Introdução à Programação

Separação de Preocupações

Separação de Preocupações

" …**"the separation of concerns"**, which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by "focusing one's attention upon some aspect": it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant. …"

Dijkstra, E.W "On the role of scientific thought", 1974

Tópicos da Aula

- Hoje, aprenderemos a importância da **separação de preocupações** para o desenvolvimento de um sistema
  - Conceito
  - Impacto no desenvolvimento do sistema
  - Utilização do conceito no desenvolvimento de sistemas
    - Código de Negócio
    - Código de GUI
    - Separando código de negócio e código de GUI
- Veremos, também, outra técnica de refactoring
  - *Extract class*
  - Utilizando *extract class* para separar negócio e GUI

Separação de Preocupações

- Podemos dizer que **separação de preocupações** consiste no processo de quebrar a solução de um problema em partes bem definidas (as mais independentes possíveis) de modo a facilitar o desenvolvimento da solução
  - Pode-se focar em cada parte do problema individualmente
- O princípio é que para superar a complexidade de um sistema, deve-se resolver uma questão (ou preocupação) importante por vez

Preocupação (Concern)

- **Preocupação** é uma parte do problema que queremos tratar como uma unidade conceitual única na solução do software
  - Uma preocupação não deve ser nem mais nem menos do que deveria ser
  - Cada preocupação deve ser logicamente bem delimitada
  - Deve conter somente atributos e comportamento inerentes a sua essência

Importância da Separação de Preocupações

- Separação de preocupações → Modularidade
  - Preocupações podem ser tratados nos módulos
  - Preocupações independentes permitem desenvolvimento paralelo
    - Aumenta produtividade

Um sistema pode ser composto por muitos módulos, mas não obedecer ao princípio da separação de preocupações!

Preocupações devem ser identificadas para que os módulos sejam logicamente bem delimitados!

Importância da Separação de Preocupações

- Separação de preocupações → Reusabilidade e Extensibilidade
  - Módulos bem delimitados e com nenhuma ou poucas dependências com outros módulos podem ser mais facilmente reutilizados e estendidos
- Separação de preocupações → Facilidade de Entendimento (Legibilidade)
  - Módulo com um propósito bem definido ajuda o seu entendimento
- Separação de preocupações → Facilidade de Manutenção
  - Melhora a reusabilidade e extensibilidade
  - Facilita a detecção de módulos com bugs

Considerações sobre Separação de Preocupações

- Identificação e separação de preocupações pode ser uma tarefa difícil dependendo da complexidade do sistema
- Conceito de preocupação é relativo a um determinado problema
  - O que é uma preocupação em um sistema pode não ser para outro
- Nem sempre conseguimos separar a solução em preocupações independentes (cross-cutting concerns)
  - Exemplos: tratamento de erros, auditoria, rastreamento, etc
  - Novas técnicas tentam lidar com cross-cutting concerns
    - Programação orientada a Aspectos (AOP)
    - Separação Multidimensional de Preocupações (SMDC)

Separando Preocupações: Calculando Média Aritmética

Objetivo da Aplicação:
Cálculo da Média Aritmética de um conjunto de números inteiros

Negócio

Interface Gráfica que permite entrada de dados e mostra a média aritmética

GUI

Código de Negócio

```
public class MinhaJanela extends Window {
   TextField campoResultado;
   int[] numeros;
   int proximaPosicaoVaga;
   int tamanhoDoArray;
   ...

   private void acaoBotaoCalcularMedia() {
        int soma,media;
        soma = 0;
        for (int i = 0;i < proximaPosicaoVaga ; i++) {
              soma = soma + numeros[i];
        }
        media = soma/proximaPosicaoVaga;
        campoResultado.setInt(media);
```

São inerentes à aplicação, independem da interface com o usuário

Código da GUI

```
public class MinhaJanela extends Window {
   TextField campoResultado;
   TextField campoEntrada;
   int[] numeros;
   int proximaPosicaoVaga;
   ...

   private void acaoBotaoNovoNumero() {
       int entrada = campoEntrada.getInt();
       numeros[proximaPosicaoVaga] = entrada;
       proximaPosicaoVaga++;
       botaoCalcularMedia.setEnabled(true);
       campoEntrada.setText("");
       if (arrayCheio())
            botaoNovoNumero.setEnabled(false);
```

Código para a apresentação da aplicação (entrada e saída de dados)

Problemas da Aplicação

- A aplicação para calcular a média mistura o código do negócio com o da GUI
  - Métodos são confusos, pois misturam muitos aspectos diferentes
  - Código da classe tende a ser muito grande
    - Novos comportamentos tanto do negócio como da GUI são inseridos na mesma classe
  - Dificulta reusabilidade
  - Dificulta a manutenção

Separando Negócio e GUI

- Deve-se separar as diferentes preocupações
  - Criar classes separadas para cada preocupação
  - Classe de negócio cuida do cálculo da média e do gerenciamento de dados
    - Veremos mais adiante no curso que, na verdade, a parte de gerenciamento e armazenamento de dados deve ser separado do negócio
    - Código de negócio deve ser independente de GUI, armazenamento de dados, comunicação, etc
  - Classe da GUI cuida do comportamento da interface gráfica e faz chamadas a objetos da classe de negócio

Extract Class

- A técnica de refactoring ***extract class*** consiste em tirar preocupações (métodos e atributos logicamente relacionados) de uma classe e jogar em uma nova classe

Aplicando Extract Class para Separar Código de Negócio

```
public class ConjuntoDeInteiros {
   int[] numeros;
   int proximaPosicaoVaga, tamanhoArray;
   ...
   public void inserir(int e){
       numeros[proximaPosicaoVaga] = e;
       proximaPosicaoVaga = proximaPosicaoVaga++;
   }
   public void zerar() {
       proximaPosicaoVaga = 0;
   }
   public int calcularMedia() {
       int soma = 0, media;
       for (int i = 0;i < proximaPosicaoVaga; i++)
             soma = soma + numeros[i];
       media = soma/proximaPosicaoVaga;
       return media;
   }
```

Cria-se uma nova classe com atributos e métodos do negócio

Mudando a Classe da GUI para Usar Classe de Negócio

```
public class MinhaJanela extends Window {
   TextField campoResultado;
   ConjuntoDeInteiros numeros;
   ...
```

Criando um atributo da classe de negócio

```
private void acaoBotaoNovoNumero() {
   int entrada = campoEntrada.getInt();
   numeros.inserir(entrada);
   botaoCalcularMedia.setEnabled(true);
   campoEntrada.setText("");
   ...
}
private void acaoBotaoCalcularMedia() {
   int media = numeros.calcularMedia();
   campoResultado.setInt(media);
  }
```

Chamando métodos da classe de negócio

Resumindo

- Separação de preocupações
  - Conceito
  - Impacto no desenvolvimento do sistema
  - Código de Negócio
  - Código de GUI
  - Separando código de negócio e código de GUI
- Refactoring: Extract Class
  - Utilizando *extract class* para separar negócio e GUI