Universidade de Brasília Faculdade UnB Gama

Orientação a Objetos

Aula 6 - Encapsulamento

Daniel Porto

daniel.porto@unb.br

APRESENTAÇÃO

Encapsulamento

ENCAPSULAMENTO

Este conceito, também chamado de acessibilidade ou ocultamento por alguns estudiosos, corresponde a uma característica importante ao se trabalhar com objetos.

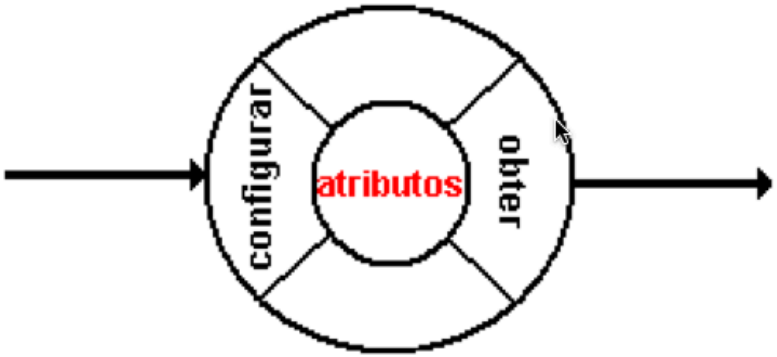
Por meio dele são combinados dados e comportamentos em um objeto para controlar e fornecer certa estabilidade e segurança ao estado do objeto (instância de uma classe).

O encapsulamento promove o comportamento de caixa preta nos objetos.

ENCAPSULAMENTO

A incorporação deste conceito em POO consiste na implementação de objetos que **nunca fornecerão acesso direto** aos seus atributos (instâncias).

Este acesso sempre acontecerá através da troca de mensagens com acionamento de métodos específicos da classe que instanciam (possui) estes atributos.



O acesso aos atributos da classe só acontece por meio dos métodos da própria classe, tanto no fornecimento quanto na alteração de seus valores.

ENCAPSULAMENTO

Este conceito garante:

- facilidade e agiliza na **manutenção** do programa
- possibilita **reutilização**
- fornece **confiabilidade**

porque o próprio objeto pode modificar totalmente como seus dados são armazenados, porém seus métodos continuarão fornecendo suas informações desejadas sem que os recursos solicitantes destes dados se importem como estes dados foram alcançados.

ENCAPSULAMENTO

Métodos Privados

Os atributos e métodos são membros de uma classe, podendo cada um deles possuir seu qualificador de acesso adequado a lógica implementada em um programa Java.

Conforme o encapsulamento, os atributos de uma classe deverão ter acesso restrito aos métodos da classe, podendo alguns **métodos também serem encapsulados**.

Esta situação ocorre com certa frequência, pois é comum a ”quebra” de um código em vários métodos, em que alguns deles não terão utilidade ”pública”, mas somente específica as operações que um método específico desta classe tenha que realizar para chegar ao seu resultado esperado.

ENCAPSULAMENTO

Normalmente, os métodos privados são definidos por:

- não serem de interesse dos usuários e serviços da aplicação desenvolvida
- não serem facilmente suportados para uma possível modificação na implementação de sua classe

Para realizar a implementação de um método privado, simplesmente troque seu qualificador de acesso para private e este método só poderá ser acessado por outros métodos definidos na sua própria classe, por exemplo:

```
private boolean verificaStatusSaldo (float valor) {
        :
}
```

ENCAPSULAMENTO

Métodos *getters* e *seters*

Sendo os atributos de uma classe privados o acesso aos mesmos só acontecerá através de métodos existentes nesta própria classe. Assim, um padrão adotado na POO em Java é a elaboração de métodos de acesso ao conteúdo dos atributos (get) e de modificação dos mesmos (set).

A elaboração destes métodos recebem o prefixo correspondente (get ou set) em seus identificadores (nomes), por exemplo:

```
public void setAno(int ano) {
        :
}
```

ENCAPSULAMENTO

O método setAno realiza a alteração do estado do atributo **ano** em um objeto que o declarou como uma variável de instância private.

