

COTIL



UNICAMP

Linguagem de Programação III

Profa. Simone Berbert Rodrigues Dapólitto

**Cap. 11 – Tratamento de
Exceções**

Tratamento de Exceções

- Uma das diferenças mais evidentes entre um bom e um mau programa é o **tratamento de exceções**, isto é, a capacidade que o seu programa tem de responder a **situações inesperadas**.
- Quando o tratamento de exceções é feito adequadamente o seu programa fica mais **robusto** e com um bom nível de **usabilidade**.

Tratamento de Exceções

- Até agora nós criamos pequenos programas sem nos preocuparmos em tratar possíveis **erros**.
- No entanto todos aqueles que programam há algum tempo sabem que o tratamento de erros (exceções) é parte **fundamental** da programação.
- Vamos começar os nossos estudos conhecendo a forma como uma **exceção** é apresentada e o seu significado. Observe as duas classes seguintes:

Tratamento de Exceções

- Classe TesteExcecoes

```
public class TesteExcecoes{  
    public static void main(String[] args) {  
        new RecebeArray(args);  
        System.out.println("Término do programa!");  
    }  
}
```

Tratamento de Exceções

- Classe RecebeArray

```
public class RecebeArray {  
  
    public RecebeArray(String[] array) {  
        imprimirPosicao0(array);  
    }  
  
    public void imprimirPosicao0(String[] array) {  
        System.out.println(array[0]);  
    }  
}
```

Tratamento de Exceções

- Ao executar a classe TesteExcecoes e essa, invocar o método main, devemos informar alguns valores para o parâmetro args, pois esse está sendo usado no código.
- Se informarmos os valores para o parâmetro args do método main, que é um array, a execução ocorrerá normalmente.

Tratamento de Exceções

- Mas se tentarmos executar a classe TesteExcecoes, sem o envio de parâmetros para o método main, na instanciação de um novo objeto RecebeArray será passado como parâmetro do construtor o array vazio.
- O construtor por sua vez invoca um método desta mesma classe (`imprimirPosicao0()`) repassando o parâmetro recebido (vazio).
- Este método tenta imprimir o elemento da posição 0, primeira posição, e o resultado obtido é uma exceção, já que o array está vazio.

Tratamento de Exceções

- Quando esta exceção acontece o programa é encerrado de forma inesperada e não é executado até o final.
- Podemos observar a visualização da execução desta aplicação na figura a seguir:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at Excecoes.RecebeArray.imprimirPosicao0(RecebeArray.java:10)
    at Excecoes.RecebeArray.<init>(RecebeArray.java:6)
    at Excecoes.TesteExcecoes.main(TesteExcecoes.java:6)
```

Tratamento de Exceções

- A exceção foi do tipo **java.Lang.ArrayIndexOutOfBoundsException** que significa a tentativa de acesso a uma posição fora dos limites do array.
- A seguir ele informa que posição foi, neste caso o índice 0 (zero).
- Embaixo da definição da exceção nós temos a **stacktrace** que contém a sequência da pilha de execução.
- Pela **stacktrace** é possível perceber quais caminhos a execução percorreu até alcançar a **exceção**.

Tratamento de Exceções

- A **primeira** linha da stacktrace contém a **definição da exceção**.
- A **segunda** linha contém o **último trecho de código executado** com o **número da linha** no código fonte. Neste caso foi a invocação do método imprimirPosicao0() da classe RecebeArray, que no código fonte (arquivo RecebeArray.java) refere-se a linha 10.

Tratamento de Exceções

- A fim de construirmos programas mais robustos, poderíamos **tratar situações** como esta. Para isto iremos utilizar o bloco **try/catch**:

```
try{  
    //Código perigoso  
} catch(Exception e) {  
    //tratamento da exceção  
}
```

Tratamento de Exceções

- O bloco **try/catch** define dois trechos de código:
 - Um sujeito a erros (**try**)
 - Outro que responsável pelo tratamento do erro caso aconteça (**catch**).
- Vamos modificar agora a nossa classe RecebeArray e adicionar este bloco ao método `imprimirPosicao0()`:

Tratamento de Exceções

- Classe RecebeArray

