Unicesumar

Programação I

Encapsulamento

Prof. Me. Fábio Perfetto

Definição de Classes em Java

Como foi demonstrado, uma classe tem dois grupos de elementos:
• seus Atributos, também denominados de variáveis-membro ou campos; e
• seus Métodos, operações ou funções-membro.

A sintaxe Java para a definição de uma "classe elementar" é bastante simples, constituindo-se de uma declaração seguida de um corpo delimitado por chaves, no qual residem as declarações de todos os atributos e métodos da classe, como mostra a seguinte estrutura de código:

```
[Modificador] class NomeDaClasse {
// Corpo da Classe
// ------------------------------------
// Atributos, ou variáveis da Classe
:
// Métodos, ou funções da Classe
:
}
```

PrimeiraClasse.java

O Modificador especifica a acessibilidade da classe (opcional), se presente, pode ser uma combinação de public e abstract ou final.

No momento, é necessário entender que uma classe declarada com um especificador de acesso public, indica que todo o conteúdo (atributos e métodos) público da classe pode ser utilizado livremente (sem restrições) pelas classes das aplicações, por outras classes do mesmo pacote (package) ou de outro pacote. Veja, a seguir, a construção de uma nova classe pública, denominada **PrimeiraClasse**:

```
public class PrimeiraClasse {
// Corpo da Classe
// ---------------------------------------
// Declaração de Atributos
:
// Declaração de Métodos
:
}
```

Salvando o Código da Classe

Ao salvar o código correspondente a uma classe em um arquivo, deve-se ter os seguintes cuidados:

1. Em um arquivo fonte Java, pode existir várias definições diferentes de classes, mas apenas uma delas pode ser pública. Se desejarmos várias classes públicas, então, cada uma delas deverá ser salva, separadamente, em arquivos distintos.
2. O "nome do arquivo" deve ser sempre o "nome da classe" pública que ele contém, observando-se cuidadosamente o mesmo uso de letras maiúsculas e minúsculas, tanto para o nome da classe como para o nome do arquivo (Java é uma linguagem sensível ao uso de letras maiúsculas e minúsculas em tudo).

Portanto, o nome de um arquivo, para conter a classe pública PrimeiraClasse, deve ser PrimeiraClasse.java. O que inicialmente pode parecer uma restrição é, na verdade, um mecanismo inteligente e simples para manter nomes realmente representativos para os arquivos de um projeto.

Regras para Nomes de Classes

Lembrando: O nome de uma variável ou classe em Java pode ser formado por uma sequência de um ou mais caracteres alfabéticos e numéricos, iniciados por uma letra ou ainda pelos caracteres de sublinhar '_' ou cifrão '**$**'.

Em Java, recomenda-se que a declaração de classe utilize nomes iniciados com letras maiúsculas, "diferenciando-se" dos nomes de variáveis ou instâncias de objetos que devem iniciar com uma letra minúscula. Caso o nome seja composto de mais de uma palavra, todas as iniciais de cada palavra também deverão ser iniciadas com letras maiúsculas, tal como nos exemplos:

Bola	Socket	Filter
BolaDeFutebol	ServerSocket	DataChangeObserver

A utilização de caracteres numéricos no nome de classes também é livre, ao passo que o uso do traço de sublinhar (*underscore* '_') não é indicado.

Tal como no caso de variáveis, é recomendável uma escolha criteriosa para os nomes das classes, de forma que estes sejam verdadeiramente significativos, expressando de maneira simples a ideia central da classe ou dos objetos que a mesma representa.

