Programação Modular
ESTILO DE PROGRAMAÇÃO

Roberto da Silva Bigonha
Mariza A.S.Bigonha
Laboratório de Linguagens de Programação
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

Agosto de 2019

Sumário

PROGRAMAÇÃO POR CONTRATO

Contrato de Classe

- Contrato de uma classe é a coleção de métodos e campos por ela exportados.
- A semântica dos elementos exportados, i.e., suas pré- e pós-condições, faz parte do contrato.
- Campos e métodos privados não fazem parte do contrato.
- Contrato é o que o projetista da classe promete que ela fará.

Condições de Execução de um Método

- Pré-Condição:

  Condição a ser satisfeita pelos valores dos parâmetros do método e o estado do objeto.

- Pós-Condição:

  Condição a ser satisfeita pelo estado objeto produzido e o valor retornado pelo método

Programação por Contrato ...

- Pré-Condição define as condições sob as quais a chamada a uma rotina é legítima.
- Pós-Condição define as condições que a rotina deve garantir no seu retorno.
- O retorno normal de um método significa cumprimento do contrato.
- Falhas no cumprimento do contrato devem sempre causar lançamento de exceções.

Conceito de Subcontratação.

- A definição de subclasse equivale a **estender** o contrato da superclasse.
- O contrato herdado pode ser redefinido de que não se viole o contrato da superclasse.
- Na redefinição da implementação de um contrato pode-se **enfraquecer** as pré-condições e **fortalecer** as pós-condições de métodos.
  - Condição mais forte é mais restrita

Fortalecimento de Pré-Condições

- Exemplo: substituir caixa-humano por caixa-eletrônico
  - Caixa-Humano: autoriza retiradas até R$50,00 sem verificar existência de fundos
  - Caixa-Eletrônico: não as autoriza
  - Há **violação contrato porque** fortaleceu-se incorretamente a pré-condição do método

Enfraquecimento de Pós-Condições

- Exemplo: método retorna um inteiro no intervalo de 10 a 20.
- O valor retornado inteiro pode ser usado para indexar arranjos cujo tamanho seja menor ou igual a 20
- Não há necessidade de testar o valor de índice retornado pelo método antes do uso.
- Problema: método redefinido retorna um valor entre 1 e 100, violando contrato

Programação por Contrato

- O relacionamento entre uma classe e seus clientes é uma relação de direitos e obrigações
- É uma relação vinculada a idéia de contrato
- Definir cláusulas de pré-condição e de pós-condição para todo método r e observar que:
  - *Se o usuário se comprometer a chamar o método* r *com sua pré-condição satisfeita, então* r *garante devolver um estado final (do objeto associado) com sua pós-condição satisfeita, ou então declara explicitamente seu insucesso.*

QUALIDADE DE CLASSES

Qualidade de Uma Classe

- Classes Bem-Construídas:
  - Classes TAD
  - Classes Empacotadoras
  - Classes Estrutura de Dados
- Classes Mal-Construídas:
  - Classes Depositárias
  - Classes Fantasma
  - Classes Polivalente

Classes TAD

Uma classe TAD bem-construída:

- Implementa um único e relevante contrato
- Seus métodos operam seus campos de forma relevante e significativa.
- Somente os métodos do contrato são públicos.
- Demais métodos são privados
- Todos os seus campos são privados
- Não dá visibilidade *protegido* nem a campos nem a métodos

Classes Empacotadoras

- Incorpora funções que seriam de módulos com abstração de procedimentos.
- Uma classe *empacotadora* bem-construída:
  - encapsula um conjunto de rotinas ou constantes simbólicas relacionadas.
  - não possui campos
  - pode possuir também uma coleção de rotinas privadas.

