53 45 52 45 49 20 46 49 45 4c 20
41 4f 53 20 50 52 45 43 45 49 54
4f 53 20 44 41 20 48 4f 4e 52 41
20 45 20 44 41 20 43 49 c3 8a 4e
43 49 41 2c 20 50 52 4f 4d 4f 56
45 4e 44 4f 20 4f 20 55 53 4f 20
45 20 4f 20 44 45 53 45 4e 56 4f
4c 56 49 4d 45 4e 54 4f 20 44 41
20 49 4e 46 4f 52 4d c3 81 54 49
43 41 20 45 4d 20 42 45 4e 45 46
c3 8d 43 49 4f 20 44 4f 20 43 49
44 41 44 c3 83 4f 20 45 20 44 41
20 53 4f 43 49 45 44 41 44 45 2e

RESIDÊNCIA DE

SOFTWARE

CAPACITAR
TREINAR
EMPREGAR

TRANSFORMAR

Firjan SENAI

serratec
Parque Tecnológico Região Serrana

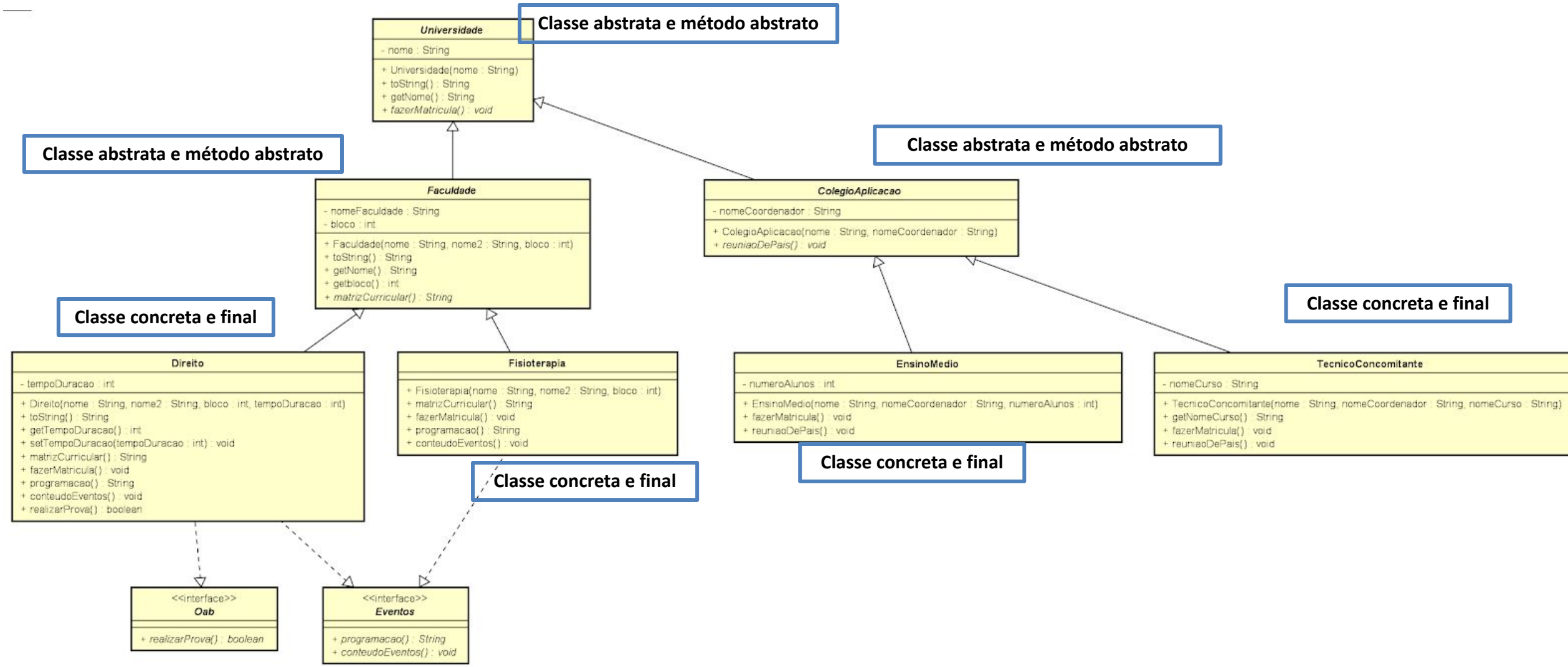
JAVA I
