CENTRO PAULA SOUZA
COMPETÊNCIA EM EDUCAÇÃO PÚBLICA PROFISSIONAL
40 ANOS

GOVERNO DE SÃO PAULO

Capacitação técnica em informática

Analise e desenvolvimento de aplicações orientadas a objeto com Java SE

Objetivo

- Capacitar docentes do Centro Paula Souza a ministrarem as disciplinas DSII e DSIII
- Conteúdo:
  - Desenvolvimento de softwares orientado a objetos
  - Linguagem de apoio: Java SE (JDK 6 Update 13)
  - IDE: Eclipse 3.4.2
  - Sistema operacional: Microsoft Windows XP/Vista

Tratamento de erros

- Na execução de um programa podem ocorrer erros que podem tanto causar a interrupção do programa quanto produzir resultados incorretos ou comportamento inesperado.

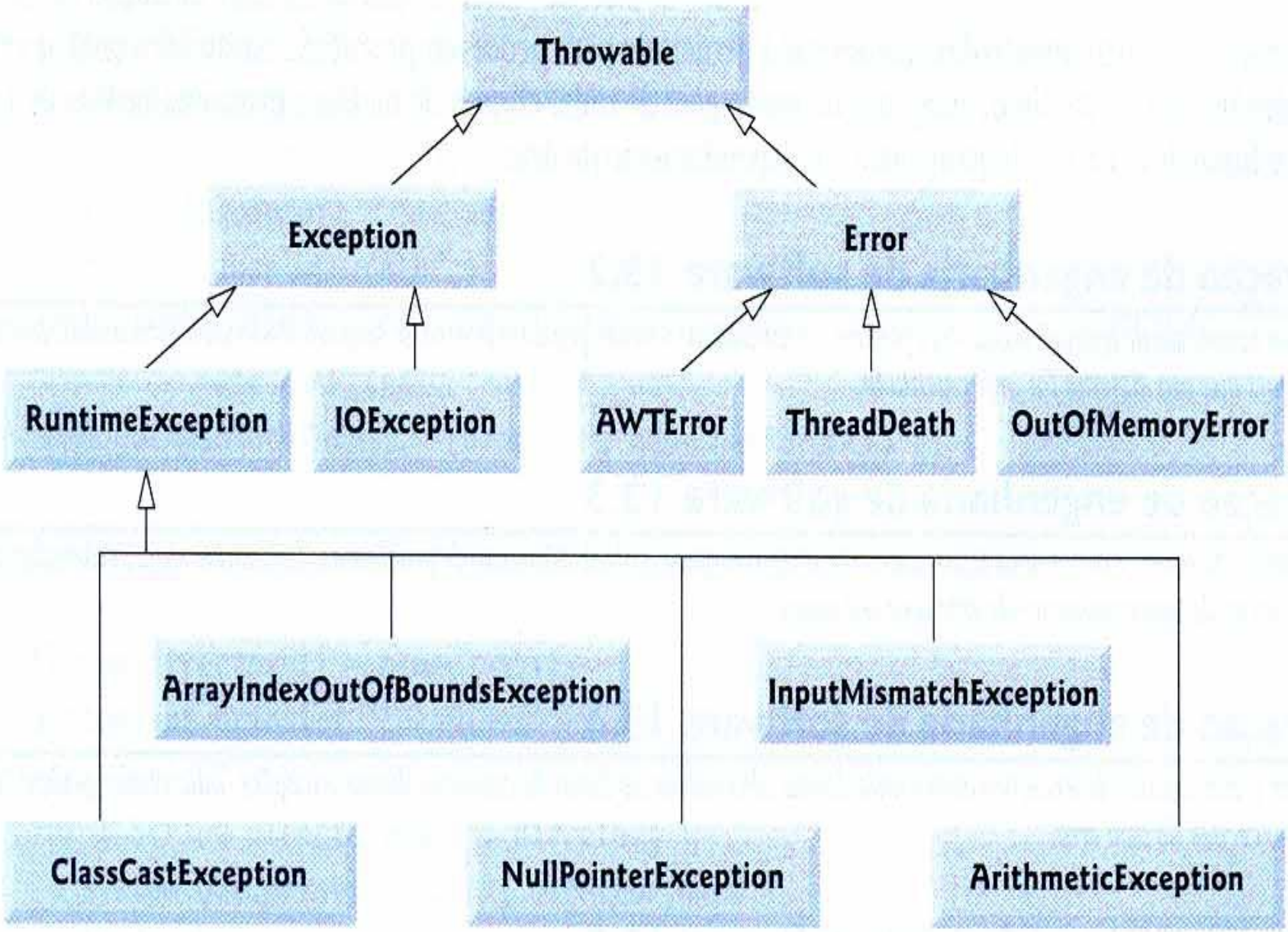
- Existem erros de lógica e erros de execução.

