Análise e Projeto de software

Renan Thiago da Silva Rosa

201304940032

Faculdade de Computação

Universidade Federal do Pará

renannojosa@gmail.com

1. Introdução

Um arquiteto de software, quando solicitado para projetar algo novo, primeiro aborda a funcionalidade principal do núcleo, que em uma aplicação de negócios é a lógica básica de negócios. Em um aplicativo bancário, Por exemplo, módulos de núcleo são projetados para gerenciar as transações bancárias que cada cliente faz. Em um aplicativo de varejo, os módulos principais lidam com as compras e gerenciamento de estoque. Em ambas as aplicações, as preocupações em todo o sistema envolvem recursos como registro, autorização, persistência e outros elementos comuns a muitos dos principais módulos de negócios.

Vejamos outro exemplo de software. Se o arquiteto está projetando uma aplicação de robótica, as principais preocupações são o gerenciamento de movimentos e a computação do caminho. As preocupações comuns a muitos dos módulos principais envolvem recursos como log, gerenciamento remoto e otimização de caminho.
Essas preocupações em todo o sistema que abrangem vários módulos são chamadas de preocupações transversais (*crosscutting concerns*). A programação orientada a aspectos (AOP) gerencia essas preocupações transversais.

2. Preocupações Transversais em Aspectj

Na implementação AspectJ do AOP, o compilador AspectJ usa os módulos contendo as regras de combinação, que abordam as preocupações transversais, para adicionar novos comportamentos aos módulos que abordam as preocupações principais - tudo sem fazer modificações no código-fonte dos módulos principais; A combinação ocorre apenas no código de byte que o compilador produz.

AspectJ usa extensões para a linguagem de programação Java para especificar as regras de combinação para o *crosscutting*. As extensões são projetadas de tal forma que um programador Java deve se sentir em casa, enquanto usá-los. As extensões AspectJ usam as seguintes construções para especificar as regras de combinação de forma programática; Eles são os blocos de construção que formam os módulos que expressam a implementação da preocupação transversal.

2.1 Join point

Um join point (ponto de junção) é um ponto identificável na execução de um programa. Poderia ser uma chamada para um método ou uma atribuição para um membro de um objeto. Em AspectJ, tudo gira em torno de juntar pontos, já que eles são os lugares onde o crosscutting.

2.2 Pointcut

Um pointcut é uma construção de programa que seleciona pontos de junção e coleta o contexto nesses pontos. Por exemplo, um ponto de corte pode selecionar um ponto de junção que é uma chamada para um método e também pode capturar o contexto do método, como o objeto de destino no qual o método foi chamado e os argumentos do método.

Podemos escrever um ponto de corte que irá capturar a execução do método `Debit()` na classe `Account` mostrada abaixo:

```
accountOperation ():call(void Account.Debit(..));
```

2.3 Advice

Advice é o código a ser executado em um ponto de junção que tenha sido selecionado por um pointcut.

Os conselhos podem ser executados antes, depois ou ao redor do ponto de junção. Usando um Advice, podemos registrar uma mensagem antes de executar o código em determinados pontos de junção que estão espalhados por vários módulos. O corpo de Advice é muito parecido com um corpo de método - ele encapsula a lógica a ser executada ao atingir um ponto de junção.

Usando o ponto anterior, podemos escrever um conselho que imprimirá uma mensagem antes da execução do método `credit()` na classe `Account`:

```
before() : execution(void Account.credit(float)) {
showMessageDialog(null, "About to perform credit operation");
```

2.4 Aspect

O Aspect é a unidade central do AspectJ, da mesma forma que uma classe é a unidade central em Java. Ele contém o código que expressa as regras de combinação para crosscutting. Pointcuts, Advices e outros são combinados em um aspecto. Além dos elementos AspectJ, os aspectos podem conter dados, métodos e membros de classe aninhados, assim como uma classe Java normal.

Podemos juntar todos os exemplos de código desta seção em um aspecto da seguinte maneira:

```
public aspect ExceptionAspect {

pointcut accountPassword():call(void Account.setpassword(..));

after() throwing (NumberFormatException nfe):
  call(void Account.*(..))
      {
JOptionPane.showMessageDialog(null,nfe.getMessage(),"Erro",0);
      }
```

3.0 Tratamento de Exceções

Na classe `Account`, um ponto de interese é representado pelo método credit que lança uma exceção. As exceções constituem uma preocupação transversal que podem ser tratadas em separado, deixando na classe somente suas atividades centrais e delegando outras atividades aos aspectos.

```
credit(String amount)throws NumberFormatException
```

A classe `Account` no código 3.1 representa uma versão simplificada de uma conta bancária. Ele contém métodos para executar as operações de débito e crédito, bem como obter e definir o saldo da conta.

