

# SCC0504 – Programação Orientada a Objetos

## Exceções

Luiz Eduardo Virgilio da Silva  
ICMC, USP

**Parte do material foi obtido com os professores:**  
José Fernando Junior (ICMC/USP)



# Sumário

- Introdução
- Tipos de exceção
- Capturando e tratando exceções
- Repassando exceções
- Criando novos tipos de exceção
- Vantagens

# Introdução

- As linguagens modernas de programação possuem um poderoso esquema para tratamento de erros baseado em **tratamento de exceções**
- Existem erros que são detectáveis no momento da **compilação**, por falha do programador
  - Compile-time errors
- Exemplos
  - Associar valores a variáveis de tipos diferentes
  - Invocar um método inexistente
  - Tentar acessar um campo com acesso restrito
  - Tentar acessar uma variável de instância em um método estático
  - ...

# Introdução

- Há erros, porém, que podem surgir durante a **execução** do programa
  - Runtime errors
- Exemplos
  - Extrapolar limites de um vetor
  - Divisão por zero
  - Casting explícito inadequado
  - Tentar acessar um arquivo que não existe
  - Tentar gravar em um disco cheio
  - Tentar acessar um diretório que não há permissão
  - ...

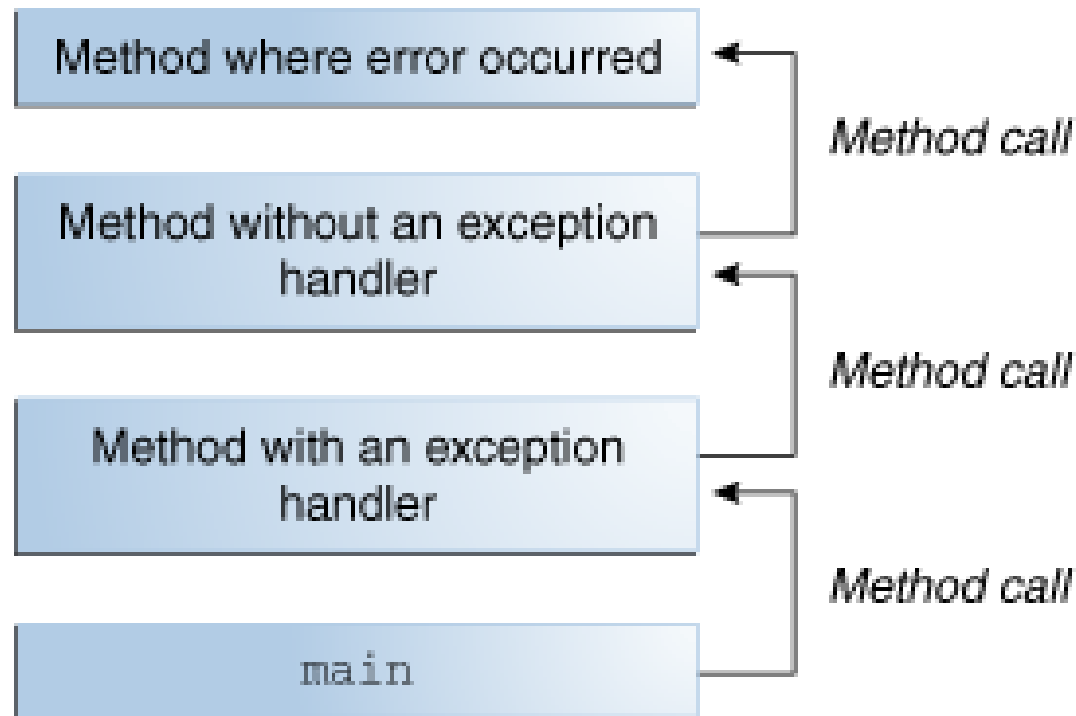
# Introdução

- O sistema de tratamento de exceções se refere aos erros em **tempo de execução**
- Toda vez que um erro ocorre, uma exceção (objeto) é criada e lançada
- A exceção (objeto) encapsula as informações do erro
- Essa exceção deve ser capturada em algum momento
  - Cascata de chamadas de métodos
  - Bloco try-catch, try-catch-finally