Tratamento de Erros

06/08/2020

RECAPTULANDO

- Java
  - Introdução
  - Condições
  - Repetições
  - Operadores
  - Encapsulamento
  - Classes e objetos
  - Construtores
  - Modificadores de Acesso
  - Herança
  - Classes abstratas
  - Intefaces
  - Enum
  - Static
  - Final

REVISÃO DE CONCEITOS DAS AULAS ANTERIORES



EXCEÇÃO

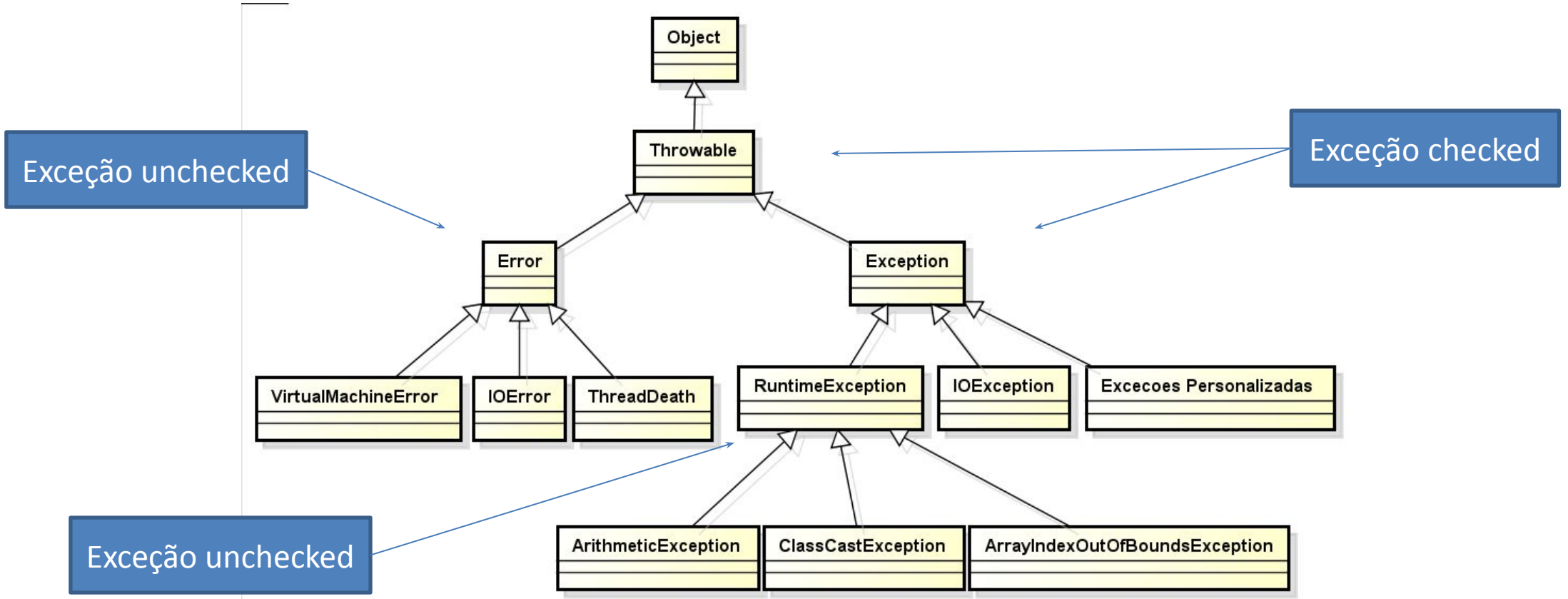
- Uma exceção representa uma situação que normalmente não ocorre e representa algo de estranho ou inesperado no sistema.
  - Exemplo:
    - Divisão por zero
    - Referência a nulo
    - Acesso a uma posição do vetor que não existe

