Tutorial Entendendo Orientação à Objetos no Java[1]

1 OBJETOS E CLASSES

1.1 CLASSES

- Uma ***classe*** é um modelo usado para definir vários objetos com características semelhantes. Um programa é constituído de uma classe ou de um conjunto de classes. Os elementos básicos de uma classe são chamados ***membros*** da classe e podem ser divididos em duas categorias:
    - As ***variáveis***, que especificam o estado da classe ou de um objeto instância desta classe.
    - Os ***métodos***, que especificam os mecanismos pelos quais a classe ou um objeto instância desta classe podem operar.

      O esqueleto de uma classe apresenta-se da seguinte maneira:

```
class NomeDaClasse{
   ...
   TipoDaVariavel1 variavel1;
   TipoDaVariavel2 variavel2;
   ...

   TipoDeRetorno1 metodo1(){
     ...
   }

   TipoDeRetorno2 metodo2(){
     ...
   }
   ...
}
```

    - É claro que, além das variáveis mencionadas acima, que estão definidas fora de qualquer método, haverá em geral também variáveis definidas dentro de um determinado método e cujo escopo será limitado a este método, ou possivelmente a um sub-bloco deste método. Estas variáveis são chamadas ***locais***.

[1] Este tutorial foi baseado no texto do Prof. Michel Betz – Universidade Federal do Rio Grande do Sul

Ao contrário, as variáveis das quais tratamos aqui são chamadas ***globais***. Veja adiante nesta aula uma discussão um pouco mais completa desta distinção.

- Para que uma instância de uma classe possa ser criada por qualquer outra classe, a classe em questão deve ser declarada ***pública***, o que é feito acrescentando-se a palavra-chave **public** à declaração da classe:

```
public class NomeDaClasse{
    ...
}
```

Classes que constituem aplicativos independentes e applets devem ser públicas. Uma classe que não for declarada pública somente poderá ser acessada por outras classes do mesmo pacote. Deve-se notar que cada arquivo fonte pode conter ***somente uma classe pública***. Caso o arquivo contenha uma classe pública, ele deve possuir o mesmo nome que esta classe, com a terminação **.java**.

1.2 OBJETOS

- Um ***objeto*** é uma ***instância*** de uma classe, ou seja, uma realização concreta e particular da mesma. Um objeto precisa ser ***criado***. Para que seja possível acessar as variáveis e os métodos de um objeto, é preciso atribuir uma ***referência*** ao objeto. O ***tipo*** de uma referência, ou seja, a classe à qual pertence o objeto ao qual ela vai referir-se, precisa ser ***declarado***.
- **Declaração:** a seguinte instrução declara que a variável **nomeDoObjeto** refere-se a um objeto instância da classe **NomeDaClasse**:

```
NomeDaClasse nomeDoObjeto;
```

- **Criação:** a seguinte instrução cria (em memória) um novo objeto instância da classe **NomeDaClasse**, que será referenciado pela variável **nomeDoObjeto** previamente declarada:

```
nomeDoObjeto = new NomeDaClasse();
```

- As duas instruções acima podem ser combinadas numa só:

```
NomeDaClasse nomeDoObjeto = new NomeDaClasse();
```

- Vale notar que a atribuição de uma referência a um objeto não é obrigatória. Há casos em que basta criar o objeto e passá-lo como argumento de um método, com um comando do tipo:

```
algumMetodo( new AlgumaClasse() );
```

2 VARIÁVEIS

2.1 VARIÁVEIS DE INSTÂNCIA E VARIÁVEIS DE CLASSE

- Uma ***variável de instância*** é uma variável cujo valor é ***específico ao objeto*** e não à classe. Uma variável de instância em geral possui um valor diferente em cada objeto representante da classe.
- Uma ***variável de classe*** ou ***variável estática*** é uma variável cujo valor é ***comum a todos os objetos*** representantes da classe. Mudar o valor de uma variável de classe em um objeto automaticamente muda o valor para todos os objetos instâncias da mesma classe. Um exemplo óbvio de uma variável de classe seria o número de instâncias desta classe que já foram criadas.
- Uma variável é considerada como de instância por "default". Para declarar uma variável de classe, acrescenta-se a palavra-chave **static**. Exemplo:

```
static int numeroDeInstanciasDestaClasse;
```

