Programação Orientada a Objetos

UNIDADE 2

ser educacional
gente criando o futuro

GUIA DE ESTUDO

UNIDADE 2

PROGRAMAÇÃO ORIENTADA A OBJETOS

PARA INÍCIO DE CONVERSA

Olá, caro(a) estudante! Vamos dar início à unidade 2 do nosso curso de Programação Orientada a Objetos. Nesse segundo guia de estudo da respectiva disciplina, vamos falar um pouco sobre os conceitos básicos e termologias usadas na programação orientada a objetos. Entre os conceitos que veremos, podemos destacar: classes, objetos, atributos, métodos, construtores, sobrecarga; instanciação e referência de objetos; envio de mensagens; ciclo de vida de um objeto.

É importante lembrar mais uma vez que esse guia de estudo deve ser usado como uma referência para seus estudos. Livros e outras referências devem ser usados para aprofundar o assunto. E mais uma vez é importante ressaltar que os exercícios são fundamentais para o aprendizado da disciplina de programação orientada a objetos.

Vamos começar então!

CONCEITOS BÁSICOS E TERMINOLOGIAS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

Nessa unidade veremos vários conceitos usados em orientação a objetos.

Vimos na unidade passada, que a orientação a objetos tem como base os objetos e classes. No nosso curso iremos usar a linguagem Java como base para a implementação dos códigos.

Essa linguagem é bastante usada e também é muito parecida com outras linguagens que usam orientação a objeto, como C++ e C#.

Classe

Uma classe em orientação a objeto pode ser definida de várias formas. Vimos em outra disciplina, na parte de análise e projeto orientado a objeto, o termo classe sendo usado para modelar os sistemas. Essa modelagem pode ser detalhada até o nível de implementação (programação).

Em programação orientada a objetos podemos conceituar classe como uma forma de modelar uma entidade que contempla tanto a modelagem dos dados que caracterizam o objeto, como também os códigos que formam os métodos dessa classe.

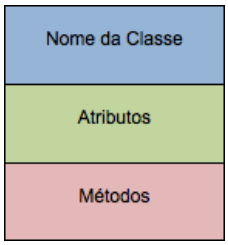
A classe funciona como uma fôrma, um molde, para construção de objetos. Nesse molde temos as características dos atributos, como os seus nomes e tipos associados. Temos também as operações que esse objeto pode executar ou oferecer a outros objetos. Essas operações são conhecidas como métodos da classe. Os métodos funcionam como funções ou procedimentos locais dessa entidade. Esses métodos são usados para manipular os dados do objeto, ou seja, seus atributos. Esses métodos estarão disponíveis em todos os objetos gerados por essa classe. A atuação dos métodos é restrita aos atributos e variáveis locais da classe.

Podemos ver uma classe como uma forma de descrever objetos com os mesmos atributos e métodos. Ou seja, um molde para criar vários objetos.

DICA

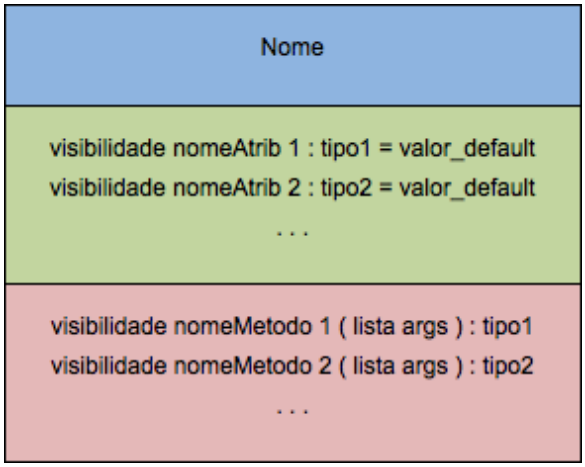
Veremos depois que um objeto individual é uma entidade concreta que executa algum papel no sistema como um todo. Enquanto uma classe captura a estrutura e o comportamento comum a todos os objetos que são relacionados a essa determinada classe.