EXCEÇÃO

- Uma exceção representa uma situação que normalmente não ocorre e representa algo de estranho ou inesperado no sistema.
  - Podem ocorrer:
    - Por erros de lógica
    - Acesso a um recurso que não esteja disponível ou não existe.
  - Exemplo:
    - Divisão por zero
    - Abrir arquivo ou banco de dados que não existe
    - Acesso a uma posição do vetor que não existe

EXCEÇÃO

- A classe mãe de todas as exceções é a **Throwable**. Apenas objetos dessa classe ou de suas subclasses podem ser gerados, propagados e capturados através de exceções.



EXCEÇÃO

- **Classe Error:** A classe Error e suas descendentes representam situações anormais que poderiam acontecer na máquina virtual.

- **Classe Exception:** A classe Exception e suas descendentes representam situações não comuns que podem ocorrer durante a execução de um programa.

- **Classe RuntimeException:** Esse tipo de exceção é conhecido como não verificada (unchecked). Sendo assim, não é requisito declarar uma cláusula try/catch.

TRATAMENTO DE EXCEÇÃO

- Processo de gerenciar as exceções que podem ocorrer no programa.
  - Objetivo: Evitar que o programa aborte inesperadamente ou informar ao usuário sobre algum problema na execução
  - Em linguagens antigas retornava-se um código de erro e de acordo com o número de erro poderia ser consultado qual erro ocorreu
  - As linguagens modernas já possuem comandos para tratamento dentro da própria linguagem

MANIPULANDO EXCEÇÃO

- Existem algumas formas de manipular as exceções. Vamos começar pelo throw.

THROW

DivisaoPorZero.java

```
package aula;

public class DivisaoPorZero {
    public static int divisao(int i, int j){
        return i/j;
    }

    public static void main(String[] args) {
      System.out.println(divisao(20,0));
    }
}
```

Console

```
<terminated> DivisaoPorZero [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (24/05/2012 09:5
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at aula.DivisaoPorZero.divisao(DivisaoPorZero.java:5)
        at aula.DivisaoPorZero.main(DivisaoPorZero.java:9)
```

MANIPULANDO EXCEÇÃO

- A função da chamada **throw** é lançar uma exceção assim que a mesma ocorre, deixando a cargo do método tratar possíveis erros de programação vindos da passagem errada de parâmetros.

DivisaoPorZero.java

```
package aula;

public class DivisaoPorZero {
    public static int divisao(int i, int j){
        if (j == 0)
         throw new ArithmeticException("Erro !! Você dividiu um número por zero");
        return i/j;
    }

    public static void main(String[] args) {
      System.out.println(divisao(20,0));
    }
}
```

A Exceção precisa ser criada com **new** e lançada com **throw**

THROWS

- O **throws** repassa o erro para o método que executou a chamada neste caso o main. Esse procedimento evita que erros não tratados causem danos futuros ao sistema.

DivisaoPorZero.java

```
package aula;

public class DivisaoPorZero {
    public static int divisao(int i, int j) throws ArithmeticException {
        return i / j;

    }

    public static void main(String[] args) {
        System.out.println(divisao(20, 0));
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at aula.OutroTesteDivisao.divisao(OutroTesteDivisao.java:6)
        at aula.OutroTesteDivisao.main(OutroTesteDivisao.java:11)
```

A Exceção deverá ser tratada por um try/catch

O main no exemplo acima não tratando o erro, faz com que a JVM libere como saída o erro **padrão**. Para corrigir este problema é necessário a utilização do **throws** em conjunto com as cláusulas **try/catch**.

A função do **try/catch** é criar um procedimento de tentativa e erro. Tudo que estiver dentro do corpo do **try** será executado como tentativa, caso um erro for detectado, uma cláusula **catch** é executada referenciando-se ao erro definido pela cláusula **throws**.

