Tratamento de exceções

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

Protegendo seu programa de entradas inválidas

- Antigo paradigma: *"Garbage in, garbage out"* (lixo entra, lixo sai).
  - Não é suficiente para um software de produção.
  - Programa nunca deve produzir lixo, independentemente da entrada.
- Novos paradigmas de proteção (**Princípios de McConnell**[1]):
  - *"Garbage in, nothing out"* (lixo entra, nada sai).
  - *"Garbage in, error message out"* (lixo entra, mensagem de erro sai).
  - *"No garbage allowed in"* (nenhum lixo é permitido entrar).

[1] MCCONNELL, Steve. *Code complete: um guia prático para a construção de software*, 2005. Bookman.

Princípios de McConnell

- **Verifique dados de todas as fontes externas.**
  - Arquivos, usuários, rede...
  - Faixas de valores para tipos numéricos (int, float, ...);
  - Formato de texto em Strings (comprimento, valores restritos, ...).
- **Verifique parâmetros nas chamadas de métodos.**
  - Semelhante ao anterior, mas dados vêm de outras rotinas.
  - Evitar propagação de valores incorretos.
  - Testar o parâmetro dentro da função.
- **Decida como tratar entradas com problema.**
  - Detectado o parâmetro inválido, decida o que fazer.
  - Diferentes abordagens se adequam a diferentes situações.

Robustez × corretude

- Sempre é melhor ser totalmente robusto, certo?
  - Manter o programa funcionando a qualquer custo.
  - Um editor de texto que se fecha caso você dê *backspace* em um documento vazio.
  - Um visualizador de imagens que termina caso haja algum problema com uma foto digital.
  - Um MP3 player que para de funcionar se o arquivo não está completo
- Nem sempre: um sistema de impressão de raio-X digital. Caso um valor esperado não seja encontrado. . . Manter o sistema funcionando e deixar de imprimir?

Robustez

Robustez

Habilidade do software em reagir apropriadamente a situações anormais.

Programação por contrato.

- Método chamado:
  - ou executa,
  - ou falha.
- Falha: situação excepcional
  - Tratamento varia com o tipo de erro.
  - Pode-se produzir uma `Exception` ou `Error` em Java.

Tratamento de Exceções

- Vantagens
  - Separa tratamento do erro do código normal.
  - Propaga erros na pilha de chamada de funções.
  - Agrupa e diferencia tipos de erros.
- Modelo de programação:
  - Em caso de situação anormal, programa lança (`throw`) uma exceção.
  - Bloco de comandos em que a exceção foi lançada tem duas opções:
    - Capturar (`catch`) e tratar a exceção.
    - Propagar a exceção para o bloco que o chamou (`throws`).
  - Finalmente (`finally`) executa código invariante, que deve ser executado independentemente se a execução foi bem sucedida ou se houve exceção.

Código sujo: sem exceções

```
int LeArquivo {
    int codigoErro = 0;
    AbraArquivo();
    if (ArquivoFoiAberto) {
        ObtenhaTamanhoArquivo();
        if (TamanhoFoiObtido) {
            AloqueMemoria();
            if (MemoriafoiAlocada) {
                LeArquivoNaMemoria();
                if (LeituraFalhou) { codigoErro = -1; }
            } else {  codigoErro = -2; }
        } else {  codigoErro = -3;
        }
        FecheArquivo();
        if (ArquivoNaoFechou && errorCode == 0) {
            codigoErro = -4;
        } else { codigoErro = codigoErro and -4;
        }
    } else { codigoErro = -5; }
    return codigoErro;
}
```

Código limpo: com exceções

```
LeArquivo {
    try {
      AbraArquivo;
      ObtenhaTamanhoArquivo;
      AloqueMemoria;
      LeArquivoNaMemoria;
      FecheArquivo;
    } catch (FalhaAberturaArquivo) {
        FaçaAlgumaCoisa;
    } catch (FalhaObtencaoTamanhoArquivo) {
        FacaAlgumaCoisa;
    } catch (FalhaAlocacaoMemoria) {
        FacaAlgumaCoisa;
    } catch (FalhaLeArquivo) {
        FacaAlgumaCoisa;
    } catch (FalhaFechamentoArquivo) {
        FacaAlgumaCoisa;
    }
}
```

Hierarquia de exceções em Java

- *Checked Exceptions*
  - Não descendem de `RuntimeException`
- *Unchecked Exceptions*
  - Descendem de `RuntimeException`
- Compilador verifica as *Checked Exceptions*
- Programador tem duas alternativas
  - Trata a exceção (`try/catch`)
  - Delega a exceção (`throws`)
- A classe `Trowable` é a raiz da hierarquia de classes de exceções.
- A classe `Exception` é uma extensão de `Trowable`. Normalmente novas exceções estendem de `Exception`.
- Classe `Exception` possui apenas uma `String` para armazenar a mensagem de erro de uma exceção.

Tratamento de Exceções em Java

Quatro passos devem ser aprendidos:

- Como criar sua própria exceção?
- Como lançar uma exceção?
- Como propagar uma exceção?
- Como capturar e tratar uma exceção?

Criando Tipos de Exceções

- Exceção deve estender `Exception`.

Por exemplo:

```
public class ExcecaoListaCheia extends Exception {
    public ExcecaoListaCheia() {
        super("A lista está cheia.");
    }
}
```

Lançando uma exceção

- Exceções são lançadas pela cláusula `throw`.