Caro(a) aluno(a), para facilitar a compreensão usaremos UML (Unified Modeling Language), uma vez que já foi visto na outra disciplina, como forma de representação para uma classe no diagrama de classes. Essa especificação é feita usando um diagrama retangular dividido em três regiões, conforme a figura a baixo:



Fonte: Adaptação do autor

Detalhando e exemplificando esse diagrama podemos ter:



Fonte: Adaptação do autor

Nome da classe

É um identificador para a classe, dessa forma podemos referenciá-la posteriormente, por exemplo, no momento da criação de um objeto, na hora de usar como base para o mecanismo de herança, etc. Como padrão, os nomes das classes devem começar com a primeira letra maiúscula.

Atributos

É um conjunto de propriedades da classe, estrutura com os dados para armazenar as informações dos objetos. Para cada atributo, especifica-se:

- nomeAtrib: um identificador, nome, para o atributo. Como padrão o nome do atributo começa com a primeira letra minúscula.
- tipo: o tipo do atributo é usado para classificar a estrutura de dados desse atributo (inteiro, real, caráter, lógico, etc.).
- valor_default: opcionalmente, pode-se especificar um valor inicial para cada atributo.
- visibilidade: opcionalmente, pode-se especificar o nível de acessibilidade de um atributo de um objeto a partir de outros objetos.

Valores possíveis são:

- (privativo), nenhuma visibilidade externa, apenas os métodos da classe tem acesso a esses atributos;
+ (público), visibilidade externa total, o seja, os atributos podem ser acessados por qualquer objeto; e
# (protegido), visibilidade externa limitada.

Métodos

É o conjunto de operações, funcionalidades da classe. Para cada método, especifica-se sua assinatura (informações de acesso do método, sem sua implementação), composta por:

- NomeMetodo: um identificador para o método, ou seja, o nome do método. Esse nome, por padrão, deve ter a primeira letra minúscula.
- Tipo: quando o método tem um valor de retorno, o tipo desse valor deve ser especificado. No caso de não retornar nada, usa-se na maioria das linguagens, "void".
- Lista args: é a lista de argumentos. Quando o método recebe parâmetros para sua execução, é definido um tipo para cada parâmetro. São informações adicionais passadas para um método. Informações adicionais fornecidas na chamada de método com argumentos.
- Visibilidade: como para atributos, define o quão visível é um método a partir de objetos de outras classes.

Objeto

Objetos são instâncias de classe, um exemplar de uma classe. Um programa desenvolvido com uma linguagem de programação orientada a objetos manipula estruturas de dados através dos objetos da mesma forma que um programa em linguagem imperativa tradicional utiliza variáveis. Assim como as variáveis

são associadas a seus tipos, os objetos estão associados a suas classes. É através de objetos que (praticamente) todo o processamento ocorre em aplicações desenvolvidas com linguagens de programação orientadas a objetos.

Um objeto é um elemento computacional que representa, no domínio da solução, alguma entidade (abstrata ou concreta) do domínio de interesse do problema que está em análise. Objetos similares são agrupados em classes. Um programa orientado a objetos é composto por um conjunto de objetos que interagem através de "trocas de mensagens". Na prática, essa troca de mensagem traduz-se na invocação de métodos entre objetos.

A programação é feita pelo envio de mensagens. A solução de um problema consiste em identificar os objetos, mensagens e sequências de mensagens para efetuar a solução. Cada objeto do mundo real transforma-se em um "objeto" de software. Viabiliza a montagem de sistemas a partir de componentes. Os objetos são construídos, ou instanciados, através de uma chamada ao seu método construtor. Vamos ver agora como e o que são esses métodos construtores.

Construtores

Para criar um objeto, é necessário construí-lo, ou seja, instancia-lo. Para essa tarefa temos métodos especiais que são os CONTRUTORES.

Construtores inicializam um objeto. Além de alocar espaço em memória para o objeto, os construtores têm a tarefa de inicializar o objeto, atribuindo valores iniciais aos seus atributos, executando operações necessárias para o funcionamento do objeto, etc.

