Desenvolvimento de Sistemas

Tratamento de exceções em linguagem de programação: comandos, classes, aplicabilidade

Por vezes, programas de computador resultam em erro, e esse erro pode terminar com a execução do programa ou o travamento do sistema por completo. Para contornar isso, algumas linguagens de programação, como Java, contêm o **tratamento de exceção**, que é uma maneira segura de tratar desses erros sem a quebra total do programa.

Imagine os seguintes exemplos que poderiam causar problemas em um programa de computador:

- ➔ Um novo usuário, ao executar o seu cadastro em um sistema, acaba inserindo uma entrada de dados anormal em um campo de cadastro, como um caractere especial em um campo de data ou um hífen em um campo que não aceita esse tipo de caractere.

- ➔ Um arquivo que deve ser lido pelo programa para gravar novos dados encontra-se ausente ou corrompido
- ➔ Um programa tenta executar uma divisão por 0 (zero).

Nos casos mencionados, o tratamento de exceção evitaria com graciosidade que o programa quebrasse totalmente e, ou, parasse de responder. Ou seja, o tratamento de exceção observa e trata erros previsíveis, mostrando,

por exemplo, uma mensagem amigável ao usuário ao invés de encerrar a execução do programa abruptamente.

Como funcionam as exceções

Uma exceção em Java nada mais é do que um **objeto** de uma classe específica (que, como você verá a seguir, é descendente de **Throwable**). Esse objeto especial pode ser criado em situações específicas em que se detecta uma situação de falha. Por exemplo, imagine que você está programando um *software* que lê arquivos no formato TXT; caso algum usuário acabe enviando um arquivo de extensões DOC ou XLS, que não são suportadas, é possível criar e **lançar** uma exceção para sinalizar à execução do programa que ocorreu um problema.

Lançar exceção significa basicamente interromper a execução do código atual, do código que chamou esse código, do código que invocou este outro e assim por diante, até alcançar algum código que faça o **tratamento** dessa exceção.

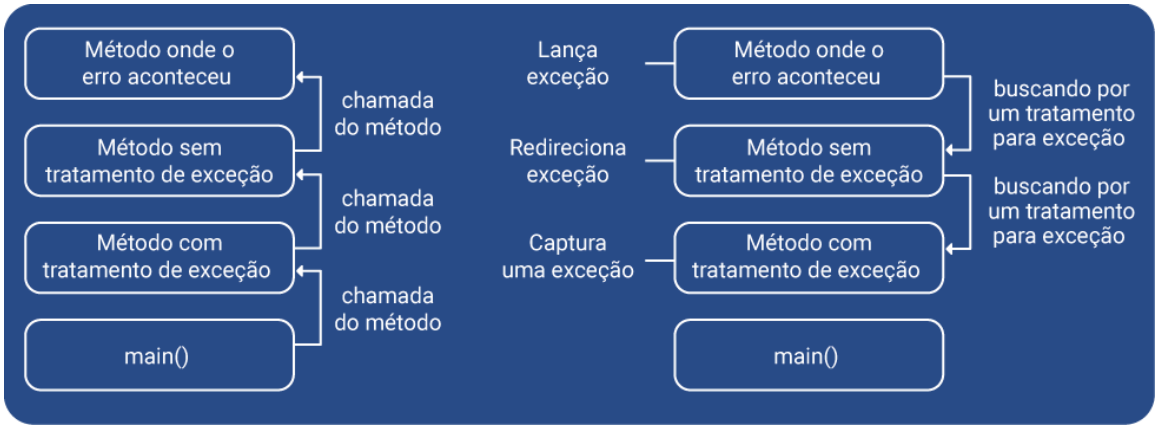

Figura 1 – Fluxo de uma exceção

Fonte: Senac EAD (2022)

Tratar exceção é basicamente proteger um código suscetível à falha (neste exemplo, poderia ser o código responsável por realizar a leitura do arquivo que ele espera que seja TXT) e realizar uma ação específica para essa falha – por exemplo, mostrando uma mensagem ao usuário ou registrando em arquivo log o problema que ocorreu.

Caso **não haja tratamento** para uma exceção lançada, ela alcançará o nível mais básico do programa (o método **main()** em Java), mostrará em console a exceção com mensagens compreensíveis ao programador, mas pouco amigáveis ao usuário, e poderá encerrar a execução do programa. Além de situações em que você mesmo cria objetos de exceção no código desenvolvido, existem casos em que Java cria exceções internamente em operações implementadas na linguagem. Por exemplo, quando ocorre a manipulação de um vetor e utiliza-se um índice inválido, internamente Java cria um objeto de exceção para alertar que não é possível acessar a posição inexistente do vetor.