Classes Estrutura de Dados

- Classes que definem e agrupam um conjunto de informação formada por dados heterogêneos
- Seus campos são acessados individualmente para leitura e escrita
- Não possui métodos, exceto as funções construtoras
- As boas classes ED devem ser de exclusivo uso local e privado
- Em Java são classes privadas, estáticas e aninhadas
- Exemplo:

```
private static class Node {
        public  Info valor;
        public  Node proximo;
        public  Node(Info v) { valor = v;}
}
```


Classes Depositárias

- São classes públicas de poucas instâncias.
- Seus objetos são depósitos de dados de uso geral.
- Seus campos não são relacionados.
- Seus métodos são em sua maioria de *definição de propriedades*, i.e., do tipo void setX(T a) e T getX()
- Não há métodos que operem sobre seus campos de forma relevante, por exemplo, causando mudança de estado.
- Exemplo:

```
class Orange {
        public float temperatura;   private double juros;
        public void setJuros(double a) { juros = a;}
        public double getJuros( ) {return juros;}
}
```

Classes Fantasma

- São classes públicas que não possuem declarações de campos.
- Somente possuem métodos **não-relacionados**.
- Seus métodos apenas operam sobre seus parâmetros, portanto sobre dados de outros objetos.
- Exemplo:

```
class Phantom {
        public void matricula(String nome, Curso c) { ... }
        public double imposto(double salario) { ... }
        public double temperatura(float t) { ...}
        public String fornecedor(Equipamento e)  {...}
}
```


Classes Polivalentes

- Implementam mais de um **contrato**.
- Seus métodos e campos podem ser particionados em subcontratos.
- Melhor se implementada como classes TAD distintas, uma para cada subcontrato

- **Exemplo**

```
class Esquizo {
      private int[ ] pilha;
      public void push(int x) {...}
      public void int pop() {...}
      public void empty1(int x) {...}

      private int[ ] lista;
      public void insert(int x) {...}
      public void remove(int x) {...}
      public void empty2(int x) {...}
 }
```

ORGANIZAÇÃO DE CONSTANTES

Constantes Simbólicas

- Inclusão de constantes como valores literais no código fonte dificulta manutenção
- Uso de strings literais dificulta internacionalização do software
- Somente deve-se incluir constantes literais que nunca são afetadas por mudanças na especificação, e.g., 12 = meses do ano
- Usar sistematicamente constantes simbólicas no lugar de contantes literais

Tipos de Constantes Simbólicas

1. Quanto ao Acoplamento:
   - ❑ De Sistema: de interesse geral
   - ❑ De Biblioteca: de interesse da biblioteca e de seus usuários
   - ❑ De Módulo: de interesse do módulo e seus usuários

2. Quanto ao Escopo:
   - ❑ pública: visível fora do componente
   - ❑ encapsulada: uso somente local ao componente

Constantes Simbólicas de Sistema

- ❑ São constantes dependentes de plataforma
- ❑ Devem ter o escopo de todo o sistema
- ❑ Não pertencem a um módulo ou biblioteca específicos
- ❑ Em geral, a alteração do seu valor não implica em manutenção dos módulos que a utilizam
- ❑ Devem ser agrupadas em um único pacote
- ❑ Pacote de constantes simbólicas consiste em um conjunto de classes públicas empacotadoras
- ❑ Cada classe define um conjunto coeso de constantes simbólicas públicas relacionadas

Constantes Simbólicas de Sistema ...

```
package CS;
public class CE1{
    public static final T C1 = valor1;
    .....
    public static final T Cn = valorn;
}
public class CE2{
    public static final T K1 = valor1;
    .....
    public static final T Kn = valorn;
} ...
```


Constantes Simbólicas Públicas de Biblioteca

```
package BibliotecaA;
public class CE1{
    public static final T C1 = valor1;
    .....
    public static final T Cn = valorn;
}
public class CE2{
    public static final T K1 = valor1;
    .....
    public static final T Kn = valorn;
} ...
```

Constantes Simbólicas Encapsuladas de Biblioteca

```
package BibliotecaA;
class CE1{
    static final T C1 = valor1;
    .....
    static final T Cn = valorn;
}
class CE2{
    static final T K1 = valor1;
    .....
    static final T Kn = valorn;
} ...
```


Constantes Simbólicas Públicas de Módulos

```
public class M1 {
    public static class CE1{
        public static final T C1 = valor1;
        .....
        public static final T Cn = valorn;
    }
    public static class CE2{
        public static final T K1 = valor1;
        .....
        public static final T Kn = valorn;
    } ...
}
```