```
public class RecebeArray {  
    public RecebeArray(String[] array) {  
        imprimirPosicao0(array);  
    }  
  
    public void imprimirPosicao0(String[] array) {  
        try{  
            System.out.println(array[0]);  
        } catch(ArrayIndexOutOfBoundsException e) {  
            System.out.println("Erro: Array vazio, execute o  
                programa novamente passando ao menos um  
                parâmetro.");  
        }  
    }  
}
```

Tratamento de Exceções

- E agora ao executarmos novamente este mesmo programa o usuário irá receber uma mensagem mais significativa, observe a execução abaixo:

Erro: Array vazio, execute o programa novamente passando ao menos um parâmetro.
Término do programa!

Tratamento de Exceções

- Pronto! Agora ao executar o programa sem parâmetros o **usuário saberá o que fazer**.
- Uma vez que a exceção foi tratada, o programa continua o fluxo de execução e termina normalmente.
- Devemos observar o seguinte, esta não é a única exceção que pode acontecer neste método, vamos modificar o método main() de forma que ele encaminhe como parâmetro um valor nulo (null).

Tratamento de Exceções

```
public class TesteExcecoes{  
    public static void main(String[] args) {  
        new RecebeArray(null);  
        System.out.println("Termino do programa!");  
    }  
}
```

Tratamento de Exceções

- Repare que agora a nova instância da classe RecebeArray está recebendo como parâmetro uma referência nula, isto ocasionará uma exceção do tipo **java.lang.NullPointerException**.
- Vejamos o resultado da execução deste código:

```
Exception in thread "main" java.lang.NullPointerException
    at Excecoes.RecebeArray.imprimirPosicao0(RecebeArray.java:11)
    at Excecoes.RecebeArray.<init>(RecebeArray.java:6)
    at Excecoes.TesteExcecoes.main(TesteExcecoes.java:6)
```

Tratamento de Exceções

- O programa encerrou de forma **inesperada** – a mensagem “Termino do programa!” não foi apresentada.
- O bloco **try/catch** não foi capaz de **capturar a exceção** lançada pelo código.
- Isso aconteceu pois ele foi construído de forma que apenas exceções do tipo `java.lang.ArrayIndexOutOfBoundsException` sejam tratadas.

Tratamento de Exceções

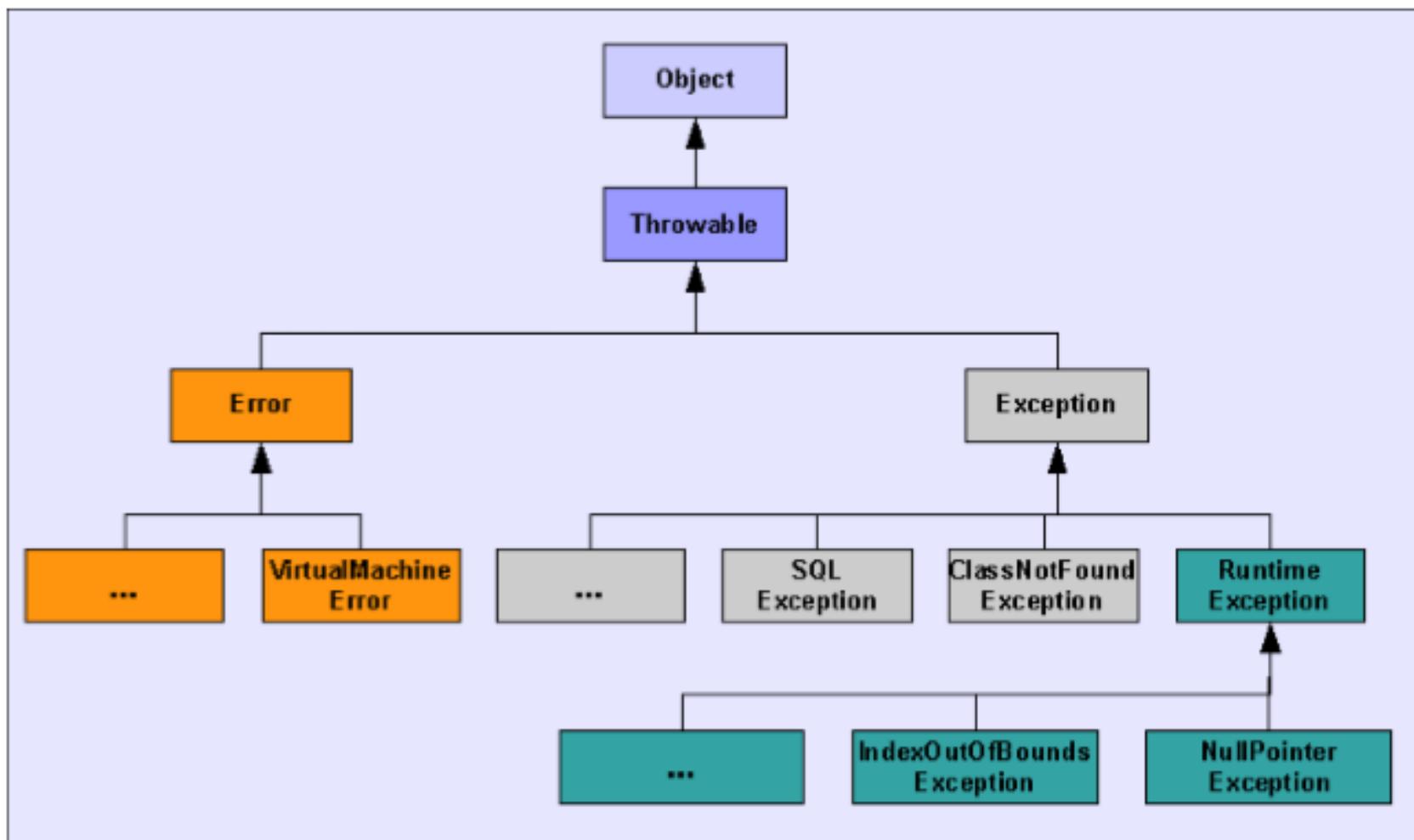
- Para corrigir este problema podemos encadear mais um bloco catch que capture todos os tipos de exceções, vejamos:

Tratamento de Exceções

