Programação I

Aula 3: Classes e objetos

Apresentação

Começaremos a trabalhar com os conceitos de orientação a objetos, com a definição de classes, atributos e métodos.

Em seguida, abordaremos os métodos de acesso *setters* e *getters*, bem como diferenciaremos os conceitos de classes e objetos.

Por fim, você desenvolverá suas primeiras aplicações orientadas a objetos.

Objetivos

- Analisar os conceitos de classes, objetos, atributos e métodos;
- Identificar os conceitos de métodos de acessos;
- Desenvolver aplicações orientadas a objetos.

Programação orientada a objetos

A programação orientada a objetos tem como principal conceito representar, em um sistema computacional, um objeto da vida real.

Esta representação deve descrever o objeto quanto às suas características e ações que poderá realizar dentro do sistema.

Não devemos nos preocupar com todas as características presentes no objeto, mas com aquelas que serão necessárias ao sistema (requisitos).

Exemplo

Por exemplo, a placa de um automóvel é importante para um sistema de estacionamento, assim como a hora de chegada e saída.

Em alguns casos, o fabricante, modelo e a cor do automóvel poderão ser importantes, mas dificilmente iremos cadastrar o número do chassi do mesmo. Como o número do chassi não é facilmente visto e seu cadastramento dependeria da documentação do automóvel ou de uma análise para a identificação, que seria difícil, uma vez que é um identificador com muitas letras e números, acabaria por gerar filas e insatisfação dos clientes.

Entretanto, para o sistema de cadastramento do DETRAN, por exemplo, o número do chassi é uma das informações mais importantes. Dessa forma, identificarmos a placa é importante como descritor do automóvel para o sistema de estacionamento, já o chassi não.

Por isso, devemos analisar cada objeto separadamente e quais são as características importantes para o sistema em que o objeto será utilizado. Como outro exemplo, podemos notar que a matrícula, nome e CR de um aluno são importantes para o sistema acadêmico, mas o time para o qual o aluno torce ou sua religião não são. Por isso, os descritores **time** e **religião** não são importantes para o objeto **Aluno** em um sistema acadêmico.

Classes

As classes Java são responsáveis pelo conjunto de códigos para a criação de objetos e aplicações. Uma classe Java deve descrever as características e ações que o objeto possui ao ser representado em um sistema computacional, levando em consideração as cara

Atributos

Atributo é conceitualmente um descritor do objeto e deve representar uma característica dele. O conjunto de atributos do objeto deve representar todas as características importantes do objeto para o sistema.

Exemplo

String matricula; // atributo para armazenamento da matrícula
String nome; // atributo para armazenamento do nome
double cr; // atributo para armazenamento do cr

Métodos

Método é uma ação, um conjunto de instruções a serem executadas por um objeto para realizar uma determinada tarefa.

O conjunto de métodos de um objeto deve descrever todas as ações (tarefas ou funções) que o objeto poderá realizar dentro do sistema.

Exemplo

```
public int soma(int n1, int n2){
        int soma;
        soma = n1 + n2;
        return soma;
}
public void imprimeAumento(double salario, int percentual){
        double aumento;
        aumento = salario + salario * percentual / 100.0;
        System.out.println("O salário com aumento é: " + aumento);
}
```

Objetos

A classe modela o objeto de acordo com as necessidades do sistema para a sua descrição e suas ações. A partir de uma mesma classe, vários objetos diferentes, mas com características semelhantes, podem ser criados em um mesmo sistema ou em diferentes sistemas.

Se consideramos a classe **Aluno**, podemos criar a partir desta classe dezenas, centenas ou mesmo milhares de objetos Alunos com características semelhantes, tais como matrícula, nome e CR, mas com propriedades (valores próprios nos atributos de cada objeto) diferentes.

Os objetos só existem durante a execução do sistema, pois estes só existirão como referência na memória do computador neste momento. Dizemos também que os objetos só existem “em tempo de execução”, uma vez que o sistema ao ser encerrado terá toda a sua memória apagada. Consequentemente, todas as suas variáveis e objetos não existirão mais.