THROWS

- O **try/catch** do método main trata possíveis erros à chamada para o método **divisao**. Como o **throws** faz referência à exceção **ArithmeticException**, o **catch** foi criado para exibir uma mensagem para esta exceção.

DivisaoPorZero.java

```
package aula;

public class DivisaoPorZero {
    public static int divisao(int i, int j) throws ArithmeticException {
        return i / j;

    }

    public static void main(String[] args) {
        try {
            System.out.println(divisao(20, 0));
        } catch (ArithmeticException a) {
            System.out.println("Erro !! Você dividiu um número por zero");
        }
    }
}
```

- Como são tratadas no Java?
  - Bloco try/catch
  - São tratadas como objetos que de acordo com sua classe sabemos a exceção que ocorreu

```
public void nomeMetodo() {
    try {
        // bloco onde pode ocorrer a excecao
    }catch(Exception e) {
        // bloco onde é feito o tratamento, caso ocorra a excecao
    }
}
```

TRATAMENTO DE EXCEÇÃO

- E quando precisamos tratar mais de uma exceção?
  - Podemos inserir mais de um bloco *catch*

```
public void nomeMetodo() {
    try {
        // bloco onde pode ocorrer a excecao
    }catch(ArithmeticException ex) {
        // bloco onde é feito o tratamento para excecoes aritmeticas
    }catch(Exception e) {
        // bloco onde é feito o tratamento de outras excecoes
    }
}
```

- E se quisermos executar algo caso ocorra ou não a exceção?
  - Adicionamos o bloco *finally* após o *catch*

```
public void nomeMetodo() {
    try {
        // bloco onde pode ocorrer a excecao
    }catch(ArithmeticException ex) {
        // bloco onde é feito o tratamento para excecoes aritmeticas
    }catch(Exception e) {
        // bloco onde é feito o tratamento de outras excecoes
    }finally {
        //Comandos finais a executar independente de ter ocorrido excecao
        //O bloco finally é opcional
    }
}
```

EXERCÍCIO

1- Crie uma classe com o método *main* que faça a divisão entre dois números e apresente o resultado. Faça o tratamento caso o denominador seja zero.

```
public class CalculaDivisao {
    public static void main(String[] args) {
        int a = 10;
        int b = 0;
        int c = 0;
        try {
            c=a/b;
            System.out.println("Resultado: "+c);
        }catch (Exception e) {
            System.out.println("Não foi possivel calcular a divisão");
            System.out.println("Erro: "+e.getMessage());
        }
    }

}
```

ALGUMAS EXCEÇÕES COMUNS

- *NullPointerException:* Ocorre quando o Java tenta acessar um objeto na memória que ainda não foi inicializado. O objeto não tem um valor definido e está nulo
- *AritmeticException:* Este tipo de exceção ocorre quando alguma operação aritmética é inválida e a mesma gera uma exceção, operações essas que não podem ser resolvidas, como é o caso da divisão por zero.
- Array*IndexOutOfBoundsException:* ocorre sempre que você tenta especificar um índice inválido do seu array

EXERCÍCIO

2- Crie uma classe com o método *main* que pegue uma palavra ou frase e coloque as letras em maiúsculo. Crie uma variável String que recebe null e tente chamar o métodos *toUpperCase().*

```
public class TransformaMaiusculo {
    public static void main(String[] args) {
        String fraseEntrada = null;
        String fraseSaida = null;
        try {
            fraseSaida = fraseEntrada.toUpperCase();
            System.out.println("Resultado: "+fraseSaida);
        }catch (Exception e) {
            System.out.println("Falha ao transformar frase");
            System.out.println("Erro: "+e.getMessage());
        }
    }

}
```

Podemos alterar a classe no catch para NullPointerException para tratar a exceção especifica

