Material sobre Exceções[1]

Este material é complementar aos slides de "Exceções". É preciso visitar este conteúdo para reconhecer os conceitos e exemplos usados neste material.

Lançar ou disparar uma exceção é a maneira mais eficaz que um objeto tem para indicar que não é capaz de atender a uma solicitação do cliente da classe. Uma das principais vantagens é que é quase impossível que o cliente da classe ignore o fato de que uma exceção foi lançada e mesmo assim prossiga. O não-tratamento de uma exceção pelo cliente fará com que a aplicação termine imediatamente.

Há várias formas de lançar exceções. Nos slides de aula sobre o assunto, foi visto como capturar e tratar uma exceção que foi lançada usando o bloco `try-catch`. Aqui veremos outras formas de como uma exceção pode ser lançada.

1 Lançando uma exceção

Há duas etapas para lançar uma exceção. Primeiro um objeto de um tipo de exceção é criado com o comando `new`; em seguida, o objeto é lançado usando a palavra-chave `throw`. Essas duas etapas são combinadas em uma única instrução:

```
throw new TipoDeExceção("string de diagnostico optional");
```

Quando um objeto exceção é criado, uma **string de diagnóstico**, ou seja, um texto descritivo do problema que ocorreu, pode ser passada para seu construtor. Essa string torna-se, mais tarde, disponível para o receptor da exceção via o método de acesso `getMessage()` do objeto exceção ou seu método `toString()`. O método a seguir está lançando uma exceção para indicar que passar um valor *null* não faz sentido.

```
/**
* Configura um comentário para este item.
* @param comentario O comentario que será adicionado.
* @throws NullPointerException se o comentário for nulo.
*/
public void setComentario(String comentario) {
    if (comentario == null) {
        throw new NullPointerException("Parâmetro null em setComentario.");
    }
    this.comentario = comentario;
}
```

[1]Textos extraídos do livro "Programação orientada a objetos com Java" de Michael Kolling e David J. Barnes, capítulo 12.

Observe que na documentação foi incluído detalhe sobre quaisquer exceções que ele lança, utilizando a tag *@throws* da javadoc.

O código anterior mostra apenas o lançamento de uma exceção, mas o que acontece quando uma exceção é lançada? Na realidade, há dois efeitos a considerar: o efeito no método em que a exceção é lançada e o efeito no chamador.

Quando uma exceção é lançada, a execução do método de lançamento termina imediatamente - ele não continua até o final do corpo do método. Por causa disso, o compilador indicará um erro se quaisquer instruções forem escritas após uma instrução `throw`, porque elas nunca poderiam ser executadas.

O efeito de uma exceção sobre o ponto no programa que chamou o método é um pouco mais complexo. Em particular, o efeito total depende ou não se algum código foi escrito para capturar a exceção. Se não for capturada, o programa então simplesmente terminará com uma indicação de lançamento de exceção.

2 Tratamento de exceções

Os princípios do lançamento de exceções se aplicam de maneira idêntica a exceções não-verificadas e às verificadas (ver slides de aula sobre esta classificação), mas as regras particulares do Java fazem com que o tratamento de exceções torne-se um requisito obrigatório somente com exceções verificadas. Nestes casos, o compilador impõe verificações tanto ao método que lança uma exceção verificada quanto ao chamador desse método.

2.1 Exceções verificadas: cláusula throws

O primeiro requisito do compilador é que um método que lança uma exceção verificada deve declarar que faz isso em uma cláusula *throws* adicionada ao cabeçalho do método. Por exemplo, um método que lança uma `IOException` verificada poderia ter o seguinte cabeçalho:

```
public void salvarNoArquivo(String arquivoDestino) throws IOException
```

É possível utilizar uma cláusula `throws` para exceções não-verificadas, porém o compilador não exige uma.

2.2 Capturando exceções: instrução try

O segundo requisito é que o chamador de um método que lança uma exceção verificada deve estar preparado para lidar com a exceção. Lembrando que **uma exceção que é lançada precisa ser capturada e tratada**. Isso normalmente significa escrever uma rotina de tratamento de exceção na forma de um bloco *try/catch* (já visto nos slides de aula).

