Programação I

Exceções

Jorge Roberto Trento
Bacharel em Ciências da Computação - UNOESC
Especialização em Ciências da Computação - UFSC
Formação Pedagógica - Formadores de Educação Profissional – UNISUL
Especialização em Ensino Superior – FIE
Especialização em Gestão de Tecnologia da Informação – FIE

Introdução

- O termo exceção refere-se a algo excepcional que aconteceu em determinado contexto, como uma falha de funcionamento;
- Algumas falhas podem ocorrer em softwares, como esgotamento da memória do sistema, *overflow* em uma soma, entre outros;
- Nestas situações, o ideal é que o programa saiba detectar estes erros e tomar as medidas cabíveis para voltar a um estado estável e funcional.

Conceito

- O conceito de exceção é simples: se algum erro acontece, uma exceção é lançada e o código que estava sendo executado para de ser processado.
- A exceção lançada (que é por sua vez um objeto) contém informações referentes ao erro, como uma mensagem que o descreve (ex.: "arquivo teste.txt não existe").

Conceito

- Depois que um exceção é lançada (throw), o sistema (no caso a JVM) tenta encontrar algo que seja capaz de manusear e tratar essa exceção.
- Um método se qualifica a capturar a exceção (*catch*) se ele contiver um código especial para isso. Se ele não possuir esse código, o próximo método na lista de candidatos será analisado. O processo se repete até que alguém apto seja encontrado.

Exemplo

- Se um método for capaz de tratar a exceção, a execução do programa (que havia sido interrompida pela exceção) tem continuidade no exato ponto onde se inicia o código de tratamento da exceção. Se nenhum método for capaz de tratar a exceção, ela irá viajar por toda a lista até chegar na JVM (que foi que iniciou o programa). Nesse caso, a JVM termina o programa e mostra uma mensagem de erro, como a seguinte:

```
Exception in thread "main" java.lang.NoClassDefFoundError: Teste1
Caused by: java.lang.ClassNotFoundException: Teste1
 at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
 at java.security.AccessController.doPrivileged(Native Method)
 at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:307)
 at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:248)
```

Analisando a exceção

- A primeira linha da mensagem mostra que houve uma exceção na thread chamada "main", sendo que o tipo da exceção é java.lang.NoClassDefFoundError (veremos adiante sobre tipos de exceções).
- Abaixo dessa mensagem, há a lista de métodos que foram percorridos, sendo que nenhum deles foi capaz de tratar a exceção.
- O método que causou a exceção está indicado na terceira linha (método run() da classe URLClassLoader , que está na linha 202 do arquivo URLClassLoader.java). Os métodos abaixo desse foram os métodos testados (na ordem de cima para baixo) em busca de alguém apto a tratar a exceção.

```
Exception in thread "main" java.lang.NoClassDefFoundError: Teste1
Caused by: java.lang.ClassNotFoundException: Teste1
 at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
 at java.security.AccessController.doPrivileged(Native Method)
 at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:307)
 at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:248)
```

Tratar as exceções

- O código especial (chamado tratador de exceção) que a JVM busca para que um método seja capaz de tratar uma exceção é o bloco try...catch . O exemplo abaixo ilustra a utilização do bloco:

```
try {
    // código(s) que pode(m) gerar exceção(ões)
} catch () {
    // código para tratar a exceção
}
```

Exemplo Try - Catch

- O bloco é composto por duas partes importantes: o try (que contém os códigos que podem gerar exceção) e o catch (que contém o código que será executado para tratar alguma exceção levantada dentro do try). Abaixo há um exemplo de código:

```
private List<Integer> lista;
PrintWriter p = null;

try {
      p = new PrintWriter(new FileWriter("teste.txt"));
      for (int i = 0; i < 5; i++) {
            out.println("Valor em: " + i + " = " + lista.get(i));
      }
} catch (IOException e) {
      System.err.println("Pegamos uma exceção: " + e.getMessage());
}

System.out.println("Continuando...");
```

Exemplo Try - Catch

- No código anterior, o construtor da classe PrintWriter pode levantar uma exceção para indicar algum erro, como impossibilidade de ler do disco. Se esse erro acontecer, o construtor jogará a exceção e a execução do código será interrompida imediatamente. Isso quer dizer que o for() (abaixo da instanciação do objeto p) não será executado.
- Como a exceção foi jogada dentro de um bloco try , ela será pega pelo catch no momento que for jogada, e o código do catch será executado (no caso, apenas um println() dizendo que pegamos uma exceção);
- A JVM não precisa procurar em um método anterior pelo tratador de exceção, porque o código atual está apto a fazer esse tratamento. A exceção poderá ser manipulada através do objeto e (declarado dentro do catch), que contém as informações da exceção jogada pelo construtor da classe PrintWriter .

