COBENGE 2005
XXXIII - Congresso Brasileiro de Ensino de Engenharia
"Promovendo e valorizando a engenharia em um cenário de constantes mudanças"
12 a 15 de setembro - Campina Grande Pb
Promoção/Organização: ABENGE/UFCG-UFPE

A PROGRAMAÇÃO ORIENTADA A OBJETOS COMO FERRAMENTA PARA O APRENDIZADO E AUXÍLIO EM PROJETOS DE ENGENHARIA

Lincoln C. Zamboni – lincoln.zamboni@mackenzie.com.br
Universidade Presbiteriana Mackenzie – Escola de Engenharia – Depto. Propedêutica de Eng.
Rua da Consolação, 930
CEP 01302-907 – Consolação – São Paulo – SP
Edson A. R. Barros – prof_edson@mackenzie.com.br
Sergio V. D. Pamboukian – sergiop@mackenzie.com.br

***Resumo**: O estudante de engenharia possui, nos dias de hoje, muitos recursos e ferramentas computacionais para auxiliá-lo na confecção de seus projetos. Linguagens de programação estão disponíveis para o desenvolvimento de aplicações científicas de forma rápida, com interfaces gráficas atraentes e de fácil manipulação, em ambientes propícios à orientação a objetos com seu pragmatismo voltado para o reúso. Ciências básicas, como o cálculo e a física também trazem a sua contribuição para a formalização e solução de problemas específicos que ocorrem nos projetos. Neste trabalho abordamos um problema típico de engenharia nas áreas de hidráulica e hidrologia, que é o dimensionamento de galerias ou canais a partir da vazão proveniente de uma bacia com até 100 hm² e mostramos como fórmulas simples permitem o desenvolvimento de classes de objetos para reúso. Enfocamos os conceitos de orientação a objetos e o reúso de classes bem como os fundamentos sobre linguagens de programação que devem ser de conhecimento de todos os propensos futuros engenheiros.*

Palavras-chaves: Computação, Programação, Hidráulica, Hidrologia, Cálculo.

1. INTRODUÇÃO

Algumas décadas atrás, as réguas de cálculo, os nomogramas e as calculadoras mecânicas eram indispensáveis em um curso de engenharia. Com o passar do tempo, as calculadoras eletrônicas substituíram tais recursos e também se tornaram indispensáveis. Atualmente, os computadores com seus softwares são as principais ferramentas de auxilio nas mais diversas fases do projeto de engenharia.

A UML (Unified Modeling Language) permitiu a padronização, a modelagem, o desenvolvimento e a documentação dos projetos de software orientados a objetos e os ambientes com linguagens de programação também orientadas a objeto tornaram-se os maiores fomentadores do reúso de componentes.

Os conceitos abordados em disciplinas do ciclo básico e, em especial, as com conteúdo de computação e programação, podem, e devem, ser utilizadas na resolução de vários problemas de engenharia, desde os mais simples aos mais complexos.

Tratados sob a ótica da orientação a objetos, os mais simples fornecem o aprendizado básico e os mais complexos o aprofundamento e percepção da necessidade desta orientação.

Este trabalho objetiva:

- mostrar a evolução das metodologias empregadas nas linguagens de programação e enumerar os conhecimentos necessários para um estudante de engenharia;
- mostrar a existência explícita de integração de conhecimentos entre disciplinas que formam o ciclo básico da engenharia;
- mostrar a resolução de um problema típico de engenharia fazendo uso de técnicas de programação orientadas a objeto e o conseqüente reúso de tais objetos.

Ao selecionarmos um dos inúmeros problemas típicos e atraentes de engenharia na área de hidráulica e hidrologia, cabe salientar que poderíamos ter selecionado um de qualquer outra área que, com fórmulas simples e tabelas, promoveriam o desenvolvimento de classes de objetos para reúso com uma facilidade de abordagem ímpar.

2. METODOLOGIAS DE PROGRAMAÇÃO

2.1. Da Programação Estruturada à Programação Orientada a Objetos

A elaboração de algoritmos desenvolve no estudante de engenharia qualidades de planejamento, preparo e previsão que não devem ser omitidas no ensino superior.

A tão famosa fórmula de Bhaskara, vista na "equação (1)", é um exemplo trivial de algoritmo onde, a partir do conhecimento dos valores reais de *a*, *b* e *c* da equação do segundo grau $ax^2 + bx + c = 0$, determina-se, os dois valores da raiz *x*.