Constantes Simbólicas Encapsuladas de Módulos

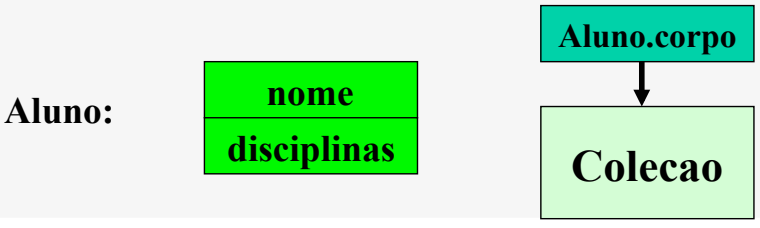
```
public class M1 {
    private static class CE1{
        static final T C1 = valor1;
        .....
        static final T Cn = valorn;
    }
    private static class CE2{
        static final T K1 = valor1;
        .....
        static final T Kn = valorn;
    } ...
}
```


EXEMPLOS DE IMPLEMENTAÇÃO DE CLASSES TAD

Controle Acadêmico

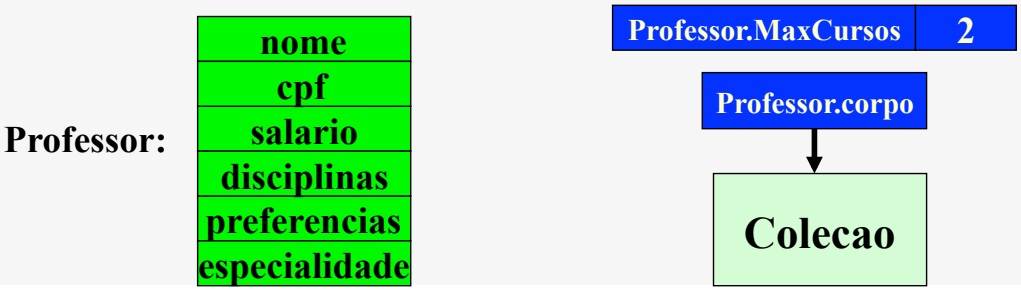
Sistema de Controle Acadêmico

```
class Aluno {
        private String nome;
        private Colecao disciplinas;

        public static Colecao corpo=new Colecao();
        public void matricular ( ) {...}
        public void trancar ( ) {...}
        public void print ( ) { ... };
        public void manter ( ) {...}
                ...
}
```



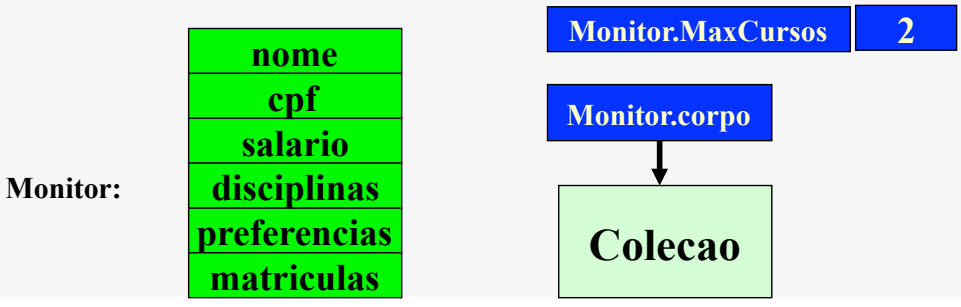
...Sistema de Controle Acadêmico

```
class Professor{
        private String nome;       private String cpf;
        private double salario;    private Colecao disciplinas;
        private Colecao preferencia;
        private static int MaxCursos = 2;
        private String especialidade;

        public static Colecao corpo=new Colecao();
        public void print ( ) { ...}
        public void manter ( ) {...}
        ...
}
```



...Sistema de Controle Acadêmico

```
class Monitor{
        private String nome;       private String cpf;
        private double salario;    private Colecao disciplinas;
        private Colecao preferencias;
        private static int MaxCursos = 2;
        private Colecao matriculas;
        public static Colecao corpo = new Colecao();
        public void print ( ) { ...}
        public void manter ( ) {...}
        ...
}
```