Tratamento de erros

- Erros de lógica.
  - Estes erros se apresentam no desenvolvimento de um algoritmo não apropriado para solucionar o problema que se propõe. Estes erros não necessariamente causam interrupção na execução do programa.
- Erros de execução.
  - São erros mais específicos em relação aos lógicos, que decorrem de uma operação inválida e causam interrupção na execução do programa.
  - Ao se causar um erro a execução é interrompida e é enviado um sinal ao programa indicando que a operação não pode ser realizada.

Tratamento de erros

- As linguagens de programação possuem formas para identificar e tratar os erros de execução.
- Em Java eles são detectados pela JVM e um objeto de uma classe que caracteriza o erro é criada.
- Notifica-se o programa da operação inválida e caso seja possível tratá-lo, pode-se acessar o objeto que caracteriza o erro.
- Os erros são caracterizados por objetos de classes específicas que pertencem a hierarquia da classe Throwable.
- Vejamos a seguir uma parte da familia Throwable.


Throwable
Exception
Error
RuntimeException
IOException
AWTError
ThreadDeath
OutOfMemoryError
ArrayIndexOutOfBoundsException
InputMismatchException
ClassCastException
NullPointerException
ArithmeticException

Tratamento de erros em Java

- A classe Throwable é a superclasse da classe Exception e, portanto, também é a superclasse de todas as exceções.
- Somente objetos Throwable podem ser utilizados como mecanismo de tratamento de exceções.
- A classe Throwable tem duas subclasses:
  - Exceptions
  - Error

Classe Error

- A classe Error, com suas subclasses, é utilizada para indicar erros graves que não se espera que sejam tratados pelo programas.
- Errors raramente acontecem e não são de responsabilidade da aplicação.
- Exemplos de Errors.
  - Erros internos da JVM
  - Falta de memória

Classe Exception

- Normalmente chamamos os erros em Java de exceptions.
- Uma exceção representa uma situação que normalmente não ocorre (ou não deveria ocorrer) e representa algo de estranho ou inesperado no sistema
- O Java distingue entre duas categorias de exceções:
  - Checked (verificadas)
  - Unchecked (não verificadas)

Unchecked Exceptions

- Uma exceção não verificada é aquela em que o compilador Java não verifica o código para determinar se ela foi capturada ou declarada.
- Em outras palavras, o programador não é obrigado a inserir o tratamento de erro.
- Em geral, o programador pode impedir a ocorrência de exceções não verificadas pela codificação adequada.

Unchecked Exceptions

- Todos os tipos de exceção que são subclasses diretas ou indiretas da classe RuntimeException são exceções não verificadas.

- Exemplos de Unchecked Exceptions.
  - Entrada de tipos incompatíveis (Leitura de uma String em um atributo double, por exemplo)
  - Acesso a índice inexistente em um array.
  - Chamada a um método de um objeto nulo.

Checked Exceptions

- Uma exceção verificada (ao contrário da não verificada) é aquela em que o compilador Java verifica o código para determinar se ela foi capturada ou declarada e obriga o programador a inserir um tratamento de erro.
- Todos os tipos de exceção que herdam da classe Exception, mas não da RuntimeException, são exceções verificadas.
- Exemplos de Checked Exceptions.
  - Abrir um arquivo para leitura (onde pode ocorrer o erro do arquivo não existir).

Tratamentos de Checked Exceptions

- Existem duas formas de tratar uma Checked Exception:
  - Utilizando a cláusula throws
  - Utilizando a estrutura try-catch-finally

Throws

- O Throws delega para quem chamou a responsabilidade de tratar o erro, ou seja, a obrigatoriedade de tratamento é passada para a classe que fará a chamada ao método.

try – catch – finally

- É a principal estrutura para captura de erros em Java onde o código ira tentar (try) executar o bloco “perigoso” e, caso ocorra algum problema, o erro gerado será pego (catched).
- Um catch possui um parâmetro de exceção (identificação do erro) seguido por um bloco de código que o captura e possibilita o tratamento.
- É possível definir vários catch para cada try.
- O finally é opcional e, se for usado, é colocado depois do último catch.
- Havendo ou não uma exception (identificada no bloco try) o bloco finally sempre será executado.

try – catch – finally

```
try{

    // Bloco de código que pode gerar erro

}catch (Exception1 e) { // O objeto "e" conterá o erro
    // Tratamento da exception 01

}catch (Exception2 e) {
    // Tratamento da exception 02

}catch (Exception3 e) {
    // Tratamento da exception 03

}finally{
    // Bloco que sempre será executado
}
```