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a} \qquad (1)$$

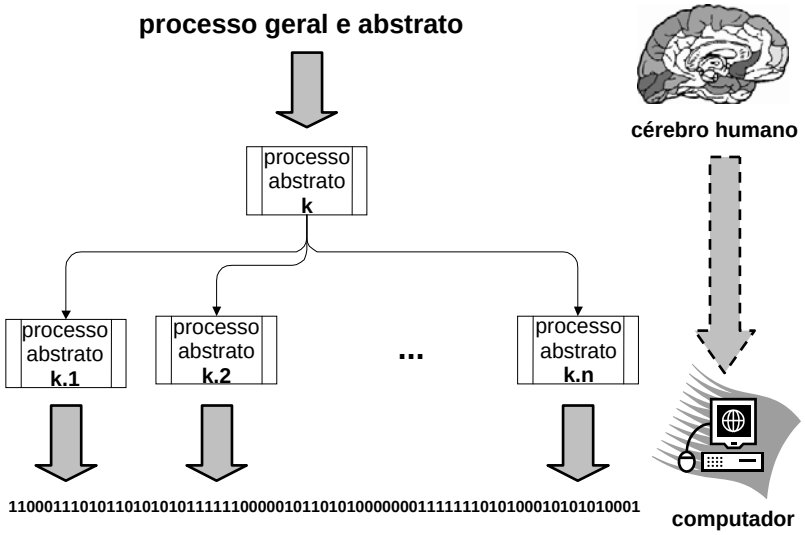


Figura 1 – Domínio da complexidade

Segundo BARROS ET AL. (2003), a complexidade de um programa pode ser dominada através da programação estruturada, uma metodologia que se compõe de alguns princípios elementares (veja a “Figura 1”):

- princípio da abstração é a concepção ou visão do programa separado de sua realidade. É a simplificação de fatos, descrevendo o que está sendo feito sem explicar como está sendo feito;
- princípio da formalidade possibilita analisar os programas de forma matemática. Fornece uma abordagem rigorosa e metódica. Possibilita a transmissão de idéias e instruções sem ambigüidades e permite que estas sejam automatizadas;
- princípio da divisão é a subdivisão organizacional de um programa em um conjunto de partes menores e independentes, mais fáceis de serem entendidas, resolvidas, manipuladas e testadas individualmente;
- princípio da hierarquia é a organização hierárquica que está relacionada com o princípio da divisão. A organização das partes em uma estrutura hierárquica do tipo árvore sempre aumenta a compreensibilidade.

A partir de um processo geral e abstrato do que o programa deve fazer, este é dividido em vários processos menores também abstratos. Qualquer uma destes processos pode, de forma hierárquica, ser dividido em mais processos abstratos e assim sucessivamente. A metodologia termina por chegar a um nível tal de formalidade no qual o processo pode ser implementado no computador através de uma linguagem de programação.

Na linguagem C++, por exemplo, um processo é implementado como uma função. As funções aceitam entradas e saídas de forma a generalizar o processo aplicando-se a quaisquer dados de tipos determinados. As entradas (*e*) são os argumentos e dados globais usados na função. As saídas (*s*) são o valor de retorno (as funções podem ou não possuir valor de retorno), modificações feitas através de ponteiros e referências, bem como mudanças em dados globais (veja a “Figura 2”).

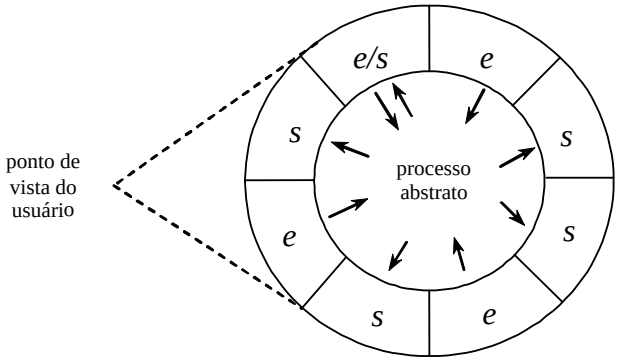


Figura 2 – Funções como módulos de construção

A idéia da Programação Orientada a Objetos surgiu nos anos 60, porém apenas os programadores experientes e de alto conhecimento técnico se utilizavam dela, pois os conceitos de programação eram novos e a elaboração dos algoritmos era complicada. Só após o aparecimento dos ambientes gráficos (como o Windows) e das linguagens “Visuais” (como Delphi, C++ Builder, Visual C++, Visual Basic, etc.) os programadores se sentiram mais à vontade para utilizar este tipo de programação. Objetos são metáforas naturais para objetos físicos e entidades abstratas. Expressar a computação em termos de objetos reduz a lacuna entre o mundo real e o programa.

Os objetos são formados por dados e processos, combinando-os de forma que os processos acessem ou modifiquem os dados. As classes são tipos de objetos declarados pelo usuário. Elas agregam uma declaração de dados e uma declaração de processos (métodos)

para processar os dados. As classes descrevem grupos de objetos com propriedades semelhantes, comportamento, relacionamentos e semântica comuns. Todos os objetos da mesma classe têm processos comuns mas dados distintos. Cada objeto é criado a partir de uma classe, via uma especificação do valor para seus dados.