Hierarquia de exceções em Java

Na linguagem Java, todas as exceções são representadas por classes e todas as classes de exceção derivam de uma classe chamada **Throwable** sendo assim, ao ocorrer uma exceção, algum objeto de algum tipo será instanciado (criado). Existe também na linguagem dois tipos de subclasses que são **Throwable**: as classes de exceção e as de erro. Exceções do tipo erro dizem respeito a erros que ocorrem na JVM (Java Virtual Machine, ou em português, Máquina Virtual Java) e não no programa em questão. Esse tipo de exceção está fora do controle do programador e não será tratado por ele. Logo, este texto tratará da subclasse de exceção (**Exception**), que são as exceções

que podem, e devem, ser tratadas pelo programador, como nos casos citados anteriormente. Uma importante subclasse de exceção é a **RuntimeException**, que é usada para representar diversos erros comuns durante a execução de um programa em Java.

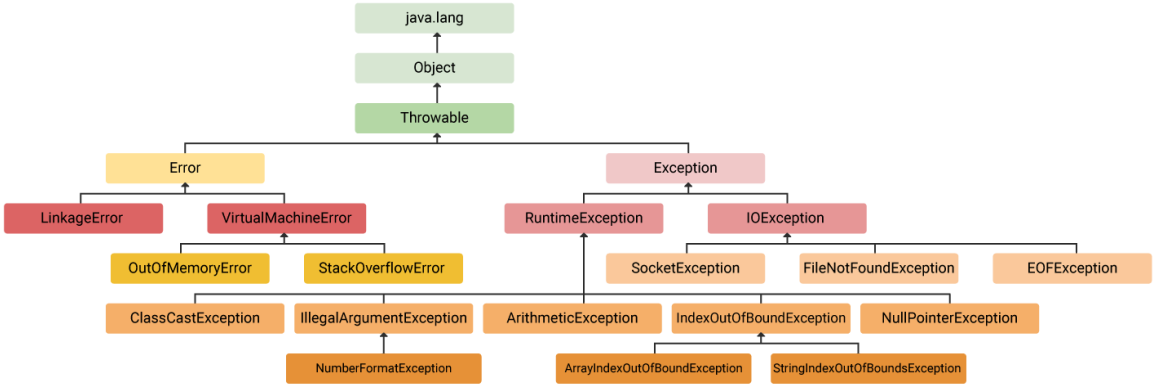

Figura 2 – Hierarquia de classes de exceção e de erros

Fonte: Adaptado de Punchihewa (2021)

Comandos e aplicabilidade

Todo o tratamento de exceção em Java ocorre por meio de cinco palavras-chave: ***try***, ***catch***, ***throw***, ***throws*** e ***finally***. Estas formam uma espécie de subsistema em que uma acaba referenciando a outra.

Partes do programa e trechos de código que você desejar monitorar por exceções deverão sempre conter um bloco de código ***try***. Caso ocorra uma exceção nesse bloco, o seu código poderá capturar essa exceção usando o comando ***catch***. Para “lançar” manualmente uma exceção, use o comando ***throw***. Em alguns casos, será preciso lançar uma exceção fora de um método e, para isso, utiliza-se o comando ***throws***. Além disso, todo código que precisa ser executado ao sair de um bloco try será colocado no bloco ***finally***.

Classes

Algumas classes do Java são exclusivas para tratar certos tipos de exceção.

Confira uma pequena lista com as mais classe mais comuns e um breve exemplo de utilização delas.

1. NullPointerException

Uma exceção do tipo **NullPointerException** é lançada quando um programa Java tenta processar um objeto com um valor do tipo **null**.

Por exemplo:

```
public class Exemplo {
  public void fazAlgo() {
    Integer numero = null;

    if (numero > 0) {
      System.out.println("O numero e positivo");
    }
  }
}
```

Nesse exemplo, por conta de o objeto **número** conter o valor **null**, uma exceção será disparada pela classe **NullPointerException**.

2. ArrayIndexOutOfBoundsException

Uma exceção do tipo **ArrayIndexOutOfBoundsException** ocorre quando um programa Java tenta processar (inserir ou pesquisar) em um vetor fora do seu índice, ou seja, tenta pesquisar ou adicionar um valor em uma posição do vetor que não existe.

Já foi apresentado um exemplo dessa situação neste conteúdo, mas confiram mais um:

```
public class Exemplo {
  public void processArray() {
    List names = new ArrayList <>();
    names.add("João");
    names.add("Maria");
    return names.get(5);
  }
}
```

Nesse exemplo há um **arraylist** que contém duas posições, mas tenta retornar um valor na sua quinta posição, que, neste caso, não existe.

3. IllegalStateException

O **IllegalStateException** é lançado quando um método está sendo chamado em um momento ilegal ou inadequado. Uma ocorrência comum dessa exceção é lançada ao tentar remover um item da lista enquanto você está processando essa lista, como demonstrado a seguir:

```
public class Exemplo {
  public void sendoprocessadoArray() {
    List names = new ArrayList<>();
    names.add("Eric"); //adicionando na lista
    names.add("Sydney");

    Iterator iterator = names.iterator();

    while (iterator.hasNext()) {
      iterator.remove();
    }
  }
}
```

Nesse exemplo, uma exceção será disparada, pois está sendo chamado o método de remover da lista dentro de um *while* de incluir na lista.

4. ClassCastException

A classe **ClassCastException** é lançada quando se tenta incluir um objeto dentro de outro, sendo que eles não pertencem à mesma classe hierárquica. Pode acontecer por exemplo quando se tenta colocar um objeto do tipo *long* dentro de um do tipo *string*.