Observe que este método recebe um parâmetro (**int ano**) que será o responsável pelo novo valor que este atributo receberá.

A simples obtenção do valor armazenado neste atributo poderia ser conseguida através do acionamento do método denominado getAno, que possivelmente não receberia nenhum parâmetro, mas retornaria o valor da variável de instância denominada **ano**.

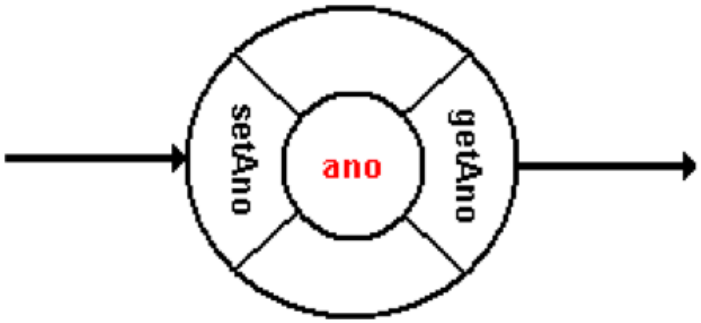
```
public int getAno( ) {
        :
}
```

ENCAPSULAMENTO

Observe que o qualificador de acesso destes dois métodos (setAno e getAno) são públicos, apesar do atributo acessado (ano) ser privado (private).

Dessa forma, o atributo só poderá ser acessado por outras classes através de seus próprios métodos, respectivamente acionados de acordo com a necessidade do processamento a ser realizado:

- acesso ou obtenção de seu conteúdo (get)
- alteração ou configuração de seu conteúdo (set)



ENCAPSULAMENTO

```java
/** Síntese
  *    Objetivo: registrar uma data
  *    Entrada:  dia, mês, ano
  *    Saída:    data completa com dia da semana
 */
import java.util.Scanner;
public class RegistraData {
    public static void main(String[] args) {
        int valor;   // usada na leitura de dia, mês e ano
        Data data = new Data();
        Scanner ler = new Scanner(System.in);
        do {
            System.out.print("Informe o ano: ");
            valor = ler.nextInt();
        } while (valor < 1);
        data.setAno(valor);

        do {
            System.out.print("Informe o mês: ");
            valor = ler.nextInt();
        } while ((valor < 1) | (valor > 12));
        data.setMes(valor);
```

ENCAPSULAMENTO

```
23          // continuação do exemplo anterior
24
25          System.out.print("Informe o dia: ");
26          do {
27              valor = ler.nextInt();
28              if (!data.validaDia(valor))
29                  System.out.print("Dia inválido, digite outro: ");
30              else
31                  data.setDia(valor);
32          } while (!data.validaDia(valor));
33
34          mostraData(data.getDiaSemana(),data.getDia(),
35                     data.getMes(), data.getAno());
36      } // encerra o método principal
37
38      // Outro método da classe RegistraData
39      public static void mostraData(String diaSemana,int dia,int mes,int ano){
40          System.out.print("\nDia "+dia+"/"+mes+"/"+ano+" - "+diaSemana);
41      }
42  }
```

ENCAPSULAMENTO

```java
/** Síntese
 *    Atributos: dia, mês, ano, dia da semana
 *    Métodos: getAno(), getMes(),getDia(),setAno(int),
 *             getDiaSemana(),setMes(int), setDia(int),
 *             validaDia(int), semana(String)
 */
import java.util.Date; // classe de manipulação de data
import java.util.Scanner;

public class Data {
    // define registro de Data (atributos)
    // Similar a Struct em C
    private int dia;
    private int mes;
    private int ano;
    private String diaSemana;

    // métodos de acesso aos atributos da classe Data

    // obtém ou fornece o conteúdo do atributo ano
    public int getAno() {
        return this.ano;
    }
```