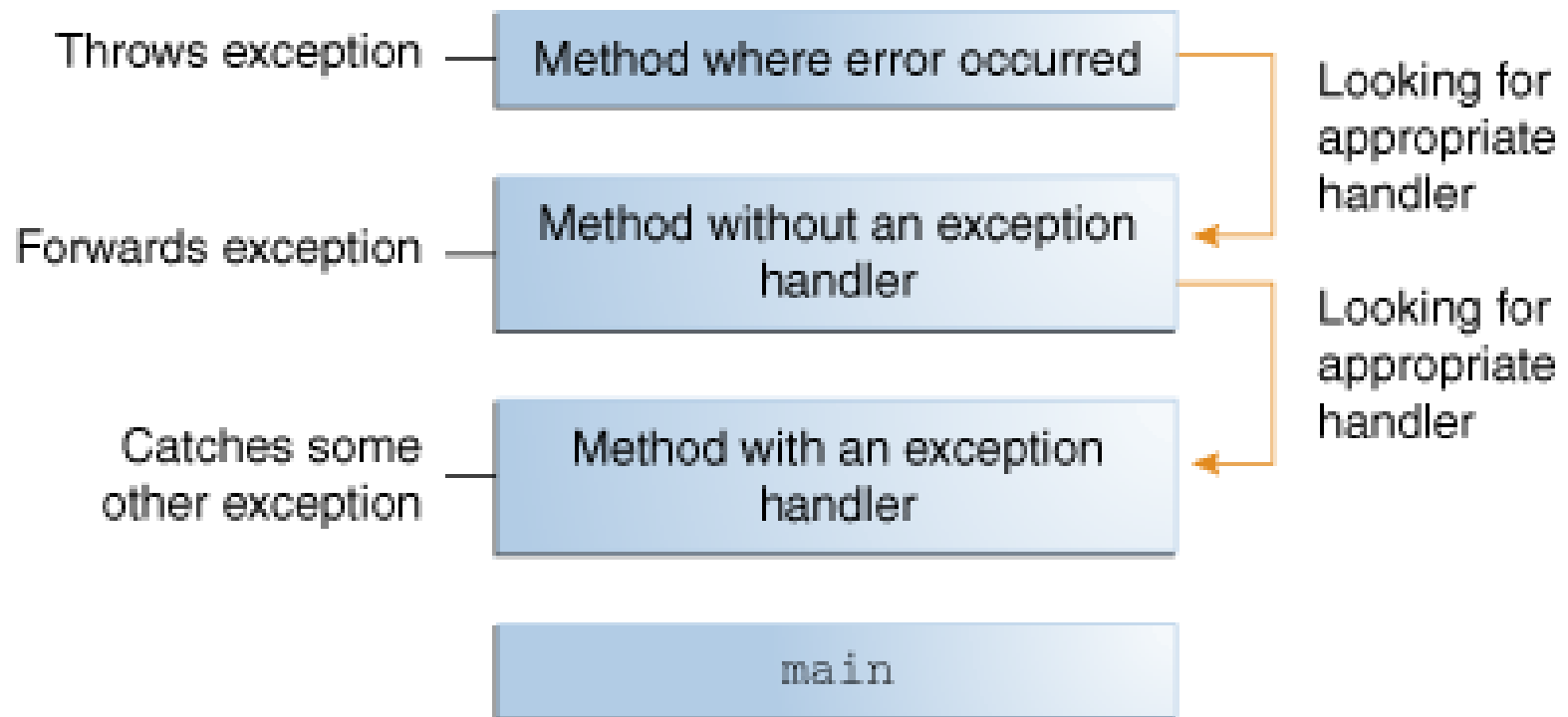
# Introdução

- Quando surge um erro na execução de um método, o método cria um objeto de exceção e passa para o sistema de *runtime* do java (**lança** a exceção)
- A execução do programa para naquele ponto e o *runtime* procura o trecho de código mais próximo (stack) capaz de **capturar** uma exceção daquele tipo
- Se a busca chegar ao *main* e não encontrar quem capture a exceção, o programa termina