Código 3.1 Account.java

```
package pacote;
import pacote.InsufficientBalanceException;
public abstract class Account {
    private int balance;
    private int accountNumber;
    private int password;


    public Account(int accountNumber) {
        accountNumber = accountNumber;
    }

    public void credit(String amount)throws
NumberFormatException{
        int number = Integer.parseInt(amount);
        setBalance(getBalance() + number);
    }

    public void debit(int amount) throws
InsufficientBalanceException
    {
        int balance = getBalance();

        if (balance < amount)
        {
            throw new InsufficientBalanceException("Total
balance not sufficient");
        }
        else
        {
            setBalance(balance - amount);
        }
    }

    public void setpassword(String str) throws
NumberFormatException
    {
        int number = Integer.parseInt(str);
        password = number;

    }

    public int getBalance() {
        return balance;
    }


    public void setBalance(int value) {
        balance = value;
    }
}
```

O método `debit()` da classe `Account` declara que pode lançar InsufficientBalanceException quando o saldo da conta não é suficiente para executar a operação. O código 3.2 mostra a implementação desta exceção.

Código 3.2 InsufficientBalanceException.java

```
package pacote;

public class InsufficientBalanceException extends Exception {
    public InsufficientBalanceException(String message)
    {
        super(message);
    }
}
```

A classe `SavingsAccount` é uma especialização da classe de `Account` que representa a conta de poupança. No nosso exemplo no código 3.3, serve para mostrar que o ponto de junção para classes está conectado com a relação de herança.

Código 3.3 SavingsAccount.java

```
package pacote;

public class SavingsAccount extends Account {
    public SavingsAccount(int accountNumber) {
        super(accountNumber);
    }
}
```

As exceções que podem ser lançadas mostradas no código 3.1 serão tratadas no aspecto `ExceptionAspect` descrito no código 3.4

Código 3.4 ExceptionAspect.aj

```
package pacote;
import javax.swing.JOptionPane;

public aspect ExceptionAspect {

    pointcut accountOperation ():call(void Account.Debit(..));
    pointcut accountPassword():call(void
Account.setpassword(..));

    after() throwing (InsufficientBalanceException ibe):
        call(void Account.*(..))
        {

JOptionPane.showMessageDialog(null,ibe.getMessage(),"Erro",0);
        }

    after() throwing (NumberFormatException nfe):
        call(void Account.*(..))
        {

JOptionPane.showMessageDialog(null,nfe.getMessage(),"Erro",0);}
```

```
}
```

O pointcut `accountOperation` engloba o join point do código 3.1 que poderá lançar uma exceção. Este join poin é a chamada do método `Debit()`.De maneira semelhante ao pointcut descrito acima, o pointcut `accountPassword` tem como join poin a chamada do método `setpassword()`.

O Advice será executado após uma chamada para qualquer método na classe `Account` que lança uma exceção. Se um método retornar normalmente, o aviso não será executado. O Advice tem a seguinte sintaxe:

`after() throwing (<ExceptionType exceptionObject>)`

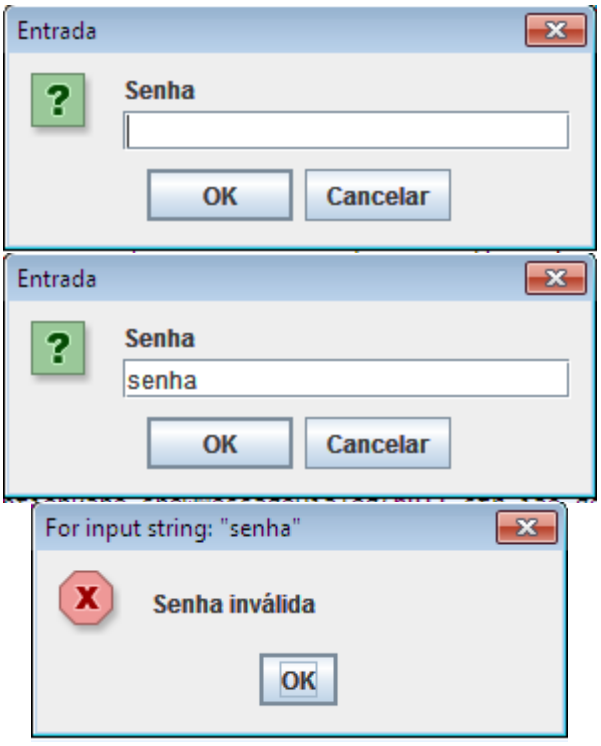
Caso o método lance uma exceção o Advice retorna uma mensagem apropriada. O código 3.5 é um programa simples que causará a execução dos join points.