Como podemos ver em BARROS ET AL. (2003), três idéias fundamentais caracterizam a programação orientada a objetos:

- encapsulamento: acesso ou modificação dos dados de forma a ocultar os detalhes da implementação destes com o uso de uma interface de processos. Assim a mesma classe pode ter diferentes implementações, em diferentes tempos, sem afetar o código que a utiliza. Os dados são definidos pela interface, e os detalhes de sua implementação são abstraídos;
- hereditariedade: organiza classes para reuso, com classes derivadas herdando processos de classes base (processos não necessitam recodificação para classes de componentes comuns). Estende os processos da programação procedimental para uma estrutura de dados e processos. Novas classes e seus processos podem ser definidos e extensões de classes existentes;
- polimorfismo: o processo a ser executado sobre o objeto depende do objeto real (eventualmente identificado por ponteiro) em tempo de execução. Um processo para girar uma figura, por exemplo, vai chamar o código correto para girar um triângulo, um retângulo, etc.

2.2. Conhecimentos fundamentais para o estudante de engenharia

Nas disciplinas de computação e programação existentes nos cursos de engenharia, procura-se fornecer ao aluno alguns conceitos fundamentais sobre uma determinada linguagem de programação para que a mesma, em conjunto com métodos matemáticos aprendidos em outras disciplinas, possam auxiliá-lo na solução de problemas específicos de sua área. Estes conceitos trazidos para o âmbito da linguagem C++, e que podem ser facilmente migrados para qualquer outra linguagem orientada a objetos como Java, C#, Object Pascal e outras, são:

- instruções de controle do fluxo: seqüência, condição (if/else, switch) e repetição (for, do/while e while);
- tipos básicos de dados: int, double, char, bool, float, long int, long double, etc;
- reúso de classes de objetos de interface gráfica: formulários, rótulos, botões, caixas de edição, de checagem, de listagem, combinadas, grades, botões de rádio, etc.
- arranjos e estruturas: ponteiros, operadores [], new e delete para alocação dinâmica;
- classes:
  - construtor padrão, construtor de cópia, destruidor, operador de atribuição, encapsulamento e métodos de acesso;
  - polimorfismo, métodos virtuais e virtuais puros;
  - hereditariedade;
- classes da Standard Template Library (STL): string, vector, list, etc.

É interessante que os conceitos de orientação a objetos sejam apresentados em duas fases: a primeira reusando classes de objetos presentes no ambiente integrado da linguagem e a segunda desenvolvendo classes para reúso.

Além disso, devemos enfatizar o uso de unidades do SI (Sistema Internacional de Unidades) nas fórmulas de engenharia, a documentação interna do código com comentários e a organização do programa em unidades.

3. DIMENSIONAMENTO DE GALERIAS E CANAIS A PARTIR DA VAZÃO PROVENIENTE DE UMA BACIA

Para ilustrar o uso da programação orientada a objetos como ferramenta auxiliar no desenvolvimento de projetos de engenharia, mostraremos a seguir um exemplo prático que consiste no desenvolvimento de classes de objetos para reúso no dimensionamento de galerias ou canais a partir da vazão proveniente de uma bacia com até 100 hm².

Utilizaremos as fórmulas de Occhipinti, "equação (2)", Racional, "equação (3)", e Ganguillet-Kutter, "equação (4)", extraídas de AZEVEDO NETO E ALVAREZ (1973) e também o Método da Bisseção que está detalhado item 4. As unidades do SI seguiram os quadros constantes em ROZEMBERG (2002).

$$I = \frac{27{,}96 \cdot T^{0{,}112}}{(t+15)^{0{,}86 \cdot T^{-0{,}0144}}} \tag{2}$$

$$Q = \frac{C \cdot A \cdot I}{6} \tag{3}$$

$$Q = \frac{k + \frac{1}{n}}{1 + k \cdot \frac{n}{\sqrt{R_h}}} \cdot A_m \cdot \sqrt{R_h \cdot i} \tag{4}$$

Nas "equações (2) e (3)" temos:

- I = intensidade da precipitação crítica (mm/min) para a cidade de São Paulo;
- T = intervalo de recorrência, ou período de retorno (anos);
- t = tempo de concentração (h);
- Q = vazão da bacia (m³/s);
- C = coeficiente de escoamento superficial, ou de run-off (adimensional);
- A = área da bacia contribuinte a montante (hm²).