ENCAPSULAMENTO

```
24      // continuação do exemplo anterior
25
26      // configura ou altera o conteúdo do atributo ano
27      public void setAno(int anoParametro) {
28          this.ano = anoParametro;
29      }
30      public int getMes() {
31          return this.mes;
32      }
33      public void setMes(int mesParametro) {
34          this.mes = mesParametro;
35      }
36      public int getDia() {
37          return this.dia;
38      }
39      public void setDia(int diaParametro) {
40          this.dia = diaParametro;
41      }
42      public String getDiaSemana() {
43          return this.diaSemana;
44      }
```

ENCAPSULAMENTO

```
45          // continuação...
46          protected boolean validaDia(int dia) {
47              boolean valida = false;
48              switch (this.mes) {
49                  case 1:
50                  case 3:
51                  case 5:
52                  case 7:
53                  case 8:
54                  case 10:
55                  case 12:
56                  if (dia > 0 && dia <= 31)
57                      valida = true;
58                      break;
59                  case 4:
60                  case 6:
61                  case 9:
62                  case 11:
63                      if (dia > 0 && dia <= 30)
64                      valida = true;
65                      break;
```

ENCAPSULAMENTO

```
66                  case 2 :
67                      if (dia > 0 && dia <= 28)
68                          valida= true;
69                      else
70                          if (dia == 29) {
71                              if (((ano%4 == 0) && (ano%100 != 0)) || (ano%400 == 0))
72                                  valida = true;
73                          }
74              } // encerra switch
75              if(valida) {
76                  StringBuilder hoje = new StringBuilder();
77                  hoje.append(this.mes);
78                  hoje.append("/");
79                  hoje.append(dia);
80                  hoje.append("/");
81                  hoje.append(this.ano);
82                  semana(hoje.toString());
83              }
84              return valida;
85          }
86          private void semana(String dataDesejada) {
87              Date data = new Date(dataDesejada);
88              switch(data.getDay()) {
89                  case 0:
90                      diaSemana = "Domingo";
```

ENCAPSULAMENTO

```
91              // continuação do exemplo anterior
92
93                  case 1:
94                      diaSemana = "Segunda-feira";
95                      break;
96                  case 2:
97                      diaSemana = "Terça-feira";
98                      break;
99                  case 3:
100                     diaSemana = "Quarta-feira";
101                     break;
102                 case 4:
103                     diaSemana = "Quinta-feira";
104                     break;
105                 case 5:
106                     diaSemana = "Sexta-feira";
107                     break;
108                 case 6:
109                     diaSemana = "Sábado";
110             }
111         }
112
113     } // fim da classe Data
```

ENCAPSULAMENTO

DEPRECATED

Observe em seu programa, no IDE Eclipse, que a classe Date e o método getDay() aparecem de maneira diferente das demais instruções. Como informação deste ambiente é realizado um alerta de que estes recursos são desencorajados ao uso (~~deprecated~~), normalmente, por sua descontinuidade em novas versões Java, em que elas devem ter sido substituídas por novas classes e métodos.

Por exemplo no trecho do código anterior:

```
88          switch(data.getDay()) { // getDay() é "Deprecated"
89               case 0:
90                  diaSemana = "Domingo";
91                  break;
92                       :
```

ENCAPSULAMENTO

Observe no programa anterior que o acesso aos atributos da classe Data acontecem somente por meio de seus métodos, em que cada um deles respeita o padrão de acesso e modificação através do prefixo set e get em seus identificadores.

Analisando a classe Data também é possível observar como Java cria **Estruturas de Dados Compostas Heterogêneas** (registros). Estas estruturas são criadas e manipuladas pelo programa por meio de suas classes.

```
public class Data() { //elementos de Data (estrutura heterogênea)
        private int dia;
        private int mes;
        private int ano;
        private String diaSemana;
}
```

ENCAPSULAMENTO

Analisando o registro Data, do exemplo anterior, é possível observar que os métodos get e set foram elaborados com êxito, porém um atributo privado ficou sem a especificação destes métodos padrões (diaSemana).

No entanto, o valor coerente a ser armazenado neste elemento do registro é gerado e apresentado na execução do programa.