Exemplo Try - Catch

- Se o construtor da classe PrintWriter não levantar uma exceção, o código do try será executado inteiramente: o objeto p será instanciado e o for() será executado normalmente. Quando ele terminar, o código do programa seguirá executando (no caso, o println("Continuando...") será executado). Nesse caso, o código dentro do catch não será executado, visto que nenhuma exceção foi jogada.
- O bloco try...catch é o responsável por conter e tratar uma exceção jogada (o código dentro do try pode jogar a exceção, sendo que o catch a pega e trata).
- Se a exceção for jogada e não houver um try...catch envolvendo o método que gerou a exceção, ela será jogada novamente para o método anterior (que foi quem chamou o método atual). Se esse método não possuir um try...catch também, a exceção será jogada novamente para o próximo método.
- Se ninguém tratar a exceção, ela cairá no try...catch mais externo existente, que é o da JVM (que em seu catch apenas imprime o erro e encerra o programa).

Tratamento múltiplo

- Determinados métodos podem jogar mais de um tipo de exceção, especialmente quando desempenham tarefas complexas. Um exemplo é o método que obtém um elemento de uma lista, que pode gerar as seguintes exceções (exemplo): elemento não encontrado, lista vazia ou parâmetro de busca incorreto (esperado int e um float foi dado, por exemplo).
- Quando o programa deve reagir de forma diferente para cada uma dessas possíveis exceções, é conveniente utilizar um try com vários catchs (um para cada exceção a ser capturada).
- O código abaixo mostra o mesmo exemplo anterior, com a diferença que há mais de uma exceção sendo capturada.

Exemplo – Tratamento Múltiplo


```
private List<Integer> lista;
PrintWriter p = null;

try {
      p = new PrintWriter(new FileWriter("teste.txt"));
      for (int i = 0; i < 5; i++) {
            out.println("Valor em: " + i + " = " + lista.get(i));
      }
} catch (FileNotFoundException e) {
      System.err.println("Arquivo não encontrado! " + e.getMessage());

} catch (IOException e) {
      System.err.println("Pegamos uma exceção: " + e.getMessage());
}

System.out.println("Continuando...");
```

Exemplo – Tratamento Múltiplo

- Se o construtor da classe PrintWriter não conseguir encontrar o arquivo, ele irá jogar uma exceção do tipo FileNotFoundException , que será pega e tratada pelo primeiro catch .
- O primeiro catch será utilizado porque a JVM irá comparar o tipo do objeto da exceção jogada com o tipo informado no catch (através do uso de instanceof).
- Se o objeto da exceção jogada for uma instância da classe indicada no catch , então esse catch será usado, caso contrário o próximo catch será testado. Se nenhum catch conseguir pegar a exceção, um erro de compilação será gerado.
- É importante notar que o teste é feito com **instanceof** e que todos os conceitos de polimorfismo se aplicam aqui.
- Isso quer dizer que se FileNotFoundException herda de IOException , por exemplo, então qualquer objeto da classe FileNotFoundException também é uma instância de IOException.

Exceções no Java

- A linguagem Java disponibiliza diversas classes que são exceções prontas para uso, como FileNotFoundException .
- Para ser jogada como uma exceção, a classe precisa herdar a classe Throwable(jogável) - que é a superclasse de todas as exceções).
- Dessa forma, se o programador deseja pegar qualquer exceção levantada dentro do try (independente do seu tipo), basta utilizar uma classe muito genérica no catch , como é o caso da Throwable ou da Exception (que herda de Throwable também).