EXEMPLO

Aluno.java

```java
// Classe Aluno
public class Aluno { // declaração e início da classe
 // Atributos devem ser identificados começando por letras minúsculas
 String matricula, nome;
 double cr;

// Métodos devem ser identificados começando por letras minúsculas
public void imprimir( ){
   System.out.println("Matrícula: " + matricula);
   System.out.println("Nome : " + nome);
   System.out.println("CR : " + cr);
 }
} // término da classe
```

Aplicações Java

Aplicações em Java são classes especiais que possuem um método *main*(). O método *main* é responsável por criar os objetos e realizar a combinação de diferentes classes para atender às necessidades de um sistema.

Em cada sistema, temos apenas uma aplicação, que será responsável pela lógica de criação e uso das classes. A comunicação entre os objetos ocorre por meio de trocas de mensagens, que são expressas com o uso de métodos. Uma aplicação, então, cria objetos a partir de uma ou mais classes e usa os métodos dos objetos para realizar as ações que atenderão às necessidades dos usuários.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

AppAluno.java

```
//Aplicação para uso da Classe Aluno
public class AppAluno { // declaração e início da classe
    public static void main (String[] args){// método inicial da App
          Aluno aluno1 = new Aluno(); // Criação ou instanciação do objeto aluno1
          Aluno aluno2 = new Aluno(); // Criação ou instanciação do objeto aluno2
          Aluno aluno3 = new Aluno(); // Criação ou instanciação do objeto aluno3
          //definindo valores para os atributos do aluno1
          aluno1.matricula = "1001";
          aluno1.nome = "André";
          aluno1.cr = 6.7;
          //definindo valores para os atributos do aluno2
          aluno2.matricula = "1002";
          aluno2.nome = "Maria";
          aluno2.cr = 7.5 ;
          //definindo valores para os atributos do aluno3
          aluno3.matricula = "1003";
          aluno3.nome = "João";
          aluno3.cr = 7.0;
          //exibindo os valores dos atributos de cada aluno:
          aluno1.imprimir();
          aluno2.imprimir();
          aluno3.imprimir();
    }
}
```

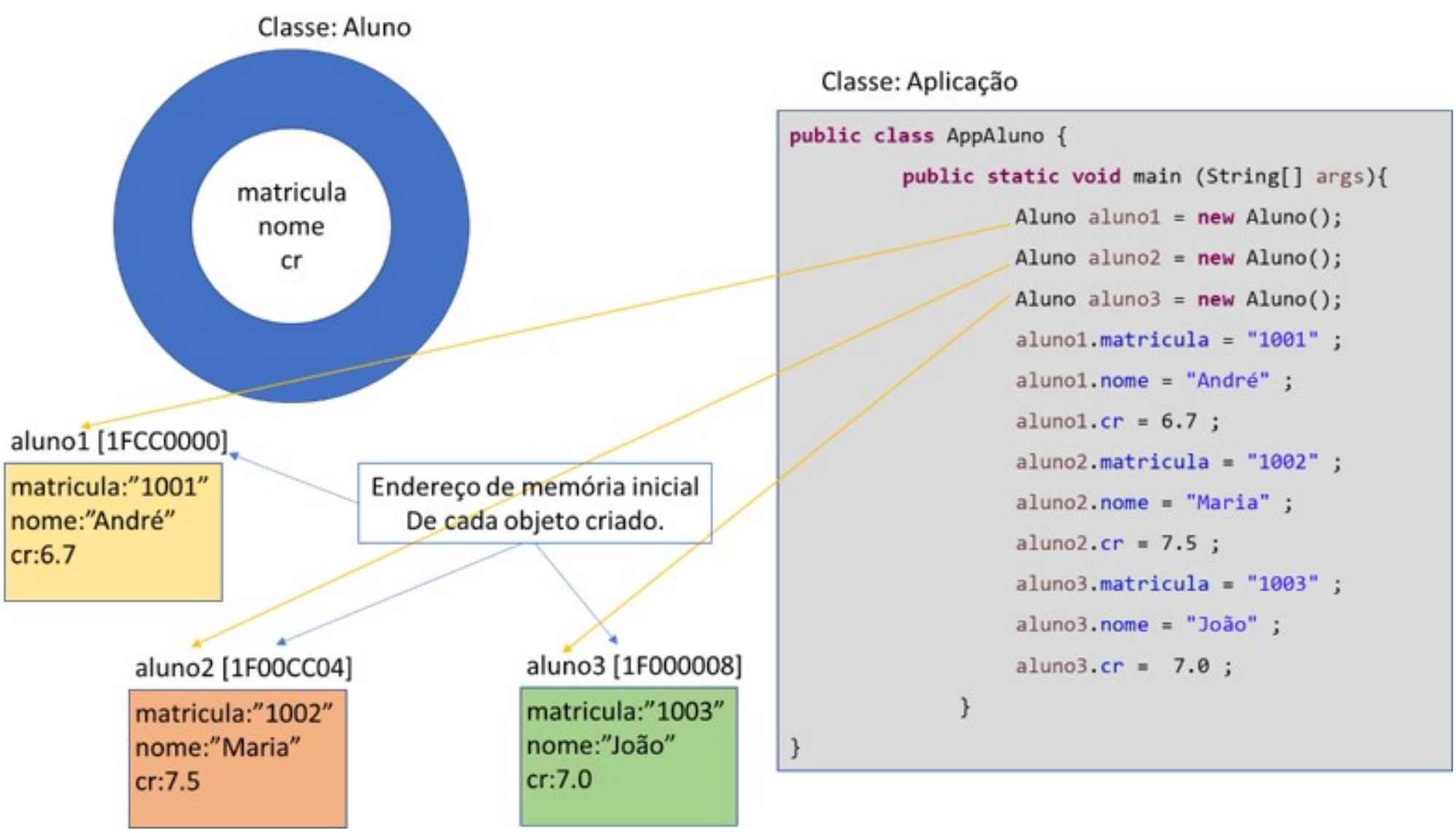
Notas:

1. Cada classe pública (*public*) deve ser criada em um arquivo próprio e o nome da classe deve ser o mesmo do arquivo. Ou seja, a classe Aluno deve ser criada no arquivo Aluno.java e a classe da aplicação AppAluno deve ser criada no arquivo AppAluno.java. No projeto, seja no Eclipse ou no Netbeans, deverão ser criadas duas classes, uma para o Aluno e outra para a aplicação;

2. Foi criada apenas uma classe Aluno, mas a partir dela poderemos criar quantos objetos quisermos;

3. Na aplicação foram criados três diferentes objetos do tipo Aluno. Isso faz com que cada objeto Aluno (aluno1, aluno2 e aluno3) seja criado na memória em locais diferentes (endereços) e possuam espaço de alocação de memória diferentes para cada atributo de cada objeto;

4. Cada objeto criado é independente do outro e possui valores próprios para os seus atributos (propriedades). Como a ação é realizada pelo objeto, cada método fará a ação sobre os atributos do objeto indicado, evitando que haja qualquer tipo de alteração indevida nos valores de cada um.



A classe **Aluno** e os objetos criados a partir dela.

Biblioteca de classes e reaproveitamento de código

A classe **Aluno** passou a ser uma biblioteca, e esta classe pode ser reutilizada em diversas outras aplicações. Esse conceito é um dos mais importantes na programação orientada a objetos, pois reduz o trabalho. Qualquer classe criada poderá ser reaproveitada inúmeras vezes por diversas aplicações, poupando esforço de desenvolvimento e facilitando a manutenção.

Cada classe criada se torna uma parte da sua biblioteca de classes e, conforme você vai criando novas classes, a sua biblioteca tende a aumentar. Dessa forma, quando você for criar novas aplicações, terá à sua disposição uma séria de classes já prontas e disponíveis para reaproveitar, sem precisar de novas.

Se você precisar realizar qualquer melhoria em uma classe da sua biblioteca, você poderá realizar sem problemas, pois qualquer inclusão não afetará o uso desta classe nas aplicações antigas, mantendo a compatibilidade entre todas as aplicações.

Facilidade de manutenção

Com base no reaproveitamento de código da programação orientada a objetos, podemos realizar alterações de melhoria, atualização ou qualquer manutenção em uma classe. Isso fará com que todas as aplicações sejam atualizadas quando forem recompiladas.

Métodos *Setters* e *Getters*

Por questões de segurança e falta de controle, não é comum realizar acessos diretos aos atributos de um objeto, por isso são criados métodos específicos para receber o valor e realizar a atribuição (Setters), ou para a recuperação (Getters) de um valor armazenado nos atributos de um objeto. Este processo pode evitar que valores incorretos sejam atribuídos sem qualquer chance de análise.

