Unidade 5: Projeto de Programas

Noções de Concepção e Programação

Prof. Daniel Caetano

Objetivo: Apresentar e aplicar alguns conceitos da análise funcional.

Bibliografia: YOURDON, 1990.

INTRODUÇÃO

Nas aulas anteriores foram apresentados os conceitos fundamentais da linguagem Java, conceitos que, de maneira geral, podem ser aplicados a qualquer linguagem de programação estruturada.

Na última aula foram apresentados os conceitos de métodos e funções, que têm um papel muito relevante no desenvolvimento e organização de sistemas de todos os tipos de tamanhos. Entretanto, a divisão funcional das atividades de um sistema não é algo novo, remontando às origens das linguagens de programação.

Nesta aula reveremos brevemente alguns conceitos relativos à análise de fluxo de dados e à análise funcional, para então exercitar sua aplicação no desenvolvimento de um pequeno software.

1. BREVE HISTÓRICO DAS LINGUAGENS

Início: Linguagem de Máquina - Wireup e Digitação
Depois: Assembly: palavras que são traduzidas em códigos de máquina. Assemblada.
1954: FORTRAN: primeira linguagem de alto nível completa. Código Spaghetti. Compilada.
1960: ALGOL60: primeira linguagem de alto nível estruturada em blocos. Compilada.
1970: Pascal: popular linguagem de alto nível estruturada. Compilada.
1970: Smalltalk: primeira linguagem orientada a objetos de sucesso. Compilada.
1971: C: popular linguagem de alto nível estruturada. Compilada.
1975: BASIC: popular linguagem de alto nível "spaghetti". Interpretada.
1983: C++: popular linguagem orientada a objetos. Compilada
1986: Object Pascal (Mac) / Actor (Windows): linguagens orientadas a objetos já com libs voltadas ao desenvolvimento de GUIs. Compiladas.
1991: Visual BASIC: implementação voltada a GUI do velho BASIC. Interpretada/VBRUN
1991: Oak: embrião da linguagem Java. Interpretada.
1991: Python: Linguagem de programação multi-paradigma, interpretada.
1995: PHP: primeira linguagem real voltada ao desenvolvimento Web. Interpretada.
1995: Java: popular linguagem orientada a objetos multiplataforma. Interpretada.
1995: Ruby: Linguagem de programação multi-paradigma, interpretada.
2001: C#: derivada de C++ e Java, orientad a objetos, especifica da plataforma .Net.

A evolução das linguagens, em geral, acontece antes da evolução de paradigma de projeto; na verdade, o aparecimento de linguagens que seguem novas filosofias, em geral, promovem mudanças de paradigma de projeto.

Se, no início, imperava o "caos", este foi substituído pela análise estruturada e, recentemente, pela análise orientada a objetos. Recentemente surgiu um novo paradigma, a orientação a aspectos, mas este ainda está em fase embrionária.

2. VISÃO GERAL SOBRE ANÁLISE E PROJETO ESTRUTURADO

A análise e projeto nem sempre existiram na forma como conhecemos. Quanto mais voltarmos no passado, mais comum era a atitude de "sentar e programar". Entretanto, com o passar dos anos, essa atitude foi se tornando cada vez mais problemática, à medida em que os softwares foram crescendo.

O "sentar e programar" limitava a capacidade de previsão do desenvolvedor, dificultando a tarefa de criar um software que fosse adaptável a situações futuras. Da mesma forma, a manutenção se tornava difícil, uma vez que os softwares desenvolvidos normalmente não contavam com uma modularização coerente. O trabalho em equipe tornava-se um pesadelo.

Destas dificuldades, surgiu a questão: como implementar um sistema?

Objetivo: resolver com sucesso um problema do mundo real.
Passo 1: Compreensão do domínio do problema, cada um de seus detalhes, com definições mais precisas dos objetivos: esse processo foi chamado de Análise de Sistemas.
- Fundamental para atender as necessidades do usuário.
- Fundamental para prever necessidades futuras (expansibilidade, difícil prever)

Passo 2: Propor um modelo simplificado de implementação
Passo 3: Propor um modelo detalhado de implementação
Passo 4: Implementar
Passo 5: Testar
Passo 6: Implantar
Passo 7: Período de Manutenção