2.2 RESTRIÇÕES DE ACESSO

- Algumas palavras-chaves são diponíveis para ampliar ou restringir o acesso a uma variável. Estas palavras-chaves são acrescentadas à declaração da variável.
- Uma variável que pode ser acessada por ***qualquer outra classe*** é dita ***pública***, e é declarada usando-se a palavra-chave **public**.
- Uma variável que pode ser acessada ***somente por métodos da própria classe*** é dita ***privada***, e é declarada usando-se a palavra-chave **private**.
- Uma variável que, além de poder ser acessada por métodos da própria classe, também pode ser acessada ***pelas subclasses*** da classe na qual ela é declarada, é dita ***protegida*** e é declarada usando-se a palavra-chave **protected**. [O conceito de subclasse será desenvolvido na próxima aula.]
- Uma variável para a qual não foi especificada nenhuma destas palavras-chaves é dita ***amigável*** e pode ser acessada por todas as classes que pertencem ao ***mesmo pacote***. Pacotes são agrupamentos de classes. Como já sabemos, as classes de biblioteca da SUN estão organizadas em pacotes. O programador também pode criar os seus próprios pacotes.
- Os vários tipos de declaração de acesso estão exemplificados abaixo:

```
/* a classe definida neste arquivo pertence ao pacote
chamado algumPacote */
package algumPacote;

public class NomeDaClasse{
```

```
/* a variável w é acessível por qualquer classe */
  public int w;

/* a variável x é acessível pelos métodos da classe
NomeDaClasse e das suas subclasses */
  protected int x;

/* a variável y é acessível pelos métodos da classe
NomeDaClasse e das outras classes que pertencem ao pacote
chamado algumPacote */
  int y;

/* a variável z é acessível somente pelos métodos da classe
NomeDaClasse */
  private int z;
    ...
}
```

- As restrições de acesso desempenham um papel fundamental na programação orientada a objeto. Embora possa parecer mais simples e simpático permitir a qualquer objeto o acesso a todas as variáveis de qualquer outro objeto, isto resultaria em código muito pouco robusto. Recomenda-se, pelo contrário, ***limitar o acesso a qualquer variável o quanto for possível, dada a função da variável em questão***. Mesmo no caso de variáveis que precisam ser acessíveis a todos os objetos, há uma alternativa preferível ao acesso irrestrito outorgado pela palavra-chave **public**, como veremos na próxima seção.

2.3 COMO UM OBJETO ACESSA AS VARIÁVEIS DE OUTRO OBJETO

- Supondo que as restrições de acesso discutidas acima o permitam, uma classe pode utilizar ou modificar uma variável pertencente a um objeto através do ***operador ponto***.
- Por exemplo, após declarar e criar o objeto **nomeDoObjeto**, representante da classe **NomeDaclasse**, podemos ir buscar a variável **variavel** deste objeto e atribui-la à variável **var** da classe que estamos desenvolvendo:

```
  TipoDaVariavel var;
  var = nomeDoObjeto.variavel;
```

Evidentemente, **var** e **variavel** devem ser do mesmo tipo (chamado **TipoDaVariavel** no exemplo acima).

- Também podemos atribuir à variável **variavel** do objeto **nomeDoObjeto** o valor de uma variável **var** da nossa classe [entende-se aqui "valor" no sentido geral, podendo ser uma referência a um objeto]:

```
  TipoDaVariavel var = algumValor;
  nomeDoObjeto.variavel = var;
```

- Se **variavel** for uma referência a um outro objeto, pode-se concatenar operadores pontos para alcançar variáveis deste objeto:

```
TipoDaVariavel var;
var = nomeDoObjeto.variavel.outraVariavel;
```

Neste caso, é claro, **var** e **outraVariavel** devem ser do mesmo tipo.

- Se a variável a ser acessada for uma variável de classe, pode-se usar a classe, em vez de uma instância particular, para acessar a variável:

```
TipoDaVariavel var;
var = NomeDaClasse.variavel;
```

- Embora esta seja a maneira a mais direta de permitir que um objeto tenha acesso às variáveis de outro objeto, não é a mais segura, pois permite que um objeto modifique uma variável do outro sem restrição. Uma maneira mais segura consiste em declarar a variável como privada, mas incluir na classe métodos especiais controlando o acesso à variável:

```
public class NomeDaClasse{

  private TipoDaVariavel variavel;
    ...

  public TipoDaVariavel getVariavel(){
     return variavel;
  }

  public void setVariavel( TipoDaVariavel valor ){
/* aqui pode-se colocar testes para determinar se o valor é
aceitável */
     ...
    variavel = valor;
  }

}
```

Assim, pode-se colocar no método **setVariavel** código para conferir que o valor passado como argumento é adequado. Omitindo o método **setVariavel**, impedimos qua a variável seja modificada por qualquer outro objeto, embora ela possa ser lida chamando o método **getVariavel**.