Observe:

```
public class Exemplo {
  public void forcandoUmCastErrado() {
    Long value = 1967L;
    String name = (String) value;
  }
}
```

5. ArithmeticException

A classe **ArithmeticException** será chamada quando uma condição aritmética não permitida for executada, por exemplo, no caso de uma tentativa de divisão por zero.

Confira:

```
return numero / 0;
```

Nesse exemplo, um retorno em uma variável que tenta ser dividida por zera lançará a exceção.

Essas foram as cinco classes mais comumente utilizadas e disparadas em Java.

Lançando suas próprias exceções

Até agora você viu exceções que foram geradas e captadas (*catch*) pela JVM, no entanto, você pode lançar (*throw*) suas próprias exceções.

Veja como lançar uma **ArithmeticException** manualmente:

```
// Lançando uma exceção manualmente com o throw

class ThrowDemo{
  public static void main(String args[]){
try{
      System.out.println("Antes de lançar a exceção");
      throw new ArithmeticException(); // Lancando a excecao
    }
    catch (ArithmeticException exc){
      //Capturando a exceção
      System.out.println("Antes de lançar a exceção");
    }

    System.out.println("Aqui já estamos fora do bloco try/catch");
    }
}
```

A saída desse programa seria a seguinte:

```
Antes de lançar a exceção

Exceção capturada

Aqui já estamos fora do bloco try/catch

------------------------------------------------------------------------

BUILD SUCCESS

------------------------------------------------------------------------
```

Algumas vezes, pode ser preciso executar uma rotina após uma exceção ser executada, por exemplo, fechar um arquivo, desconectar de um servidor etc. Para esses casos, use o bloco ***finally***.

O bloco ***finally*** será executado sempre que um bloco ***try/catch*** terminar sua execução, tenha ou não ocorrido uma exceção.

Analise o exemplo:

```
//Usando o bloco finally para garantir a execução depois de um try/catch

class UseFinally{
  public static void geraExcecao(int escolha){
    int t;
    int nums[] = new int[2];

    System.out.println("Recebendo" + escolha);

    try{
      switch(escolha){
    case 0:
      t = 10 / escolha; // Gerando o erro de divisão por zero
      break;
    case 1:
      nums[4] = 4; // Gerando o erro de índice fora do escopo
      break;
    case 2:
      return; // Retornando ao bloco try
      }
    }
    catch (ArithmeticException exc){
      //Capturando a exceção
      System.out.println("Divisao por zero detectada");
      return; //Return do catch
    }
    catch (ArrayIndexOutOfBoundsException exc){
      //Capturando a exceção
      System.out.println("Erro de índice fora do escopo");
    }
    finally{
      System.out.println("Saindo do bloco try");
    }

  }
}


class FinallyDemo{
  public static void main(String args[]){
      for (int i=0; i < 3; i++){
    UseFinally.geraExcecao(i);
    System.out.println();
      }
  }
}
```

```
class FinallyDemo{
  public static void main(String args[]){
    for (int i=0; i < 3; i++){
    UseFinally.geraExcecao(i);
    System.out.println();
    }
  }
}
```

A saída desse programa seria o seguinte:

```
Recebendo 0

Divisão por zero detectada



Recebendo 1

Erro de índice fora do escopo



Recebendo 2

Saindo do bloco try

------------------------------------------------------------------------

BUILD SUCCESS

------------------------------------------------------------------------
```

Usando o throws

O **throws** é utilizado sempre que houver a necessidade de, ao executar um método, este (o método) deva tratar essa exceção.

```
public class JavaTester{
  public int division(int a, int b) throws ArithmeticException{
      int t = a/b;
      return t;
  }
  public static void main(String args[]){
      JavaTester obj = new JavaTester();
      try{
        System.out.println(obj.division(15,0));
      }
      catch(ArithmeticException e){
        System.out.println("Voce nao deveria tentar dividir por zero");
      }
  }
}
```

A saída desse programa seria a seguinte:

```
Você não deveria tentar dividir por zero

------------------------------------------------------------------------

BUILD SUCCESS

------------------------------------------------------------------------
```

Encerramento

Neste conteúdo, você pôde aprender que aplicações modernas e sistemas de informação são complicadas estruturas de dados, e que, para contornar e evitar problemas de execução em seu *software*, o tratamento de exceções é

uma prática indispensável.

Você aprendeu que, tratando esses dados, é possível contornar problemas tanto previsíveis quanto imprevisíveis, acelerar a solução do problema, descobrir a origem deste e, não menos importante, salvar evidências de que o problema ocorreu.

Você também viu que problemas de *performance* podem ocorrer devido a falhas não tratadas, além disso, quando essas exceções são devidamente tratadas, garante-se um *software* mais seguro.

Quando você tratar corretamente esses problemas com as ferramentas certas, garantirá a execução desejada e o fluxo correto do seu *software*.

Em suma, exceções fornecem os meios para separar os detalhes do que fazer quando algo fora do comum acontece a partir da lógica principal de um programa.