No caso do exemplo feito na seção anterior, a chamada ao método `salvarNoArquivo()` deve ser feita dentro de um bloco *try/catch* para que seja capturada e tratada como mostra o exemplo a seguir:

```
try{
   info.salvarNoArquivo(nomeArquivo);
   falhou = false;
} catch (IOException e){
   System.out.println("Não foi possível salvar no " + nomeArquivo);
   falhou = true;
}
```

No caso do exemplo feito na Seção 1, a chamada ao método `setComentatio()` deve ser feita da seguinte forma:

```
try {
    String comentario = leitor.next();
    item.setComentario(null);
} catch (NullPointerException e) {
    System.out.println("O comentário não pode ser nulo.");
}
```

Nos dois casos, se a exceção for lançada, o fluxo do código vai para uma ação que notifica o usuário do método sobre o erro, e o programa pode continuar com a ação mais apropriada, a ser definida pelo programador.

3 Definindo novas classes de exceção

Onde as classes de exceção padrão não descrevem satisfatoriamente a natureza do problema, novas classes de exceção mais descritivas pode ser definidas utilizando a herança. Novas classes de exceção verificadas podem ser definidas como subclasses de qualquer classe de exceção verificada existente (como `Exception`) e novas exceções não-verificadas seriam subclasses na hierarquia `RuntimeException`, como pode ser visto na Figura 1.

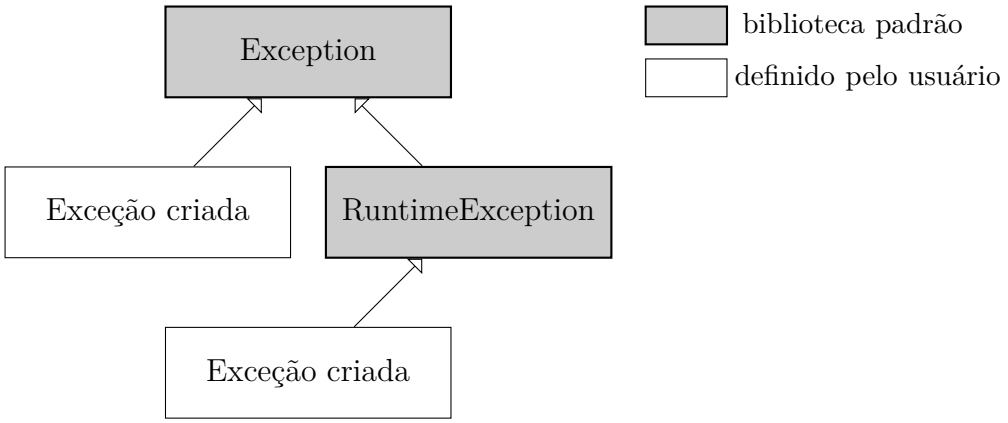

Figura 1: Hierarquia de exceções

Todas as classes de exceção existentes suportam a inclusão de uma string de diagnóstico passada para o construtor. Entretanto, uma das principais razões para definir novas classes de exceção é incluir mais informações dentro do objeto exceção para suportar um diagnóstico de erro e uma recuperação após erro.

Uma nova classe de exceção pode ter normalmente quatro construtores, sendo que os dois primeiros são os mais utilizados.

- Um construtor que não recebe argumentos e faz uma chamada ao construtor da superclasse que exibe uma mensagem de erro padrão;
- um que recebe uma string de diagnóstico personalizada e a passa para o construtor da superclasse;
- um que recebe uma string de diagnóstico personalizada e um objeto `Throwable` (para encadear exceções) e passa ambas para o construtor da superclasse, e
- um que recebe um objeto `Throwable` e a passa para o construtor da superclasse.

A seguir um exemplo da criação de uma nova classe de exceção, com a inclusão dos quatro construtores:

```
public class ImparException extends RuntimeException{
    public ImparException(){
        super();
    }
    public ImparException(String m){
        super(m);
    }
    public ImparException(String m,Throwable e){
        super(m,e);
    }
    public ImparException(Throwable e){
        super(e);
    }
}
```

É possível adicionar mais métodos para que a exceção fique mais personalizada como, por exemplo, a sobrescrita do método `toString()` que deve retornar um string com a descrição da classe e com informações apropriadas para ajudar a diagnosticar a razão do erro.