TAD Pilha de Inteiros

TAD PilhadeInteiros

```
class PilhaDeInteiros {
    private static final int FUNDO = -1;
    private int ultimo; private int[ ] item;
    private int top = Fundo;
    public static class CS {...}
    public static class ExMax extends Exception { ... }
    public static class ExMin  extends Exception { ... }
    public PilhadeInteiros(int tamanho) {...}
    public void push(int v) throws ExMax {...}
    public int pop( ) throws ExMin {... }
    public boolean vazia ( ) { ... }
    public static void main(String[ ] args) { ... }
}
```

Constantes Simbólicas de Módulo

```
public static class CS {
        public static final byte ESTOURO=1;
        public static final byte EMFALTA=2;
}
```


Classe de Exceções

```
public static class ExMax extends Exception {
        private byte id;
        public ExMax(byte id) {this.id = id;}
        public byte tipo( ) {return id;}
}

public static class ExMin extends Exception {
        private byte id;
        public ExMin(byte id)  { this.id = id; }
        public byte tipo( ) {return id;}
}
```

Operações de PilhadeInteiros

```
public PilhadeInteiros(int tamanho) {
        if (tamanho < 0) tamanho = 0;
        ultimo = tamanho -1 ;
        itens = new int[tamanho];
}
```


Operações de PilhadeInteiros...

```
public void push(int v) throws ExMax {
    if (top==ultimo) throw new ExMax(CS.ESTOURO);
    item[++top]= v;
}

public int pop( ) throws ExMin {
     if(vazia ( )) throw new ExMin(CS.EMFALTA);
     return item[top--];
}

public boolean vazia( ) {   return (top == FUNDO)
}
```

Teste Interno da PilhaDeInteiros

```
public static void main(String[ ] args) {
  PilhadeInteiros p1 = new PilhaDeInteiros(10),
  PilhaDeInteiros p2 = new PilhadeInteiros(50);
  PilhaDeInteiros p3 = new PilhaDeInteiros(5);
  PrintStream o = o;
  o.println("Pilhas de int, "+ "tamanhos 10, 50 e 5");  o.println( );
  try{ o.println("Insere 10 elems em pi:");
      for (int i = 0;i < 10; i++) { p1.push(i); o.println("topo de p1 = "+p1.top) }  }
  o.println( );
  o.println("Transfere de p1 para p2:");
  o.println("topo de p1 = + p1.top + ", topo de p2 = "+ p2.top);
  while (! P1.vazio( ))  {
       p2.push(p1.pop( ));
       o.println("topo de p1 = "+ p1.top + ", topo de p2 = "+ p2.top);
  }
```


Teste Interno da PilhaDeInteiros...

```
  o.println( );
  o.println("Transfere de p2 para out: ");
  while (! P2.vazio( )) o.print(p2.pop( ) + "  ");
  o.println( ); o.println( );
  o.println("Empilha 10 em p3: ");
  try{for (int i = 0;i < 10; i++) { p1.push(i);  p3.pusho(i); }
  catch(ExMax e)
         { o.println("p3:Exceção de pilha "+ "cheia, id = "+ e.tipo()); }
  o.println( );  o.println("Desempilha 10 de p3:");
  try { for (int i = 0;i < 10; i++)  p3.pop( ); o.println( ); }
  catch (ExMin e)
          {o.println("p3: Exceção de pilha + "vazia, id = " + e.tipo());}
  catch (Exception e) { } o.println("Acabou bem");
}
```

ADT PilhaDeInteiros ...

Pilhas de int, tamanhos 10, 50 e 5

Insere 10 elems em p1:

```
Topo de p1 = 0
Topo de p1 = 1
Topo de p1 = 2
Topo de p1 = 3
Topo de p1 = 4
Topo de p1 = 5
Topo de p1 = 6
Topo de p1 = 7
Topo de p1 = 8
Topo de p1 = 9
```

Transfere tudo de p1 para p2:

```
Topo de p1 = 9, top de p2 = -1
Topo de p1 = 8, top de p2 = 0
Topo de p1 = 7, top de p2 = 1
Topo de p1 = 6, top de p2 = 2
Topo de p1 = 5, top de p2 = 3
Topo de p1 = 4, top de p2 = 4
Topo de p1 = 3, top de p2 = 5
Topo de p1 = 2, top de p2 = 6
Topo de p1 = 1, top de p2 = 7
Topo de p1 = 0, top de p2 = 8
Topo de p1 = -1, top de p2 = 9
```


