UNIP
UNIVERSIDADE PAULISTA

ALPOO


Teoria - Java

Exceptions

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

1

UNIP
UNIVERSIDADE PAULISTA

Disciplina: ALPOO


Tema: Exceptions

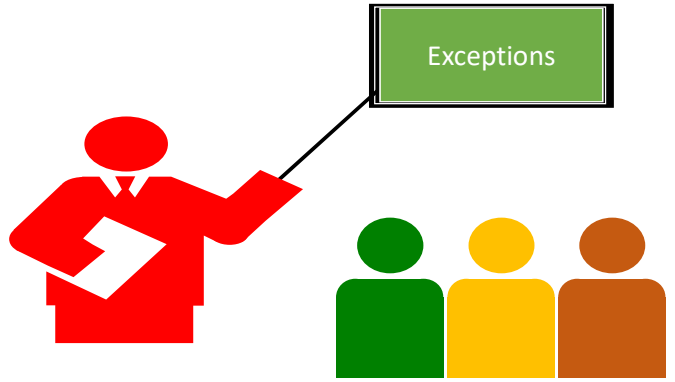
Professor: Ricardo Drudi

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

2

UNIP
UNIVERSIDADE PAULISTA




Exceptions

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

3

UNIP
UNIVERSIDADE PAULISTA

Exceptions

- Exceptions são eventos que quebram o fluxo normal do programa.
- Devem ser capturadas e tratadas de modo que não cheguem ao usuário final.
- Para lançar uma exceção: `throws`
- Para capturar uma exceção: `try / catch / finally`

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

4

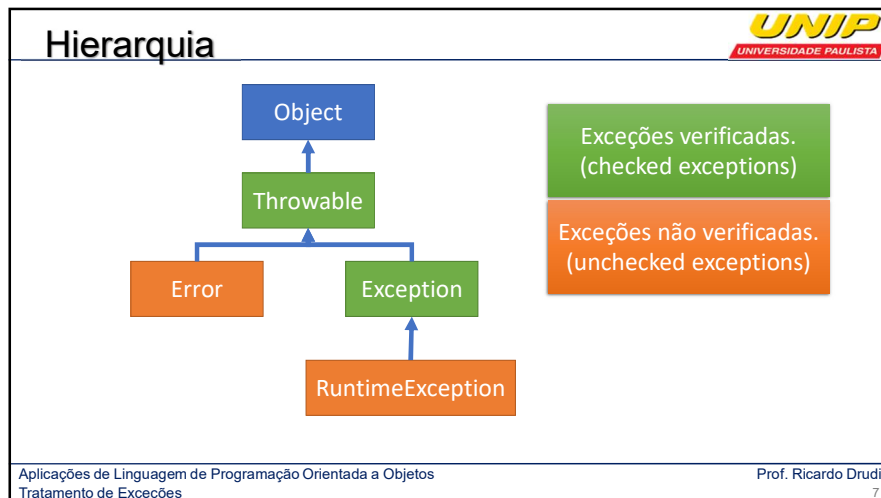
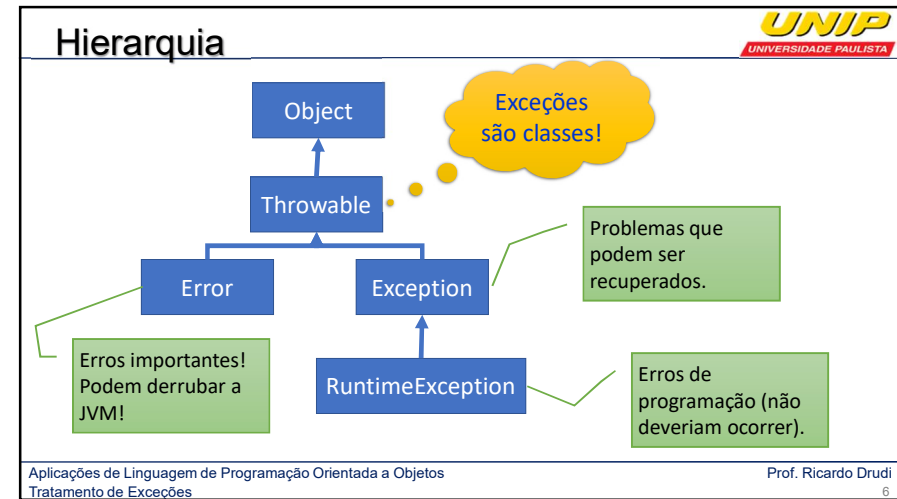
Exemplo

```
try {
    pst = con.prepareStatement("INSERT INTO Agenda
                               (Nome,Email) VALUES (?,?)");
    pst.setString(1, agenda.getNome());
    pst.setString(2, agenda.getEmail());
    pst.executeUpdate();
} catch (SQLException ex) {
    throw new RuntimeException("Erro no INSERT.");
} finally {
    ConnectionFactory.closeConnection(con, pst);
}
```