Por exemplo:

```
public void adicionar(Object o) {
    if (this.tamanho() == MAX)
        throw new ExcecaoListaCheia();
}
```

- A cláusula `throws`.
  - Métodos devem declarar qual tipo de exceção ele pode lançar.
  - Pode-se usar uma lista de exceções separadas por vírgula.
  - Só é possível se lançar uma exceção se esta foi previamente declarada na cláusula `throws`.

Fluxo de execução de código

```
void método() {
    try {
        código 1;

        // ———————— excecao EX lançada ————————

        código 2; // não será executado.
    } catch (EX e) {
        código 3; // irá capturar a exceção EX.
    } finally {
        código 4; // será sempre executado.
    }
    código 5; // não será executado, caso seja lançada uma exceção
inesperada que não esteja sendo tratada por um bloco catch.
}
```

Tratando exceções

- O corpo de `try` é executado até uma exceção ser lançada ou até finalizar com sucesso.
- Caso ocorra uma exceção a cláusula `catch` que trata aquele tipo de exceção é executada.
- Se houver cláusula `finally`, seu código será executado no final de tudo.
- Cláusulas `finally` são executadas com ou sem a ocorrência de exceções.
  - São especialmente úteis para atividades de limpeza.

Exemplo: Exceção de estoque

```
public class ExcecaoEstoqueExcedido extends Exception {
   private int quant;

   public int getQuant() {
      return quant;
   }

   public ExcecaoEstoqueExcedido(int quant, int max) {
      super("O estoque de " + quant + " excedeu o limite de "
            + max + ".");
      this.quant = quant;
   }
}

public class ExcecaoEstoqueNegativo extends Exception {

   public ExcecaoEstoqueNegativo() {
      super("O estoque deve possuir um valor positivo.");
   }
}
```

Exemplo: Exceção de estoque

```
public class Produto {
    ...
    public void setQuant(int q) throws ExcecaoEstoqueNegativo,
                                       ExcecaoEstoqueExcedido {
        if (q < 0)
            throw new ExcecaoEstoqueNegativo();
        else if (q > MAX_ESTOQUE)
            throw new ExcecaoEstoqueExcedido(q, Produto.MAX_ESTOQUE);
        else quant = q;
    }
    ...
    public Produto(String d, float p, int q, LocalDateTime f) throws
                    ExcecaoEstoqueNegativo, ExcecaoEstoqueExcedido {
      setDescricao(d);
      setPreco(p);
      setQuant(q);
      setDataFabricacao(f);
      id = ++cont;
      instancias++;
    }
}
```

Exemplo: Exceção de estoque

```
public class BemDuravel extends Produto {
    ...
.    public BemDuravel(String d, float p, int q, LocalDateTime f, int g)
                    throws ExcecaoEstoqueNegativo, ExcecaoEstoqueExcedido {
        super(d, p, q, f);
        setMesesGarantia(g);
    }
}

public class BemDeConsumo extends Produto {
    ...
    public BemDeConsumo(String d, float p, int q, LocalDateTime f, LocalDate v)
                        throws ExcecaoEstoqueNegativo, ExcecaoEstoqueExcedido  {
        super(d, p, q, f);
        setDataValidade(v);
    }
}
```

Exemplo: Exceção de estoque

```
´public class Aplicacao {}
   public static void main(String args[]) {
      try {
         Estoque estoque = new Estoque();

         adicionarProduto(estoque);
         adicionarProduto(estoque);
         ...
         estoque.adicionar(
            new BemDeConsumo("Leite", 4.00F, 120,
                LocalDateTime.now(), LocalDate.now().plusMonths(6)));
         ...
         p.setQuant(p.getQuant() + 200);

         ...
      } catch (ExcecaoEstoqueExcedido e) {
         JOptionPane.showMessageDialog(null, e.getMessage(), "Erro de estoque excedido",
                                       JOptionPane.ERROR_MESSAGE);
         e.printStackTrace();
      } catch (ExcecaoEstoqueNegativo e) {
         JOptionPane.showMessageDialog(null, e.getMessage(), "Erro de estoque negativo",
                                       JOptionPane.ERROR_MESSAGE);
         e.printStackTrace();
      } finally {
         System.out.println("Sempre executado.");
      }
   }
}
```

Aspectos de desempenho

- Exceções devem ser evitadas em casos de erro esperados: fim de arquivo, por exemplo.
- Exceções são úteis quando dados de entrada não podem ser completamente verificados.
- Exceções são úteis quando não se sabe o que fazer quando um erro é detectado: se dados inválidos, o que fazer? Inicializar com valores padrão?
- Boa prática de programação: se seu método é capaz de tratar uma exceção, então trate-a, ao invés de passar a exceção. Aumenta legibilidade.
- Dica de desempenho: se um erro pode ser processado localmente, trate-o, ao invés de lançar uma exceção. Exceções são caras, se comparadas ao processamento local.

Multi-catch

- Permite que dois tipos de exceção sejam capturados pela mesma instrução **catch.**

```
class MultiCatch {
  public static void main(String args[]) {
    int a = 88, b = 0;
    int result;
    char chrs[] = { 'A', 'B', 'C' };

    for(int i = 0; i < 2; i++) {
       try {
          if(i == 0)
             result = a / b;      // gera uma ArithmeticException
          else
             chrs[5] = 'X';       // gera uma ArrayIndexOutOfBoundsException

       // Captura ambas as exceções.
       } catch(ArithmeticException | ArrayIndexOutOfBoundsException e) {
          System.out.println("Excecao capturada: " + e);
       }
    }
    System.out.println("Apos multi-catch.");
  }
}
```