Na "equação (4)" temos:

- Q = vazão da galeria ou canal (m^3/s);
- k = variável auxiliar vista na "equação (7)";
- i = inclinação da galeria ou canal (adimensional);
- n = coeficiente de rugosidade da galeria ou canal (adimensional) definido pela classe de rugosidade vista na "Tabela 1";
- A_m = área molhada da galeria ou canal (m^2) conforme "Tabela 2";
- R_h = raio hidráulico da galeria ou canal (m) calculado conforme a "equação (8)";
- p = perímetro molhado da galeria ou canal (m) conforme "Tabela 2".

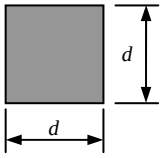
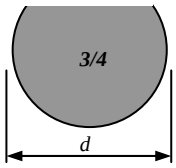
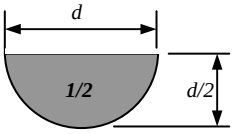
$$k = 23 + \frac{0{,}00155}{i} \tag{7}$$

$$R_h = \frac{A_m}{p} \tag{8}$$

Tabela 1 – Classes e coeficientes de rugosidade

Classe de rugosidade (*CR*)	**Material da seção**	**Coeficiente de rugosidade (*n*)**
A	barro vitrificado	0,014
B	cimento alisado	0,011
C	concreto	0,013
D	canais lamosos	0,025
E	canais dragados	0,028

Tabela 2 – Áreas e perímetros molhados

Nome da seção	**Geometria**	**Área molhada** A_m	**Perímetro molhado** p
quadrada		d^2	$3 \cdot d$
circular ¾		$0{,}589048622548086 \cdot d^2$	$1{,}9866515 \cdot d$
circular ½		$0{,}392699081698724 \cdot d^2$	$1{,}57079632679490 \cdot d$

4. TEOREMA DE BOLZANO E MÉTODO DA BISSEÇÃO

Para determinar a vazão através da fórmula de Ganguillet-Kutter (4) temos que utilizar um dos métodos do Cálculo Numérico para a determinação de raízes de uma função. O método utilizado no exemplo deste artigo é o da bisseção, descrito a seguir.

Segundo o teorema de Bolzano, se $y = f(x)$ é uma função contínua no intervalo $[a,b]$ $(a < b)$ e $f(a) \cdot f(b) < 0$, então existe uma raiz α em $]a,b[$, isto é, $f(\alpha) = 0$.

A “Figura 3” ilustra de forma geométrica o teorema de Bolzano.

Em ZAMBONI e MONEZZI JUNIOR (2002), podemos ver a demonstração deste teorema e também do método da bisseção utilizado para a busca da raiz.

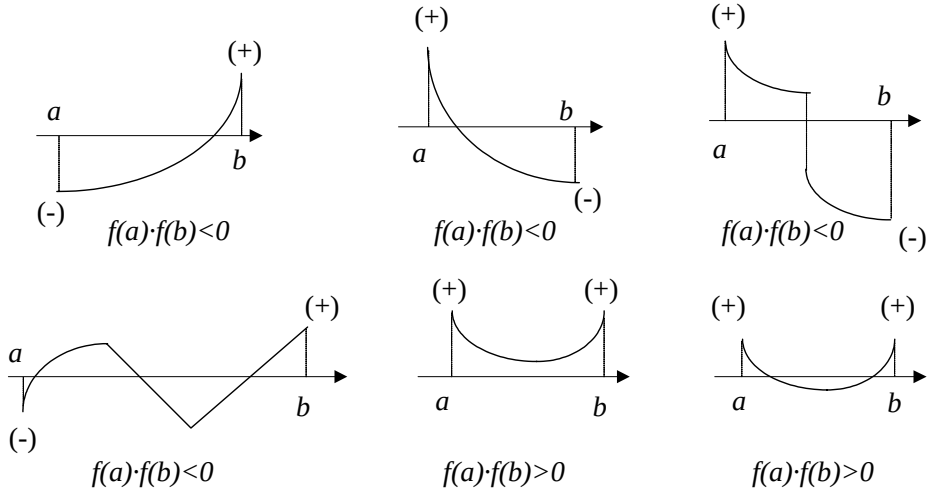


Figura 3 – Ilustrações do Teorema de Bolzano

Vamos montar duas seqüências reais $a_1, a_2, a_3, \ldots$ e $b_1, b_2, b_3, \ldots$ da seguinte forma:

- $a_1 = a$ *e* $b_1 = b$
- *para* $i = 1, 2, 3, \ldots$

$\alpha_i = \dfrac{a_i + b_i}{2}$ se $f(\alpha_i) \cdot f(b_i) \leq 0$, então $a_{i+1} = \alpha_i$ e $b_{i+1} = b_i$

se $f(\alpha_i) \cdot f(a) < 0$, então $a_{i+1} = a_i$ e $b_{i+1} = \alpha_i$

As seqüências reais formadas têm as seguintes propriedades:

1. A seqüência a_i é crescente, isto é, $a_i \leq a_{i+1}$ e b_i é decrescente, isto é, $b_{i+1} \leq b_i$.
2. São limitadas, isto é, $a_i < b$ e $a < b_i$.
3. $b_i - a_i = \dfrac{b-a}{2^{i-1}}$.