Atributos, ou Propriedades (1/3)

As classes podem ter zero, um ou mais atributos, que são variáveis destinadas a armazenar informações intrinsecamente associadas aos objetos representados. Por exemplo, ao falarmos de um "Objeto Bola" qualquer, seu tamanho é uma variável associada, pois toda bola, sendo um objeto concreto, deve ter um tamanho. O tamanho pode ser definido mais rigorasamente como uma medida de raio pois essa medida está associada naturalmente à sua forma geométrica. Assim sendo, é bastante natural definirmos uma classe **Bola** que possua como um atributo uma variável destinada a armazenar seu tamanho ou raio, como abaixo:

```
// Bola.java
public class Bola {
// Corpo da Classe
// --------------
// Atributos
  double raio;
}
```

Como mostra o trecho de código dado, a adição de um atributo a uma classe correspondente à simples declaração de uma variável de um certo tipo, cujo nome deveria indicar seu propósito. Tal declaração deve ser colocada dentro do corpo da classe, ou seja, dentro das chaves que delimitam a declaração da classe. Muitas vezes, nos referimos aos atributos de uma classe como seus campos (*fields*) ou também como suas variáveis-membro (*members*).

Atributos, ou Propriedades (2/3)

A porção dos dados manipulados por uma aplicação, ou classe Java.

Atributo é uma propriedade nomeada de um tipo.

A definição de *dados* ou *atributos* é idêntica à descrição dos campos de um registro, isto é, uma lista de identificadores com tipos associados.

Define o "estado" de um objeto (variáveis de instância).

Os atributos descrevem as características, propriedades, dos objetos analisados ⇔ Estrutura de Dados.

Atributos, ou Propriedades (3/3)

A sintaxe utilizada para definir um atributo de um objeto é:

```
[modificador] tipo nome [ = default];
```

onde:

- **modificador** (opcional), uma combinação de especifacor de acesso (public, protected ou private); final e static.
- **tipo** deve ser um dos tipos de dados da linguagem Java ou o nome de uma classe.
- **nome** deve ser um identificador válido.
- **default** (opcional) é a especificação de um "valor inicial" para a variável.

exemplos de declarações:

```
public int campoInt;                // atributo com acesso público do tipo inteiro
protected boolean campoBoolean;     // atributo com acesso protegido do tipo lógico
private char campoChar;             // atributo com acesso privado do tipo char
static byte campoByte;              // atributo com acesso estático do tipo byte
String campoString;                 // campo com acesso no pacote do tipo String
```

Variáveis de Instância (final)

Algumas variáveis de instância (ou atributos) precisam ser modificáveis e algumas não. Utiliza-se a palavra chave final para especificar o fato de que uma variável não é modificável (constante) e que qualquer tentativa de modificar é um erro.

```
public class Exemplo1 {

    public static void main(String[] args) {
      Vetor objVetor = new Vetor(10);

      objVetor.alimentarVetor();
      objVetor.imprimirVetor();
    }
}

class Vetor {
  private final int maxTam;
  public int v[];

  public Vetor(int n) {
    maxTam = n;
  }

  public void alimentarVetor() {
    v = new int[maxTam];
    for (int i=0; i<maxTam; i++) {
      v[i] = (int)Math.round(Math.random()*20);
    }
  }

  public void imprimirVetor() {
    for (int i=0; i<maxTam; i++) {
      System.out.printf("v[%d] = %d\n", i, v[i]);
    }
  }
}
```

```
Saída - aulaTres (run)
run:
v[0] = 19
v[1] = 13
v[2] = 1
v[3] = 19
v[4] = 18
v[5] = 10
v[6] = 9
v[7] = 8
v[8] = 6
v[9] = 3
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

As variáveis final podem ser inicializadas apenas na sua declaração ou no método construtor da classe.

Métodos, ou operações (1/3)

Enquanto os atributos permitem armazenar dados associados aos objetos, ou seja, valores que descrevem a aparência ou o estado de um certo objeto, os métodos (*methods*) ou funções-membro realizam operações sobre os atributos de uma classe ou são capazes de especificar ações ou transformações possíveis para um objeto. Isso significa que os métodos conferem um caráter dinâmico aos objetos, pois permitem que os objetos exibam um comportamento que, em muitos casos, pode mimetizar (imitar) o comportamento de um objeto real ou concreto.

Outra maneira de entender o que são os métodos é imaginar que os objetos são capazes de enviar e receber mensagens, de tal forma que possamos construir programas (ou aplicações) nos quais os objetos trocam mensagens, proporcionando o comportamento desejado. A ideia central contida na mensagem que pode ser enviada ao objeto é a mesma quando ligamos um rádio: acionar o botão que liga esse aparelho corresponde a enviar uma mensagem ao rádio: “ligue”. Quando fazemos isso, não é necessário compreender em detalhes como funciona o rádio, mas apenas entender superficialmente como operá-lo ⇔ Conceito de “Caixa Petra”.

Métodos, ou operações (2/3)

- *Método* é um serviço que é requisitado a um objeto como parte de seu comportamento em resposta a estímulos (procedimento algorítmico).
- Um método é formado por uma interface e sua implementação. A interface descreve as características externas do método, sua parte visível como: *nome*, *parâmetros* e *valor retornado*. A implementação contém o código efetivo para a operação, isto é, uma sequência de instruções da linguagem.

Os métodos descrevem o comportamento, como agem e reagem os objetos analisados ⇔ Funções ou Procedimentos (Algoritmos).

Métodos, ou operações (3/3)

A forma genérica para a definição de um método em uma classe é a seguinte:

```
[modificador] tipo nome([parâmetros]) {
    // Corpo do Método
}
```

onde:

- **modificador** (opcional), uma combinação de especifador de acesso (public, protected ou private); abstract ou final e static.
- **tipo** é um indicador do valor de retorno (void quando o método não possuir um valor de retorno = procedimento).
- **nome** do método deve ser um identificador válido.
- **parâmetros** (opcional) são representados por uma lista de parâmetros separados por vírgulas, onde cada parâmetro obedece à forma: tipo nome.

Métodos são essencialmente subrotinas que podem manipular atributos de objetos para os quais o método foi definido. Além dos atributos de objetos, métodos podem definir e manipular variáveis locais; também podem receber parâmetros por valor através da lista de argumentos. Uma boa prática de programação é manter a funcionalidade de um método simples, desempenhando uma única tarefa. O nome do método deve refletir de modo adequado a tarefa realizada.

```
// Classe elementar "Matematica.java" implementada para demonstrar
// a definição de atributos e métodos.

public class Matematica
{

   public static int a, b;

   public static int adicao() {
      return (a + b);
   }

   public static int subtracao() {
      return (a - b);
   }

   public static int multiplicacao() {
      return (a * b);
   }

   public static int divisao() {
      return (a / b);
   }
}
```

Declaração da Classe **Matematica**

Atributos, ou variáveis da Classe

"Protocolo" de **Métodos** da Classe

Corpo da Classe

Referenciando Atributos e Métodos

Para denotar ou referenciar os atributos ou métodos de uma classe ou objeto deve-se utilizar um operador, denominado seletor, simbolizado por um caractere ponto '.' como segue:

Atributos

NomeDoObjeto, ou NomeDaClasse.nomeDoAtributo

por exemplo: Matematica.a = 10;

System.out.println(Matematica.a);

Métodos

NomeDoObjeto, ou NomeDaClasse.nomeDoMétodo([argumentos])

por exemplo: System.out.println(Matematica.adicao());

Os parênteses após o nome do método têm duplo propósito: um é diferenciar a construção ou o uso dos métodos da declaração de atributos, e o outro é "permitir" que sejam "especificados valores auxiliares que podem ser enviados" em anexo à mensagem (denominados de argumentos), para informar mais precisamente a forma como a ação deve ser realizada.

```java
// Usando a classe "Matematica", que na realidade representa uma
// biblioteca de códigos, já que seus atributos e métodos são static,
// não necessitando uma instância, ou objeto da classe para utilizar
// estes recursos.
import java.util.Scanner;
public class UsandoMatematica {

  public static void main(String args[]) {
   Scanner ler = new Scanner(System.in);

// Fazendo referência aos atributos ou variáveis membro
// NomeDaClasse.nomeDoAtributo
    System.out.println("Informe o valor da variável (a):");
    Matematica.a = ler.nextInt();

// NomeDaClasse.nomeDoAtributo
    System.out.println("Informe o valor da variável (b):");
    Matematica.b = ler.nextInt();

