ACH 2003 - Computação Orientada a Objetos

Mecanismo para tratamento de exceções

Prof. Flávio Luiz Coutinho
flcoutinho@usp.br

Mecanismo para tratamento de exceções

É um mecanismo através do qual o código em execução dentro de um método pode sinalizar para quem o chamou (direta ou indiretamente) que algo atípico ocorreu.

Mecanismo para tratamento de exceções

É um mecanismo através do qual o código em execução dentro de um método pode sinalizar para quem o chamou (direta ou indiretamente) que algo atípico ocorreu.

Quando um método lança uma exceção, de um modo geral, o compilador nos obriga a tratá-la de alguma forma. Isso reduz a chance de nós, programadores, negligenciarmos o tratamento de situações atípicas que podem ocorrer durante a execução de um programa.

Mecanismo para tratamento de exceções

É um mecanismo através do qual o código em execução dentro de um método pode sinalizar para quem o chamou (direta ou indiretamente) que algo atípico ocorreu.

Quando um método lança uma exceção, de um modo geral, o compilador nos obriga a tratá-la de alguma forma. Isso reduz a chance de nós, programadores, negligenciarmos o tratamento de situações atípicas que podem ocorrer durante a execução de um programa.

Vamos assumir, por hora, que todas as exceções devem ser tratadas...

Exemplo: programa para cálculo da média de 3 notas

Sem uso de exceções (versão zero):

- lógica principal misturada com lógica para tratamento de erros.
- Para cada operação realizada, verifica-se se a mesma foi bem sucedida.
- O programa só prossegue a execução em caso positivo.
- Há mais código para tratamento de potenciais erros, do que código que faz o trabalho principal propriamente dito.
- Ao mesmo tempo, espera-se que tais instruções para tratamento de erro executem pouquíssimas vezes, pois a expectativa é que situações problemáticas ocorram apenas raramente.

Exemplo: programa para cálculo da média de 3 notas

Sem uso de exceções (versão 1):

- Delegamos a verificação do valor lido ao próprio método **leDouble.** Mas…
- A mistura de código responsável pela lógica principal com lógica para tratamento de erro ainda persiste.
- Mais ainda: para permitir que o tratamento seja feito pelo método **media**, precisamos sobrecarregar o papel do retorno do método **leDouble** (uso de *magic number* para sinalizar que há algum problema).
- Por outro lado, deixar o tratamento da situação problemática para o próprio **leDouble** não parece fazer muito sentido também (o papel dele é informar o problema, não tratá-lo. Quem deve lidar com o problema é quem chama o método).

Exemplo: programa para cálculo da média de 3 notas

Com uso de exceções (versões 2 3 e 4):

- O método **leDouble** cria e lança uma exceção quando se depara com um valor lido fora do intervalo esperado.
- O lançamento da exceção interrompe imediatamente a execução da chamada a **leDouble** (ou seja, a execução não finaliza normalmente, e nada é devolvido através do return).
- A exceção “chega” até o método **media** (em execução). A chegada da exceção interrompe a execução das instruções definidas no bloco **try**, e o fluxo de execução é desviado para o bloco **catch** (que está apto a capturar a exceção que foi lançada). Após a execução do bloco **catch**, a execução prossegue a partir da instrução seguinte a este bloco.

Exemplo: programa para cálculo da média de 3 notas

Com uso de exceções (versões 2 3 e 4), temos os seguintes benefícios:

- No método **media**, há separação total da lógica principal (código dentro do bloco **try**), da lógica responsável por lidar com as situações problemáticas (bloco **catch**).
- O compilador informa da possibilidade de a exceção ocorrer durante uma chamada a **leDouble**, e “nos obriga” a implementar o que deve ser feito, caso ela ocorra. Isso reduz a chance de um erro ser negligenciado (se, na versão 1, deixássemos de verificar o retorno do método **leDouble** por simples desconhecimento, nós não seríamos avisados do problema em potencial).

Ponto de vista do **método que lança** uma exceção:

Um método que pode eventualmente lançar uma exceção quando algo “dá errado”, deve declarar a possibilidade de lançá-la (throws).

Quando a situação problemática é detectada em tempo de execução, um objeto que encapsula as informações sobre o problema é instanciado, e lançado (throw).

O lançamento de uma exceção finaliza imediatamente a execução do método que a lança. O objeto que encapsula as informações da exceção é “devolvido por uma via alternativa” para o método anterior na cadeia de chamadas.

Ponto de vista do **método que recebe** a exceção:

Quando uma exceção “chega” a um método, este deve (em geral):

- **Capturar e tratar** a exceção (blocos catch, associados a um bloco try)

ou

- **Deixar a exceção “seguir seu caminho”**, passando para o método anterior na cadeia de chamadas. Dizemos que, neste caso, o método em questão lança a exceção de forma indireta e, mesmo neste caso, o método deve indicar que isso pode acontecer através do throws.

Independente da escolha, em ambos os casos se está ciente de que a exceção pode ocorrer, e alguma atitude quanto a isso é tomada.

Benefícios

- Potenciais situações de erro são negligenciadas com menor chance.
- Separação do código relativo à lógica principal da lógica para tratar situações excepcionais e/ou de erro. Evita-se a escrita de código que verifica o sucesso/falha de cada operação executada.
- Flexibilidade no tratamento das situações excepcionais:
  - Para definir qual método dentro da cadeia de chamadas deve ser o responsável pelo tratamento do problema.
  - É possível dividir a responsabilidade pelo tratamento entre mais de um método.
  - É fácil discriminar os tipos de exceção que ocorrem e quem trata cada tipo (tratamentos específicos para cada tipo de exceção).

Checked vs. Unchecked exceptions

- Nem todas as exceções precisam ser, obrigatoriamente, tratadas.
- A obrigatoriedade se aplica apenas a um subconjunto de todos os objetos que podem ser lançados:
  - Derivados de Exception, mas não de RuntimeException.
  - Exceções derivadas de RuntimeException não precisam ser tratadas (obrigatoriamente), mas podem ser eventualmente.
  - Erros (que funcionam da mesma forma que exceções) também não precisam ser obrigatoriamente tratados (mas podem ser eventualmente).