Os construtores são métodos definidos dentro de cada classe, assim como qualquer outro método.

Na linguagem JAVA o construtores têm o mesmo nome das suas classes. Ou seja, uma classe chamada "Pessoa" terá seus construtores com esse mesmo nome: Pessoa ( ). Digo "seus" construtores, pois uma classe pode ter mais de um construtor. Nesse caso teremos uma sobrecarga no método construtor. Veremos mais sobre sobrecarga adiante.

Como os construtores armazenam valores iniciais nos atributos do objeto, Eles frequentemente recebem valores de parâmetros externos para definir os atributos. Quando o método construtor não recebe nenhum parâmetro externo, dizemos que é um construtor sem parâmetros.

Caro(a) estudante, caso não seja definido nenhum construtor, o compilador cria o construtor padrão, sem parâmetros. O construtor padrão inicializa os atributos de classe para seus valores padrões. Caso seja definido qualquer construtor, o construtor padrão não será adicionado pelo compilador. Caso deseje-se que a classe ainda possua um construtor sem parâmetros, é necessário declará-lo explicitamente.

DICA

É sempre bom fornecer um construtor para assegurar que as variáveis de instância da sua classe (os atributos) sejam adequadamente inicializadas com valores significativos quando cada novo objeto de sua classe for criado. A menos que a inicialização-padrão de variáveis de instância de sua classe seja aceitável e que não seja necessário fazer nenhuma ação na inicialização do objeto, nesse caso, bastaria o construtor padrão fornecido pelo compilador JAVA.

EXEMPLO

Veremos agora um exemplo do uso de construtores em JAVA:

Pessoa.java

```
1  public class Pessoa {
2
3          private String nome ;    // Nome completo da pessoa.
4          private int idade ;      // Idade da pessoa.
5          private char sexo ;      // Sexo da pessoa: 'M' para masculino e 'F' para feminino.
6          private double altura ;  // Altura da pessoa: valor real expresso em metros.
7
8          // Construtor da classe Pessoa com quatro parâmetros.
9          public Pessoa ( String n , int i , char s , double a ) {
10                 nome = n ;
11                 idade = i ;
12                 sexo = s ;
13                 altura = a ;
14         } // Fim do construtor
15
16         // Construtor da classe Pessoa sem parâmetros.
17         // Definido explicitamente para substituir o
18         // construto vazio padrão do compilador JAVA.
19         public Pessoa ( ) {
20                 // Vazio, apenas para substituir o construtor padrão !
21         }
22
23         // Construtor da classe Pessoa com dois parâmetros, apenas o nome e a idade.
24         public Pessoa ( String nome , int idade ) {
25                 this.nome = nome ;
26                 this.idade = idade ;
27         } // Fim do construtor
28
29         // Outros métodos . . .
30 } // Fim da classe Pessoa
```

Fonte: Adaptação do autor baseado em FRANÇA NETO, L. R.

No exemplo da classe Pessoa têm três construtores. No primeiro (linhas 9-14), temos um construtor com quatro parâmetros que serão usados para iniciar os valores dos atributos nome, idade, sexo e altura. Quando criamos o nosso próprio construtor, o construtor padrão gerado pelo compilador JAVA é desativado. Para ter um construtor vazio, tipo o padrão, definimos o segundo construtor (linhas 19-21) sem parâmetros para compensar o que não é mais fornecido pelo compilador JAVA. Em seguida, temos o terceiro construtor (linhas 24-27) para ser usado apenas passando como parâmetros o nome e a idade. Os outros atributos receberão os valores padrões do JAVA. Nesse caso, usamos também a palavra-chave this (também chamada de referência this), que é usada quando um objeto acessa uma referência a si própria (você pode ler mais sobre a referência this no capítulo 8 do livro do Deitel – Java como Programar). Nesse caso, do construtor a referência this está sendo usada para diferenciar os parâmetros dos atributos, que tem o mesmo nome.

