Exceções

Profa. Karen Selbach Borges

Última Atualização: Mai/2023

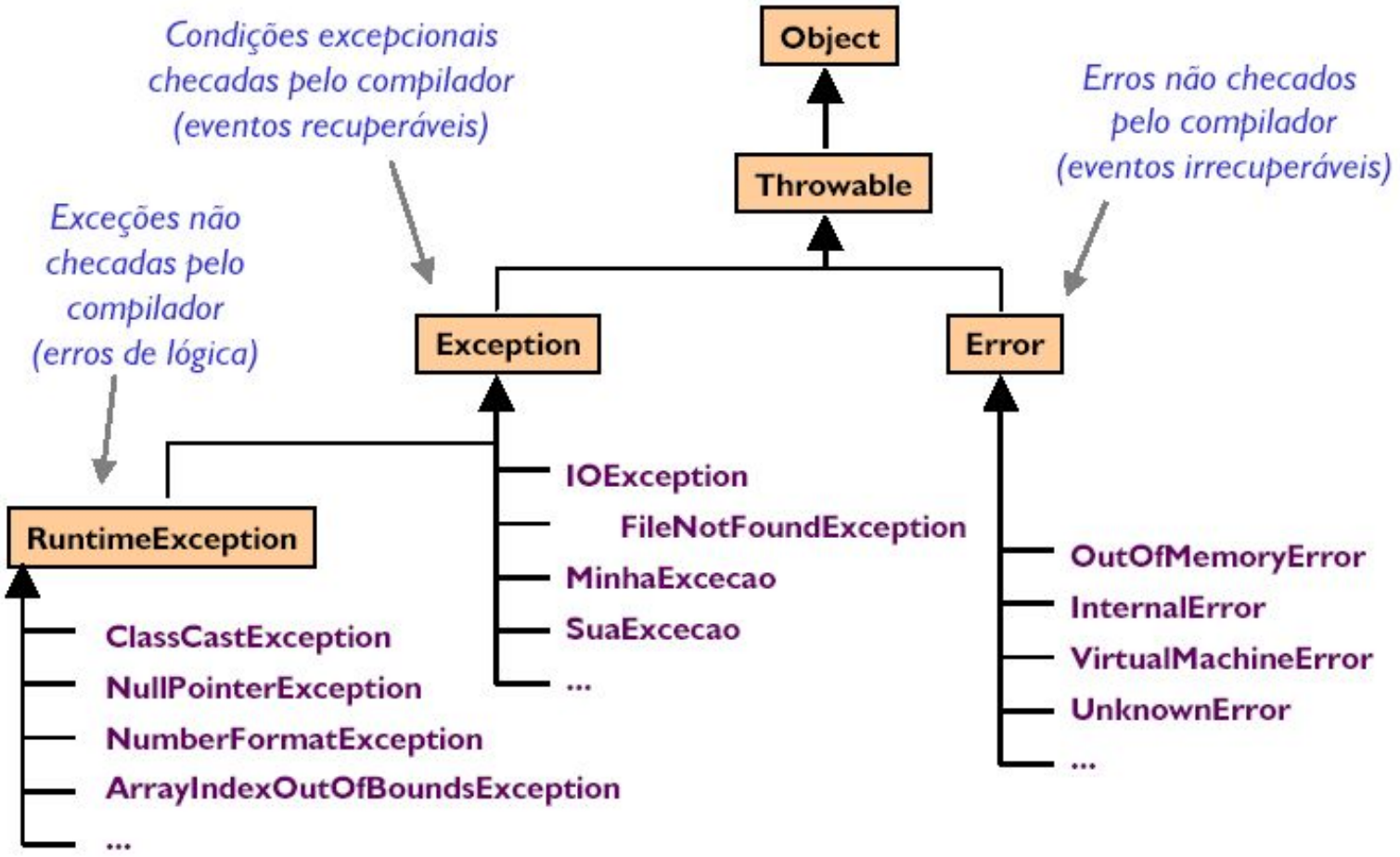
Exceções – visão geral

- Erros e exceções representam:
  - Situações anormais (exceções)
  - Situações inválidas (erros)
- Ocorrem durante a compilação ou em tempo de execução.

Introdução

- Os erros mais comumente encontrados são:
  - Erros de lógica de programação : devem ser corrigidos pelo programador.
    - Ex : limites do vetor ultrapassados, divisão por zero
  - Erros devido a condições do ambiente de execução : fogem do controle do programador, mas devem ser tratados em tempo de execução.
    - Ex : arquivo não encontrado, rede fora do ar, etc.
  - Erros graves, sem possibilidade de recuperação : fogem do controle do programador e nada pode ser feito.
    - Ex : falta de memória, erro interno da JVM, espaço em disco esgotado, etc.