// Ativando os métodos ou funções membro:
// NomeDaClasse.nomeDoMétodo()
    System.out.println(... + Matematica.adicao());
    System.out.println(... + Matematica.subtracao());
    System.out.println(... + Matematica.multiplicacao());
    System.out.println(... + Matematica.divisao());
  }
}
```

Métodos set e get

Campos private de uma classe podem ser manipulados somente pelos métodos dessa classe.

As classes costumam fornecer métodos public para permitir a clientes da classe **configurar** (**set**, isto é, atribuir valores a) ou **obter** (**get**, isto é, obter os valores de) variáveis de instância private.

Vantagens:
O método **get** pode controlar como o cliente da classe pode acessar a variável de instância private. Por exemplo, um método **get** poderia controlar o formato dos dados que ele retorna e assim proteger o código do cliente na representação dos dados real.

Um método public **set** pode, e deve, avaliar cuidadosamente as tentativas de modificar o valor da variável a fim de assegurar que o novo valor é apropriado para esse item de dados. Por exemplo, uma tentativa de configurar (**set**) o mês do ano com um valor fora dos limites entre 1 e 12 deverá ser rejeitada.

Parâmetros dos Métodos (1/2)

Na declaração dos métodos, pode ser especificada uma lista de tipos e nomes de parâmetros, tal como declarações de variáveis, que receberão os valores fornecidos aos métodos. Tal lista é denominada de lista de parâmetros formais, lista de parâmetros ou simplesmente parâmetros. Se o método receber parâmetros, os mesmos deverão ser colocados no interior dos parênteses na instrução de chamada, separados por vírgulas; caso contrário, *os parênteses permanecerão vazios*, esses valores são denominados de argumentos ou parâmetros reais, ou ainda, parâmetros efetivos.

Consideramos novamente a sintaxe geral dos métodos:

```
[modificador] tipo nome([parâmetros]) {
    // Corpo do Método
}
```

Sendo uma relação de tipos e nomes dos parâmetros que receberão o valor dos argumentos desejados, uma lista de parâmetros formais tem a forma:

Tipo1 par1, Tipo2 par2, …, TipoN parN

Parâmetros dos Métodos (2/2)

Um método pode receber nenhum ou tantos argumentos quantos forem desejados. Os tipos dos parâmetros são os mesmos que podem ser empregados nas variáveis simples ou nos atributos das classes. Os nomes dos parâmetros também são arbitrários, e deve-se ressaltar que constituem variáveis locais, pertencentes ao escopo definido pelo corpo do método, podendo ter nomes idênticos ao de variáveis locais de outros métodos.

Em Síntese:

- são variáveis opcionalmente passadas a um método
- um método pode ter zero ou mais parâmetros
- são definidos no cabeçalho do método
- através da passagem de parâmetros é feita a transferência de informações entre os métodos sejam: constantes, variáveis, ou expressões, ao invés de somente o valor de variáveis globais ou atributos da classe.
- esta utilização formaliza a “comunicação” entre os métodos.

```java
// Demonstrando a declaração de "parâmetros" na definição de métodos.
import java.util.Scanner;
public class Parametro {
  public static void main(String args[]) {
    Scanner ler = new Scanner(System.in);

    System.out.println("Informe o valor da variavel (a):");
    int a = ler.nextInt();
    System.out.println("Informe o valor da variavel (b):");
    int b = ler.nextInt();

// Ativando o método e enviando o argumentos respectivos.
    System.out.println("..." + Matematica.operacao(a, '+', b));
    System.out.println("..." + Matematica.operacao(a, '-', b));
    System.out.println("..." + Matematica.operacao(a, '*', b));
    System.out.println("..." + Matematica.operacao(a, '/', b));
  }
}
```

```java
class Matematica {
// Método declarado com 3 (três) parâmetros formais.
  public static int operacao(int a, char op, int b) {
    switch (op) {
      case '+': return(a + b);
      case '-': return(a - b);
      case '*': return(a * b);
      case '/': return(a / b);
       default: return(0);
    }
  }
}
```

Valor de Retorno dos Métodos

Um outra questão relacionada a declaração de métodos (subrotinas) é a possibilidade de o método devolver mensagens ou não, isto é, "se o método é capaz de retornar algum tipo de valor".

Métodos em Java têm sua execução encerrada de duas maneiras possíveis:
1. quando um método não tem um valor de retorno (declarado com o tipo de retorno `void`): a execução é encerrada quando o bloco do corpo do método chega ao final;

```
public static void main(String args[]) {
   :
} // fim do corpo do método
```

2. encerra a execução do método através do comando `return`.

