UNIVERSIDADE DE ITAÚNA CIÊNCIA DA COMPUTAÇÃO - 6º PERÍODO LINGUAGENS DE PROGRAMAÇÃO

Tratamento de Exceções

Tratamento de Eventos

ARLEY AUGUSTO E SILVA

IAGO ANTUNES FERREIRA

Itaúna

17/11//2022

Tratamento de Exceções

1. O que é?

Uma exceção denota um comportamento anormal, indesejado, que ocorre raramente e requer alguma ação imediata em uma parte do programa. Esta condição pode ser um erro, tal como um overflow aritmético, ou pode ser uma ocorrência anormal que não é considerada um erro, tal como o fim do arquivo. Condições que são consideradas exceções são chamadas de síncronas, o que significa que elas ocorrem em locais previsíveis do programa. Por exemplo, o overflow ocorrerá somente onde está sendo feita uma operação aritmética, e o fim do arquivo ocorrerá somente quando um arquivo estiver sendo lido. Isto distingue exceções de condições que são assíncronas, isto é, que podem ocorrer a qualquer momento. Uma interrupção gerada pelo usuário ou um sinal do dispositivo são exemplos de condições assíncronas, que são gerenciadas mais apropriadamente através das características de concorrência de uma LP, uma vez que são geradas através de algum processo que está executando concorrentemente.

Exceções possuem um sentido mais amplo do que simplesmente erros computacionais. Elas referem-se a qualquer tipo de comportamento anômalo que, intuitivamente e informalmente, correspondem a um desvio do curso de ações esperado, previsto pelo programador. O conceito de “desvio” não pode ser colocado rigorosamente. Ele representa uma decisão de projeto tomada pelo programador, que decide que certos estados são “normais” e “esperados”, enquanto outros são “anormais”. Assim, a ocorrência de uma exceção não significa necessariamente que ocorreu um erro catastrófico, mas sim que a unidade que está sendo executada não pode proceder da maneira definida pelo programador.

- Exceção — uma indicação de um problema que ocorre durante a execução de um programa.

- Tratamento de exceções — resolver exceções que poderiam ocorrer para que o programa continue ou termine elegantemente.

- O tratamento de exceções permite que os programadores criem programas mais robustos e tolerantes a falhas.

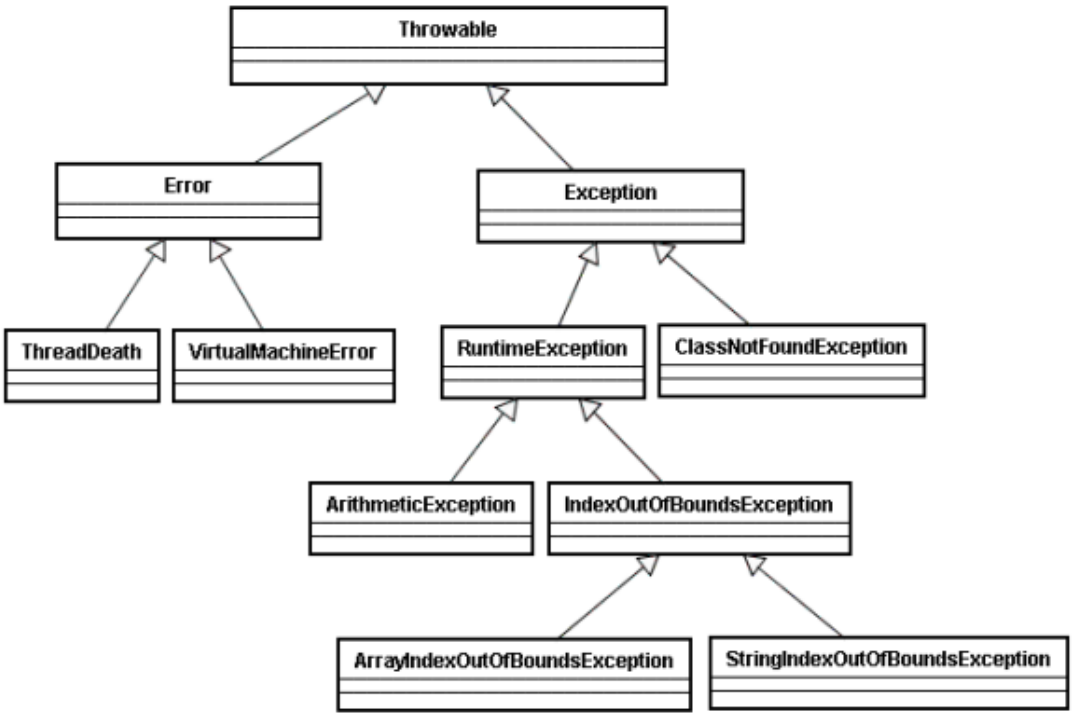
Exemplos:

– ArrayIndexOutOfBoundsException — é feita uma tentativa de acessar um elemento depois do final de um array.