O Eclipse e as Checked Exceptions

```
public class Leitura {

    public void abrirArq() {
        new java.io.FileReader("arquivo.txt");
    }

}
```

```
public class Leitura {

    public void abrirArq() {
        new java.io.FileReader("arquivo.txt");
    }

}
```

Add throws declaration
Surround with try/catch
Assign statement to new local variable (Ctrl+2, L direct acces
Assign statement to new field (Ctrl+2, F direct access)

O Eclipse e as Checked Exceptions

```
public class Leitura {

    public void abrirArq() throws FileNotFoundException {
        new java.io.FileReader("arquivo.txt");
    }

}
```

```
public class Principal {

    public static void main(String[] args) {

        Leitura ler = new Leitura();

        try {

            ler.abrirArq();

        } catch (FileNotFoundException e) {
            JOptionPane.showMessageDialog(null, "Arquivo não encontrado!", "Erro!", 0);
            e.printStackTrace();
        }

    }

}
```

O Eclipse e as Checked Exceptions

```
public class Leitura {

    public void abrirArq() {

        try {

            new java.io.FileReader("arquivo.txt");

        } catch (FileNotFoundException e) {

            JOptionPane.showMessageDialog(null, "Arquivo não encontrado!", "Erro!", 0);
            e.printStackTrace();
        }
    }

}
```

```
public class Principal {

    public static void main(String[] args) {

        Leitura ler = new Leitura();

        ler.abrirArq();

    }

}
```

Exemplos e exceptions

- ArithmeticException.
  - Resultado de uma operação matemática inválida.
- NullPointerException.
  - Tentativa de acessar um objeto ou método antes do mesmo ser instanciado.
- ArrayIndexOutOfBoundsException.
  - Tentativa de acessar um elemento de um vetor além de sua dimensão (tamanho) original.

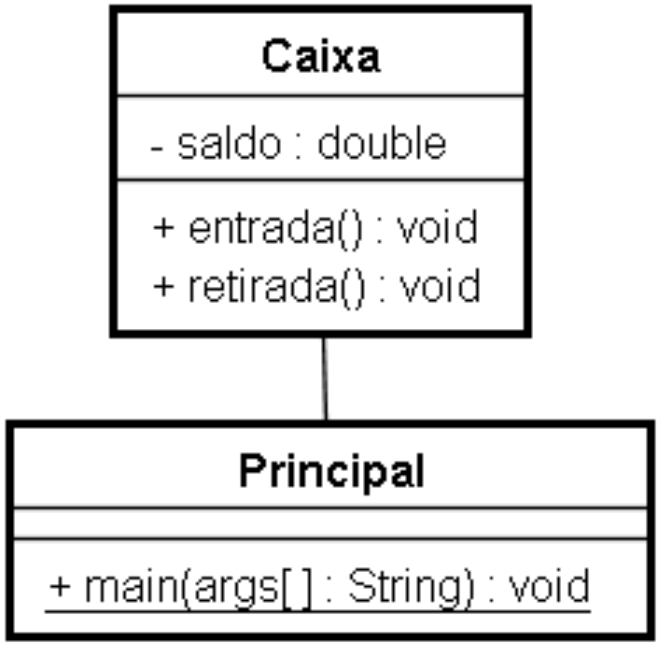
Exemplos e exceptions

- NumberFormatException.
  - Incompatibilidade de tipos numéricos.
- FileNotFoundException.
  - Arquivo não encontrado.
- ClassCastException.
  - Tentativa de conversão incompatível.

Exemplos e exceptions

- Exception.
  - É possivel capturar o erro recorrendo a classe mãe das exceções,ou seja, qualquer erro será interceptado.
  - Esse tipo de tratamento é desaconselhável em função da falta de especificação do erro ocorrido.

Exemplo


Caixa
- saldo : double
+ entrada() : void
+ retirada() : void
Principal
+ main(args[ ] : String) : void

Exemplo 01

- try-catch-finally.

```
// Trata o erro caso o usuario digite algo diferente de números
public void entrada(){

    try {
        this.setSaldo(this.saldo + Double.parseDouble(JOptionPane.showInputDialog("Digite o valor da entrada: ")));

    // Se o usuario digitar uma letra ou um caractere especial o Java irá gerar um erro de tipo de dado inválido.
    } catch (NumberFormatException erro){
        JOptionPane.showMessageDialog(null, "Digite apenas numeros!", "Tipo invalido!", 0);

    }finally{
        JOptionPane.showMessageDialog(null, "Mensagem no finally do metodo entrada",
                                            "Finally - por aqui sempre passa", 0);

    }
```