Exemplo da console (execução):

Informe o ano: 2000
Informe o mês: 2
Informe o dia: 29

Dia 29/2/2000 - **Terça-feira**

ENCAPSULAMENTO

A prática do desenvolvimento destes métodos **get** e **set** na programação em Java deve ser condicionada a situação lógica do programa que atenderá as necessidades existentes no problema envolvido.

Dessa forma, estes métodos devem ser planejados antes de serem implementados, não sendo uma ”boa prática” de programação, e nem da realização do encapsulamento, a simples implementação destes métodos para qualquer atributo existente em uma classe.

Exemplo: suponha que o usuário deste programa tivesse informado que sexta-feira fosse o dia da semana para 29/2/2000, através de seu método setDiaSemana, enquanto o getDiaSemana mostraria este dia gerando um dado inconsistente (errado) a nossa realidade.

ENCAPSULAMENTO

Instrução this

O uso do this em um método referência o objeto no qual este método esteja operando. Como exemplo observe o método **setAno** que altera o valor do atributo private ano definido na classe Data através do this.

O uso desta instrução referencia os atributos existentes no objeto de Data no qual o método foi acionado (objeto alvo).

Por exemplo: Suponha a execução da chamada de método obj1.calculaTotal(obj2), em que cada ocorrência do this representa a referência ao objeto referido por obj1; a expressão this.aux representa o acesso a variável aux do objeto referido por **obj1** (objeto corrente).

ENCAPSULAMENTO

Algumas outras aplicações do **this** são possíveis na POO em Java, mas neste momento é interessante conhecer seu uso também sobre métodos construtores.

Entre as características especiais deste método ainda existe a possibilidade dele acionar um **outro método construtor** na mesma classe, em sua primeira linha de código. Por exemplo:

```
1  class Aluno {
2      Aluno (String nome) {
3          this(nome, Matricula.getNovaMatricula());
4      }
5      public Aluno(String nome, int matricula) {
6          chamada = nome;
7          codigo = matricula;
8              :
9      }
10 }
```

EXERCÍCIO DE FIXAÇÃO

1) Desenvolva um programa orientado a objeto que possua um método construtor para classe Aluno. Esta classe é composta por dois atributos (matrícula inteira e nome - String) e pode criar quantos alunos o usuário desejar, respeitando o limite máximo de 100 cadastros de Alunos. Seu programa deverá encapsular esta classe Aluno e fazer as respectivas validações para estes dados em métodos específicos para cada atributo (serviços). O valor máximo de cadastro estará definido na constante MAXIMO que estará declarada em outra classe executável que gerenciará a aplicação elaborada e que possibilitará sua execução. Quando o usuário desejar encerrar estes cadastros, ou o limite for atingido, seu programa deverá realizar um salto de 20 linhas na console e apresentar de maneira tabelar (forma de tabela) todos os cadastros realizados. Todos os métodos presentes na classe Aluno não poderão ser estáticos.

ENCAPSULAMENTO

Compreendendo alguns Qualificadores de Acesso

Os membros de uma classe (atributos e métodos) podem ser definidos como **static** ou não. Os criados como static pertencem a classe e não estão condicionados a criação de seus objetos para serem utilizados.

A sintaxe para o acesso a um membro de uma classe é:

identificadorObjeto.identificadorAtributo

ou

identificadorObjeto.identificadorMétodo()

Para membros **static** o acesso pode ser pela classe:

identificadorClasse.identificadorAtributo

ou

identificadorClasse.identificadorMétodo()

ENCAPSULAMENTO

```java
/** Síntese
 *    Objetivo: cadastro de produto de uma loja
 *    Entrada:  sem entrada
 *    Saída:    código,preço unitário e nome do produto
 */
public class Loja {
    private static class Produto {  // define estrutura
        public int codigo;             // heterogênea
        public double preco;
        public String nome;
    }

    public static void main(String[] args) {
        Produto prod;   // cria uma referência a Produto
        prod = new Produto(); // aciona construtor alocação
        prod.codigo = 1;
        prod.preco = 300.00;
        prod.nome = "filmadora";
        System.out.print("Código: " + prod.codigo + " =>");
        System.out.print("\t" + prod.nome + "\tR$" + prod.preco);
    }
}
```