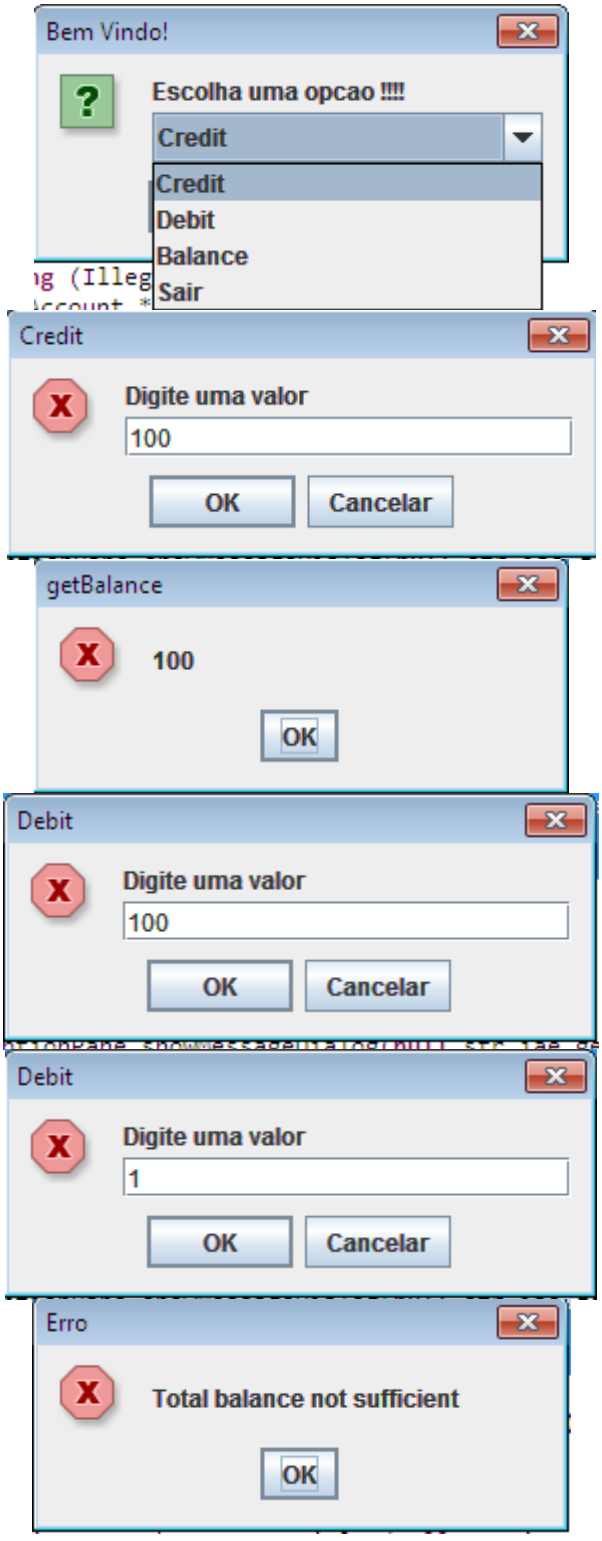
Código 3.5 Test.java

```
package pacote;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class Test {
    static JFrame frame = new JFrame();
     static String str;
     static int valor;

    public static void main(String[] args) throws
InsufficientBalanceException {
        String[] option = {"Credit", "Debit", "Balance",
"Sair"};
        SavingsAccount account = new SavingsAccount(12456);

        str = JOptionPane.showInputDialog(null, "Senha");
        account.setpassword(str);

        while(true)
        {
            String escolha = (String)
JOptionPane.showInputDialog(frame, "Escolha uma opcao !!!!",
"Bem Vindo!", JOptionPane.QUESTION_MESSAGE, null, option,
option[0]);
            if (escolha == null)
            {
                escolha = "0";
            }
            switch (escolha)
            {
                case "Credit":
                    str = JOptionPane.showInputDialog(null,
"Digite uma valor");
                     account.credit(str);
                    break;

                case "Debit":
                    String str =
JOptionPane.showInputDialog(null, "Digite uma valor");
```

```
                    valor = Integer.parseInt( str.trim()   );
                    account.debit(valor);

                    break;

                case "Balance":
                    JOptionPane.showMessageDialog(frame,
"account.getBalance()");
                    break;

                case "Sair":
                    JOptionPane.showMessageDialog(frame, "adeus
");
                    System.exit(0);
            }

        }



    }
}
```

Exemplos de execução:




Bem Vindo!
Escolha uma opcao !!!!
Credit
Credit
Debit
Balance
Sair
Credit
Digite uma valor
100
OK
Cancelar
getBalance
100
OK
Debit
Digite uma valor
100
OK
Cancelar
Debit
Digite uma valor
1
OK
Cancelar
Erro
Total balance not sufficient
OK