Métodos *Setters*

São métodos especiais que recebem o valor do atributo e, por serem métodos, podem analisar se são válidos, sendo responsáveis pela atribuição. Quando o atributo é protegido (privado), é necessário um método para realizar a atribuição.

Características dos métodos *Setters*:

- **São sempre do tipo *void***, pois métodos Setters não devem retornar nada;
- **Devem ser públicos** para que a aplicação tenha acesso ao método;
- **Devem começar pela palavra *set*** e o nome do atributo: como tem mais de uma palavra, cada nova palavra no nome deve começar por letra maiúscula;
- **Possui sempre um parâmetro do mesmo tipo do atributo que receberá o valor**, pois ambos (parâmetro e atributo) devem ser do mesmo tipo.

A verificação do valor a ser atribuído não pode ser realizada quando efetuamos uma atribuição direta:

```
Aluno a = new Aluno();
a.cr = -5.0;
```

O uso de um método *Setter* neste caso evitará que seja atribuído um valor inválido para o CR, no caso -5.0;

Exemplo

```
public void setCr(double c){
    if(c >=0.0 && c <= 10.0) {
        cr = c;
    }
}
// na aplicação:
a.setCr( -5.0 );
//
```

- Note que o parâmetro c recebe o valor a ser atribuído ao CR (-5.0), mas antes de atribuir é realizada uma verificação do valor para averiguar se o mesmo é válido. No caso, o valor do parâmetro é menor do que zero.
- Como sabemos que um CR não pode ser negativo, a atribuição não será realizada, assim como a tentativa de realizar a atribuição de um CR maior do que 10 (dez) também não permitirá que a atribuição ocorra.

Apenas atribuições com valores válidos poderão ser realizadas neste caso.

Métodos *Getters*

São métodos especiais que retornam o valor armazenado no atributo, evitando acesso direto a ele pela aplicação. Assim como visto no método *Setter*, a proteção do atributo (*private*) fará com que a aplicação não tenha acesso direto a ele, fazendo com que seja necessário um método público para recuperar o valor atribuído ao mesmo.

Características dos métodos *Getters*:

- **São sempre do mesmo tipo do atributo** que será retornado, nunca do tipo void;

- **Devem ser públicos** para que a aplicação tenha acesso ao método;

- **Devem começar pela palavra get e o nome do atributo**: como tem mais de uma palavra, cada nova palavra no nome deve começar por letra maiúscula;

- **Não possui parâmetro**: esses métodos nunca receberão parâmetros, uma vez que não farão atribuições ou ações com parâmetros, realizando apenas o retorno do valor armazenado no atributo.

Exemplo

```
public double getCr( ){
        return cr;
}
```

Note que não existe parâmetro, o método apenas deve retornar o valor armazenado e por isso não pode ser void, sendo o tipo de retorno do mesmo tipo do atributo que será retornado, e a ação é a de retorno (return).

```
// na aplicação:
double conceito = a.getCr();
```

No futuro, os atributos das nossas classes serão protegidos contra acesso direto (privado), impedindo que a aplicação possa acessar diretamente um atributo. Dessa forma, é necessário que usemos os métodos Setters e Getters para atribuir e recuperar os valores do atributo.

EXEMPLO

Aluno.java (versão com métodos Setters e Getters.)

```
public class Aluno {
   // Atributos devem ser identificados começando por letras minúsculas
 String matricula, nome;
 double cr;

// Métodos devem ser identificados começando por letras minúsculas
 public void setMatricula(String m){
   if(!m.isEmpty()) { // se o parâmetro m NÃO (!) estiver vazio
      matricula = m; // será feita a atribuição
   }
 }
 public String getMatricula(){
   return matricula; // retorna a matrícula
 }
   public void setNome(String n){
      if(!n.isEmpty()) { // se o parâmetro n NÃO (!) estiver vazio
   nome = n; // será feita a atribuição
 }
 }
   public String getNome(){
 return nome; // retorna o nome
 }
   public void setCr(double c){
      if (c >=0 && c<=10){ // se o parãmetro c for válido
   cr = c; // o valor de c será atribuído
 }
 }
   public double getCr(){
 return cr; // retorna o CR
   }
   public void imprimir( ){
         // os métodos Getters foram usados aqui
         System.out.println("Matrícula: " + getMatricula());
         System.out.println("Nome : " + getNome());
         System.out.println("CR : " + getCr());
 }
}
```

