POO
Tratamento de exceções

Aula 13 – Parte 2

Erros de execução

- Durante a **execução** de um aplicativo podem ocorrer situações anormais, estranhas ao propósito da aplicação.
- **Aplicações profissionais** devem **detectar** e **sanar** essas ocorrências o mais cedo possível, antes de perder a integridade da aplicação.
- Java oferece o conceito de **exceção** como mecanismo para tratar e corrigir falhas de execução.

Condições anormais

- Falhas físicas ou lógicas:
  - Defeito de HW, Falta de espaço em disco
  - Impressora desconectada/ sem papel
  - Perda de conexão de rede
- Falha de Sistema
  - Memória disponível
- Falha de ambiente:
  - Arquivo protegido
  - Violação de segurança

Condições anormais

- Falha nos dados
  - Erros de digitação
  - Dados corrompidos
- Erro de implementação ou codificação
  - Objetos usados antes da criação
  - Erro de indexação
- Quando ocorre alguma dessas falhas, o que acontece com o programa Java?

Exceção – Notificação de Erros

- Uma condição anormal pode ser diagnosticada internamente pelo programa Java ou pelo programador.
- Essa falha é representada no ambiente Java como um objeto do tipo **exceção**.
- Esse tipo de objeto Exceção **notifica** e **carrega** informação sobre uma ocorrência especial.

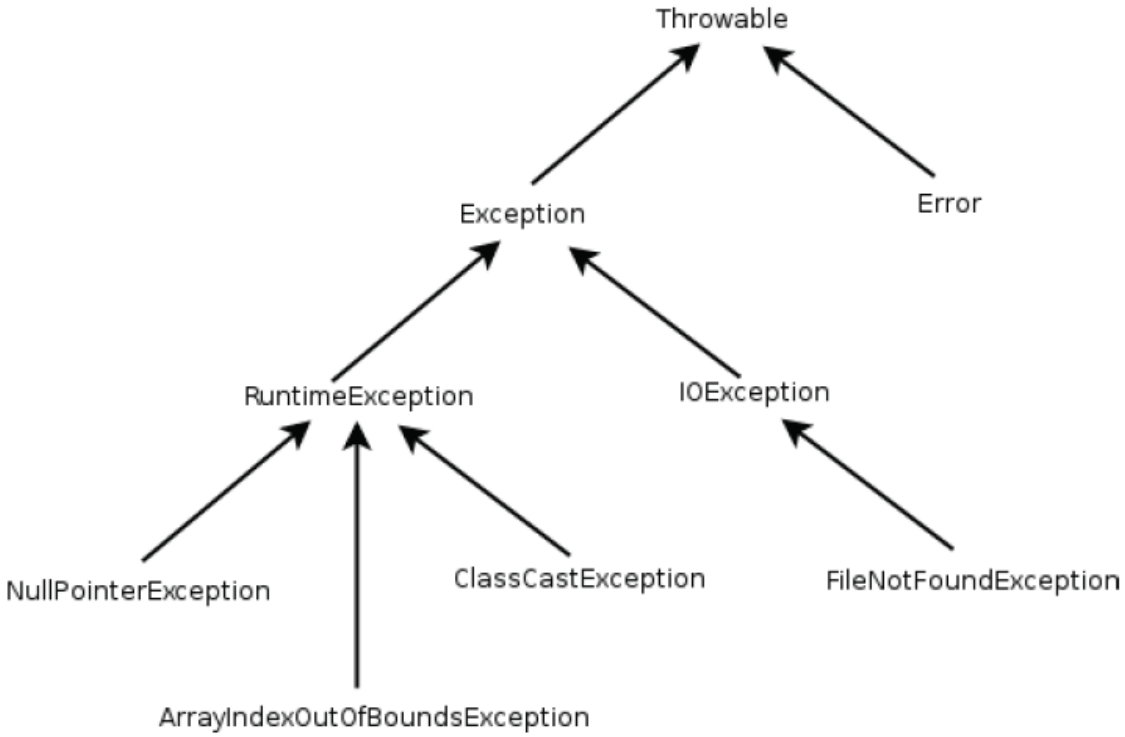
Exceções – Criação e tratamento

- Exceções podem ser criadas e lançadas:
  - Por falhas do sistema;
  - Por métodos da API
  - Pelo próprio programador (métodos das nossas classes);

- Diante de uma exceção notificada podemos:
  1. Não tomar nenhuma providência (o que fizemos até agora!);
  2. Capturar e tratar;
  3. Apenas capturar (e não tratar);
  4. Relançar;

Hierarquia de exceções

- Todas as exceções são derivadas direta ou indiretamente da classe **Throwable**.
- Alguns da família:



Tratamento de Exceções

- Para tratarmos comportamentos anormais devemos proteger um bloco de código com as cláusulas:
  - try { }: tenta executar o bloco de comandos
  - catch{ }: captura em caso de falha no bloco try
  - finally { }: trecho de código sempre executado independente se houve erro ou não.