```
public class RecebeArray {  
    public RecebeArray(String[] array) {  
        imprimirPosicao0(array);  
    }  
  
    public void imprimirPosicao0(String[] array) {  
        try {  
            System.out.println(array[0]);  
        } catch(ArrayIndexOutOfBoundsException e) {  
            System.out.println("Erro: Array vazio, execute o  
            programa novamente passando ao menos um parametro.");  
        } catch(NullPointerException e2) {  
            System.out.println("Valor nulo informado.");  
        }  
    }  
}
```

Tratamento de Exceções

- Todas as exceções em Java derivam da classe Throwable conforme hierarquia a seguir:



Tratamento de Exceções

- Observe que, conforme dito em módulos anteriores, todas as classes em Java estendem de alguma forma – direta ou indiretamente – da classe Object.
- Nesta imagem estão representadas as três modalidades de exceções existentes na linguagem Java:
 - Unchecked Exception (verde)
 - Checked Exception (cinza)
 - Error (laranja)

Tratamento de Exceções

- **Error:** Hierarquia em laranja na imagem, representam situações incomuns, que não são causadas pelo programa, indicam situações que não acontecem usualmente durante a execução de um programa (Ex: Estouro da pilha de execução – StackOverflowError);

Tratamento de Exceções

- **Checked Exception:** Hierarquia em cinza, representam situações que, geralmente, não são erros de programação e sim indisponibilidade de recursos ou condição necessária para a correta execução inexistente (Ex: Em aplicações distribuídas existe dependência externa de rede de comunicação – NoRouteToHostException).
- As checked exceptions são tratadas obrigatoriamente, isto é, o compilador só compila a classe se houver um tratamento (bloco try/catch) para aquele tipo de exceção.

Tratamento de Exceções

- **Unchecked Exception (RuntimeException):**
Hierarquia em verde, representam situações que, geralmente, identificam erros de programação (programa não é robusto) ou mesmo situações incomuns/difíceis de tratar (Ex: Acessar índice inválido em um array – `ArrayIndexOutOfBoundsException`);

Tratamento de Exceções

- Segue quadro com relação de algumas das mais comuns exceções:

Exceção	Quando acontece
ArrayIndexOutOfBoundsException	Tentativa de acesso a posição inexistente no array.
ClassCastException	Tentativa de efetuar um cast em uma referência que não é classe ou subclasse do tipo desejado.
IllegalArgumentException	Argumento formatado de forma diferente do esperado pelo método.
IllegalStateException	O estado do ambiente não permite a execução da operação desejada.
NullPointerException	Acesso a objeto que é referenciado por uma variável cujo valor é null.
NumberFormatException	Tentativa de converter uma String inválida em número.
StackOverflowError	Normalmente acontece quando existe uma chamada recursiva muito profunda.
NoClassDefFoundError	Indica que a JVM não conseguiu localizar uma classe necessária a execução.

Jogando (throw) exceções

- Possivelmente em algum trecho do nosso programa, se uma determinada condição acontecer, nós queiramos **lançar uma exceção**.

Jogando (throw) exceções

- Vejamos um exemplo:
 - Suponha o método abastecer() pertencente a uma classe que representa um veículo.
 - Este método recebe como parâmetro um Double que representa o valor que está sendo abastecido, logo não pode haver situações onde o valor seja negativo.
 - Caso esta situação aconteça nós iremos lançar uma **IllegalArgumentException**.

Jogando (throw) exceções

```
public void abastecer(Double litros) {  
  
    if (litros < 0) {  
  
        throw new IllegalArgumentException("Era  
        esperado um valor maior que 0:  
        "+litros);  
  
    }  
  
    else {  
  
        tanque += litros;  
  
    }  
}
```

Jogando (throw) exceções

```
public class appVeiculos {  
  
    public static void main(String args[] ) {  
        Veiculo v = new Veiculo();  
        v.abastecer(-1);  
    }  
}
```

Jogando (throw) exceções

- A palavra reservada **throw** é responsável por “jogar”/lançar a exceção.
- Logo a seguir, nós instanciamos um objeto do tipo **IllegalArgumentException** e passamos como parâmetro do método construtor o texto que deverá aparecer quando a stacktrace for impressa.
- Veja o resultado da execução deste método passando o valor -1 (um negativo):

Jogando (throw) exceções

```
Exception in thread "main" java.lang.IllegalArgumentException: Era esperado um valor maior que 0: -1.0
at Excecoes.Veiculo.abastecer(Veiculo.java:10)
at Excecoes.appVeiculos.main(appVeiculos.java:8)
```

O bloco **finally**

- Os blocos try e catch podem conter uma terceira cláusula chamada **finally** que indica o que deve ser feito após o término do bloco try ou de um catch qualquer.

O bloco finally

- É interessante colocar algo que é **imprescindível** de ser executado, caso o que você queria fazer **tenha dado certo, ou não.**
- O caso mais comum é o de **liberar um recurso** no finally, como um arquivo ou conexão com banco de dados, para que possamos ter a certeza de que aquele arquivo (ou conexão) vá ser fechado, mesmo que algo tenha falhado no decorrer do código.

O bloco finally

- No exemplo a seguir, o bloco finally será sempre executado, independentemente de tudo ocorrer bem ou de acontecer algum problema:

O bloco finally

```
try {
    // bloco try
} catch (IOException ex) {
    // bloco catch 1
} catch (SQLException sqlex) {
    // bloco catch 2
} finally {
    // bloco que sempre
    // executado, independente será se houve ou
    // não exception e se ela foi tratada ou não
}
```

Exercícios !

Bibliografia

- Material do site Java Starter - www.t2ti.com – acessado em 07/05/2015
- Curso Fundamentos de Java oferecido pela Softblue: www.softblue.com.br – acessado em 07/05/2015