# Introdução



# Introdução





# Exceções

- Todo código em Java que pode gerar exceções **deve** lidar com essas exceções de duas possíveis maneiras
  - Declarando um bloco **try-catch** dentro do método, para tratar a exceção
  - Declarar que o método lança (**throws**) os tipos de exceção que o código gera, delegando o tratamento para outro nível na pilha (repassa o erro)
- Se nenhuma das duas alternativas for implementada, o código não compila
- Nem todas as exceções devem ser obrigatoriamente tratadas (lançadas)
  - Depende do tipo

# Exceções

- Existem três tipos básicos de objetos de exceção em Java, da qual todas as outras exceções são derivadas
  - Exception (*checked*)
  - RuntimeException (*unchecked*)
  - Error (*unchecked*)
- Essas três classes derivam da classe **Throwable**

# Exceções

- Exceções do tipo **Exception** (*checked*) **precisam ser tratadas**, pois são situações excepcionais em um programa que podem ser contornadas
  - Ex: abertura de um arquivo cujo nome foi informado errado pelo usuário
  - Abertura do arquivo falha, gerando uma exceção que pode ser contornada pelo programador
    - Ex: Perguntar o nome novamente

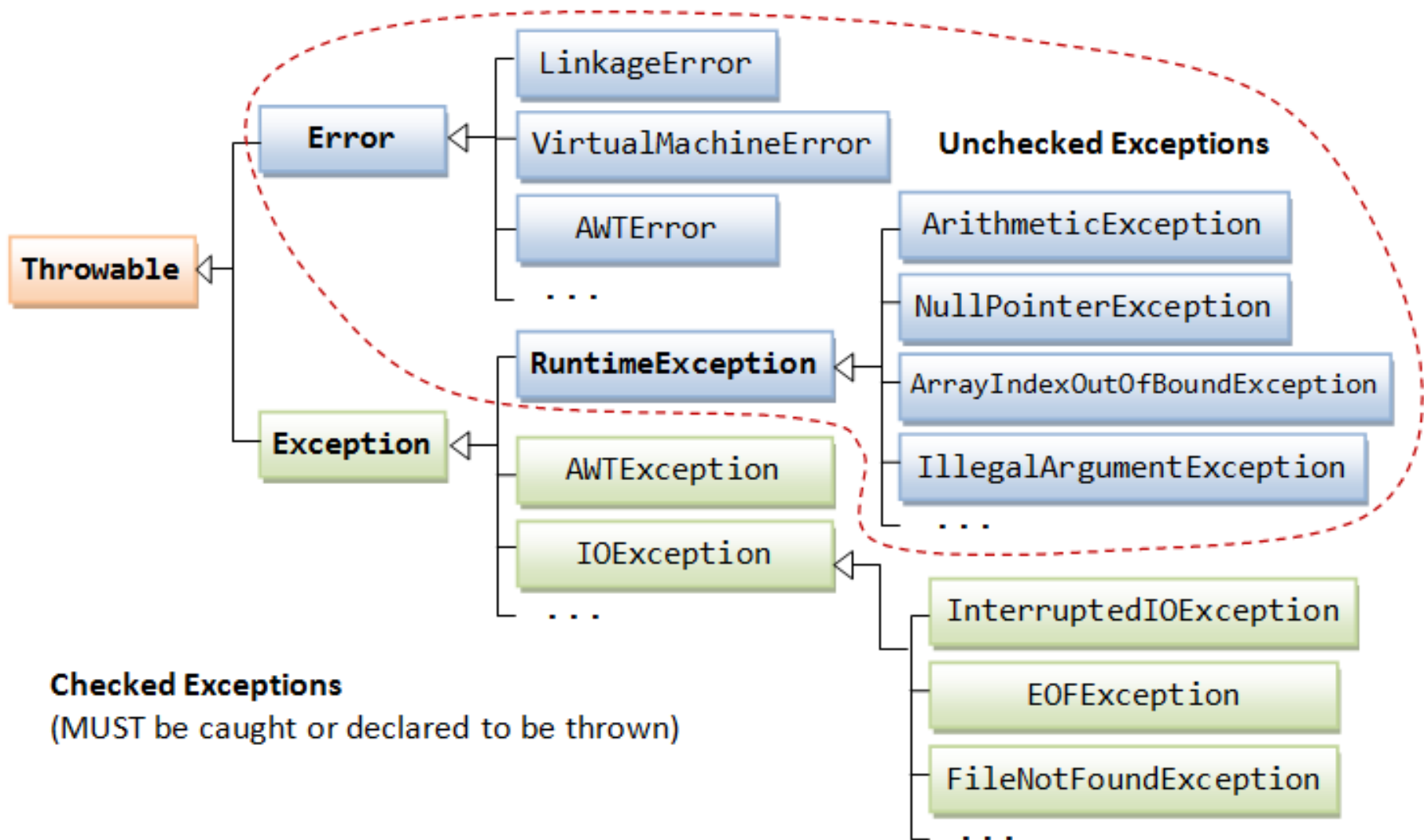
# Exceções

- Exceções do tipo **Error** (*unchecked*) representam erros **externos** à aplicação, que não são contornáveis pelo programador
  - Ex: falha na leitura de um arquivo por um problema de hardware
  - Programador não pode resolver
- A aplicação pode capturar esse erro se quiser notificar ao usuário
  - Não é obrigatório (*unchecked*)
  - Neste caso, é aceitável que o Java apenas imprima uma mensagem de erro e termine o programa

# Exceções

- Exceções do tipo **RuntimeException** (*unchecked*) representam erros **internos** na aplicação, mas que em geral não são tratáveis pelo programador
  - Erros na lógica de programação
  - Uso incorreto da API Java
  - Ex: por algum motivo, o programa lê o nome de um arquivo do usuário mas passa **null** para o construtor do arquivo