AppAluno.java (nova versão)

```
//Aplicação para uso da Classe Aluno
public class AppAluno { // declaração e início da classe
 public static void main (String[] args){// método inicial da App
        Aluno aluno1 = new Aluno(); // Criação ou instanciação do objeto aluno1
        Aluno aluno2 = new Aluno(); // Criação ou instanciação do objeto aluno2
        Aluno aluno3 = new Aluno(); // Criação ou instanciação do objeto aluno3
        //definindo valores para os atributos do aluno1
        aluno1.setMatricula( "1001" );
        aluno1.setNome( "André" );
        aluno1.setCr( 6.7 ) ;
        //definindo valores para os atributos do aluno2
        aluno2.setMatricula( "1002" );
        aluno2.setNome( "Maria" );
        aluno2.setCr( 7.5 );
        //definindo valores para os atributos do aluno3
        aluno3.setMatricula( "" ); // valor vazio, não será atribuído
        aluno3.setNome( "" ); // valor vazio, não será atribuído
        aluno3.setCr( 12 ); // valor do CR inválido, não será atribuído
        //exibindo os valores dos atributos de cada aluno:
        aluno1.imprimir();
        aluno2.imprimir();
        aluno3.imprimir();
    }
}
```

Notas:

1. Os valores dos atributos dos alunos 1 e 2 serão atribuídos normalmente, mas os valores do aluno3 não, porque a matrícula e o nome estão vazios e o CR não é válido;

2. Os valores foram atribuídos utilizando os métodos *Setters*, que verificaram se os valores eram válidos para só então realizar as atribuições;

3. Os métodos *Getters* foram usados na própria classe Aluno para buscar os valores armazenados nos atributos do objeto no método imprimir.

Exemplo prático

A classe **Carro** possui os atributos e métodos a seguir, crie a classe Carro e a aplicação AppCarro, realize a entrada de dados na aplicação através do teclado, e ao final imprima os dados dos respectivos carros (através do método imprimir()).

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Classe: Carro.

Atributos	Métodos
Fabricante : texto	- *Setters* para todos os atributos
Modelo : texto	- *Getters* para todos os atributos
Cor : texto	- Imprimir () // imprime todos os dados do carro
Placa : texto	
Valor : real	
NumeroPortas : inteiro	
AnoFabricacao : inteiro	
AnoModelo : inteiro	

Solução do exercício prático

Arquivo da classe Carro

Carro.java

```
public class Carro {
        // use as regras da boa prática em programação Java
        // para os identificadores da classe, dos atributos e dos métodos
        String fabricante, modelo, cor, placa;
        double valor;
        int numeroPortas, anoFabricacao, anoModelo;
        public String getFabricante () {
             return fabricante;
        }
        public void setFabricante (String fab) {
   if(!fab.isEmpty()) {
      fabricante = fab;
   }
 }
        public String getModelo () {
             return modelo;
        }
        public void setModelo (String mod) {
if(!mod.isEmpty()) {
   modelo = mod;
}
     }
     public String getCor () {
          return cor;
     }
     public void setCor (String co) {
if(!co.isEmpty()) {
          cor = co;
}
     }
     public String getPlaca () {
          return placa;
     }
     public void setPlaca (String pla) {
if(!pla.isEmpty()) {
          placa = pla;
     }
}
        public double getValor () {
             return valor;
        }
        public void setValor (double val) {
   if(val > 0) {
        valor = val;
   }
        }
        public int getNumeroPortas () {
             return numeroPortas;
}
        public void setNumeroPortas (int nump) {
   if(nump > 0) {
             numeroPortas = nump;
   }
        }
        public int getAnoFabricacao () {
             return anoFabricacao;
}
        public void setAnoFabricacao (int anof) {
```

```
    if(anof > 0) {
              anoFabricacao = anof;
    }
          }
          public int getAnoModelo () {
               return anoModelo;
}
          public void setAnoModelo (int anom) {
    if(anom > 0) {
              anoModelo = anom;
    }
          }
          public void imprimir () {
             //String fabricante, modelo, cor, placa;
             //double valor;
             //int numeroPortas, anoFabricacao, anoModelo;
             System.out.println( "Fabricante : " + getFabricante() );
             System.out.println( "Modelo : " + getModelo() );
             System.out.println( "Cor : " + getCor() );
             System.out.println( "Placa : " + getPlaca() );
             System.out.println( "Valor : " + getValor() );
             System.out.println( "Número de Portas : " + getNumeroPortas() );
             System.out.println( "Ano de fabricação: " + getAnoFabricacao() );
             System.out.println( "Ano do Modelo : " + getAnoModelo() );
          }
}
```

Arquivo da Aplicação (AppCarro.java).

