Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Coordenação de Pós-Graduação em Informática - COPIN

PROJETO DE SOFTWARE ORIENTADO A OBJETO

# RESUMO - EXCEPTIONS

Maio de 2008

João Arthur Brunet Monteiro
Mestrado em Ciências da Computação - CEEI/UFCG
Campina Grande, Maio de 2008

# 1 Resumo

Este documento tem por objetivo apresentar alguns conceitos a respeito de exceções que demandam tratamento explícito em tempo de compilação (*checked exceptions*) ou não (*unchecked exceptions*). Além disso apresentarei a minha opinião sobre o assunto, levando em consideração qual desses dois tipos de exceções deve ser usado na construção de um software.

Várias linguagens de programação possuem mecaninsmos para lidar com exceções durante a construção e execução do programa. No entanto, segundo Eckel [1], Java é a única a usar explicitamente *checked exceptions* na especificação de contratos (e. g., assinatura de métodos). A Sun [2] recomenda o uso de *unchecked exceptions* somente para erros de programação, tais como, divisão por zero ou acesso a objetos cuja referência é nula.

No entanto, na literatura foram encontradas várias opiniões contrárias ao tutorial da Sun. Eckel acredita que o uso de *checked exceptions* é problema pois obriga, através do compilador, o programador a poluir o código com tratamento de erros que ele pudesse talvez não querer tratar. Como exemplo que se contrapõe a Java, o autor relata que Python também possui exceções e que a escolha de tratar ou não é de responsabilidade do programador.

Em relação a este ponto, acredito que algumas exceções em Java não deveriam ser checadas em tempo de compilação, como exemplo, a exceção IOException, que denota erro de entrada/saída. Acredito que, neste caso, não é responsabilidade do programador tratar este tipo de erro, pois pouco pode ser feito se ele ocorrer.

Waldhoff [4], entre outros autores [3], também mostram uma visão contrária ao uso de exceções que devem ser checadas em tempo de compilação. Waldhoff acredita que quanto maior a distância entre o método que lançou a exceção e o método realmente interessado em tratar essa exceção, maior será a poluição do código com tratamentos ou lançamentos de exceções desnecessários. Isto porque todo método que estiver entre o método que lança a exceção e o método interessado em tratá-la terá que lança-la adiante. Neste caso, concordo com o autor que a poluição de código gerada é desnecessária, por isso se faz necessário nesta situação o uso de *unchecked exceptions*.

Particularmente acredito que o uso de *checked exceptions* seja benéfico para aumentar a clareza do contrato exposto por um método, além de tornar visível para o cliente do método quais são os possíveis erros relacionados a execução do mesmo. De fato, *checked exceptions* obrigam os programadores a pensarem em tratamento de erro, enquanto que *unchecked exceptions* dão uma maior liberdade para os mesmos.

Defendo que o uso de *checked exceptions* deve ser aplicado para erros

relacionados a lógica de negócio do sistema, quebras de contrato etc. Esta opinião é baseada no fato de que acredito ser importante para os programadores terem ciência dos erros relacionados a lógica de negócio em tempo de compilação, uma vez que a especificação desses erros fazem parte da especificação do sistema.

# Referências


[1] E. B. Does java need checked exceptions? http://www.mindview.net/etc/.

[2] S. Microsystems. Unchecked exceptions - the controversy. the java tutorials. http://java.sun.com/docs/books/tutorial/essential/exceptions/runtime.html, 2008.

[3] B. Trost. Checked exceptions are of dubious value: http://www.c2.com/cgi/wiki?checkedexceptionsareofdubiousvalue, 2007.

[4] R. Waldhoff. Java's checked exceptions were a mistake (and here's what i would like to do about it): http://radio.weblogs.com/javascheckedexceptionswereamistake.html, 2003.