MÚLTIPLAS EXCEÇÕES

- *Como tratar?*

```
public class TratandoExcecao {
    public static void main(String[] args) {
        int vetor[] = {10,3,5};
        int a = 10;
        int b = 0;
        int c = 0;
        String str = null;
        try {
            c=a/b;
            System.out.println("Resultado: "+c);
            System.out.println("Resultado: "+vetor[3]);
            System.out.println("Resultado: "+str.equals("xpto"));
        }catch (ArithmeticException e) {
            System.out.println("Falha ao realizar divisao");
            System.out.println("Erro: "+e.getMessage());
        }catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Falha ao acessar o indice do vetor");
            System.out.println("Erro: "+e.getMessage());
        }catch (Exception e) {
            System.out.println("Falha no programa");
            System.out.println("Erro: "+e.getMessage());
        }
    }
}
```

Criamos um bloco para cada exceção que pode ocorrer

OBS.:Devemos deixar o bloco catch que contém a classe de exceção mais específica no início

EXCEÇÕES CHECKED E UNCHECKED

- As **checked exceptions** são exceções que devem ser tratadas pelo programa usando try/catch ou delegadas através da clausula throws. O compilador checará se a exceção está sendo devidamente tratada.

Arquivo.java

```
package aula;

import java.io.FileReader;

public class Arquivo {
    public static void main(String[] args) {
        FileReader pw = new FileReader(aula14.txt);
    }
}
```

A Classe FileReader está no pacote java.io temos que importar usando a combinação CTRL+SHIFT+O

Arquivo.java

```
package aula;

import java.io.FileNotFoundException;
import java.io.FileReader;

public class Arquivo {
    public static void main(String[] args) throws FileNotFoundException {
        FileReader pw = new FileReader("aula14.txt");
    }
}
```

O código acima não irá compilar, o compilador avisa que temos que tratar uma exceção FileReader clique no quick fix do eclipse que ele fará a implementação. Teremos que tratar futuramente com try/catch

EXCEÇÕES CHECKED E UNCHECKED

- Podemos também tratar no próprio método sem passar para frente.

Arquivo.java

```
package aula;

import java.io.FileNotFoundException;
import java.io.FileReader;

public class Arquivo {
    public static void main(String[] args) {
        try {
            FileReader pw = new FileReader("aula14.txt");
        } catch (FileNotFoundException f) {
            System.out.println("Arquivo nao existe");
        }

    }
}
```

EXCEÇÕES CHECKED E UNCHECKED

- As **unchecked exceptions** não exigem nenhum tratamento por parte do programador apesar de poderem ser tratadas. O compilador não checa se as exceções estão sendo tratadas. Todas as classes que são filhas de RuntimeException não precisam ser tratadas, assim como aquelas que são filhas da classe Error.

EXEMPLO

*ContaCorrente.java

```java
package aula;

public class ContaCorrente implements Conta {
    private String numero;
    private double saldo;

    public ContaCorrente(String numero, double saldo) {
        this.numero = numero;
        this.saldo = saldo;
    }

    public double getSaldo() {
        return saldo;
    }

    @Override
    public String toString() {
        return "Conta:" + numero + " Saldo:" + saldo;
    }

    @Override
    public void depositaConta(double valor) {
        this.saldo += valor;
    }

    @Override
    public boolean saqueConta(double valor) {
        if (this.saldo < valor) {
            return false;
        } else {
            this.saldo -= valor;
            return true;
        }
    }
}
```

Conta.java

```java
package aula;

public interface Conta {
        public boolean saqueConta(double valor);
        public void depositaConta(double valor);

}
```

TestaConta.java

```java
package aula;

public class TestaConta {
    public static void main(String[] args) {
        Conta cc = new ContaCorrente("23334-98", 2300.);

        cc.depositaConta(1000);
        if (!cc.saqueConta(400))
            System.out.println("Saque nao efetuado");
        System.out.println(cc.toString());
    }
}
```

EXEMPLO

Utilizando uma exceção **unchecked** podemos passar o erro para que seja tratado pelo método que o chamar. Altere o método **saqueConta**

```
public boolean saqueConta(double valor) {
    if (this.saldo < valor) {
        throw new RuntimeException();
    } else {
        this.saldo -= valor;
        return true;
    }
}
```