– ClassCastException — ocorre uma tentativa de fazer uma coerção em um objeto que não tem um relacionamento é um com o tipo especificado no operador de coerção.

– NullPointerException — quando uma referência null é utilizada onde um objeto é esperado.

2. Hierarquia



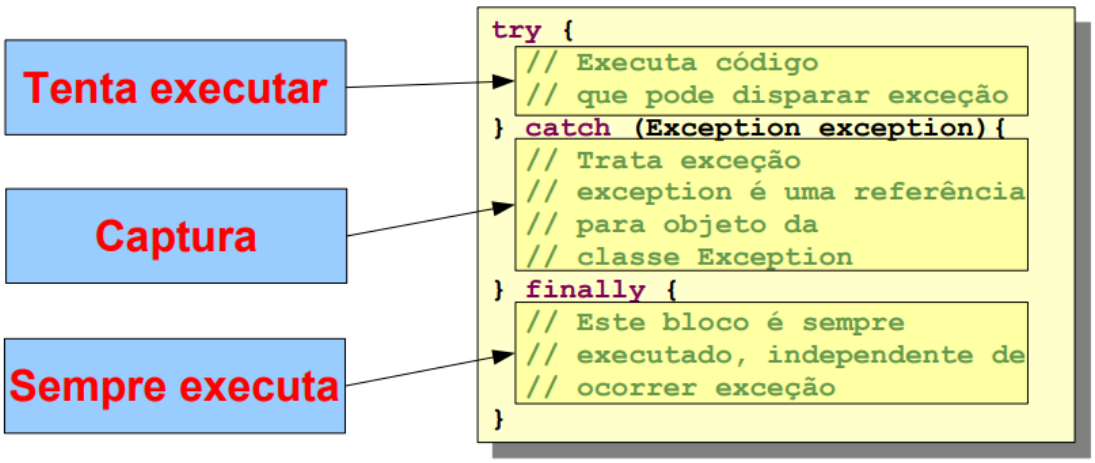
3. Funcionamento

Um método deve informar que exceções ele pode disparar (throw): cláusula throws na definição do método

Um bloco que tenta (try) chamar um método que pode disparar uma exceção deve tratá-la: chamada normal de um método, mas que deve estar em um bloco try {...} catch {...}

Uma exceção é um objeto que deve ser capturado (catch): é nesse bloco que a exceção deve ser tratada. Um trecho de código pode ser executado sempre: bloco finally.

Um bloco deve capturar uma exceção para tratá-la bloco try-catch-finally.



Pode haver mais de um bloco catch

Cada bloco trata um tipo específico de exceção

O bloco try contém métodos que podem disparar todas as exceções

Um método pode disparar mais de uma exceção

As classes de exceção seguem a ordem da mais especializada para a mais genérica.

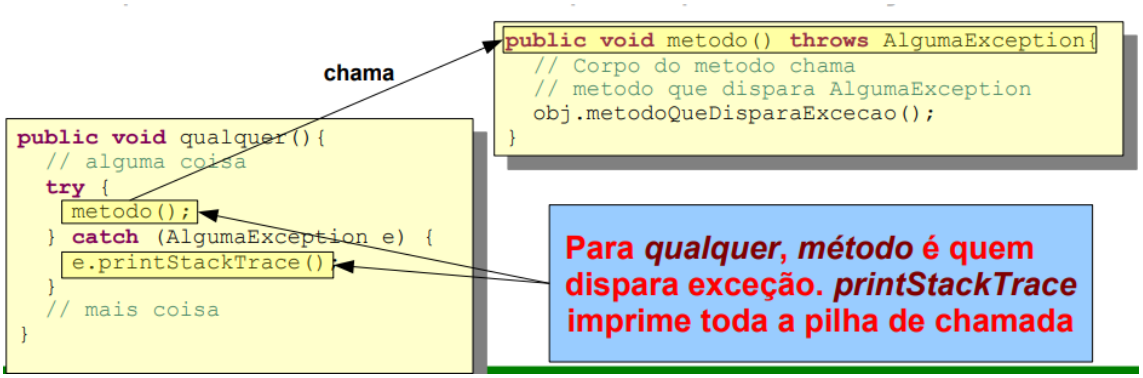
```
try {
// bloco de comandos
} catch (Excecao1 ex1) {
// Trata exceção1
} catch (Excecao2 ex2) {
// Trata exceção2
} catch (Excecao3 ex3){
// Trata exceção3
}
```

Um método pode repassar uma exceção

Ele chama um método que dispara uma exceção, mas não quer tratar.