Note que criamos três métodos construtores, todos com o mesmo nome. É possível criar métodos com o mesmo nome na mesma classe. Isso é visto como sobrecarga de métodos. Vermos como isso é possível adiante.

Sobrecarga

Chamamos de sobrecarga de métodos quando temos na mesma classe métodos com o mesmo nome, contanto que tenham diferentes conjuntos de parâmetros (assinatura diferente). Ou seja, diferentes tipos, números e ordem dos parâmetros. Quando os métodos com sobrecarga são chamados o compilador JAVA avalia seus parâmetros (ordem, quantidades, tipos) para saber qual método correto deve executar. Usamos sobrecarga de métodos para realizar a mesma tarefa ou tarefas semelhantes com o mesmo nome do método, mas sobre tipos diferentes ou com número diferentes de argumentos.

EXEMPLO

Veremos abaixo um exemplo de uma classe que usa sobrecarga de métodos. Essa classe tem métodos para calcular o quadrado e o dobro de números inteiros e reais. Esse exemplo é uma adaptação do exemplo de sobrecarga de métodos do capítulo 6 do livro do Deitel, Java: Como programar.

MetodoSobrecarga.java

```
1  // MetodoSobrecarga.java
2  // Declarações de métodos sobrecarregados.
3
4  public class MetodoSobrecarga
5  {
6    // teste de métodos square sobrecarregados.
7    public void testOverloadedMethods ( )
8    {
9      System.out.printf ( " O quadrado do inteiro 7 é  %d\n " , square ( 7 ) ) ;
10     System.out.printf ( " O quadrado do real 7.5 é %f\n " , square ( 7.5 ) ) ;
11     System.out.printf ( " O dobro do inteiro 5 é  %d\n " , dobro ( 5 ) ) ;
12   } // fim do método testOverloadedMethods.
13
14   // método square com argumento de int.
15   public int square ( int intValue )
16   {
17     System.out.printf ( " \nChamada ao método square com o argumento inteiro : %d\n " , intValue ) ;
```

```
18      return intValue * intValue ;
19    } // fim do método square com argumento de int.
20
21    // método square com argumento Double.
22    public double square ( double doubleValue )
23    {
24      System.out.printf ( " \nChamada ao método square com o argumento real : %f\n " , doubleValue ) ;
25      return doubleValue * doubleValue ;
26    } // fim do método square com argumento Double.
27
28    // método dobro com argumento de int.
29    public int dobro ( int intValue )
30    {
31      System.out.printf ( " \nChamada ao método dobro com o argumento inteiro : %d\n " , intValue ) ;
32      return 2 * intValue ;
33    } // fim do método dobro com argumento de int.
34
35
36  } // fim da classe MetodoSobrecarga.
```

MetodoSobrecargaTeste.java

```
1   // MetodoSobrecargaTeste.java
2   // Aplicativo para testar a classe MetodoSobrecarga.
3
4   public class MetodoSobrecargaTeste
5   {
6     public static void main ( String args[] )
7     {
8       MetodoSobrecarga methodOverload = new MetodoSobrecarga ( ) ;
9       methodOverload.testOverloadedMethods ( ) ;
10    } // fim de main.
11  } // fim da classe MetodoSobrecargaTeste.
```

Fonte: Adaptação do autor baseado em FRANÇA NETO, L. R.

Na classe Método Sobrecarga temos três métodos, sendo que o método "square" tem duas implementações, uma que calcula o quadrado de um número com valores inteiros e outra que calcula o quadrado de um número com valores reais. Para o compilador saber qual método é chamado, ele analisa a assinatura do método. Se o método recebe um parâmetro int, então ele chama o método com argumento inteiro (linhas 15 - 19). Se for passado como parâmetro um valor double, então ele chama o outro com argumento real (linhas 22 – 26). Veja que a diferença entre os dois métodos são os parâmetros e os tipos de retorno:

```
public int square ( int intValue )
public double square ( double doubleValue )
```