Exemplo 02

- try-catch-finally.
  - Recuperando detalhes do erro através do objeto (erro) gerado

```
public void entrada(){

    try {
        this.setSaldo(this.saldo + Double.parseDouble(JOptionPane.showInputDialog("Digite o valor da entrada: ")));

    } catch (NumberFormatException erro){
        // Apresenta mensagem personalizada
        JOptionPane.showMessageDialog(null, "Digite apenas numeros!", "Tipo invalido!", 0);
        // Apresenta a mensagem gerada pela exception NumberFormatException armazenada em erro
        JOptionPane.showMessageDialog(null, erro.getMessage());
        // Recupera a listagem da pilha de erro gerada
        JOptionPane.showMessageDialog(null,erro.getStackTrace());
        // Apresenta em console a listagem da pilha de erro gerada
        erro.printStackTrace();


    }finally{
        JOptionPane.showMessageDialog(null, "Mensagem no finally do metodo entrada",
                                            "Finally - por aqui sempre passa", 0);
    }

}
```

Throw

- A qualquer momento o programador pode fazer uma chamada a um erro através da instrução throw.
- Uma instrução throw especifica um objeto a ser lançado que pode ser qualquer membro da hierarquia de Throwable.
- Normalmente utilizado para forçar a chamada de um erro independentemente dele ter ocorrido

Exemplo 03

```
public void entrada(){

    double valor = Double.parseDouble(JOptionPane.showInputDialog("Digite o valor da entrada: "));

    try {
        // Testa se o valor lido foi zero ou negativo
        if(valor==0 || valor<0){
            // se foi, chama (força) o erro desejado (apesar dele não ter ocorrido)
            throw new IllegalArgumentException();
        }

        this.setSaldo(this.saldo + valor);

    } catch (NumberFormatException erro){
        JOptionPane.showMessageDialog(null, "Digite apenas numeros!", "Tipo invalido!", 0);

    } catch (IllegalArgumentException erro){
        JOptionPane.showMessageDialog(null, "A movimentacao nao pode ser zero nem negativo!",
                                      "Tipo invalido!", 0);

    }finally{
        JOptionPane.showMessageDialog(null, "Mensagem no finally do metodo entrada",
                                      "Finally - por aqui sempre passa", 0);
    }

}
```

Exceptions “personalizadas”

- O programador também pode criar suas próprias classes de erros.
- Para ser reconhecida pelo compilador Java como uma classe de erro ela deve pertencer a hierarquia de Throwable, mais especificamente de Exception.
- Lembrando que toda subclasse de Exception que não for subclasse de RuntimeException é uma Checked Exception, ou seja, seu tratamento é obrigatório.

Exemplo 04

- Classe MovimentoNegativo

```
// Implementacao da exception "personalizada" que trata a
// entrada de valores negativos extendendo da classe Exception

public class MovimentoNegativo extends Exception {

    // O método toString é o padrão para emissão de mensagens
    public String toString(){
        return "O valor nao pode ser negativo!";
    }

}
```

Exemplo 04

- Classe MovimentoZero

```
// Implementacao da exception "personalizada" que trata a
// entrada de valores zerados extendendo da classe Excepton

public class MovimentoZero extends Exception {

    public String toString(){
        // O metodo toString eh o padrao para emissao de mensagens
        return "O valor nao pode ser zero!";
    }
}
```

Exemplo 04

- Método entrada (Classe Caixa)

```
// Implementa as classes de tratamento de erros "personalizados"
// MovimentoZero e MovimentoNegativo através do comando throws
// Se um dos erros for instanciado, a responsabilidade de trata-lo será do método chamador (main)
public void entrada() throws MovimentoZero,MovimentoNegativo{

    double valor = Double.parseDouble(JOptionPane.showInputDialog("Digite o valor da entrada: "));

    // Verifica o valor lido
    // se for zero instancia o erro MovimentoZero
    if(valor==0) throw new MovimentoZero();
    // se for negativo instancia o erro MovimentoNegativo
    if(valor<0) throw new MovimentoNegativo();


    try {
        this.setSaldo(this.saldo + valor);
    } catch (NumberFormatException erro){
        JOptionPane.showMessageDialog(null, "Digite apenas numeros!", "Tipo invalido!", 0);
    }

}
```

Exemplo 04

- Método main (Classe Principal)

```
try{
    c1.entrada();

// se a exception gerada foi a MovimentoZero
} catch (MovimentoZero erro) {
    // Apresenta a descricao gerada contida no parametro erro
    JOptionPane.showMessageDialog(null, erro, "Erro zero!", 0);

// se a exception gerada foi a MovimentoNegativo
} catch (MovimentoNegativo erro) {
    // Apresenta a descricao gerada contida no parametro erro
    JOptionPane.showMessageDialog(null, erro, "Erro negativo!", 0);
}
```