2.1. Análise de Sistemas "Clássica"

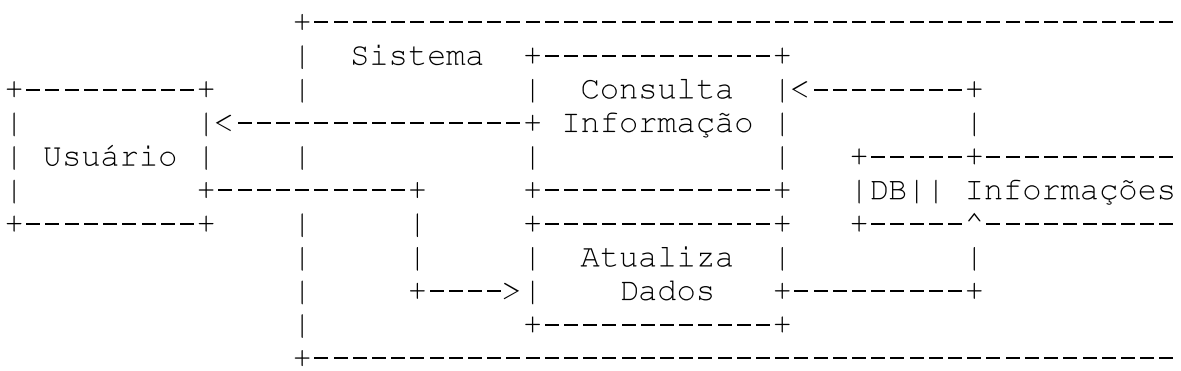
Linguagem de Máquina, Assembly, linguagens não estruturadas:
- Fluxogramas (Diagramas de Blocos, uma entrada, múltiplas saídas).

Programação Estruturada

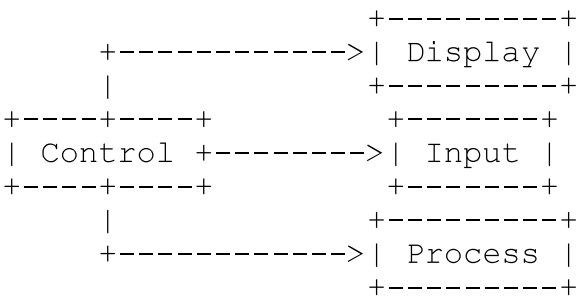
- Duas vertentes básicas:
  => Fluxo de dados
  - Módulos definidos pelos dados que são processados.
  => Decomposição funcional
  - Módulos definidos pelas funções, agrupadas por finalidade

Pensemos num Sistema de Informações genérico. Deve permitir que o usuário processe informações (adição/consulta).

Um exemplo de fluxo de dados (DFD):



Um exemplo de blocos funcionais:

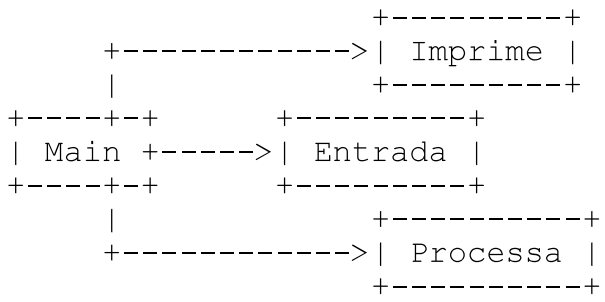


Não havia consenso sobre qual das metodologias usar e o resultado era uma documentação pouco coerente, causando sérios problemas em projetos grandes, com muitas equipes. Apesar dos partidários do DFD, em geral projetistas das basses de dados, costumeiramente "ganharem a briga", isso também não era necessariamente algo bom: em geral o diagrama de blocos funcionais se adapta muito melhor à programação estruturada funcional.

Enquanto ainda se discutia qual dos dois modelos era mais adequado, em meados da década de 60 surgiam as primeiras linguagens orientadas a objeto, já no início da década de 70 ganhando alguma força com o SmallTalk. Mas isso só complicou as coisas: apesar das vantagens evidentes das linguagens orientadas a objetos, nenhum dos modelos existentes e predominantes aderia bem a elas. Em outras palavras: os modelos existentes eram, semanticamente, muito distantes da implementação em linguagem OO.

3. IMPLEMENTANDO EM CÓDIGO

Vamos construir um programa que colete as notas de um aluno e calcule sua média usando um paradigma de divisão funcional simples, como o apresentado no diagrama anterior:



Iniciaremos especificando cada uma dessas funções.

A) imprime()

O método imprime deve receber como prâmetro a informação a ser impressa, na forma e texto. Ela não retorna nada. Sua assinatura deve ser:

public static void imprime(String texto)

B) entrada()

O método entrada deve receber um dado do usuário, no formato double. Para isso, ele deve ter como parâmetros a mensagem que irá apresentar ao usuário e deverá retornar um número no formato double. Sua assinatura deve ser:

public static double entrada(String texto)

C) processa()

O método processa deve receber os dados AV1, AV2, AV3 e FREQ e retornar o boolean false se o aluno está reprovado e true se ele estiver aprovado. Sua assinatura deve ser esta:

public static boolean(double av1, double av2, double av3, double freq)

Este é o primeiro nível da especificação. Vamos agora detalhar o código de cada trecho, testando-os um a um. Para isso, começaremos implementando um metodo **main** simplesmente para teste.