AppCarro.java

```
import java.util.Scanner;

public class AppCarro {
 public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner entrada = new Scanner( System.in );
        Carro car1 = new Carro();
        System.out.println("--- Entrada de Dados - Primeiro Carro ---:");
        System.out.println("Digite o Fabricante do carro:");
        car1.setFabricante( entrada.nextLine() );
        System.out.println("Digite o Modelo do carro:");
        car1.setModelo( entrada.nextLine() );
        System.out.println("Digite a Cor do carro:");
        car1.setCor( entrada.nextLine() );
        System.out.println("Digite a Placa do carro:");
        car1.setPlaca( entrada.nextLine() );
        System.out.println("Digite o Valor do carro:");
        car1.setValor( Double.parseDouble( entrada.nextLine()) );
        System.out.println("Digite o Número de Portas do carro:");
        car1.setNumeroPortas( Integer.parseInt( entrada.nextLine()) );
        System.out.println("Digite o Ano de fabricação do carro:");
        car1.setAnoFabricacao( Integer.parseInt( entrada.nextLine()) );
        System.out.println("Digite o Ano do Modelo do carro:");
        car1.setAnoModelo( Integer.parseInt( entrada.nextLine()) );

        Carro car2 = new Carro();
        System.out.println("--- Entrada de Dados - Segundo Carro ---:");
        System.out.println("Digite o Fabricante do carro:");
        car2.setFabricante( entrada.nextLine() );
        System.out.println("Digite o Modelo do carro:");
        car2.setModelo( entrada.nextLine() );
        System.out.println("Digite a Cor do carro:");
        car2.setCor( entrada.nextLine() );
        System.out.println("Digite a Placa do carro:");
        car2.setPlaca( entrada.nextLine() );
        System.out.println("Digite o Valor do carro:");
        car2.setValor( Double.parseDouble( entrada.nextLine()) );
        System.out.println("Digite o Número de Portas do carro:");
        car2.setNumeroPortas( Integer.parseInt( entrada.nextLine()) );
        System.out.println("Digite o Ano de fabricação do carro:");
        car2.setAnoFabricacao( Integer.parseInt( entrada.nextLine()) );
        System.out.println("Digite o Ano do Modelo do carro:");
        car2.setAnoModelo( Integer.parseInt( entrada.nextLine()) );

        // Saída de dados
        System.out.println("--- Entrada de Dados - Primeiro Carro ---:");
        car1.imprimir();
        System.out.println("--- Entrada de Dados - Segundo Carro ---:");
        System.out.println("----- Dados do Segundo Carro -----");
        car2.imprimir();
 }
}
```

Notas:

Você pode ver que temos algumas repetições de código para realizar a entrada de dados de cada objeto. Se aumentarmos o número de objetos, aumentaremos consideravelmente o tamanho do código.

Teste realizado:

--- Entrada de Dados - Primeiro Carro ---

Digite o Fabricante do carro :
Ford
Digite o Modelo do carro :
Ecosport
Digite a Cor do carro :
Branca
Digite a Placa do carro :
RIO1A00
Digite o Valor do carro :
98000
Digite o Número de Portas do carro :
4
Digite o Ano de fabricação do carro :
2018
Digite o Ano do Modelo do carro :
2019
Digite o Fabricante do carro :
Fiat
Digite o Modelo do carro :
Argo
Digite a Cor do carro :
Preta
Digite a Placa do carro :
RIO2A00
Digite o Valor do carro :
60000
Digite o Número de Portas do carro :
4
Digite o Ano de fabricação do carro :
2019
Digite o Ano do Modelo do carro :
2019

--------- Dados do Primeiro Carro ----------

Fabricante : Ford
Modelo : Ecosport
Cor : Branca
Placa : RIO1A00
Valor : 98000.0
Número de Portas : 4
Ano de fabricação: 2018
Ano do Modelo : 2019

--------- Dados do Segundo Carro ----------

Fabricante : Fiat
Modelo : Argo
Cor : Preta

Placa : RIO2A00

Valor : 60000.0

Número de Portas : 4

Ano de fabricação: 2019

Ano do Modelo : 2019

Para resolver este problema e evitarmos a redundância de códigos, vamos incluir um novo método na classe Carro, um método para a entrada de dados. Desta forma, evitamos a redundância dos códigos de entrada de dados.