No caso do método "dobro", esse tem o mesmo parâmetro e tipo de retorno do método "square" para números inteiros. Só que nesse caso não há sobrecarga, pois o nome dos métodos são diferentes.

```
public int square ( int intValue )
public int dobro ( int intValue )
```

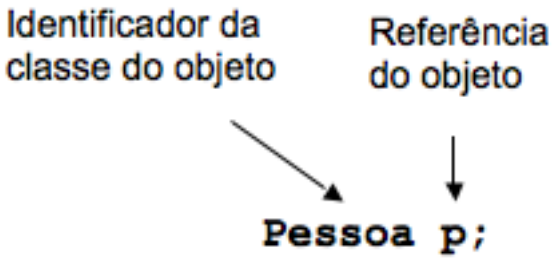
Para completar nosso exemplo temos a classe MetodoSobrecargaTeste que usamos para criar um objeto e chamar o método "testOverloadedMethods ( )". Esse método testa as chamadas aos métodos sobrecarregados. Veremos em seguida como criar objetos, ou seja, instanciar uma classe.

INSTANCIAÇÃO E REFERÊNCIA DE OBJETOS

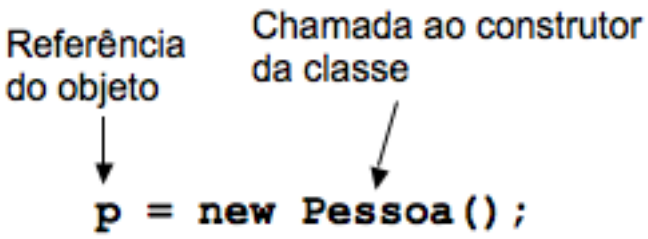
Vamos ver agora como criar um objeto, ou seja, instanciar esse objeto com base em uma classe. Como vimos anteriormente, um objeto é uma instância de uma classe. Ele é a materialização dessa classe. Para criar um objeto em JAVA usaremos a palavra reservada new para chamar o construtor da classe. Assim como nós declaramos uma variável em JAVA, iremos declarar a referência a um objeto. Basta dar um nome a essa referência do objeto e em seguida, o nome da classe que ele pertence. Às vezes, para simplificar, chamamos a referência do objeto (o identificador) simplesmente de objeto.

EXEMPLO

Veja o exemplo abaixo:



Nesse exemplo temos que p é uma referência para um objeto da classe Pessoa. Esse objeto ainda não existe. Não foi criado ainda. Para criar é necessário chamar o método construtor da classe. Veja com isso pode ser feito:



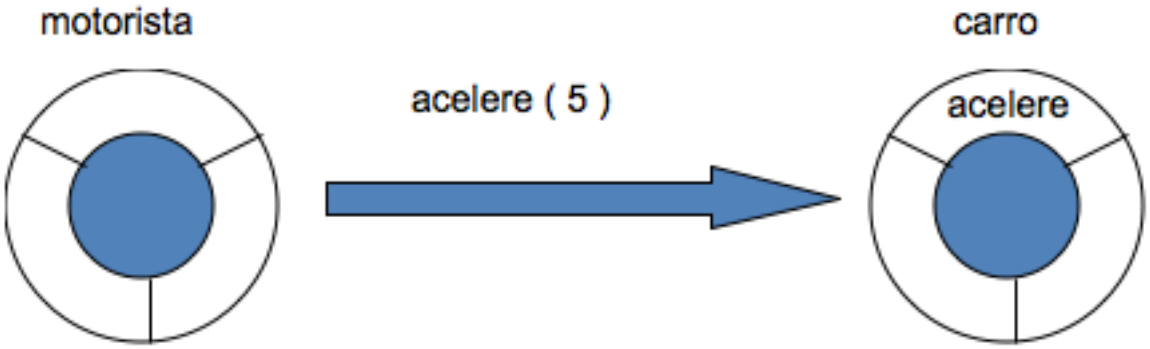
Foi feita então uma chamada ao construtor da classe Pessoa. Nesse caso ao construtor sem parâmetros, uma vez que temos no nosso exemplo da classe Pessoa, três construtores (ver código anterior). Dizemos que p agora é uma referência a um objeto (instanciado) da classe Pessoa. Ou para simplificar nossa comunicação, podemos dizer, de maneira informal, que p é um objeto da classe Pessoa. Veremos mais adiante um exemplo completo, usando a classe Pessoa e a criação, manipulação de objetos dessa classe.