Classificação das Exceções



Classificação das Exceções

- **Exceções Não-Verificadas** : são as exceções que derivam das classes *Error* e *RuntimeException*.
  - As exceções derivadas da classe *Error* **não devem** ser tratadas, pois a solução de tais problemas está fora do domínio do programador.
  - As exceções derivadas da classe *RuntimeException* **não precisam** ser tratadas. Elas deveriam ser tratadas a nível de código.

Classificação das Exceções

- **Exceções Verificadas** : são as exceções derivadas da classe *Exception*
  - Devem, **obrigatoriamente**, ser tratadas.
  - Ocorrem devido a circunstâncias que o programador não pode evitar.
  - Em geral envolvem entrada e saída de dados.

Tratamento de Exceções

- Antecipação de problemas pelo programador
- Situações de erro podem ser revertidas
- Solução ideal: tratamento de problemas separado do código normal (classes de exceção)
- Mecanismo: sistemas de tratamento de exceções
- Um sistema de tratamento de exceções deve ser capaz de: detectar e sinalizar (disparar), capturar e tratar uma exceção (ativar tratador)

Exceções – visão geral

- Diz-se que uma exceção é *lançada para* sinalizar alguma falha
- O lançamento de uma exceção causa uma interrupção abrupta do trecho de código que a gerou
- O controle da execução volta para o primeiro trecho de código (na pilha de chamadas) apto a tratar a exceção lançada

Exceções – visão geral

```
class X {
   .
   .
   .
   public void metodoGerador () {
    código que gera a exceção;
   }
}
```

Propaga a exceção para o método que chamou o método gerador da exceção

```
class Y {
   X teste = new X();

   public void metodoReceptor () {
    teste.metodoGerador();
   }
}
```

Encerra a aplicação

OU

```
class X {
   .
   .
   .
   public void metodoGerador () throws Exception {
    código que gera a exceção;
   }
}
```

Lança a exceção para o método que chamou o método gerador da exceção

```
class Y {
   X teste = new X();

   public void metodoReceptor () {
    try {
      teste.metodoGerador();
    }
    catch (Exception e) {
      código que trata a exceção;
    }
   }
}
```

Segue a execução da aplicação

Exceções – visão geral

- Para usarmos exceções precisamos de:
  - uma representação para a exceção
  - uma forma de lançar a exceção
  - uma forma de tratar a exceção

Representação da Exceção

- As exceções são representadas por classes que derivam de *Exception* ou de *RuntimeException*

Algumas Classes de Exceções

- **ClassNotFoundException** (java.lang): indica que houve tentativa de utilizar uma classe não definida.
- **InstantiationException** (java.lang): indica que houve tentativa de instanciar uma classe que é uma interface ou uma classe abstrata.
- **NoSuchMethodException** (java.lang): indica que houve tentativa de utilizar um método não definido.
- **NoSuchFieldException** (java.lang): indica que houve tentativa de utilizar uma variável de instância não definida.

Algumas Classes de Exceções

- **CloneNotSupportedException** (java.lang): indica que houve tentativa de clonar um objeto cuja classe não está implementando a interface Cloneable
- **IOException** (java.io): indica uma falha ou interrupção no processo de E/S
- **FileNotFoundException** (java.io): indica que houve tentativa de acessar um arquivo que não existe
- Relação disponível na documentação das APIs em **java.lang.Exception**

Lançando Exceções

- Se um método encontrar uma ou mais situações com as quais não consegue lidar, então esse deve:
  - Sinalizar que dentro do método pode ocorrer uma exceção (utiliza-se a palavra reservada *throws*)
  - Instanciar a classe de exceção mais adequada e fazer o lançamento (utiliza-se a palavra reservada *throw)*

Exemplo 1

```
public class Circulo{
     // Declaração de atributos
    private double raio;

    // Construtor da classe Ciculo
    public Circulo(double raio) throws IIlegalArgumentException {
      if (raio < 0) {
        throw new IIlegalArgumentException ("Raio não pode ser negativo");
      }
      this.raio = raio;
    }

   //Metodo que recupera o valor do raio
   public double getRaio(){
      return raio;
   }
}
```

Sinaliza a exceção

Dispara a exceção

Exemplo 2

```
import javax.swing.*;

public class Leitor{
  public double leDouble() throws NumberFormatException{
   String altura = JOptionPane.showInputDialog(null,"Informe a altura");
   double a = Double.parseDouble(altura);
  }
}
```

Apenas sinaliza a exceção

O método parseDouble irá instanciar e disparar a exceção caso o seu parâmetro não seja uma String representando um número double (ex: 1,9 quando deveria ser 1.9)

Tratamento da Exceção

- O tratamento da exceção se dá pela captura do objeto de exceção (no bloco try) e a execução da rotina de recuperação de falhas (no bloco catch).