Alterar o TesteConta

```
package aula;

public class TestaConta {
    public static void main(String[] args) {
        Conta cc = new ContaCorrente("23334-98", 2300.);

        cc.depositaConta(1000);
        cc.saqueConta(15400);
        System.out.println(cc.toString());
    }
}
```

RuntimeException é a exceção mãe de todas as exceptions unchecked sendo muito genérica. Devemos usar uma exceção mais especifica como a **IllegalArgumentException**. Ela é uma **Exception** unchecked pois estende de RuntimeException. Utilizamos esta exceção quando um argumento sempre é inválido como, por exemplo, números negativos, referências nulas e outros.

```
throw new IllegalArgumentException("Saldo negativo");
```

CRIANDO NOSSAS EXCEÇÕES

SaldoNegativo.java

```
package aula;

public class SaldoNegativo extends RuntimeException {
    public SaldoNegativo(String message) {
        super(message);
    }
}
```

Altera o que esta em destaque na **ContaCorrente**

```
public boolean saqueConta(double valor) {
    if (this.saldo < valor) {
        throw new SaldoNegativo("Saldo Negativo");
    } else {
        this.saldo -= valor;
        return true;
    }
```

Alterar o que esta em destaque na classe **TestaConta**

TestaConta.java

```
package aula;

public class TestaConta {
    public static void main(String[] args) {
        Conta cc = new ContaCorrente("23334-98", 2300.);

        cc.depositaConta(1000);
        try{
            cc.saqueConta(15400);
        } catch (SaldoNegativo s) {
            System.out.println(s.getMessage());
        }
        System.out.println(cc.toString());
    }
}
```

EXERCÍCIO

Faça um programa que tenha dois vetores de tamanhos diferentes do tipo int e inicie eles com os valores (1,2,3,4,5) e (3,0,2). Depois disso crie um for para percorrer os dois vetores fazendo a divisão do primeiro numero do primeiro vetor com o primeiro numero do segundo vetor e assim por diante. Observe o erro que ocorre.

```
public class Main {
    public static void main(String[] args) {
        int[] numerador = {1,2,3,4,5};
        int[] denominador = {3,0,2};

        for (int i = 0; i < numerador.length; i++) {
            System.out.println("dividindo: "+numerador[i]+"/"+denominador[i]+" = "+ (numerador[i]/denominador[i]));
        }
    }
}
```

EXERCÍCIO

No exercício anterior, faça o tratamento das possíveis exceções que possa ocorrer para que o programa não seja interrompido

```
public class Main {
    public static void main(String[] args) {
        int[] numerador = {1,2,3,4,5};
        int[] denominador = {3,0,2};

        for (int i = 0; i < numerador.length; i++) {
            try {
                System.out.println("dividindo: "+numerador[i]+"/"+denominador[i]+" = "+ (numerador[i]/denominador[i]));
            }catch (ArithmeticException e) {
                System.out.println("Ocorreu um problema na divisao");
            }catch(ArrayIndexOutOfBoundsException e) {
                System.out.println("Ocorreu um problema ao acessar um indice do vetor que nao existe");
            }
        }
    }
}
```

EXERCÍCIO

Faça um programa que leia um número inteiro. Caso o usuário digite outro tipo de número ou caractere ocorra o tratamento da exceção e apresente a mensagem ”Falha ao ler o número”. E em caso de sucesso, apresente o número digitado.

```
public class Main {

public static void main(String[] args) {

        Main m = new Main();
        System.out.println("Digite um numero inteiro:");
        try {
            double num = m.lerNumero();
            System.out.println("Voce digitou: "+ num);
        } catch (Exception e) {
            System.out.println("Falha ao ler seu numero");
        }
    }

    public double lerNumero() throws Exception{
        Scanner in = new Scanner(System.in);
        double num = in.nextDouble();
        in.close();
        return num;
    }
}
```