ADT PilhaDeInteiros

Transfere de p2 para out: 0 1 2 3 4 5 6 7 8 9

```
Empilha 10 em p3:

p3: Exceção de pilha cheia, id = 1
```

```
Desempilha 10 de p3:

p3:Exceção de pilha vazia, id = 2
```

Acabou bem

ADT PilhaDeObjetos

```
class PilhaDeObjetos1 {
        private static final int FUNDO = -1;
        private int Ultimo, top = Fundo;  private Object[  ]  item;
        public PilhaDeObjetos1(int MaxTam) {
              this.Ultimo = MaxTam -1 ;  item = new Object[MaxTam];
        }
        public void push(Object v) throws Ex_Max {
              if (top == Ultimo) throw new Ex_Max(CS.ESTOURO);
              item[++top]= v;
        }
        public Object pop( ) throws Ex_Min {
              if(vazio( )) throw new Ex_Min(CS.EMFALTA);
              return item[top--];
        }
        public boolean vazio( ) { return (top == FUNDO) }
}
```


ADT PilhaDeObjetos

```
class PilhaDeObjetos2 {
    private static class Elemento {
            Elemento link; Object info;
            Elemento(Object info){this.info = info;}
     }
    private Elemento item; private Elemento topo;
    public void push(Object v) throws ExMax{
            try{Elemento novo = new Elemento(v); }
            catch(Exception e){throw new ExMax(Estouro);}
            novo.link = topo;  topo = novo;
    }
   public Object pop( ) throws ExMin {
          Object info;
          if(vazio( )) throw new ExMin(EmFalta);
          info = top.info;  top = top.link;  return info;
   }
  public boolean vazio( ) {  return ( topo == null); }
}
```

EFEITO COLATERAL EM FUNÇÕES

O Que é Efeito Colateral?

- O que é efeito colateral em Função?
- A obrigação da função é simplesmente retornar um valor, mas além disto ela muda o estado do objeto.
- Todo procedimento deve ter efeito colateral.
- O procedimento só existe pelo efeito colateral.
- Um procedimento sem efeito colateral não serve para nada.
- Efeito colateral em procedimentos pode mudar valor de parâmetros e/ou de variáveis globais

Efeito Colateral é Nocivo

- Geralmente **Efeito Colateral em Funções** é nocivo a boa qualidade da programação:
  - (getint( ) + getint ( ) = = 2 * getint())
  - (x + f(x) == f(x) + x)
  - a[++i] = a[++i] = a[++i] = 0;
  - Asserção: a[i=0] > 0

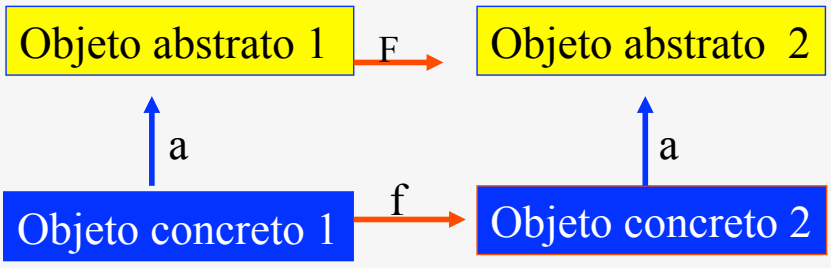
Efeito Colateral Útil

- Bom exemplo de uso de efeito colateral:
  randomSeed(seed) ; ... ; x = nextRandom( )
- Efeito colateral pode ser evitado - solução sem efeito colateral em função:
  rand.create(seed); rand.next( ); x = rand.value( )
- Foi criado o objeto rand com as operações: next e value, separando-se a questão de retornar o número gerado da de gerar o número.
  - rand.value( ) retorna sempre o mesmo número porque foi o único gerado.
  - rand.next(x) retorna sempre o próximo número.
  - rand.create(seed) inicializa o número aleatório com uma semente.