- Nesse caso, a exceção gerada pode ser capturada pelo programador
  - Porém, faz mais sentido que o código seja corrigido

# Exceções



# Capturando e Tratando Exceções

```
import java.io.*;
import java.util.List;
import java.util.ArrayList;

public class ListOfNumbers {
    private List<Integer> list;
    private static final int SIZE = 10;

    public ListOfNumbers () {
        list = new ArrayList<Integer>(SIZE);
        for (int i = 0; i < SIZE; i++)
            list.add(new Integer(i));
    }

    public void writeList() {
        PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++)
            out.println("Value at: " + i + " = " + list.get(i));

        out.close();
    }
}
```

# Capturando e Tratando Exceções

```
import java.io.*;
import java.util.List;
import java.util.ArrayList;

public class ListOfNumbers {
    private List<Integer> list;
    private static final int SIZE = 10;

    public ListOfNumbers () {
        list = new ArrayList<Integer>(SIZE);
        for (int i = 0; i < SIZE; i++)
            list.add(new Integer(i));
    }

    public void writeList() {
        PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++)
            out.println("Value at: " + i + " = " + list.get(i));

        out.close();
    }
}
```

IOException  
(checked)

IndexOutOfBoundsException  
(unchecked)



# Capturando e Tratando Exceções

```
import java.io.*;
import java.util.List;
import java.util.ArrayList;

public class ListOfNumbers {
    private List<Integer> list;
    private static final int SIZE = 10;

    public ListOfNumbers () {
        list = new ArrayList<Integer>(SIZE);
        for (int i = 0; i < SIZE; i++)
            list.add(new Integer(i));
    }

    public void writeList() {
        PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++)
            out.println("Value at: " + i + " = " + list.get(i));

        out.close();
    }
}
```

Não compila!