Classe: Carro.

Atributos	Métodos
Fabricante : texto	- Setters para todos os atributos
Modelo : texto	- Getters para todos os atributos
Cor : texto	- Imprimir () // imprime todos os dados do carro
Placa : texto	- **EntradaDados () // realiza a entrada de dados do carro**
Valor : real	
NumeroPortas : inteiro	
AnoFabricacao : inteiro	
AnoModelo : inteiro	

Nova solução do exercício prático, com a inclusão do método entradaDados na classe Carro:

Arquivo da classe Carro atualizado.

Carro.java

```
import java.util.Scanner;

public class Carro {
	// use as regras da boa prática em programação Java
	// para os identificadores da classe, dos atributos e dos métodos
	String fabricante, modelo, cor, placa;
	double valor;
	int numeroPortas, anoFabricacao, anoModelo;
	public String getFabricante () {
		return fabricante;
	}
	public void setFabricante (String fab) {
   if(!fab.isEmpty()) {
		fabricante = fab;
   }
 }
	public String getModelo () {
		return modelo;
	}
	public void setModelo (String mod) {
   if(!mod.isEmpty()) {
		modelo = mod;
	}
   }
	public String getCor () {
		return cor;
}
	public void setCor (String co) {
   if(!co.isEmpty()) {
		cor = co;
   }
	}
	public String getPlaca () {
		return placa;
}
	public void setPlaca (String pla) {
   if(!pla.isEmpty()) {
		placa = pla;
   }
	}
	public double getValor () {
		return valor;
}
	public void setValor (double val) {
   if(val > 0) {
		valor = val;
   }
	}
	public int getNumeroPortas () {
		return numeroPortas;
}
	public void setNumeroPortas (int nump) {
   if(nump > 0) {
		numeroPortas = nump;
   }
	}
	public int getAnoFabricacao () {
		return anoFabricacao;
```

```
}
        public void setAnoFabricacao (int anof) {
   if(anof > 0) {
            anoFabricacao = anof;
   }
        }
        public int getAnoModelo () {
            return anoModelo;
}
        public void setAnoModelo (int anom) {
   if(anom > 0) {
            anoModelo = anom;
   }
        }
        public void imprimir (){
        System.out.println( "-------------------------------------");
        System.out.println( "Fabricante : " + getFabricante() );
        System.out.println( "Modelo : " + getModelo() );
        System.out.println( "Cor : " + getCor() );
        System.out.println( "Placa : " + getPlaca() );
        System.out.println( "Valor : " + getValor() );
        System.out.println( "Número de Portas : " + getNumeroPortas() );
        System.out.println( "Ano de fabricação: " + getAnoFabricacao() );
        System.out.println( "Ano do Modelo : " + getAnoModelo() );
        }
        public void entradaDados () {
        Scanner entrada = new Scanner( System.in );
        // O objeto Scanner deve ficar local ao método
        // o objeto Scanner para entrada de dados não é um atributo do carro
        // é apenas um objeto auxiliar a entrada de dados
        System.out.println("Digite o Fabricante do carro :");
        setFabricante( entrada.nextLine() );
        System.out.println("Digite o Modelo do carro :");
        setModelo( entrada.nextLine() );
        System.out.println("Digite a Cor do carro :");
        setCor( entrada.nextLine() );
        System.out.println("Digite a Placa do carro :");
        setPlaca( entrada.nextLine() );
        System.out.println("Digite o Valor do carro :");
        setValor( Double.parseDouble( entrada.nextLine()) );
        System.out.println("Digite o Número de Portas do carro :");
        setNumeroPortas( Integer.parseInt( entrada.nextLine()) );
        System.out.println("Digite o Ano de fabricação do carro :");
        setAnoFabricacao( Integer.parseInt( entrada.nextLine()) );
        System.out.println("Digite o Ano do Modelo do carro :");
        setAnoModelo( Integer.parseInt( entrada.nextLine()) );
        }
}
```

Arquivo da Aplicação.