Pelas propriedades acima concluímos que as seqüências possuem limites iguais, isto é, $\lim_{i \to \infty} a_i = \lim_{i \to \infty} b_i = \alpha$.

Como $y = f(x)$ é contínua em $[a,b]$, temos $\lim_{i \to \infty} f(a_i) = \lim_{i \to \infty} f(b_i) = f(\alpha)$.

Assim $f(\alpha) \cdot f(b) \leq 0$ e $f(\alpha) \cdot f(a) < 0$. Logo $(f(\alpha))^2 \cdot f(a) \cdot f(b) \geq 0$.

Como $f(a) \cdot f(b) < 0$ (hipótese do teorema) e $(f(a))^2 \geq 0$, só poderemos ter $(f(a))^2 = 0$, isto é, $f(\alpha) = 0$. Logo α é a raiz procurada.

O método da bisseção consiste em construirmos as duas seqüências reais $a_1, a_2, a_3, \ldots$ e $b_1, b_2, b_3, \ldots$, porém não até o infinito. Pararemos suas construções quando atingirmos um valor $\alpha_n = \dfrac{a_n + b_n}{2}$, tal que $|\alpha - \alpha_n| < \varepsilon$, onde ε é um erro preestabelecido. Observe que, sendo $a_n < \alpha < b_n$, teremos $-\dfrac{b_n - a_n}{2} < \alpha - \underbrace{\dfrac{a_n + b_n}{2}}_{\alpha_n} < \dfrac{b_n - a_n}{2}$ que simplificado nos leva a uma cota para a diferença entre a raiz aproximada e a exata, $|\alpha - \alpha_n| < \dfrac{b_n - a_n}{2}$. Se $\dfrac{b_n - a_n}{2} \leq \varepsilon$, teremos $|\alpha - \alpha_n| < \varepsilon$, isto é, α_n é uma aproximação para a raiz α com erro inferior a ε.

O número de iterações n pode ser previsto facilmente considerado: $\frac{b-a}{2} \leq \varepsilon$, onde $\varepsilon > 0$. Logo, $2^n \geq \frac{b-a}{\varepsilon}$ e, então $n \geq \frac{\ln\left(\frac{b-a}{\varepsilon}\right)}{\ln 2}$.

Exemplo

Usando o método da bisseção, podemos determinar a raiz da função de Ganguillet-Kutter vista na "equação (9)" com erro inferior a 0,01. Nela consideramos $Q = 2{,}0\ m^3/s$, $i = 0{,}002$, $n = 0{,}013$ e seção quadrada de lado x.

$$f(x) = Q - \frac{k + \frac{1}{n}}{1 + k \cdot \frac{n}{\sqrt{R_h}}} \cdot A_m \cdot \sqrt{R_h \cdot i} \qquad (9)$$

O gráfico da função pode ser visto na "Figura 4" que também mostra a localização aproximada da raiz na função de Ganguillet-Kutter (9).

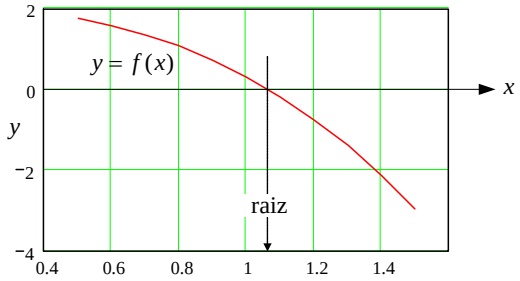


Figura 4 – Raiz da função

Como $f(x)$ é contínua para $x > 0$, $f(0{,}5) = 1{,}738$ e $f(1{,}5) = -2{,}986$, então, pelo teorema de Bolzano, existe uma raiz no intervalo $]0{,}5;1{,}5[$.

A "Tabela 3" ilustra a obtenção das seqüências reais $a_1, a_2, a_3, \ldots$ e $b_1, b_2, b_3, \ldots$ dadas na demonstração do teorema de Bolzano, ou melhor, do método da bisseção.

Tabela 3 – Bisseção da função

i	$(+)$ a_i	$\alpha_i = \frac{a_i + b_i}{2}$	$(-)$ b_i	$\frac{b_i - a_i}{2}$
1	0,500	1,000	1,500	0,500
2	1,000	1,250	1,500	0,250
3	1,000	1,125	1,250	0,125
4	1,000	1,063	1,125	0,063
5	1,063	1,094	1,125	0,032
6	1,063	1,079	1,094	0,016
7	1,063	1,071	1,079	$0{,}008 < \varepsilon = 0{,}010$

Logo, a raiz aproximada com erro inferior a 0,01 é $\alpha_7 = 1{,}071$. Observemos que $n \geq \dfrac{\ln\left(\dfrac{b-a}{\varepsilon}\right)}{\ln 2} = \dfrac{\ln\left(\dfrac{1{,}5-0{,}5}{0{,}01}\right)}{\ln 2} = 6{,}644$, logo $n = 7$ é a melhor escolha.