Exemplo de Sintaxe

```
try{
    ...    /*Bloco de código*/
}
catch (IndexOutOfBoundsException e){
    ...    /*Trata erro mais específico: Índice fora do limite*/
}
catch (AritmeticException e){
    ...    /*Trata erro mais genérico: Aritmético*/
}
finally {
   ...     /*Bloco de código final*/
}
```

Caso 1 – Captura e Tratamento

```
import java.util.Scanner;
public class TesteException
{
    public static void main(String args[])
    {
        int a, b, c;
        try
        {
            System.out.println("Digite 2 valores:");
            Scanner entrada = new Scanner(System.in);
            a = entrada.nextInt();
            b = entrada.nextInt();
            c = a/b;
            System.out.println(" c = "+c);
        }
        catch (ArithmeticException e)
        {
            System.out.println("Erro aritmetico!");
        }
        catch (RuntimeException e)
        {
            System.out.println("Erro inesperado!");
        }
        finally
        {
            System.out.println("Execucao concluida");
        }
    } // Fim de main()
} // Fim de classe
```

Caso 2 - Captura de exceção sem tratamento

```
try{
    ...   /*Bloco de código*/
}
catch ( Exception e) {
        /*deixar em branco*/
}
```

Como gerar uma exceção?

- Nos próximos slides vamos aprender:
  - Como um método pode gerar e lançar uma exceção;
  - Como relançar uma exceção gerada por outro método;
  - Como usar essas opções na prática!

Gerando e lançando uma Exceção

- O ***metodo1*** cria e lança uma exceção quando a condição for verdadeira.

```
public void metodo1 ( ) throws Exception
{
        if (<condicao>)
                throw new Exception (“Erro");
        else ...
}
```

Exemplo prático – Classe Circulo

```
// lança Exceção para raio inválido
public void setRaio (double raio) throws Exception
{
        if (raio<0)
                throw new Exception ("Circulo com raio negativo");
        else
                this.raio = raio;
}
```

Caso 3 - Relançando Exceção (gerada por outro método)

- Neste exemplo, o ***metodo2( )*** não cria uma exceção mas pode lançar uma possível exceção caso ocorra alguma falha na chamada ao ***metodo1( )***.

```
public void metodo2 ( ) throws Exception
{
        ...
        metodo1( ); // lança exceção
        ...
}
```

Exemplo prático – Classe Circulo

```
// construtor relança Exceção do método setRaio
public Circulo(double raio) throws Exception {
        setRaio(raio);
}

// lança Exceção para raio inválido
public void setRaio (double raio) throws Exception
{
        if (raio<0)
                throw new Exception ("Circulo com raio negativo");
        else
                this.raio = raio;
}
```

Captura e tratamento da Exceção relançada

- Neste exemplo, a possível exceção lançada pelo ***metodo2( )*** é capturada e tratada pelo programa principal.

```
public static void main(String[] args) {
    try {
            ...
            metodo2 ( );
            ...
    }
    catch (Exception e) {
            System.out.println("Erro identificado: ”+ e.getMessage());
    }
}
```

Uso da Classe Circulo

```
//captura e trata a exceção lançada pelo construtor
public static void main(String[] args) {
    Circulo c1;
    try {
        c1 = new Circulo(-5);
        c1.imprime();
    }
    catch (Exception e) {
        System.out.println("Erro ao criar circulo: "+ e.getMessage());
    }
}
```

Uma dúvida: neste exemplo, o objeto c1 será instanciado ou não?

Atividade Extra (opcional) – Exceção

- Escreva a família Formas Geométricas (incluindo círculo, triangulo e retângulo) de modo a lançar exceções quando parâmetros passados aos ***setters*** forem inválidos.
- Entrega até dia 03/06