ENCAPSULAMENTO

A classe Loja possui dentro de si a definição privada da classe Produto

Produto esta definida como **private** (acesso somente de códigos existentes dentro da classe Loja)

Como **main()** é **static** ele só pode usar variáveis e classes que tenham sido declaradas dentro dele ou que também sejam **static**, sendo por esta razão que Produto é **static**

Os atributos de Produto foram definidos como **public**, o que indica que quem conseguir acesso a esta classe poderá manipular seus atributos diretamente (sem get/set)

Estes atributos não são **static**, sendo necessária a criação de um objeto Produto para que os mesmos sejam utilizados

A variável **prod** é criada para receber o endereço do novo objeto Produto, sendo somente por meio dela efetuada a manipulação direta dos atributos de Produto

ENCAPSULAMENTO

Acesso a Membros da Classe e do Objeto

Um método pode acessar os dados privados do objeto no qual é chamado, além de também conseguir acessar todos os dados privados de todos os objetos gerados por sua mesma classe.

Outra informação importante, e que confunde muitos programadores, esta relacionada a passagem de parâmetros em Java quando envolve variáveis de referência.

Muitos indivíduos acreditam que esta passagem é feita por referência, porém **Java só passa parâmetros por valor**, mesmo as variáveis que fazem referência de memória a outros objetos.

ENCAPSULAMENTO

No intuito de elucidar a POO incorporando o uso de estruturas de dados e o encapsulamento acompanhe a evolução do exemplo anterior sobre o registro de produtos em uma loja de magazine.

```
1   /** Síntese
2    *    Objetivo: cadastro de produto de uma loja
3    *    Entrada:  sem entrada
4    *    Saída:    código,preço unitário e nome do produto
5    */
6   public class RegistraProduto {
7       public static void main(String[] args) {
8           Produto prod = new Produto(); // cria referência e
9           prod.setCodigo(1);            // o objeto prod
10          prod.setPreco(300);
11          prod.setNome("filmadora");
12          System.out.print("Código: " + prod.getCodigo() + " =>");
13          System.out.print("\t" + prod.getNome() + "\tR$" + prod.getPreco());
14      }
15  }
```

ENCAPSULAMENTO

```java
/** Síntese
 *   Atributos: código, preço, nome
 *   Métodos: getCodigo(), getPreco(), getNome()
 *      setCodigo(int),setPreco(double),setNome(String)
 */
public class Produto {
    // define estrutura heterogênea Produto
    private int codigo;
    private double preco;
    private String nome;

    // métodos set e get para Produto
    public int getCodigo() {
        return codigo;
    }
    public void setCodigo(int vCodigo) {
        this.codigo = vCodigo;
    }
    public double getPreco() {
        return preco;
    }
    public void setPreco(double vPreco) {
        this.preco = vPreco;
    }
```

ENCAPSULAMENTO

```
25        // continuação do exemplo anterior
26
27        public String getNome() {
28            return nome;
29        }
30        public void setNome(String vNome) {
31            this.nome = vNome;
32        }
33    }
```

Programa implementado com 2 arquivos físicos distintos e 2 classes públicas (RegistraProduto e Produto).

Apesar da classe Produto ser pública seus atributos são **privados** e só podem ser acessados pela própria classe.

Os métodos **get** e **set** da classe Produto são públicos.

Na classe RegistraProduto foi criado um objeto Produto identificado por **prod**, o que possibilitou acesso a todos os métodos públicos desta classe.

Observe que o acesso aos métodos é sempre pelo objeto **prod**.