O operador ponto também serve para chamar os métodos de outro objeto (veja adiante nesta aula), de maneira que a classe que deseja manipular as variáveis do outro objeto conterá agora comandos do tipo:

```
  TipoDaVariavel var, varp;
  var = nomeDoObjeto.getVariavel();
  ...
  nomeDoObjeto.setVariavel( varp );
```

Esta técnica de programação é chamada ***encapsulação*** de variáveis.

2.4 VARIÁVEIS GLOBAIS E VARIÁVEIS LOCAIS

- As variáveis discutidas acima são declaradas fora de qualquer método (usualmente no cabeçalho da classe) e são acessíveis por qualquer método da classe. Tais variáveis são chamadas ***globais***. Muitas vezes, variáveis auxiliares são declaradas dentro de um determinado método, ou até dentro de um bloco menor. Tais variáveis são chamadas ***locais***. Elas existem somente durante a execução daquele método ou bloco. A parte de código que "enxerga" uma determinada variável é chamada o***escopo*** da variável. Assim, o escopo de uma variável global é a classe inteira, e o escopo de uma variável local é o método, ou um bloco contido dentro do método, ao qual ela pertence.
- Exemplo:

```
class NomeDaClasse{

/* a variável abaixo é global */
   TipoDaVariavel variavel1;
   ...

   TipoDeRetorno nomeDoMetodo()
   {

/* a variável abaixo é local; está definida somente dentro
deste método */
     TipoDaVariavel variavel2;

     for( int i = 0; i < 10; i++ ){

/* a variável i é local, definida só dentro deste bloco */
       ... //
     }
   }

}
```

- Embora não pareça uma boa ideia, é possível dar a uma variável local um nome que já foi atribuído a uma variável global. Neste caso, a variável local "encobre" a variável global, mas o acesso à variável global é possível usando a palavra-chave **this** (que fornece uma ***referência ao próprio objeto***) e o operador ponto:

```
class NomeDaClasse{
   TipoDaVariavel variavel; // variável global
   ...

   TipoDeRetorno nomeDoMetodo()
   {
     TipoDaVariavel variavel; // variável local
   ...

/* a variável abaixo recebe o valor de variavel local */
     variavel2 = variavel;

/* a variável abaixo recebe o valor de variavel global */
     variavel3 = this.variavel;
   }

}
```

Alguns programadores fazem uso desta construção em métodos **set**:

```
public class NomeDaClasse{
   private TipoDaVariavel variavel;

   public void setVariavel( TipoDaVariavel variavel )
   {
     this.variavel = variavel;
   }

}
```

2.5 VARIÁVEIS CONSTANTES

- Esta expressão um tanto paradoxal refere-se a variáveis cujo valor não pode mudar durante a execução do programa. Tais variáveis são caracterizadas pela palavra-chave **final**, e por convenção recebem usualmente nomes escritos inteiramente em maiúsculas.
- Por exemplo, na classe **Math** do pacote **lang**, encontramos

```
   public static final double PI;
```

que contém o valor da famosa constante matemática.

3 MÉTODOS

3.1 VALOR DE RETORNO E ARGUMENTOS

- Em geral, um método recebe argumentos cujos valores lhe são passados pelo objeto que o chamou, efetua um conjunto de operações e retorna algum resultado. A declaração do método especifica o nome do método, o tipo de retorno, o nome e o tipo de cada argumento. Os argumentos são variáveis locais do método em questão. O valor de retorno é devolvido utilizando-se a palavra-chave **return**:

```
   TipDeRetorno nomeDoMetodo(TipoDoArg1 arg1, TipoDoArg2
arg2, ...){
     TipoDeRetorno valorDeRetorno;
     ...
     return valorDeRetorno;
   }
```

  Se o método não utiliza nenhum argumento, parênteses vazias devem ser incluídas na declaração.