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

5



Checked Exceptions

- São exceções que devem ser capturadas e tratadas de alguma forma.
- Se o método que gera a exceção não a tratar, deve repassá-la adiante.

```
acesso retorno nome (parâmetros) throws Exception {
    ...
}
```

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

8

Checked Exceptions



- As exceções são uma parte fundamental de um método.
- Quem utiliza um método deve saber o tipo de retorno, seus parâmetros e **quais exceções ele pode lançar**.
- Assim, o usuário do método pode realizar o tratamento das exceções.
- Por isso o uso de exceções **checked** é fortemente recomendado.

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

9

Unchecked Exceptions



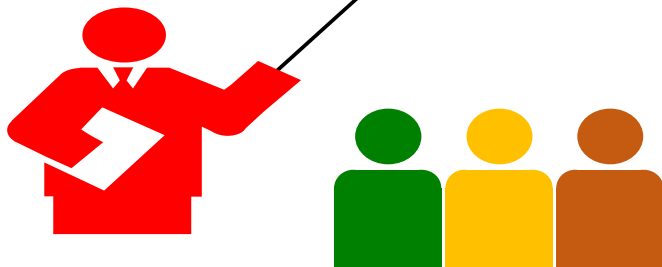
- Para que servem as classes `Error` e `RuntimeException`?
 - **Error** representa um erro importante que normalmente não pode ser tratado em tempo de programação. Geralmente são problemas de configuração ou de *hardware* que interferem no funcionamento da JVM.
 - **RuntimeException** representa um erro de programação, não é papel de quem usa o método tratar um erro interno do método.
 - Exemplos: divide by zero, null pointer, array out of bounds...

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

10

Criando Exceções



Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

11

Criando Exceções




- Na criação da instrução SQL pode ocorrer uma `SQLException`.
- No código utilizado, relançamos a exceção com `RuntimeException`, para facilitar a programação.
- Para exceções muito utilizadas é preferível criar classes específicas para cada tipo de exceção que possa ocorrer. Isso padroniza as mensagens e facilita a documentação do sistema.
- Com o erro no comando `Insert` é um erro interno de programação do método, vamos criar uma unchecked exception.

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

12

Criando Exceções



```
try {
    pst = con.prepareStatement("INSERT INTO Agenda
                               (Nome,Email) VALUES (?,?)");
    pst.setString(1, agenda.getNome());
    pst.setString(2, agenda.getEmail());
    pst.executeUpdate();
} catch (SQLException ex) {
    throw new RuntimeException("Erro no INSERT.");
} finally {
    ConnectionFactory.closeConnection(con, pst);
}
```


Ao invés de lançar uma exceção genérica, lança-se uma exceção criada especificamente para esse tipo de erro.

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

13

Criando Exceções



```
try {
    pst = con.prepareStatement("INSERT INTO Agenda
                               (Nome,Email) VALUES (?,?)");
    pst.setString(1, agenda.getNome());
    pst.setString(2, agenda.getEmail());
    pst.executeUpdate();
} catch (SQLException ex) {
    throw new SQLInsertException("Agenda");
} finally {
    ConnectionFactory.closeConnection(con, pst);
}
```


A classe `SQLInsertException` deve ser criada para lidar com esse tipo de erro.

