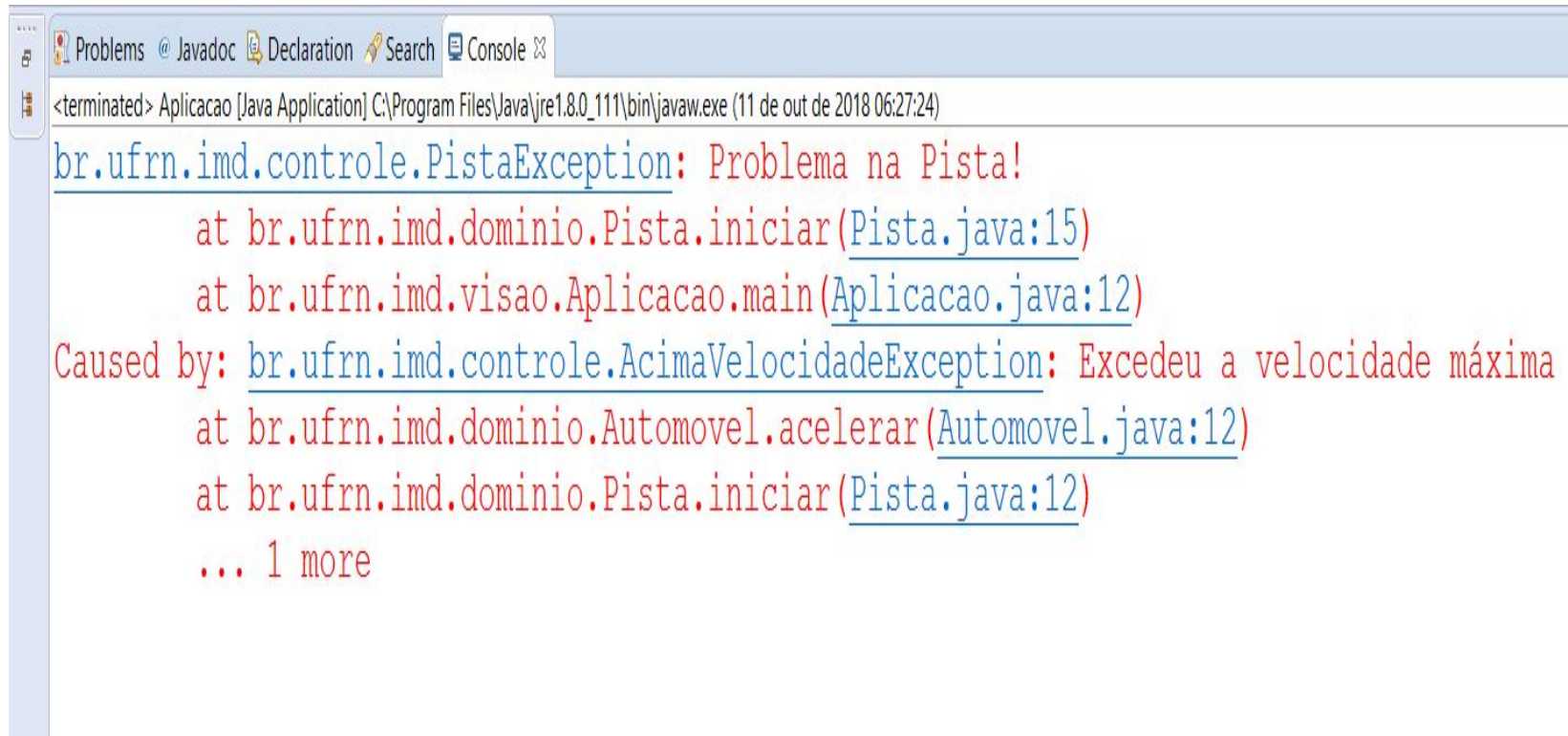
```
try
{
    instruções
    instruções de aquisição de recurso
} // fim de try
catch ( UmTipoDeExceção exceção1 )
{
    instruções de tratamento de exceções
} // fim de catch
.
.
.
catch ( OutroTipoDeExceção exceção2 )
{
    instruções de tratamento de exceções
} // fim de catch
finally
{
    instruções
    instruções de liberação de recursos
} // fim de finally
```

Compreendendo a Stacktrace

- ❑ Ferramenta poderosa para análise de erros.
- ❑ Ela fornece a causa das exceções.
- ❑ Fornece o ponto onde a exceção aconteceu.
- ❑ Mostra a pilha (stack) de chamada de métodos até o momento da exceção.
- ❑ A pilha de erros deve ser lida de baixo para cima.

Compreendendo a Stacktrace

Exemplo:



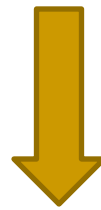
```
Problems @ Javadoc Declaration Search Console
<terminated> Aplicacao [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (11 de out de 2018 06:27:24)
br.ufrn.imd.controle.PistaException: Problema na Pista!
    at br.ufrn.imd.dominio.Pista.iniciar(Pista.java:15)
    at br.ufrn.imd.visao.Aplicacao.main(Aplicacao.java:12)
Caused by: br.ufrn.imd.controle.AcimaVelocidadeException: Excedeu a velocidade máxima
    at br.ufrn.imd.dominio.Automovel.acelerar(Automovel.java:12)
    at br.ufrn.imd.dominio.Pista.iniciar(Pista.java:12)
    ... 1 more
```

Compreendendo a Stacktrace

```
3 public class TestaExcecao {
4
5     public static void main(String[] args) throws VerificaException{
6         String frase = "Vamos Brasil";
7
8         if (frase.contains("br") || frase.contains("Br")){
9             System.out.println("Existe 'br' ou 'Br' em sua frase");
10        }
11        else {
12            throw new VerificaException();
13        }
14    }
15 }
```

Compreendendo a Stacktrace

```
3 public class VerificaException extends Exception {  
4  
5     private static final long serialVersionUID = 1L;  
6  
7     public String getMessage() {  
8         return "Não existe letra 'Br' em sua frase";  
9     }  
10 }
```



Problems | Javadoc | Declaration | Search | Console

<terminated> TestaExcecao [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (08/10/2018 17:04:48)

Exception in thread "main" [propria.VerificaException](#): Não existe letra 'Br' em sua frase
at [propria.TestaExcecao.main](#)([TestaExcecao.java:12](#))

Perguntas ...



Obrigado