5. CLASSES DESENVOLVIDAS

A “Figura 5” mostra as classes que compõem o pequeno projeto de uso de fórmulas de engenharia: a classe TBacia, as classes TGaleriaCanal e TTabelaCR com relacionamento de um para um e as outras três classes derivadas da classe TGaleriaCanal. As “Figuras 6, 7, 8 e 10” mostram as especificações das classes escritas em C++. A “Figura 9” mostra a implementação do Método da Bisseção. A “Figura 11” mostra uma pequena aplicação orientada a objetos construída instanciando-se as classes já citadas e também classes de objetos para a interface gráfica tais como formulários, caixas de agrupamento, caixas de edição, caixas de checagem, caixas de combinação, rótulos, botões, grupo de botões de rádio e grade de cadeias de caracteres.

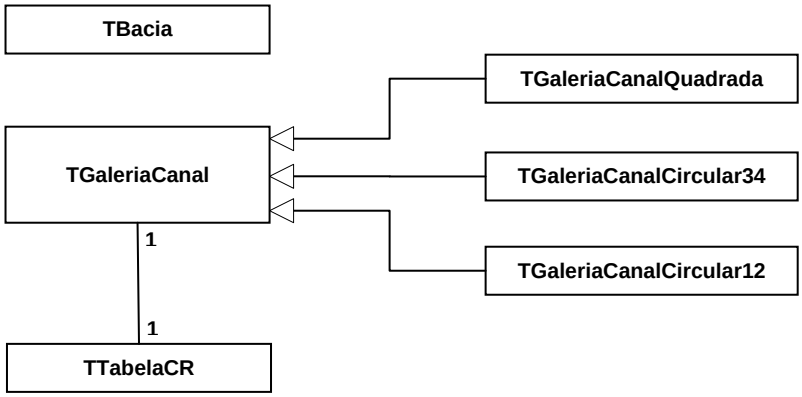


Figura 5 – Classes do problema típico

```
class TGaleriaCanal
{
  protected:
  string nome;     // Nome da seção da galeria ou canal.
  string material; // Material da seção da galeria ou canal.
  double Q;        // Vazão da galeria ou canal (m³/s).
  double i;        // Inclinação da galeria ou canal (adimensional).
  double n;        // Coeficiente de rugosidade da galeria ou canal(adim.).
  double d;        // Dimensão da galeria ou canal (m).
  double k;        // Constante para a fórmula de Ganguiliet-Kutter.
  // Método para validação da entrada.
  void Valida();
  // Método para cálculo da imagen da função de Ganguiliet-Kutter:
  double f(const double &x);
  // Método virtual puro para cálculo da área molhada da galeria ou canal (m²):
  virtual double Am(const double &x) = 0;
  // Método virtual puro para cálculo do perímetro molhado da galeria ou canal
  // (m):
  virtual double p(const double &x) = 0;
  // Método virtual para cálculo do raio hidráulico da galeria ou canal (m):
  double Rh(const double &x);

  public:
  // Métodos para mostrar e alterar os valores da galeria ou canal:
  string GetNome() {return nome;}
  void SetNome(const string &nome) {this->nome = nome;}
  string GetMaterial() {return material;}
  void SetMaterial(const string &material) {this->material = material;}
  double GetQ() {return Q;}
  void SetQ(const double &Q) {this->Q = Q; Valida();}
  double Geti() {return i;}
  void Seti(const double &i) {this->i = i; Valida();}
  double Getn() {return n;}
  void Setn(const double &n) {this->n = n; Valida();}
  // Construtor Padrão e com todos os parâmetros:
  TGaleriaCanal(const string &nome = "sem nome",
    const string &material = "concreto", const double &Q = 2.0,
    const double &i = 0.002, const double &n = 0.013);
  // Construtor com consulta a uma tabela de classe de rugosidade:
  TGaleriaCanal(const string &nome, const char &CR,
    const TTabelaCR &tcr, const double &Q, const double &i);
  // Aqui não geramos o construtor de cópia, o destruidor e o operador de
  // atribuição pois estes serão gerados implicitamente pelo compilador.
  // Método para dimensionamento da galeria ou canal (m) baseado no método da
  // bisseção:
  double Dimensiona();
};
```