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

14

Criando Exceções



- Para criar uma nova exceção **não verificada**, basta estender a classe `RuntimeException`.


```
public class SQLInsertException extends RuntimeException {
    public SQLInsertException(String mensagem) {
        JOptionPane.showMessageDialog("Erro no Insert.");
    }
}
```

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

15

Criando Exceções




- Um outro erro que pode ocorrer no método é o programador passar um objeto nulo.
- Esse tipo de erro deve ser reportado ao programador de alguma forma, para que ele saiba que o objeto não foi persistido no banco de dado.
- Assim, para esse caso, é mais aconselhável utilizar uma checked exception.
- As exceções criadas especificamente para o sistema podem ficar armazenadas em *package* próprio, facilitando sua reutilização.

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

16

Criando Exceções




```
public void insert (Agenda a) {  
    if (a == null) {  
        throw new NullPointerException("Agenda");  
    }  
    . . .  
}
```

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

17

Criando Exceções



- Para criar uma nova exceção **verificada**, basta estender a classe **Exception**.


```
public class NullPointerException extends Exception {  
    public NullPointerException(String mensagem) {  
        System.err.println("Objeto Vazio: " + mensagem);  
        printStackTrace();  
    }  
}
```

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

18

Criando Exceções



- O método que lançar a nova exception deve indicar isso em sua assinatura.

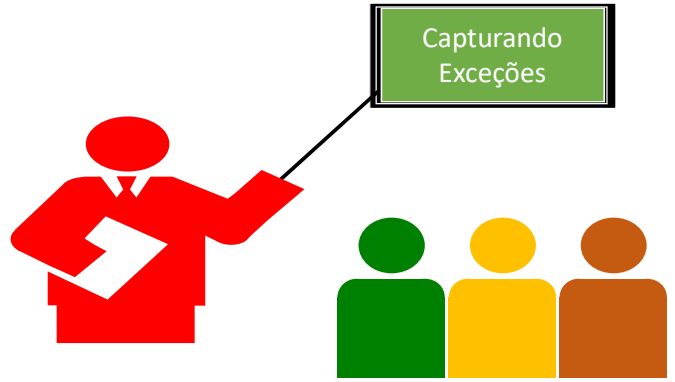

```
public void insert (Agenda a) throws NullPointerException {  
    if (a == null) {  
        throw new NullPointerException("Agenda");  
    }  
    . . .  
}
```

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

19

Criando Exceções



Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

20

Capturando Exceções



- Exceções verificadas devem obrigatoriamente ser capturadas e tratadas durante a programação.
- Para tanto existe a estrutura try/catch/finally, que lida com as exceções.
- O bloco finally é opcional na estrutura, e serve para garantir que certos códigos sejam executados em qualquer situação (com e sem a ocorrência da exceção).
- Geralmente o bloco finally é utilizado para finalização de comandos ou liberação de recursos solicitados.

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi
21

Exemplo



```
private boolean gravaTexto(String arq, String texto) {  
    boolean ok = true;  
    try {  
        FileWriter fin = new FileWrite(arq);  
        fin.write(texto);  
    } catch (IOException e) {  
        System.err.println("Arquivo de saída inexistente.");  
        ok = false;  
    } finally {  
        fin.close();  
    }  
    return (ok);  
}
```

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi
22

Try with resources



- Outra forma de garantir a liberação de recursos alocados é com a utilização da estrutura *try-with-resources*.
- Esse não é um comando diferente, mas sim uma forma diferente de utilização da estrutura try/catch.
- Para que o *try-with-resources* funcione, há 2 condições que devem ser obedecidas:
 - O recurso utilizado deve implementar a interface AutoClosable.
 - O recurso deve ser alocado na linha do comando try.

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi
23

Try with resources



```
// Abre recurso  
try {  
    // Faz alguma coisa  
} finally {  
    // Fecha recurso  
}
```

São formas equivalentes!

```
try (/*abre recurso*/) {  
}
```

O recurso será automaticamente
fechado ao final do bloco try
(mesmo que ocorra exceção).