IOException  
(checked)

IndexOutOfBoundsException  
(unchecked)

# Capturando e Tratando Exceções

- Para capturar e tratar exceções, usamos um bloco do tipo `try-catch` ou `try-catch-finally`

```
try {  
    // code  
} catch (Expt e) {  
    // code  
}
```

```
try {  
    // code  
} catch (Expt e) {  
    // code  
} finally {  
    // code  
}
```

- Dentro de `try` fica todo o código que pode gerar uma exceção

# Capturando e Tratando Exceções

- Quando uma exceção é lançada, o fluxo do programa é interrompido naquele instante
  - Se for dentro de um `try`, a execução é deslocada para o bloco `catch` que captura aquele tipo de exceção
  - Se não houver bloco `try-catch` é porque o método lança aquele tipo de exceção
  - Neste caso, o método retorna levando a exceção para quem o chamou
- Se não houver exceção, apenas o bloco `try` é executado
  - Pula os blocos `catch` e prossegue o programa

# Capturando e Tratando Exceções

- Associado a um bloco `try`, podem existir vários blocos `catch`, responsável por capturar diferentes tipos de exceção
  - Cada tipo *ExceptionType* é o nome de uma classe filha de *Throwable* (em qualquer nível)

```
try {  
    // code  
} catch (ExceptionType e) {  
    // code  
} catch (ExceptionType e) {  
    // code  
} catch (ExceptionType e) {  
    // code  
}
```

# Capturando e Tratando Exceções

- A partir do Java 7 é possível definir um bloco **catch** com mais de um tipo de exceção
  - Separados por uma barra vertical
  - Evita duplicação de código
  - O tipo “ex” é tratado como **Trowable** pelo programador
  - Mas é possível identificar o tipo recebido
    - **instanceof**

```
try {  
    // code  
} catch (IOException | SQLException ex) {  
    // code  
}
```

# Capturando e Tratando Exceções

- Adicionalmente aos blocos `try` e `catch`, é possível definir um bloco `finally`
  - Opcional
  - Sempre será executado, independente do desfecho
  - Em geral, para códigos de limpeza e finalização
  - Ex: fechamento de arquivos

# Capturando e Tratando Exceções

```
import java.io.*;
import java.util.List;
import java.util.ArrayList;

public class ListOfNumbers {
    private List<Integer> list;
    private static final int SIZE = 10;

    public ListOfNumbers () {
        list = new ArrayList<Integer>(SIZE);
        for (int i = 0; i < SIZE; i++)
            list.add(new Integer(i));
    }

    public void writeList() {
        PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++)
            out.println("Value at: " + i + " = " + list.get(i));

        out.close();
    }
}
```

Não compila!

IOException  
(checked)

IndexOutOfBoundsException  
(unchecked)

# Capturando e Tratando Exceções

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++)  
            out.println("Value at: " + i + " = " + list.get(i));  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " +  
            e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught IOException: " +  
            e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.println("PrintWriter not open");  
        }  
    }  
}
```



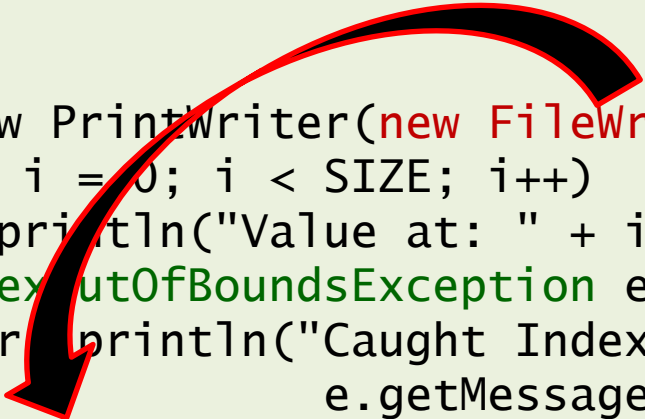
# Capturando e Tratando Exceções

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++)  
            out.println("Value at: " + i + " = " + list.get(i));  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " +  
            e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught IOException: " +  
            e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.println("PrintWriter not open");  
        }  
    }  
}
```

Se uma exceção for gerada pelo construtor de FileWriter, ele irá repassá-la.

# Capturando e Tratando Exceções

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++)  
            out.println("Value at: " + i + " = " + list.get(i));  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " +  
            e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught " +  
            e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.println("PrintWriter not open");  
        }  
    }  
}
```



A red curved arrow originates from the `list.get(i)` call in the `try` block and points to the `IndexOutOfBoundsException` catch block, illustrating the flow of an exception.

Execução do programa é transferida para o bloco que captura a exceção daquele tipo

# Capturando e Tratando Exceções

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++)  
            out.println("Value at: " + i + " = " + list.get(i));  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " +  
                            e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught IOException: " +  
                            e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.p  
        }  
    }  
}
```



A red curved arrow points from the end of the `catch (IOException e)` block to the `finally` block. A blue straight arrow points from the end of the `try` block to the `finally` block.