ENCAPSULAMENTO

Evoluindo este mesmo exemplo para o uso das estruturas de dados em conjunto (homogêneas e heterogêneas), além da leitura dos dados fornecidas pelo usuário, tem-se a seguinte implementação na classe **RegistraProduto**, pois a classe **Produto** se mantém **sem nenhuma alteração** (bem encapsulada).

```
1   /** Síntese
2    *   Objetivo: cadastros de produtos de uma loja
3    *   Entrada:  código,preço unitário,nome dos produtos
4    *   Saída: código,preço unitário,nome de cada produto
5    */
6   import java.util.Scanner;
7   public class RegistraProduto2 {
8       public static void main(String[] args) {
9           Scanner ler = new Scanner(System.in);
10          // matriz 2x3 de referencia a Produto
11          Produto prod[][] = new Produto[2][3];
12          int automatico = 0;  // códigos gerados automáticos
```

ENCAPSULAMENTO

```
13          // continuação...
14          for(int aux=0;aux < 2; aux++) {
15              for(int cont=0; cont < 3; cont++) {
16                  // cria um objeto Produto por vez
17                  prod[aux][cont] = new Produto();
18                  String auxNome = new String();
19                  double auxPreco;
20                  // código gerado automaticamente
21                  prod[aux][cont].setCodigo(++automatico);
22
23                  // Leitura do nome do Produto
24                  do {
25                      System.out.print("Digite nome do produto: ");
26                      auxNome = ler.next();
27                      auxNome = auxNome.trim();
28                  } while (auxNome == null);
29                  prod[aux][cont].setNome(auxNome);
30
31                  // Leitura do preço de cada Produto
32                  do {  System.out.print("Informe seu preço: ");
33                      auxPreco = ler.nextDouble();
34                  } while (auxPreco <= 0);
35                  prod[aux][cont].setPreco(auxPreco);
36              }
37          }
```

ENCAPSULAMENTO

```
38              // continuação...
39              for(int aux=0;aux < 2; aux++) {
40                  for(int cont=0; cont < 3; cont++) {
41                      System.out.print("Código: " +
42                                              prod[aux][cont].getCodigo() + " =>");
43                      System.out.println("\t" + prod[aux][cont].getNome()+ "\tR$"+
44                                              prod[aux][cont].getPreco());
45                  }
46              }
47          }
48      }
```

Programa implementado com 2 arquivos físicos distintos e 2 classes públicas (RegistraProduto2 e Produto).

Os atributos de Produto são privados e só podem ser acessados por seus métodos públicos **get** e **set**.

Na classe RegistraProduto2 é criada uma matriz 2x3 indicando que 6 produtos poderão ser criados e suas referências serão armazenadas nas posições desta matriz.

O acesso ao Produto é sempre pelo objeto **prod** e seus índices.

EXERCÍCIOS DE FIXAÇÃO

2) Elabore um programa que realize o cadastro de contas bancárias através do registros dos dados: número da conta; nome do cliente (principal proprietário da conta); saldo atual. Este cadastro será efetuado em um posto de atendimento bancário que pode possuir até 50 contas abertas com números completamente diferentes. Crie um menu com as opções relacionadas abaixo e as implemente adequadamente ao encapsulamento.

a) Cadastrar nova conta respeitando o limite máximo definido em uma constante em seu programa que deverá sempre ser usada em seu código, onde for conveniente;
b) Mostrar todas as contas de um determinado cliente;
c) Fechar (desativar) uma conta especifica;
d) Consultar todas as contas desativadas (máximo 50);
e) Encerrar o sistema de gerenciamento de contas.

EXERCÍCIOS DE FIXAÇÃO

3) Elabore um programa seguindo todas as características de encapsulamento que permita a uma prefeitura efetuar uma pesquisa entre os habitantes de sua cidade, coletando informações sobre o salário, idade, sexo e número de filhos. Este programa deverá ler os dados de uma quantidade indeterminada de pessoas (quantidade menor que mil, e armazenar em um registro cada dado registrado. Ao final deverá ser mostrado:

- menor idade entre os entrevistados;
- maior salário registrado;
- média do número de filhos;
- média do salário das pessoas registradas;
- média dos homens com salário superior a R$300,00;
- quantidade de pessoas que tem salário maior que a média de todas as pessoas pesquisadas.