Ele pode repassar a exceção Basta colocar a cláusula throws na assinatura do método

Para quem chama, é o método que dispara a exceção



No método que dispara a exceção:

Coloque a cláusula throws

Crie o objeto da classe de exceção Dispare (throw) a exceção

Onde a exceção será disparada depende de cada método

```
public Contato buscar(String nome)throws ContatoNaoEncontradoException{
  // Laco para procurar contato pelo nome
  for (int i = 0 ; i < quantidade ; i++)
    if (contatos[i].nome().equals(nome))
      return contatos[i];

  // Se sair do laço e não tiver retornado,
  // o contato não está cadastrado.
  // Deve disparar exceção
  throw new ContatoNaoEncontradoException();
}
```

4. Aplicação segura

Uma porção de código é considerada "segura", no contexto do tratamento de exceções, se as falhas no código, ocorridas em tempo de execução, não produzem efeitos prejudiciais, como vazamentos de memória, corrupção de dados ou saída inválida. Um código seguro deve satisfazer invariantes mesmo após a ocorrência de exceções. Os níveis de segurança relacionados ao tratamento de exceção podem ser colocados da seguinte forma:

1. **transparência à falha** : há garantia que as operações ocorrerão com sucesso e satisfarão todos os requerimentos, mesmo na presença de situações excepcionais. Este é o melhor nível de segurança.
2. **transacional** : as operações podem falhar, mas quando isso ocorre as operações não causam efeitos colaterais.[1]
3. **segurança básica**: execuções parciais de operações que falham podem causar efeitos colaterais, mas os invariantes de estado são preservados (isto é, qualquer dado gravado conterá valores válidos).
4. **segurança mínima**: execuções parciais de operações que falham podem gravar dados inválidos mas não levarão à falha completa (*crash*) do programa.
5. **sem segurança**: não há qualquer garantia. Este é o pior nível de segurança para com exceções.
6.

Normalmente um nível básico de segurança é requerido. A transparência à falhas é difícil de implementar e normalmente não é possível de atingir em bibliotecas onde o conhecimento completo da aplicação não está disponível.

Linguagens com tratamentos de exceções:

- Ada;
- Object Pascal;
- C++;
- D;
- Delphi;
- Eiffel;
- Java;
- Objective-C;
- PHP (versão 5);
- Python;
- REALbasic;
- ML;
- Ruby;
- Entre outras;

Linguagens que não oferecem mecanismos de tratamento de exceções:

- Linguagem C;
- Perl;
- Pascal;
- Modula-2;

5. Pós Contras

As exceções verificadas podem, em tempo de compilação, reduzir consideravelmente (mas não eliminar inteiramente) a ocorrência de casos em que de exceções não tratadas emergem para fora da aplicação; as exceções não verificadas (`RuntimeException`s e `Error`s) podem permanecer sem tratamento.

Porém, alguns criticam as exceções verificadas alegando que esta sintaxe requer assinaturas de métodos com extensas declarações de cláusula `throws`, que freqüentemente revelam detalhes da implementação interna e reduzem o encapsulamento, ou ainda encorajam o abuso de blocos `try/catch` mal desenhados, que podem esconder a ocorrência de exceções das rotinas que deveriam tratá-las.[8] Além disso, o uso destas exceções pode dificultar a manutenção dos programas. Por exemplo, uma interface pode ser inicialmente declarada para lançar as exceções X e Y mas, numa versão posterior do programa, percebe-se que um novo tipo de erro pode ser detectado, e é necessário, em adição, lançar a exceção Z, provocando uma mudança na interface que fará do código cliente incompatível com a nova interface.[8]

Outros consideram que os problemas citados podem ser resolvidos se o número de exceções declaradas for reduzida através da utilização de uma superclasse que generalize todas as classes de exceções potencialmente lançadas ou pela definição e declaração de tipos de exceção adequadas para o nível de abstração correspondente ao método chamado[9] através do mapeamento de exceções de nível mais baixo a estes tipos, preferencialmente envolvendo estas exceções através do uso do mecanismo de encadeamento de exceções com o objetivo de preservar a exceção raiz.

Um simples declaração `throws Exception` ou `catch (Exception e)` é sempre suficiente para satisfazer a verificação. Enquanto esta técnica é algumas vezes útil, ela efetivamente engana o mecanismo de exceções verificadas, e portanto deve ser utilizada com cuidado.

Uma visão predominante considera que as exceções não verificadas não devem ser tratadas, com exceção do nível mais externo do escopo do programa, pois elas frequentemente representam cenários em que a recuperação não é possível: as exceções de `RuntimeException` normalmente representam defeitos de programação[9] ou situações de erro (`Error`) que indicam problemas irrecuperáveis da JVM. Isto significa que, em resumo, mesmo em linguagens que suportam exceções verificadas, existem cenários em que a sua utilização não é apropriada.

6. Exceções continuáveis

As exceções "continuáveis" são relacionadas com o modelo conhecido como *"modelo do recomeço"* de tratamento de exceções, na qual algumas exceções são ditas como "continuáveis"; permitem que a execução retorne ao ponto que originou a exceção, após a subrotina de tratamento de exceções ter tomado as ações corretivas necessárias. O sistema de condições é generalizado da seguinte forma: dentro da subrotina de tratamento de condições consideradas não sérias (exceções "continuáveis" ou, em inglês, *continuable exception*), é possível saltar para pontos de reinicio pré-definidos (*restarts*) que residem entre a expressão que sinalizou a exceção e a subrotina de tratamento da condição.