Figura 6 – Classe TGaleriaCanal

```
class TGaleriaCanalQuadrada: public TGaleriaCanal
{
  protected:
  // Método virtual para cálculo da área molhada da galeria ou canal (m²):
  virtual double Am(const double &x);
  // Método virtual para cálculo do perímetro molhado da galeria ou canal (m):
  virtual double p(const double &x);

  public:
 // Construtor Padrão e com todos os parâmetros:
  TGaleriaCanalQuadrada(const string &nome = "sem nome",
    const string &material = "concreto", const double &Q = 2.0,
    const double &i = 0.002, const double &n = 0.013);
  // Construtor com consulta a uma tabela de classe de rugosidade:
  TGaleriaCanalQuadrada(const string &nome, const char &CR,
    const TTabelaCR &tcr, const double &Q, const double &i);
};
```

Figura 7 – Classe TGaleriaCanalQuadrada derivada de TGaleriaCanal

```
class TTabelaCR
{
  private:
  struct TLinha
  {
    string material; // Material da seção.
    double n;        // Coeficiente de rugosidade.
  };
  TLinha *lin; // Ponteiro para acessar a tabela na memória.
  int tam;     // Tamanho da tabela (de 0 até 26 linhas).

  public:
  // Método de obtenção da Classe de Rugosidade (CR):
  char GetCR(const int &num) {return (char)(num + 64);}
  // Métodos de obtenção do material:
  string GetMaterial(const char &CR);
  string GetMaterial(const int &num){return GetMaterial(GetCR(num));}
  // Métodos de obtenção do coeficiente de rugosidade:
  double Getn(const char &CR);
  double Getn(const int &num){return Getn(GetCR(num));}
  // Método de obtenção do tamanho da tabela:
  int GetTamanho();
  // Método de adição de uma nova linha na tabela:
  bool Adiciona(const string &material, const double &n);
  // Métodos de remoção de uma linha da tabela:
  bool Remove(const char &CR);
  bool Remove(const int &num){return Remove(GetCR(num));}
  // Método de modificação de uma linha da tabela:
  bool SetLinha(const char &CR, const string &material,
    const double &n);
  // Construtor padrão:
  TTabelaCR(const int &tam = 5);
  // Construtor de cópia:
  TTabelaCR(const TTabelaCR &t);
  // Destruidor:
  ~TTabelaCR();
  // Operador de Atribuição:
  TTabelaCR & operator =(const TTabelaCR &t);
};
```

Figura 8 – Classe TTabelaCR

```
double TGaleriaCanal::Dimensiona()
{
  double a = 0.0, b = 1.0, alfa;
  while(f(b) > 0) b++; // Encontro da inversão de sinal.
  while((b - a) / 2.0 >= 1E-15){ // Precisão para o método da bisseção.
    alfa = (a + b) / 2.0;
    if(f(alfa) * f(b) <= 0.0) a = alfa;
    else b = alfa;
  }
  return d = (a + b) / 2.0;
}
```

Figura 9 – Método da Bisseção

```
class TBacia
{
  private:
  string nome; // Nome da bacia.
  double I;    // Intensidade da precipitação crítica para São Paulo
(mm/min).
  double T;    // Intervalo de recorrência, ou período de retorno (anos).
  double t;    // Tempo de concentração (h).
  double Q;    // Vazão da bacia (m³/s).
  double C;    // Coeficiente de escoamento superficial (adimensional).
  double A;    // Área da bacia contribuinte a montante (hm²).
  // Método para cálculo de I e de Q.
  void Calc();

  public:
  // Construtor Padrão e com todos os parâmetros:
  TBacia(const string &nome = "sem nome", const double &T = 5.0,
    const double &t = 0.30, const double &C = 0.80, const double &A = 2.0);
  // Aqui não geramos o construtor de cópia, o destruidor e o operador de
  // atribuição pois estes serão gerados implicitamente pelo compilador.
  // Métodos para mostrar e alterar os valores da bacia:
  string GetNome() {return nome;}
  double GetI() {return I;}
  double GetT() {return T;}
  void SetT(const double &T) {this->T = T; Calc();}
  double Gett() {return t;}
  void Sett(const double &t) {this->t = t; Calc();}
  double GetQ() {return Q;}
  double GetC() {return C;}
  void SetC(const double &C) {this->C = C; Calc();}
  double GetA() {return A;}
  void SetA(const double &A) {this->A = A; Calc();}
  // Método operador para combinação de duas bacias.
  TBacia operator +(const TBacia &B);
};
```