Ao final da execução do bloco **catch**, o bloco **finally** é chamado.

# Capturando e Tratando Exceções

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++)  
            out.println("Value at: " + i + " = " + list.get(i));  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " +  
            e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught IOException: " +  
            e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.println("PrintWriter not open");  
        }  
    }  
}
```

Como houve falha em abrir o arquivo, não é preciso fechá-lo.

# Capturando e Tratando Exceções

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++)  
            out.println("Value at: " + i + " = " + list.get(i));  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " +  
            e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught IOException: " +  
            e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.println("Unable to open PrintWriter");  
        }  
    }  
}
```

O programa continua após o **finally**



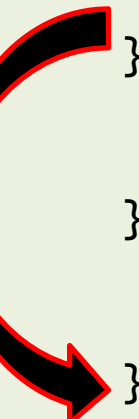
# Capturando e Tratando Exceções

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++)  
            out.println("Value at: " + i + " = " + list.get(i));  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " +  
            e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught IOException: " +  
            e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.println("PrintWriter not open");  
        }  
    }  
}
```

Se nenhuma exceção for gerada no bloco **try**, nenhum bloco **catch** é chamado.

# Capturando e Tratando Exceções

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++)  
            out.println("Value at: " + i + " = " + list.get(i));  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " +  
            e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught IOException: " +  
            e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.println("PrintWriter not open");  
        }  
    }  
}
```



Ao final da execução do bloco try, o bloco finally é chamado.

# Capturando e Tratando Exceções

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++)  
            out.println("Value at: " + i + " = " + list.get(i));  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " +  
            e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught IOException: " +  
            e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.println("PrintWriter not open");  
        }  
    }  
}
```

Como o arquivo abriu normalmente, neste caso é preciso fechá-lo.



# Capturando e Tratando Exceções

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++)  
            out.println("Value at: " + i + " = " + list.get(i));  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " +  
            e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught IOException: " +  
            e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.println("Warning: PrintWriter is null at close time.");  
        }  
    }  
}
```

O programa continua após o **finally**



# Stack Trace

- Uma informação importante que pode ser obtida com objetos de exceções é o histórico de execução
  - Nome das classes e métodos que foram chamados até ocorrer a exceção
  - Muito útil para *debug*
- É possível obter cada elemento da *stack trace* para trabalhar com ele da forma como desejarmos
  - Não trataremos aqui
- Toda exceção tem um método **printStackTrace()**
  - Pode ser chamado durante o tratamento
- Método **getMessage()** também é útil

# Repassando Exceções

- Se não quisermos capturar e tratar as exceções, podemos repassá-la na pilha de chamadas
- Para isso, o método precisa ser explicitamente declarado que lança (**throws**) um ou mais tipos de exceções
  - Obrigatório para exceções do tipo *checked*

# Repassando Exceções

- Método sem tratamento de exceções (não compila)

```
public void writeList() {  
    PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));  
    for (int i = 0; i < SIZE; i++)  
        out.println("Value at: " + i + " = " + list.get(i));  
  
    out.close();  
}
```

- Método com repasse de exceções (compila)

```
public void writeList() throws IOException {  
    PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));  
    for (int i = 0; i < SIZE; i++)  
        out.println("Value at: " + i + " = " + list.get(i));  
  
    out.close();  
}
```

# Lançando Exceções

- Antes de capturarmos ou repassarmos uma exceção, ela precisa ser criada em algum lugar
- Qualquer código pode criar uma exceção e lançá-la através do comando **throw**
  - Objetos lançáveis são instâncias da classe *Throwable*

```
public Object pop() {  
    Object obj;  
    if (size == 0) {  
        throw new EmptyStackException(); // unchecked  
    }  
    obj = objectAt(size - 1);  
    setObjectAt(size - 1, null);  
    size--;  
    return obj;  
}
```