Como pode ser visto nesse exemplo da classe TestePessoa, temos, nas linhas 17 – 19, três construtores para cada uma das referências para objetos (p1, p2, p3).

Envio de mensagens

A programação orientada a objetos identifica uma abordagem em que o programador visualiza seu programa em execução em termos de objetos que se comunicam através de trocas de mensagens. As mensagens são compostas por um nome e por parâmetros (opcional).

Os objetos interagem entre si enviando mensagens uns para os outros. O objeto que receber a mensagem responderá através da seleção e execução de um método que faz parte de seu comportamento. Após a execução, o controle volta para o objeto que enviou a mensagem. Uma mesma mensagem pode gerar ações diferentes (polimorfismo), veremos isso mais adiante. Classes bem projetadas escondem seus dados do ambiente externo (encapsulamento), sendo necessário o uso de métodos para acessar esses dados. Esses acessos são feitos através de mensagens.



Fonte: Elaborada pelo autor

As mensagens têm a seguinte estrutura:

Mensagem (destino, operação, parâmetros [se houver]).

Ex: Mensagem (carro , acelere , 5).

Em JAVA, usaremos a seguinte sintaxe para a passagem de mensagem:

```
destino.operação ( parâmetro )
carro.acelere ( 5 ) ;
```

EXEMPLO

Para exemplificar e ilustrar melhor essas operações que vimos, vamos usar a versão completa da classe Pessoa e também uma classe TestePessoa para testar essa classe.

Pessoa.java

```
1  public class Pessoa {
2
3          private String nome;     // Nome completo da pessoa.
4          private int idade;               // Idade da pessoa.
5          private char sexo;               // Sexo da pessoa: 'M' para masculino e 'F' para feminino.
6          private double altura;   // Altura da pessoa: valor real expresso em metros.
7
8          // Construtor da classe Pessoa com quatro parâmetros.
9          public Pessoa ( String n , int i , char s , double a ) {
10                 nome = n ;
11                 setIdade ( i ) ;
12                 sexo = s ;
13                 setAltura( a ) ;
14         } // Fim do construtor
15
16         // Construtor da classe Pessoa sem parâmetros.
17         // Definido explicitamente para substituir o
18         // construto vazio padrão do compilador JAVA.
19         public Pessoa ( ) {
20                 // Vazio, apenas para substituir o construtor padrão !
21         }
22
23         // Construtor da classe Pessoa com dois parâmetros, apenas o nome e a idade.
24         public Pessoa ( String nome , int idade ) {
25                 this.nome = nome ;
26                 this.setIdade( idade ) ;
27         } // Fim do construtor.
28
29         // Métodos para acesso aos atributos (campos).
30
31         // Método de acesso para o campo nome.
32         public String getNome ( ) {
33                 return nome ;
34         } // fim do método getNome.
35
36         // Método de acesso para o campo nome.
37         public void setNome ( String nome ) {
38                 this.nome = nome ;
39         } // fim do método setNome.
40
41         // Método de acesso para o campo idade.
42         public int getIdade ( ) {
43                 return idade ;
44         } // fim do método getIdade.
45
46         // Método de acesso para o campo idade.
47         public void setIdade ( int idade ) {
48                 if ( idade >= 0 ) {
49                         this.idade = idade ;
```

```
50                  }
51                  else {
52                          System.out.println (" Idade inválida ! " ) ;
53                  }
54          } // fim do método setIdade.
55
56          // Método de acesso para o campo sexo.
57          public char getSexo ( ) {
58                  return sexo ;
59          } // fim do método getSexo.
60
61          // Método de acesso para o campo sexo.
62          public void setSexo ( char sexo ) {
63                  this.sexo = sexo ;
64          } // fim do método setSexo.
65
66          // Método de acesso para o campo altura.
67          public double getAltura ( ) {
68                  return altura ;
69          } // fim do método getAltura.
70
71          // Método de acesso para o campo altura.
72          public void setAltura ( double altura ) {
73                  if ( altura > 0 ) {
74                          this.altura = altura ;
75                  }
76                  else {
77                          System.out.println ( " Altura inválida ! " ) ;
78                  }
79          } // fim do método setAltura.
80
81          // Outros métodos . . .
82
83          // Método que exibe na tela de console as informações da pessoa.
84          public void exibe ( ) {
85                  System.out.println ( " Nome   : " + nome ) ;
86                  System.out.println ( " Idade  : " + idade ) ;
87                  System.out.println ( " Sexo   : " + sexo ) ;
88                  System.out.println ( " Altura : " + altura ) ;
89          } // fim do método exibe.
90
91          // Método que soma mais um ano na idade da pessoa.
92          public void aniversario ( ) {
93                  idade = idade + 1 ;
94          } // fim do método aniversario.
95
96  } // Fim da classe Pessoa.
```