AppCarro.java

```
public class AppCarro {
    public static void main(String[] args) {
          // TODO Auto-generated method stub
          Carro car1 = new Carro();
          car1.entradaDados();
          car1.imprimir();
          Carro car2 = new Carro();
          car2.entradaDados();
          car2.imprimir();
          Carro car3 = new Carro();
          car3.entradaDados();
          car3.imprimir();
    }
}
```

Nota:

Você pode perceber agora que existe um método para a entrada de dados na classe Carro, e que ele está sendo usado por cada carro para realizar a entrada de dados pelo teclado, evitando que os códigos das entradas de dados fiquem redundantes.

Além disso, a aplicação ficou muito mais simples. Caso você tenha vários objetos carros, você não terá redundância, portanto sua aplicação ficará mais simples.

Faça um teste executando a nova aplicação e analise o resultado. Inclua mais dois objetos carros e teste novamente: você verá que a aplicação terá uma pequena mudança, mas a classe Carro ficará inalterada.

A partir deste momento, todas as classes deverão sempre conter o método entradaDados().

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Teste realizado:

```
----------------------------------

Digite o Fabricante do carro :
Ford
Digite o Modelo do carro :
Ecosport
Digite a Cor do carro :
Branca
Digite a Placa do carro :
RIO1A00
Digite o Valor do carro :
98000
Digite o Número de Portas do carro :
4
Digite o Ano de fabricação do carro :
2018
Digite o Ano do Modelo do carro :
2019

----------------------------------

Fabricante : Ford
Modelo : Ecosport
Cor : Branca
Placa : RIO1A00
Valor : 98000.0
Número de Portas : 4
Ano de fabricação: 2018
Ano do Modelo : 2019

----------------------------------

Digite o Fabricante do carro :
Fiat
Digite o Modelo do carro :
Argo
Digite a Cor do carro :
Preta
Digite a Placa do carro :
RIO2A00
Digite o Valor do carro :
60000
Digite o Número de Portas do carro :
4
Digite o Ano de fabricação do carro :
2019
Digite o Ano do Modelo do carro :
2019

----------------------------------

Fabricante : Fiat
```

Modelo : Argo
Cor : Preta
Placa : RIO2A00
Valor : 60000.0
Número de Portas : 4
Ano de fabricação: 2019
Ano do Modelo : 2019

Digite o Fabricante do carro :
Ford
Digite o Modelo do carro :
Fiesta
Digite a Cor do carro :
Prata
Digite a Placa do carro :
RIO3A00
Digite o Valor do carro :
52000
Digite o Número de Portas do carro :
4
Digite o Ano de fabricação do carro :
2018
Digite o Ano do Modelo do carro :
2018

Fabricante : Ford
Modelo : Fiesta
Cor : Prata
Placa : RIO3A00
Valor : 52000.0
Número de Portas : 4
Ano de fabricação: 2018
Ano do Modelo : 2018

Atividade

1) [Prova FAURGS - 2014 - TJ-RS – Programador] Considere a afirmação abaixo no que se refere a Linguagens Orientadas a Objetos.

Um programa em execução em uma linguagem orientada a objetos pode ser descrito como uma coleção de _____ que se _____ entre si através de _____.

Assinale a alternativa que preenche correta e respectivamente as lacunas do parágrafo acima:

a) mensagens – comunicam – objetos
b) objetos – comunicam – mensagens
c) classes – excluem – objetos
d) métodos – excluem – objetos
e) métodos – comunicam – herança

2) Dada a classe abaixo, implemente o método que permite a recuperação do valor armazenado no atributo peso do objeto atleta1.

```
public class Atleta {
 String nome;
      double peso, altura, imc;
}
```

Notas

Referências

Próxima aula

- Métodos Construtores e Polimorfismo de Sobrecarga.

Explore mais

Pesquise na internet, sites, vídeos e artigos relacionados ao conteúdo visto.

Em caso de dúvidas, converse com seu professor online por meio dos recursos disponíveis no ambiente de aprendizagem.