```
return;              // sem valor de retorno
return expressão; // retornando o resultado da expressão
```

```
public static int soma(int a, int b) {
  return (a + b); // retorna a soma dos parâmetros
}
```

Definição do Corpo de Métodos

O corpo de um método é formado por declarações de variáveis locais e comandos da linguagem de programação delimitados por chave (‘**{**‘ e ‘**}**’).

A sintaxe de declaração de variáveis locais em Java é similar àquela de declaração de atributos de objetos, sem a opção dos modificadores de visibilidade, variáveis locais têm visibilidade restrita ao método, exclusivamente.

```
int a, b;
boolean ehPrimo = true;
```

Embora não seja obrigatório, é uma boa prática de programação manter todas as declarações de variáveis no início do método.

Uma exceção aceita refere-se a blocos delimitados por iteração com `for`, onde a forma: `for (int i=1; i <=10; i++);` é aceita. Neste caso, o escopo da variável de controle “i” está restrito ao bloco da iteração.

Comandos podem representar uma expressão (uma operação a ser realizada para determinar um valor) ou um comando de controle de fluxo de execução.

```
// Estrutura modular p/ exibir todos os números primos entre 1 e
100.
public class Modular {
  public static void main(String args[]) {
    for (int n=1; n<=100; n++) {
      if (primo(n) == true)    // chama a funcao "Primo" enviando o
                               // argumento "n"
         System.out.println(n); // para chamar esta função fora do
                                // escopo da classe "Modular":
                                // if (Modular.Primo(n) == true)
    }
  }

// Função que retorna verdadeiro se o valor do parâmetro formal "n"
// corresponde a um numero primo, falso, caso contrário.
  public static boolean primo(int n) {
    boolean ehPrimo = true;
    int i = 2;
    while ((ehPrimo == true) && (i <= (n / 2))) {
      if ((n % i) == 0)
         ehPrimo = false;      // encontrou um divisor, portanto,
                               // não é primo
      else i++; // próximo divisor
    }
    return (ehPrimo); // retorna o valor da variável flag "ehPrimo"
  }
} // fim do corpo da classe "Modular"
```

2. Definição de classes em Java

Considerando novamente e, estendendo, as classes em Java são definidas através do uso da palavra-chave **class**.

Para definir uma classe, utiliza-se a construção:

```
[modificador] class NomeDaClasse {
    // corpo da classe
}
```

onde os elementos de declaração da classe representam:

- **modificador** (opcional); se presente, pode ser uma combinação de public e abstract ou final.

- **class** palavra chave da linguagem que indica a declaração de uma classe

- **NomeDaClasse** deve ser um identificador válido da linguagem.

Corpo da Classe

A definição (atributos e métodos) da classe propriamente dita está entre as chaves (‘**{**‘ e ‘**}**’) que delimitam blocos na linguagem Java.

A construção do corpo de uma classe usualmente obedece à seguinte sequência de definição:

1. As variáveis de classe (static), iniciando pelas public, seguidas pelas protected, pelas com visibilidade padrão (sem modificador) e finalmente pelas private.

2. Os atributos (ou variáveis-membro) dos objetos dessa classe, seguindo a mesma ordenação definida para as variáveis de classe.

3. Os construtores de objetos dessa classe.

4. Os métodos da classe, geralmente agrupados por funcionalidade.

Toda classe pode também ter um método **main** associado, que será utilizado pelo interpretador Java para dar início à execução de uma aplicação.

Definição de classes em Java

Agora, de forma completa, são descritos os elementos sintáticos que podem ser combinados na construção, ou definição de classes em Java:

`public` ou sem especificador (pacote)	acesso, ou visibilidade da classe
`abstract`	a classe não pode ser instanciada
`final`	classe terminal, ou folha, a classe não pode derivar outras classes
`class` **`NomeDaClasse`**	nome, ou identificador da classe
`extends` **`Super`**	a classe é derivada, ou filha, da superclasse, ou classe pai (**`Super`**)
`implements` **`Interface`**	interfaces implementadas da classe