- Se o método não retorna nenhum valor, isto deve ser declarado usando-se a palavra-chave **void**:

```
   void nomeDoMetodo( TipoDoArg1 arg1, TipoDoArg2
arg2, ... ){
     ...
   }
```

3.2 MÉTODOS ESTÁTICOS

- Por "default", um método efetua uma determinada operação sobre um determinado objeto, ou seja uma instância da classe na qual o método está declarado. Existem métodos que realizam operações genéricas, não relativas a uma instância particular. Tais métodos são chamados ***estáticos*** e são declarados acrescentando-se a palavra-chave **static** à declaração do método.
- Por exemplo, os métodos da classe **Math** do pacote **lang**, que realizam operações matemáticas sobre números, são estáticos:

```
   public static int min( int a, int b ){
     ... // retorna o menor dos 2 inteiros a e b
   }
```

3.3 RESTRIÇÕES DE ACESSO

- Algumas palavras-chaves são diponíveis para ampliar ou restringir o acesso a um método. Estas palavras-chaves são acrescentadas à declaração do método.
- Um método que pode ser acessado por ***qualquer outra classe*** é dito ***público***, e é declarado usando-se a palavra-chave **public**.
- Um método que pode ser acessado ***somente por métodos da própria classe*** é dito ***privado***, e é declarado usando-se a palavra-chave **private**.
- Um método que, além de poder ser acessado por todas as classes do mesmo pacote, também pode ser acessado pelas ***subclasses*** da classe na qual ele é declarado, é dito ***protegido*** e é declarado usando-se a palavra-chave **protected**.
- Um método para o qual não foi especificada nenhuma destas palavras-chaves é dito ***amigável*** e pode ser chamado por todas as classes que pertencem ao ***mesmo pacote***.
- Exemplo:

```
package algumPacote;

public class NomeDaClasse{

   public int metodo1( ){
     ... // acessível por qualquer classe
   }

   protected int metodo2( ){
     ... // acessível por esta classe e suas subclasses
   }

   int metodo3( ){
     ... // acessível pelas classes deste pacote
   }

   private int metodo4( ){
     ... // acessível por esta classe
   }
}
```

3.4 COMO UM OBJETO CHAMA UM MÉTODO DE OUTRO OBJETO

- Supondo que as restrições de acesso discutidas acima o permitam, uma classe que possui uma referência a um objeto pode chamar (executar) um método pertencente este objeto através do ***operador ponto"***:

```
/* o método abaixo não requer argumento e não retorna nada
*/
   nomeDoObjeto.metodo1( );
```

```
/* o método abaixo requer dois argumentos e não retorna
nada */
   nomeDoObjeto.metodo2( var1, var2 );

/* o método abaixo não requer argumento e retorna um valor
do tipo Resultado */
   Resultado resultado = objeto.metodo3( );
```

- Se a variável de retorno for um objeto, pode-se concatenar operadores pontos para chamar métodos deste objeto:

```
   nomeDoObjeto.metodo3().metodo4();
```

- Se o método for estático, usa-se a classe, em vez de uma instância particular, para chamar o método:

```
   int x = Math.min( a, b );
```

3.5 COMO CRIAR UM OBJETO - MÉTODO CONSTRUTOR

- Para criar um objeto, usa-se a palavra chave **new**, seguida de uma chamada ao ***método construtor***, cujo nome é idêntico ao da classe:

```
   NomeDaClasse nomeDoObjeto = new NomeDaClasse( );
```

- O método construtor não precisa ser incluído explicitamente na classe, mas pode ser incluído para realizar tarefas no ato da criação. A sintaxe é:

```
class NomeDaClasse{

   NomeDaClasse( ){
     ... // comandos executados na criação do objeto
   }

}
```

ou, para uma classe pública:

```
public class NomeDaClasse{

   public NomeDaClasse( ){
     ... // comandos executados na criação do objeto
   }

}
```

- O método construtor pode receber argumentos:

```
public class NomeDaClasse{

   public NomeDaClasse( Tipo1 arg1, Tipo2 arg2 ){
     ... // comandos executados na criação do objeto
   }

}
```

Valores para estes argumentos devem então ser passados ao construtor no comando de criação:

```
   NomeDaClasse nomeDoObjeto = new NomeDaClasse( valorArg1,
valorArg2 );
```

Exercícios:

1. Defina uma classe pública chamada *Soma*. Dentro desta classe, defina um método *somaValor* que recebe dois argumentos que representam valores numéricos reais. Este método fará a soma dos dois números reais e mostrará ao final o valor da soma na tela.
2. Escreva uma classe chamada *TestaSoma* que irá chamar o método *somaValor* definido na classe *Soma*, passando para o mesmo dois valores numéricos reais.