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi
24

Exemplo

```
import java.io.BufferedReader;
import java.io.FileReader;

public class Principal {
    public static void main(String[] args) {
        try {
            BufferedReader br = new BufferedReader(new FileReader("arquivo.txt"));
            System.out.println(br.readLine());
            System.out.println(br.readLine());
            br.close();
        } catch (Exception e) {
            System.out.println("Erro: " + e);
        }
    }
}
```

Abre arquivo

Fecha arquivo

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

25

Exemplo

```
import java.io.BufferedReader;
import java.io.FileReader;

public class Principal {
    public static void main(String[] args) {
        try {
            BufferedReader br = new BufferedReader(new FileReader("arquivo.txt"));
            System.out.println(br.readLine());
            System.out.println(br.readLine());
            br.close();
        } catch (Exception e) {
            System.out.println("Erro: " + e);
        }
    }
}
```

Abre arquivo

Fecha arquivo

Se for lançada uma exceção entre a abertura e o fechamento do arquivo ele não será fechado!

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

26

Exemplo

```
import java.io.BufferedReader;
import java.io.FileReader;

public class Principal {
    public static void main(String[] args) {
        try {
            BufferedReader br = new BufferedReader(new FileReader("arquivo.txt"));
            System.out.println(br.readLine());
            System.out.println(br.readLine());
            int a = 3 / 0;
            br.close();
        } catch (Exception e) {
            System.out.println("Erro: " + e);
        }
    }
}
```

Abre arquivo

Exception in thread "main"
java.lang.ArithmeticException: divide by zero

Fecha arquivo

O arquivo não é fechado, devido à exceção lançada!

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

27

Exemplo

```
import java.io.BufferedReader;
import java.io.FileReader;

public class Principal {
    public static void main(String[] args) {
        try {
            BufferedReader br = new BufferedReader(new FileReader("arquivo.txt"));
            try {
                System.out.println(br.readLine());
                System.out.println(br.readLine());
                int a = 3 / 0;
            } finally {
                br.close();
            }
        } catch (Exception e) {
            System.out.println("Erro: " + e);
        }
    }
}
```

Abre arquivo

Exception in thread "main"
java.lang.ArithmeticException: divide by zero

Fecha arquivo

Agora o método sempre fecha o arquivo!

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

28

Exemplo

```
import java.io.BufferedReader;
import java.io.FileReader;

public class Principal {

    public static void main(String[] args) {
        try {
            (BufferedReader br = new BufferedReader(new FileReader("arquivo.txt"))){
                System.out.println(br.readLine());
                System.out.println(br.readLine());
                int a = 3 / 0;
            }
        } catch (Exception e) {
            System.out.println("Erro: " + e);
        }
    }
}
```

Abre arquivo

Exception in thread "main"
java.lang.ArithmeticException: divide by zero

Agora o método sempre fecha o arquivo!
O método `close()` da classe `BufferedReader` é automaticamente chamando ao final do bloco `try`.

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

29

Qual forma usar?

- A estrutura *try-with-resources* é mais confiável, pois não há risco de que algum recurso seja esquecido.
- Por outro lado somente recursos providos por classes que implementem a *interface* `AutoCloseable` funcionam com essa estrutura.
- O *try-with-resources* também dificulta a leitura do código, pois todos os recursos utilizados devem ser alocados na própria linha do `try`.
- Em termos de funcionamento as estruturas *try-with-resources* e *try-catch-finally* são equivalentes. Use a que for mais confortável.

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

30

Muito obrigado a todos.

Copyright © 2018 Prof. Ricardo Drudi

Todos direitos reservados. A reprodução ou divulgação total ou parcial deste documento é expressamente proibida sem o consentimento formal, por escrito, do professor Ricardo Drudi.

Aplicações de Linguagem de Programação Orientada a Objetos
Tratamento de Exceções

Prof. Ricardo Drudi

31