```
{
        // Corpo da Classe
}
```

Encapsulamento

Visões de objetos:

- Interna - atributos e métodos da classe que o define;
- Externa - os serviços que um objeto proporciona e como ele interage com o resto do sistema;

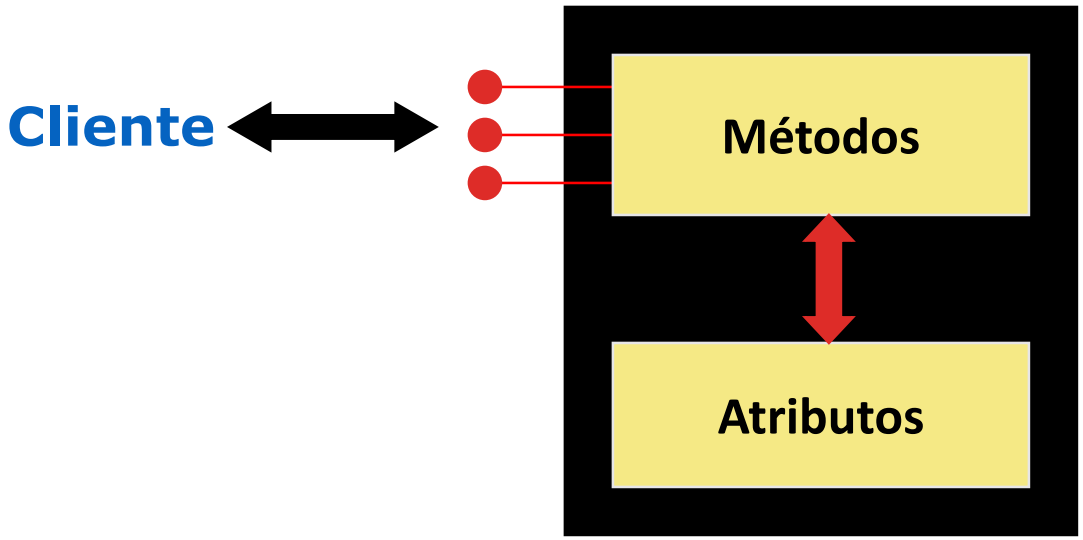
Externamente, um objeto é uma entidade *encapsulada*;

Um objeto pode usar os serviços providos por outro – mas não deve saber como estes serviços são implementados;

Qualquer mudança no estado de algum atributo do objeto usado deve ser feito através de seus métodos – dificultar (tornar impossível) a mudança direta de atributos fora da própria classe;

Encapsulamento

Pode-se pensar no objeto encapsulamento como sendo uma caixa preta;



Modificadores de Acesso

Java torna possível o encapsulamento através dos modificadores de acesso;

Um modificador é uma palavra reservada que especifica uma característica particular de um método ou atributo;

Java possui três modificadores de acesso: **`public`**, **`protected`** e **`private`**;

Modificadores de Acesso

Modificadores de acesso

private
protected
ausente (default)
public

+ restritivo

- restritivo

Modificadores de Acesso

Membros que são declarados como:

- **`public`** podem ser referenciados em qualquer parte;
- **`private`** só podem ser referenciados dentro da própria classe;
- **`Protected`** só podem ser acessados pela classe mãe e pelas subclasses.
- **`Default`** acessível nas classes do pacote.

Membros que são declarados sem modificadores de acesso têm visibilidade default e podem ser referenciados por qualquer classe do mesmo pacote;

Modificadores de Acesso

	`public`	`private`
Variables	**Violate encapsulation**	**Enforce encapsulation**
Methods	**Provide services to clients**	**Support other methods in the class**


Unicesumar