Programação Orientada a Objetos
Controle de Exceções

Rodrigo Bonifácio

10 de abril de 2014

7: Exception handling

Improved error recovery is one of the most powerful ways you can increase the robustness of your code.

Unfortunately, it's almost accepted practice to ignore error conditions, as if we're in a state of denial about errors. Some of the reason is no doubt the tediousness and code bloat of checking for many errors. For example, **printf( )** returns the number of characters that were successfully printed, but virtually no one checks this value. The proliferation of code alone would be disgusting, not to mention the difficulty it would add in reading the code.

Thinking C++, Volume 2, Bruce Eckel

Especificação (incompleta) de uma pilha (simples)

```
package br.unb.cic.adt;

public interface PilhaAbstrata {
   public void empilha(int valor);
   public int remove();
   public boolean pilhaCheia();
   public boolean pilhaVazia();
}
```

O que fazer quando um erro ocorre em uma chamada a um metodo (talvez de uma biblioteca) mas não se sabe o contexto de execução?

O que fazer quando um erro ocorre em uma chamada a um metodo (talvez de uma biblioteca) mas não se sabe o contexto de execução? No exemplo, chamar *pop()* em uma pilha vazia ou *push(int)* em uma pilha completa.

O que fazer quando um erro ocorre em uma chamada a um metodo (talvez de uma biblioteca) mas não se sabe o contexto de execução? No exemplo, chamar *pop()* em uma pilha vazia ou *push(int)* em uma pilha completa.

(a) Interromper a execução do sistema?

(b) Retornar um valor que identifica um erro?

(c) Setar uma flag global que identifica um erro?

O que fazer quando um erro ocorre em uma chamada a um metodo (talvez de uma biblioteca) mas não se sabe o contexto de execução? No exemplo, chamar *pop()* em uma pilha vazia ou *push(int)* em uma pilha completa.

(a) Interromper a execução do sistema?

(b) Retornar um valor que identifica um erro?

(c) Setar uma flag global que identifica um erro?

(d) Sinalizar que algum problema ocorreu, mas garantir que algum outro componente na pilha de execução trate o problema.

Mecanismo de controle de exceções

Se preocupa em:

- Sinalizar a ocorrência de um erro
- Relançar uma exceção para um contexto superior
- Capturar e tratar uma exceção de acordo com o contexto

Sinalizar a ocorrência de um erro

```
package br.unb.cic.adt;

public class PilhaComoArray implements PilhaAbstrata {
  private static final int MAX_SIZE = 10;

  private int pilha[];
  private int topo;

  public PilhaComoArray() {
    pilha = new int[MAX_SIZE];
    topo = 0;
  }
  ...
  public void empilha(int valor) throws ExcecaoPilha {
    if(topo >= MAX_SIZE) {
      throw new Exception(...);
    }
    pilha[topo++] = valor;
  }
```

...mas é recomendado atualizar a especificação

```java
package br.unb.cic.adt;

public interface PilhaAbstrata {
   public void empilha(int valor) throws ExcecaoPilha;
   public int remove() throws ExcecaoPilha;
   public boolean pilhaCheia();
   public boolean pilhaVazia();
}
```

Tratamento de exceções

```
try {
   int e = pilha.remove();
   System.out.println(e);
}
catch(PilhaException e) {
   // ... tratamento mais robusto aqui!
   System.out.println("Erro: " + e.getMessage());
}
```

Em C++, qualquer tipo pode ser lançado ou tratado como uma exceção, **incluindo os tipos primitivos.**

Em C++, qualquer tipo pode ser lançado ou tratado como uma exceção, **incluindo os tipos primitivos.**

- Em Java, exceções devem ser lançadas e capturadas como *Throwable*, ou uma de suas subclasses.

Tratando diferentes exceções

```
try {
  //algum codigo que pode lancar uma excecao
}
catch(ExcecaoPilha e) {
  //trata as excecoes do tipo StackException
}
catch(Exception e) {
  //trata qualquer outro tipo que seja subclasse de
}
```

Tratando e relançando uma exceção

```
try {
  //algum codigo que pode lancar uma excecao
}
catch(ExcecaoPilha) {
  //...
    throw new OutraException(e.getMessage);
}
```

Tratando e relançando uma exceção

```
try {
  //algum codigo que pode lancar uma excecao
}
catch(ExcecaoPilha) {
  //...
    throw new OutraException(e.getMessage);
}
```

Observação:

- Se uma exceção não for tratada em um contexto, ela é automaticamente relançada para um contexto superior.

Vantagens do mecanismo de exceções

- Reforça contratos, impedindo que erros sejam ignorados
- Melhora (em certo grau) a separação de preocupações
- Favorece o reúso do tratamento de exceções

Vantagens do mecanismo de exceções

- Reforça contratos, impedindo que erros sejam ignorados
- Melhora (em certo grau) a separação de preocupações
- Favorece o reúso do tratamento de exceções

Checked x Unchecked exceptions

O que fazer quando uma exceção não tratada ocorre?

Exceções padrão

- Exception
- InvalidArgumentException, ArrayIndexOutOfBounds
- IOException, FileNotFoundException, ...
- SQLException, ...

Pragmática (1/2)

- Quando a condição do erro puder ser facilmente tratada, não usar controle de exceções.
- Não usar controle de exceções para implementar fluxo de controle

Pragmática (1/2)

- Quando a condição do erro puder ser facilmente tratada, não usar controle de exceções.
- Não usar controle de exceções para implementar fluxo de controle (NoSingleElement)
- Programas extremamente simples podem lidar com algumas situações de erro com o uso de *assert* e *abort*.

Pragmática (1/2)

- Quando a condição do erro puder ser facilmente tratada, não usar controle de exceções.
- Não usar controle de exceções para implementar fluxo de controle (NoSingleElement)
- Programas extremamente simples podem lidar com algumas situações de erro com o uso de *assert* e *abort*. Particularmente prefiro o uso de *asserts* em situações de *debug*, sendo útil para o programador— não para o usuário final.
- Introduzir novas exceções em código já existente é trabalhoso. Particularmente em Java isso requer a revisão da interface de diferentes métodos.

Pragmática (2/2)

- Especifique as exceções nas definições dos métodos.
- Reuse as exceções padrão sempre que aplicado; caso contrário, crie suas próprias exceções.
- Se uma exceção fizer sentido apenas no contexto de uma classe, defina a exceção como membro da classe.
- Capture as exceções por referência, não por valor.

Programação Orientada a Objetos
Controle de Exceções

Rodrigo Bonifácio

10 de abril de 2014