Bloco Finally

- O bloco finally é utilizado como complemento do try...catch . Ele sempre é executado, independente de ter havido ou não alguma exceção. Ele é opcional e é utilizado da seguinte forma:

```
try {
} catch (IOException e) {
} catch (Exception e) {
} finally {
    // código que será sempre executado, independente
    // de haver ou não exceções
}
```

Exemplo Bloco Finally

- O bloco finally sempre é executado, independente de ter havido ou não alguma exceção. Uma utilidade para esse é bloco é evitar duplicação de código entre o try e o catch .
- Por exemplo, se dentro do try houver um código para abrir um arquivo (ex.: file.open()), esse arquivo deve ser fechado depois que for usado.
- Se o comando para fechar o arquivo estiver no final do try , ele será executado se não houver exceções. Se alguma exceção for jogada, pode ser que o fechamento do arquivo não seja feito, porque a execução do código do try será interrompida.
- Para garantir que o arquivo seja fechado, o comando para fechar o arquivo deverá ser colocado também dentro do catch. Dessa forma, o arquivo é fechado se houver e se não houver exceções.

Sem finally *versus* com finally


```
File f = new File();
f.open("teste.txt");
try {
    // manipula o arquivo...
    // faz outras coisas...
    f.close();
} catch (Exception e) {
    f.close();
}
```

```
File f = new File();
f.open("teste.txt");
try {
} catch (Exception e)
} finally {
    f.close();
}
```

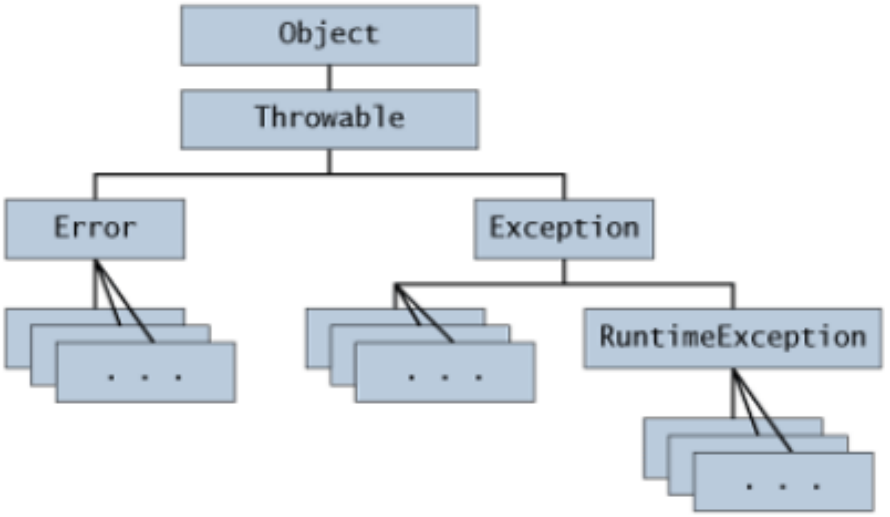
Jogando (ou lançando) exceções

- O programador pode jogar uma exceção em qualquer momento do código, basta utilizar o comando throw seguido de um objeto jogável (que herde de Throwable ou suas subclasses), conforme o exemplo abaixo.

throw new Exception();

- Conforme já mencionado, a linguagem Java possui diversas exceções prontas para serem utilizadas. O diagrama abaixo mostra a hierarquia de classes relacionadas a exceções.

Jogando (ou lançando) exceções



- No topo da hierarquia, encontra-se a classe **Object** , superclasse de todas as classes Java. Abaixo encontra-se a classe **Throwable** , que é a superclasse de qualquer classe que pode ser jogada através do comando throw .

Jogando (ou lançando) exceções

- As subclasses da classe Error (à esquerda na hierarquia) geralmente indicam algum erro muito grave e completamente anormal, possivelmente associado ao mau funcionamento da JVM.
- A classe RuntimeException e suas subclasses indicam erros que acontecem em tempo de execução, algo que não poderia ter sido previsto em tempo de compilação. Ambas as classes Error e RuntimeException (e suas subclasses) são ditas exceções não checadas (ou não verificadas), que indicam que o programador não é obrigado a utilizar um try...catch para conter qualquer código que possa levantar esse tipo de exceção. Além disso, se um método levanta esse tipo de exceção, essa informação não precisa estar especificada na declaração do método.
- A classe Exception (e suas subclasses, excluindo-se a RuntimeException e suas subclasses), por sua vez, é uma exceção checada (ou verificada), o que quer dizer que qualquer código que levante esse tipo de exceção deve ser envolvido por um try...catch . Além disso, na declaração do método que levanta a exceção, é necessário que o programador informe qual o tipo de exceção (ou exceções) que o método levanta.