3.1. Implementação do Método imprime()

O código está apresentado a seguir:

ControleNotas.java

```
class ControleNotas {

   public static void imprime(String texto) {
      System.out.println(texto);
      }

   public static void main(String[] args) {
      imprime("Um texto de teste!");
      }
   }
```

3.2. Implementação do Método entrada()

O código está apresentado a seguir:

ControleNotas.java

```
class ControleNotas {

   public static void imprime(String texto) {
      System.out.println(texto);
      }

   public static double entrada(String texto) {
      Scanner teclado = new Scanner(System.in);
      System.out.print(texto);
      return (teclado.nextDouble());
      }

   public static void main(String[] args) {
      double teste;
      teste = entrada("Digite um numero com virgula e pressione enter: ");
      imprime("Dado digitado: " + teste);
      }
   }
```

3.3. Implementação do Método processa()

O código está apresentado a seguir:

ControleNotas.java

```
class ControleNotas {

   public static void imprime(String texto) {
      System.out.println(texto);
      }

   public static double entrada(String texto) {
      Scanner teclado = new Scanner(System.in);
      System.out.print(texto);
      return (teclado.nextDouble());
      }
```

```
   public static boolean processa(double av1, double av2, double av3, double freq) {
      double media;
      if (av2 < 4.0 || freq < 75.0) return false;
      if (av1 >= av3) {
         if (av1 < 4.0) return false;
         else {
            media = (av1 + av2) / 2;
            if (media >= 6.0) return true;
            else return false;
            }
         }
      else { // av3 > av1
         if (av3 < 4.0) return false;
         else {
            media = (av3 + av2) / 2;
            if (media >= 6.0) return true;
            else return false;
            }
         }
      }

   public static void main(String[] args) {
      if (processa(3,4,8,75)==true) imprime("Ok!");
      if (processa(3,3,9,75)==false) imprime("Ok!");
      if (processa(3,4,8,70)==false) imprime("Ok!");
      if (processa(3,10,3,80)==false) imprime("Ok!");
      if (processa(10,3,10,100)==false) imprime("Ok!");
      }
   }
```

O resultado deve ser a impressão de vários "Ok!", indicando que todas as verificações foram consideradas corretas.

3.4. Implementação do Método main()

O código está apresentado a seguir:

ControleNotas.java

```
class ControleNotas {

   public static void imprime(String texto) {
      System.out.println(texto);
      }

   public static double entrada(String texto) {
      Scanner teclado = new Scanner(System.in);
      System.out.print(texto);
      return (teclado.nextDouble());
      }

   public static boolean processa(double av1, double av2, double av3, double freq) {
      double media;
      if (av2 < 4.0 || freq < 75.0) return false;
      if (av1 >= av3) {
         if (av1 < 4.0) return false;
         else {
            media = (av1 + av2) / 2;
            if (media >= 6.0) return true;
            else return false;
            }
         }
```

```
      else { // av3 > av1
         if (av3 < 4.0) return false;
         else {
            media = (av3 + av2) / 2;
            if (media >= 6.0) return true;
            else return false;
            }
         }
      }

   public static void main(String[] args) {
      double av1, av2, av3, freq;
      av1 = entrada("Digite a nota AV1: ");
      av2 = entrada("Digite a nota AV2: ");
      av3 = entrada("Digite a nota AV3: ");
      freq = entrada("Digite a frequência: ");

      if (processa(av1,av2,av3,freq)) imprime("Aluno Aprovado!");
      else imprime("Aluno reprovado!");
      }
   }
```

4. EXERCÍCIOS

A) Use como base o programa criado nesta aula para coletar a nota de vários alunos e dizer se cada um deles está aprovado ou não.

Sugestão: faça um do~while que, antes de repetir, pergunte se o usuário quer continuar (digitando 1 para prosseguir e 0 para sair).

Sugestão 2: provavelmente você precisará criar novos métodos.

B) Adapte o programa acima para, assim que a pessoa optar por sair, ou seja, não consultar mais alunos, o sistema responda a média de todos os alunos cujas notas foram digitadas.

5. BIBLIOGRAFIA

YOURDON, E. **Análise estruturada moderna.** Editora Campus, 1990.
(Just Enough Structured Analysis, http://www.yourdon.com/strucanalysis/)