TestePessoa.java

```
1  // TestePessoa.java
2  // Aplicativo para testar a classe Pessoa.
3  import java.util.Scanner;
4
5  public class TestePessoa {
6
7          public static void main ( String [ ]  args ) {
8                  // Programa principal para testar a classe Pessoa.
9
10                 // Declaração das referências dos objetos.
11                 Pessoa p1 , p2 , p3 ;
12                 int i ;
13                 double a ;
14                 Scanner leia = new Scanner ( System.in ) ; // objeto para entrada de dados via teclado.
15
16                 // Chamada aos construtores para criar os três objetos: p1 , p2 e p3.
17                 p1 = new Pessoa ( "Fulano" , 21 , 'M' , 1.7 ) ;
18                 p2 = new Pessoa ( ) ;
19                 p3 = new Pessoa ( "Beltrano" , 32 ) ;
20
21                 System.out.print ( " Digite a idade da pessoa p3 : " );
22                 i = leia.nextInt ( ) ;
23                 p3.setIdade ( i ) ;   // Chamada ao método setIdade para alterar a idade de p3.
24
25                 System.out.print ( " Digite a altura da pessoa p3 : " );
26                 a = leia.nextDouble ( ) ;
27                 p3.setAltura ( a ) ;   // Chamada ao método setAltura para alterar a altura de p3.
28
29                 p2.setNome ( "Maria" ) ;  // Chamada ao método setNome para alterar o nome de p2.
30                 p2.setIdade ( 18 ) ;      // Chamada ao método setIdade para alterar o idade de p2.
31
32                 System.out.println ( " \n Pessoa p1 : " ) ;
33                 p1.exibe ( ) ;   // Exibe na tela de console as informações da pessoa p1.
34                 System.out.println ( " \n Pessoa p2 : " ) ;
35                 p2.exibe ( ) ;   // Exibe na tela de console as informações da pessoa p2.
36                 System.out.println ( " \n Pessoa p3 : " ) ;
37                 p3.exibe ( ) ;   // Exibe na tela de console as informações da pessoa p3.
38
39                 p1.aniversario ( ) ;  // Executa o método aniversario para acrescentar 1 ano na idade de p1.
40
41                 System.out.println ( " \n Pessoa p1 : " ) ;
42                 p1.exibe ( ) ;   // Exibe na tela de console as informações da pessoa p1.
43
44                 p3.setNome ( "João" ) ;   // Chamada ao método setNome para alterar o nome de p3.
45                 System.out.println ( " \n Pessoa p3 : " ) ;
46                 p3.exibe ( ) ;   // Exibe na tela de console as informações da pessoa p3.
47                 System.out.println ( " Altura da pessoa p1 : " + p1.getAltura ( ) + " m " );
48
49                 p2 = null ; // Atribui a referência p2 o valor null, liberando o espaço alocado para o objeto p2.
```

```
50
51                  System.out.println ( " \n Pessoa p2 : " ) ;
52                  if ( p2 != null ) {
53                      p2.exibe ( )  // Exibe na tela de console as informações da pessoa p2, caso não nula.
54                  }
55                  else
56                  {
57                          System.out.println ( " p2 é uma referência nula (null) ! " );
58                  }
59
60                  leia.close ( ) ;
61          } // fim do método main.
62
63  } // fim da classe TestePessoa.
```

Fonte: Adaptado pelo autor baseado em FRANÇA NETO, L. R.