Jogando (ou lançando) exceções

- Se um método jogar uma exceção do tipo Exception , ele precisa explicitamente informar isso, caso contrário o código não compilará. Para informar isso, utiliza-se a palavra-chave throws ao lado do método (e antes de seu código), como mostra o exemplo abaixo.

```
public void escreveInfoArquivo() throws IOException {
    PrintWriter out = new PrintWriter(new FileWriter("teste.txt"));
    out.print("Meu arquivo teste");
    out.close();
}
```

Jogando (ou lançando) exceções

- No exemplo anterior, o construtor da classe PrintWriter pode levantar uma exceção do tipo IOException (ou derivado) para indicar algum erro. Como a classe IOException herda de Exception , ela é uma exceção checada, então tem-se duas opções:
- 1) envolver a instanciação do objeto out com um try...catch (para pegar e tratar a exceção) ou
- 2) repassar a exceção adiante para ser tratada pelo método que chamou o método escreveInfoArquivo() .
- Se a solução 1 for escolhida, o método não precisa dizer que joga uma exceção, porque a exceção está sendo pega e tratada dentro do próprio método. Se a solução 2 for utilizada, então o método precisa informar que joga uma exceção; sendo assim, ele precisa utilizar a palavra-chave throws e informar qual tipo de exceção joga (IOException , no caso do exemplo).
- Em determinados casos, um método pode levantar mais de uma exceção. Se elas forem exceções checadas, o método precisa informar que levanta todas essas exceções, como no exemplo abaixo (as exceções levantadas foram colocadas uma embaixo da outra por questões de legibilidade).

Jogando (ou lançando) exceções


```
public void fazAlgo() throws ExcecaoParamErrado,
ExcecaoOrdemErrada, ExcecaoImpossivelConverter{
    // código do método...
    if(teste()) {
        throw new ExcecaoParamErrado();
    }
    // ... mais código
    if(a == 1) {
        throw new ExcecaoOrdemErrada();
    }
    // ... mais códig
    throw new ExcecaoImpossivelConverter();
}
```

Jogando (ou lançando) exceções

- Da mesma forma que um bloco catch pode utilizar uma classe alta na hierarquia para poder pegar todas as exceções (independente do seu tipo), um método pode especificar que joga apenas um tipo de exceção (uma classe genérica, como Exception), ao invés de informar que joga diversas exceções.
- Utilizando-se o código do exemplo anterior como base e imaginando que ExcecaoParamErrado , ExcecaoOrdemErrada e ExcecaoImpossivelConverter herdam da classe Exception , o método poderia ser reescrito da seguinte forma:

Jogando (ou lançando) exceções


```
public void fazAlgo() throws Exception {
// código do método...
if(teste()) {
    throw new ExcecaoParamErrado();
    }
// ... mais código
    if(a == 1) {
    throw new ExcecaoOrdemErrada();
    }
    // ... mais código
throw new ExcecaoImpossivelConverter();
}
```

Exercícios

- Faça uma classe Matemática que tenha métodos para somar, subtrair, dividir e multiplicar dois números informados via teclado. Se qualquer um dos dois números informados for zero, o método levanta exceção **ParamZeroException**. Se o resultado da operação for zero, o método não retorna o valor, mas lança a exceção **ResultadoZeroException**.

Exercícios

- Faça uma classe que tenha um método estático que some os inteiros de um vetor num determinado intervalo. Se o intervalo for inválido, levanta uma exceção **IntervaloException**. Se o vetor estiver todo preenchido com zeros, levanta exceção **VetorVazioException**.

Tratamento de Exceções

Autor
Prof. Doglas André Finco
doglas.andref@uffs.edu.br

Jorge.trento@uffs.edu.br