Tratamento da Exceção

```
try
{
  // código que pode gerar exceção
}
catch (Exception e)
{
  // código que trata exceção
}
finally
{
  // tratamento geral
}
```

- No bloco try estão colocados os comandos que podem provocar o lançamento de uma exceção
- Essas exceções são capturadas em um ou mais comandos catch
- O comando finally contém código a ser executado, independente de outros comandos. É opcional, mas quando presente, é sempre executado

Exemplo

```
import javax.swing.*;

public class TestaInputDialog {
    public static void main(String args[ ]) {
       boolean ok;
       do {
                  try {
                     String altura = JOptionPane.showInputDialog(null,"Informe a
altura");
                      double a = Double.parseDouble(altura);
                      ok = true;
                }
                catch (NumberFormatException nfe) {
                      JOptionPane.showMessageDialog(null, "Caracter inválido !");
                      ok = false;
                }
       } while (ok != true);
       System.exit(0);
    }
}
```

Múltiplas Exceções

- Podem-se capturar várias exceções em um bloco *try* e tratá-las de forma diferente em diversos blocos *catch*.

```
try {
 código que gerou as exceções;
}
catch (Exception e1) {
  código que trata o primeiro tipo de exceção
}
catch (Exception e2) {
  código que trata o primeiro tipo de exceção
}
```

Exemplo

```
double [ ] alturas = new double[5];
do {
    try {
        String altura = JOptionPane.showInputDialog(null,"Informe a medida");
        double a = Double.parseDouble(altura);
        if (a < 1.5){
            JOptionPane.showMessageDialog(null, "Esta medida não é valida");
            continue;
         }
         alturas[cont] = a;
         cont++;
         ok = false;
      }
      catch (NumberFormatException nfe) {
         JOptionPane.showMessageDialog(null, "Caracter inválido !");
         ok = false;
      }
      catch (ArrayIndexOutOfBoundsException aie){
          JOptionPane.showMessageDialog(null, "Já foram informados os 5 valores !");
         ok = true;
      }
} while (ok!=true);
```

Observações

- catch (Exception e) deve ser o último bloco. Oferece tratamento genérico às exceções.
- Variáveis declaradas dentro do bloco try, são locais àquele bloco. Ou seja, fora do try não são reconhecidas.
- Não é obrigatório o catch para exceções não verificadas (`IllegalArgumentException`)
- Dentro de um catch ou finally podem haver outros try-catch.

Criando Classes de Exceção

- Pode acontecer que exista no código um problema que não consiga ser adequadamente tratado por uma das classes de exceções existentes na API Java
- Neste caso devem ser criadas classes que derivem de *Exception* ou de *RuntimeException*

```
public class CampoVazioException extends Exception {
    public CampoVazioException() {
    }

    public CampoVazioException(String msg) {
        super(msg);
    }
}
```

Importante !

- Não negligencie as exceções:

```
Ex : try {
        << codigo que gera a exceção >>
     }
     catch(Exception e) { }
```

- As exceções foram projetadas para fornecer relatórios de erros.
- “Silenciar” uma exceção pode ocultar uma situação de erro séria.

try-with-resources

- A instrução try-with-resources é uma instrução try que declara um ou mais recursos.
- Um recurso é um objeto que deve ser liberado quando o programa terminar.
- A instrução try-with-resources garante que cada recurso seja fechado no final da instrução.
- Qualquer objeto que implemente java.lang.AutoCloseable, o que inclui todos os objetos que implementam java.io.Closeable, pode ser usado como um recurso.

Exemplo sem try-with-resources


```
public String readFirstLineFromFileWithFinallyBlock(String
path) throws IOException {

    FileReader fr = new FileReader(path);
    BufferedReader br = new BufferedReader(fr);
    try {
        return br.readLine();
    } finally {
        br.close();
        fr.close();
    }
}
```

Exemplo sem try-with-resources

- No exemplo, o programa deve liberar o recurso de volta ao sistema operacional, e isso é feito chamando o método close do recurso.
- No entanto, se o programa falhar em fazer isso antes que o Garbage Collector recupere o recurso, as informações necessárias para liberar o recurso serão perdidas.
- O recurso, que ainda é considerado pelo sistema operacional como em uso, ficará bloqueado.

Exemplo com try-with-resources


```
public String readFirstLineFromFile(String path) throws
IOException {
    try (FileReader fr = new FileReader(path);
        BufferedReader br = new BufferedReader(fr)) {
            return br.readLine();
    }
}
```