Conforme o código acima, temos vários exemplos de envios de mensagens. Podemos destacar algumas mensagens:

Na linha 30: p2.setIdade (18) ;

Onde o destino seria: p2; a operação: setIdade; e o parâmetro: 18.

Na linha 39: p1.aniversario ( ) ;

Onde o destino seria: p1; a operação: aniversario; e o parâmetro: nenhum.

Ciclo de vida de um objeto

Caro(a) estudante, veremos agora como funciona o ciclo de vida de um objeto em JAVA. Os objetos em um programa JAVA são criados em tempo de execução, utilizados e destruídos. Como vimos anteriormente, JAVA usa a palavra-chave new para criar um novo objeto como uma instância de uma classe específica. Essa ação retorna como resultado uma referência para o objeto criado.

Em JAVA, um objeto não precisa ser destruído explicitamente como é necessário fazer em linguagens como PASCAL e C++. Nessas linguagens, se os objetos não forem destruídos, eles continuam utilizando a memória do sistema. No caso da linguagem JAVA, a memória alocada para um objeto é liberada por um processo de baixa prioridade denominado garbage collector. Quando esse objeto não estiver mais sendo usado e quando ele não puder mais ser acessado, esse objeto se torna "lixo".

Ou seja, a área de memória ocupada por esse objeto não mais será acessada pelos programas. Essa área de memória será liberada pelo processo de garbage collector de forma automática.

Em JAVA não é possível determinar quando um objeto será coletado. Podemos tentar acionar esse processo manualmente. Para isso, existem alguns métodos que, quando chamados em algum ponto do código do programa, sugerem que objetos não utilizados sejam coletados, em outras palavras, destruídos permanentemente. No entanto, isso não garante a execução do garbage collector nesse ponto, uma vez que outro processo de maior prioridade pode impedi-lo.

PRATICANDO

Temos então a seguinte sequência do ciclo de vida de um objeto:

1 - Criação: envolve as etapas de declaração e construção do objeto.
2 - Utilização: uso do objeto para sua função no programa.
3 - Destruição: finalização do objeto, liberando a memória utilizada por esse objeto.

Vamos ver como esse ciclo de vida pode ser visto em nosso exemplo da classe TestePessoa:

1 – Criação (Linhas 11 e 18): Pessoa p1 , p2 , p3 ;
p2 = new Pessoa ( ) ;

2 – Utilização (Linhas 29 e 30): p2.setNome ("Maria") ;
p2.setIdade (18) ;

3 – Destruição (Linha 49): p2 = null ;

No caso da destruição em JAVA não é chamado nenhum método destrutor. O fato de atribuir a referência do objeto p2 o valor "null", torna a área de memória que era referenciada por p2 sem mais nenhuma referência, portanto um "lixo". Dessa forma, quando o coletor de lixo (garbage collector) for acionado, essa área de memória será liberada.

FIQUE ATENTO!

Uma vez que a referência para objeto p2 se tornou nula na linha 49, essa referência não pode mais ser usada para executar nenhum método da classe Pessoa, exceto fazer uma chamada novamente ao construtor para criar um novo objeto.

PALAVRAS DO PROFESSOR

Bem caro(a) aluno(a), chegamos ao fim da unidade 2.

Nela vimos vários elementos usados em orientação a objetos. Aprendemos com criar uma classe, construir, usar e destruir objetos.

Espero que tenha gostado deste material, qualquer dúvida entre em contato com nossos tutores. Aguardo você na próxima unidade. Até lá!