… O Que é Efeito Colateral?

- Efeito Colateral em Java aparecer em:
- atribuições a atributos de classes
  - chamada a método x.r(…), sendo x um atributo do tipo classe e r uma rotina com efeito colateral
  - chamada a s(…), sendo s uma rotina local que produz efeito colateral

Estado Concreto X Estado Abstrato



- F = uma operação no ADT A
- f = a implementação de F na classe C
- a = função de abstração que mapeia o objeto concreto no objeto abstrato representado.

Uma implementação é **correta** se: $\mathbf{a} \circ \mathbf{f} = \mathbf{F} \circ \mathbf{a}$

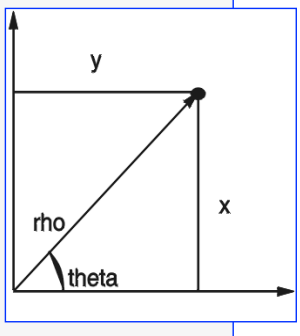
Efeito Colateral Legítimo

Efeito colateral no **estado concreto** dos objetos é **inofensivo**, mas efeito colateral que ocorre no **estado abstrato** é **nocivo**.

```
class IntQueue {
    private int counter, queue[ ];  private  boolean counterDefined;
    IntQeue(...){ ... }
     ....................
    public int numOfElements ) {
        if (!counterDefined) {
              counter = compute num de elementos;
              counterDefined = true;
         };
         return counter;
    }
}
```

numOfElements( ) pode produzir efeito colateral interno, mas externamente não há efeito colateral algum.

Classe COMPLEX com Bom Efeito Colateral




```
class Complex { public Complex( ) { ...}
                public Complex(float x, float y) {...}
                public Complex add(Complex z) {...}
                public Complex sub(Complex z) {...}
                public Complex mult(Complex z) {...}
                public Complex divide(Complex z) {...}
                public float x( ) {...}
                public float y( ) {...}
                public float rho( ) {...}
                public float theta( ) {...}
}
```

•**Qual seria a melhor representação de um número complexo?**

Classe Complex

```
class Complex {
    private Boolean cartesian, polar;
    private float privX , privY,  privRho, privTheta;
    public Complex( ) {
       privX = 0.0; privY = 0.0;
       polar = false; cartesian = true;
    }
    public Complex(float x, float y) {
       privX = x;  privY = y;
       polar = false; cartesian = true;
    }
   Operações de Complex
}
```


Operações deComplex ...

```
private void makeCartesian() {
      if (!cartesian) {
         privX = privRho*cos(privTheta);
         privY = privRho*sin(privTheta);
         cartesian = true;       }
 }
 private void make_polar() {
      if (!polar) {
          privRho = sqrt(privX*privX + privY*privY);
          privTheta = atan2(privY,privX);
          polar = true;     }
}
```

Operações de Complex ...

```
public float x( ) {  make_cartesian();   return privX;   }

public float y( ) { make_cartesian();   return privY;  }

public float rho( ) { make_polar();  return privRho;  }

public float theta( ) { make_polar(); return privTheta;  }
```


Operações de Complex

```
public Complex add(Complex z) {
        Complex t = new Complex( ); make_cartesian( );
         t.privX = privX + z.x( );  t.privY = privY + z.y( );
         t.polar = false;  t.cartesian = true;
         return t;
}

public Complex divide(Complex z) {
        Complex t = new Complex( ); make_polar( ) ;
        t.privRho = privRho / z.rho();
        t.privTheta =(privTheta-z.theta( ));
        t.cartesian = false; t.polar = true;
}
.............
```

Uso de Complex

```
class A {
      public static void main(String[] args) {
          Complex z1 = new Complex(1,2);
          Complex z2 = new Complex(3,4); Complex z3;
          float x1, x2, x3,  r1, r2, r3;

          r1 = z1.rho( ); r2 = z2.rho( );
          z3 = z1.add(z2); r3 = z3.rho( );
          x2 = z2.x( ); x3 = z3.x( );
          z1 = z3.divide(z2); x1 = z1.x( );
          r2 = z2.rho( ); x3 = z3.x( );
      }
}
```