# Criando Classes de Exceção

- A API do Java provê uma grande quantidade de classes de exceção que podem ser usadas
- Porém, pode ser que em um projeto nenhuma dessas classes represente o tipo de exceção que pode surgir
- Neste caso, podemos definir novas classes de Exceção
  - Em geral, herdam da classe **Exception** (*checked*)
  - Por convenção, o nome da classe deve terminar com Exception
    - `InvalidDateException`

# Criando Classes de Exceção

```
public class InvalidDateException extends Exception {  
    public InvalidDateException () { }  
  
    public InvalidDateException (String msg) {  
        super(msg);  
    }  
}
```

```
public class Date {  
    public void setDate(int day, int month, int year) throws  
        InvalidDateException {  
        if (day < 1 || day > 31)  
            throw new InvalidDateException("Invalid day.");  
        else if (month < 1 || month > 12)  
            throw new InvalidDateException("Invalid month.");  
    }  
}
```

# Criando Classes de Exceção

```
public class Main {  
    public static void main(String[] args) {  
        try {  
            Date d = new Date();  
            d.setDate(35,4,2015);  
            System.out.println("Date successfully created!");  
        } catch (InvalidDateException e) {  
            System.out.println(e.getMessage());  
            e.printStackTrace();  
        }  
    }  
}
```



# Vantagens

- Vantagens do uso de exceções
  - Separação entre tratamento de erro e código normal

```
int readFile {  
    int errorCode = 0;  
    open the file;  
  
    if (theFileIsOpen) {  
        determine the length of the file;  
        if (gotTheFileLength) {  
            allocate that much memory;  
            if (gotEnoughMemory) {  
                read the file into memory;  
                if (readFailed)  
                    errorCode = -1;  
            } else {  
                errorCode = -2;  
            }  
        } else {  
            errorCode = -3;  
        }  
        ...  
    }
```

# Vantagens

- Vantagens do uso de exceções
  - Separação entre tratamento de erro e código normal
    - Código normal → **try**
    - Tratamento de erro → **catch**
    - Não há **if-else**, o que torna o código mais limpo e legível

# Vantagens

- Vantagens do uso de exceções
  - Propagação do erro de forma implícita
    - ❑ Métodos não precisam retornar um tipo (inteiro, por exemplo) para sinalizar o erro
    - ❑ Repasse da exceção é automático
    - ❑ Código mais limpo
    - ❑ Cláusula **throws** apenas

```
method1 {  
    int error;  
    error = call method2;  
    if (error)  
        doErrorProcessing;  
    else  
        proceed;  
}
```

```
int method2 {  
    int error;  
    error = call method3;  
    if (error)  
        return error;  
    else  
        proceed;  
}
```

```
int method3 {  
    int error;  
    error = call readFile;  
    if (error)  
        return error;  
    else  
        proceed;  
}
```

# Vantagens

- Vantagens do uso de exceções
  - Agrupamento e diferenciação dos tipos de erros
    - Como as exceções são objetos, podemos desfrutar dos conceitos de POO
    - **IOException** consegue capturar todas as exceções I/O
      - FileNotFoundException, EOFException, ...
    - Porém, é mais interessante definir blocos **catch** específicos para cada erro
    - **Exception** captura todos as exceções *checked*
    - Se um bloco **catch** para tratamento geral das exceções for criado, este deve ser o último da lista

# Vantagens

- Vantagens do uso de exceções
  - Agrupamento e diferenciação dos tipos de erros

```
try {  
    // code  
} catch (NotSerializableException e) {  
    // error handling  
} catch (FileNotFoundException e) {  
    // error handling  
} catch (EOFException e) {  
    // error handling  
} catch (IOException e) {  
    // error handling  
} catch (Exception e) {  
    // error handling  
}
```

# Resumo

- Introdução
- Tipos de exceção
- Capturando e tratando exceções
- Repassando exceções
- Criando novos tipos de exceção
- Vantagens

# Dúvidas?



404 - Not Found