Figura 10 – Classe TBacia

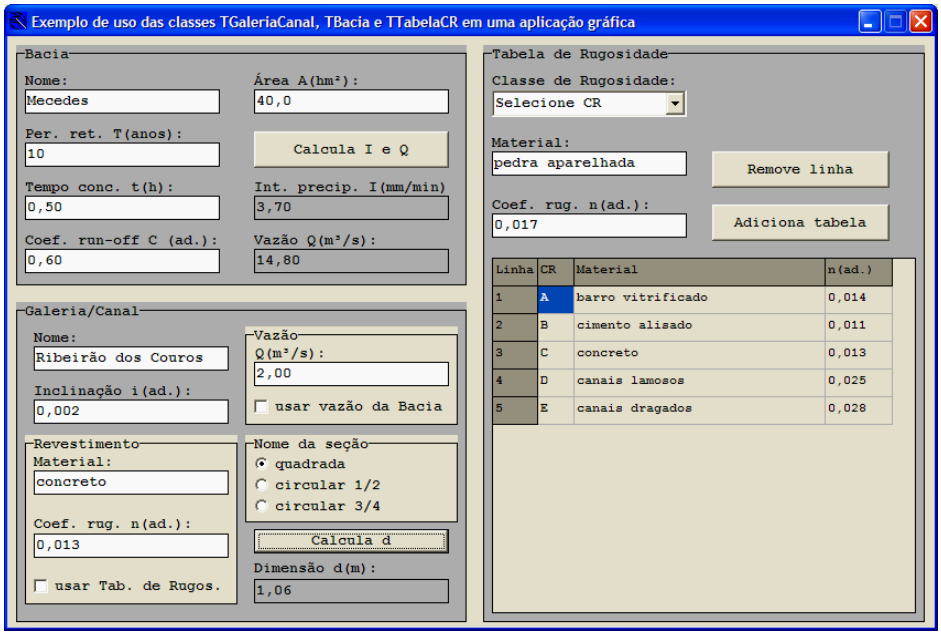


Figura 11 – Aplicação elaborada a partir das classes

6. CONSIDERAÇÕES FINAIS

Como pudemos ver, o dimensionamento de canais e galerias exemplifica de forma simples e eficaz os requisitos principais do desenvolvimento de software em termos de componentes reutilizáveis e aplicações sob a ótica da engenharia. O método da bisseção

utilizado no dimensionamento também pode ser programado de maneira fácil e rápida e ser um método de cálculo de uma classe para reúso. As linguagens de programação orientadas a objetos, através do princípio da hereditariedade e polimorfismo, permitem que as classes sejam facilmente modificadas para, por exemplo, se adequar a outros tipos de seção de galerias ou canais. O encapsulamento é mostrado de uma forma quase que desapercebida e natural.

Agradecimentos

Gostaríamos de agradecer a colaboração do Prof. Dr. Pedro José da Silva, que ministra aulas nas áreas de Hidráulica e Hidrologia na Universidade Presbiteriana Mackenzie.

REFERÊNCIAS BIBLIOGRÁFICAS

AZEVEDO NETO, J. M. de; ALVAREZ, G. A. **Manual de Hidráulica**. v.2 / 6ª. edição. São Paulo, Edgard Blücher; Brasília, INL, 1973.

BARROS, E. A. R.; PAMBOUKIAN, S. V. D.; ZAMBONI, L. C. **C++ Builder para Universitários**. São Paulo: Páginas & Letras, 2ª ed., 2003.

BARROS, Edson de Almeida Rego; PAMBOUKIAN, Sergio Vicente D.; ZAMBONI, Lincoln César. **Ensino de Computação para estudantes de Engenharia**. COBENGE2004 (Congresso Brasileiro de Ensino de Engenharia), Brasília – DF, Unb 14 a 17 set 2004.

BRAGA, B.; TUCCI, C.; TOZZI, M. **Drenagem urbana: gerenciamento, simulação, controle**. Porto Alegre: Editora da Universidade / UFRGS / ABRH, 1998.

ROZEMBERG, I.M. **O Sistema Internacional de Unidades – SI**. 2ª. Edição. São Paulo. Instituto Mauá de Tecnologia, 2002.

ZAMBONI, L. C.; MONEZZI JÚNIOR, O. **Cálculo Numérico para Universitários**. São Paulo: Páginas & Letras, 2ª ed., 2002.

Abstract*: The engineering student have, at the present time, several resources and computing tools to help himself in the conclusion of his own projects. Programming languages are available to the rapid scientific applications development, with attractive graphical interfaces and easy manipulation, in propitious environment to the object orientation with susceptible practice application to the reuse. Fundamentals sciences, like calculus and physics, brings its contribution to the formalization and solution of the specifics problems that happens in the projects. In this work we approach a typical problem in hydraulics and hydrology engineering areas, which is the computing of the dimension of a gallery or a canal having the flowing deriving from a basin with less than 100 hm² and show how simply formulas allow the development of object classes for reuse. We show the object orientation concepts and the reuse of classes and also the fundaments about programming languages which must be the knowledge of the future engineering.*

Key-words: Computation, Programming, Hydraulics, Hydrology, Calculus