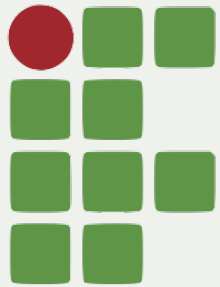
# Modificadores de acesso e método construtor

## POO29004 – Engenharia de Telecomunicações

Prof. Emerson Ribeiro de Mello

http://docente.ifsc.edu.br/mello/poo

11 de março de 2020



INSTITUTO FEDERAL
Santa Catarina
Câmpus São José

# Revisão da aula anterior

## Paradigma da programação orientada a objetos

- **Objetos interagem** com objetos através da **troca mensagens**
- A troca de mensagens ocorre através da **invocação de métodos** de objetos



# Revisão da aula anterior

## Paradigma da programação orientada a objetos

- **Objetos interagem** com objetos através da **troca mensagens**
- A troca de mensagens ocorre através da **invocação de métodos** de objetos

## Encapsulamento

- Emissor da mensagem **não precisa saber como o resultado foi obtido**, para este só importa o resultado
- O emissor precisa **conhecer quais operações o receptor sabe realizar** ou quais informações o receptor pode fornecer



# Modificadores de acesso: `public` e `private`

## Modificadores de acesso

Indicam **quais atributos** e **métodos** de um objeto estarão **visíveis aos demais objetos** do sistema



# Modificadores de acesso: `public` e `private`

- **private –** Os membros de uma classe (atributos e métodos) definidos como privados só poderão ser acessados pelos demais métodos da própria classe
- **public –** Os membros de uma classe definidos como públicos poderão ser invocados por métodos de qualquer classe



## Modificadores de acesso: `public` e `private`

- **private –** Os membros de uma classe (atributos e métodos) definidos como privados só poderão ser acessados pelos demais métodos da própria classe
- **public –** Os membros de uma classe definidos como públicos poderão ser invocados por métodos de qualquer classe

## Princípios da POO

- Geralmente **atributos** de uma classe **devem ser** declarados como **privados**
- **Métodos** geralmente devem ser **públicos**, porém há casos que um método só interessa a própria classe e assim este deve ser privado
- Isto **garante a integridade do estado do objeto**, pois somente métodos da própria classe poderão alterá-lo

# Modele uma classe para representar um Carro em um jogo

- A velocidade máxima do Carro é 200 km/h
- Para acelerar deve-se indicar o incremento que se deseja fazer na velocidade atual
- Para frear deve-se indicar o decremento que se deseja fazer na velocidade atual

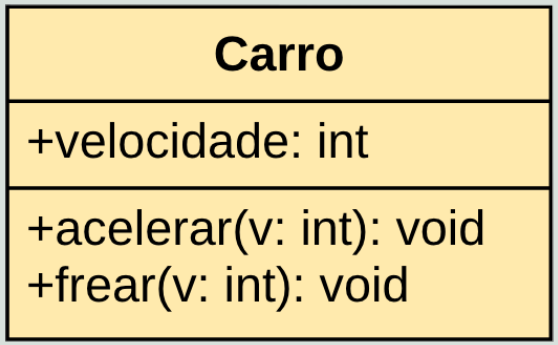


# Modele uma classe para representar um Carro em um jogo

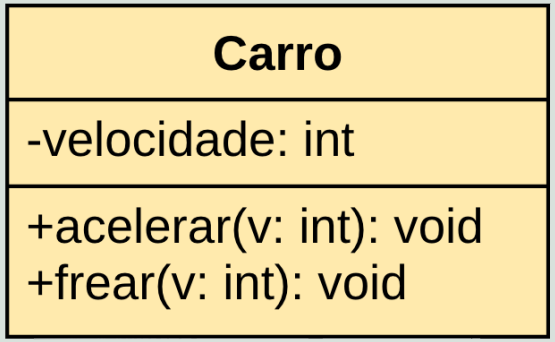
- A velocidade máxima do Carro é 200 km/h
- Para acelerar deve-se indicar o incremento que se deseja fazer na velocidade atual
- Para frear deve-se indicar o decremento que se deseja fazer na velocidade atual

## Qual modelagem seria mais adequada?

(a)




(b)



# Modificadores de acesso: exemplo não ideal

```
1  public class CarroNaoIdeal{
2    // atributos
3    public int velocidade = 0;
4
5    // metodos
6    public void acelerar(int v){
7      // o carro só pode atingir 200km/h
8      if ((velocidade + v) <= 200){
9        velocidade += v;
10     }else{
11       velocidade = 200;
12     }
13   }
14
15   public void frear(int v){
16     //....
17   }
18 }
```



# Modificadores de acesso: exemplo não ideal

```
19 public static void main(String args[]){
20   CarroNaoIdeal fusca = new CarroNaoIdeal();
21
22   // alterando a velocidade atraves dos metodos do objeto
23   fusca.acelerar(150);
24   fusca.acelerar(100);
25
26   // alterando diretamente o valor do atributo
27   fusca.velocidade = 400;
28 }
```



# Modificadores de acesso: exemplo não ideal

```
19 public static void main(String args[]){
20   CarroNaoIdeal fusca = new CarroNaoIdeal();
21
22   // alterando a velocidade atraves dos metodos do objeto
23   fusca.acelerar(150);// velocidade = 150
24   fusca.acelerar(100); // velocidade = 200
25
26   // alterando diretamente o valor do atributo
27   fusca.velocidade = 400; // velocidade = 400
28 }
```



# Modificadores de acesso: exemplo ideal

```
29 public class CarroIdeal{
30   // atributos
31   private int velocidade = 0;
32
33   // metodos
34   public void acelerar(int v){
35     // o carro só pode atingir 200km/h
36     if ((velocidade + v) <= 200){
37       velocidade += v;
38     }else{
39       velocidade = 200;
40     }
41   }
42
43   public void frear(int v){
44     //....
45   }
46 }
```



# Modificadores de acesso: exemplo ideal

```
47 public static void main(String args[]){
48   CarroIdeal fusca = new CarroIdeal();
49
50   // alterando a velocidade atraves dos metodos do objeto
51   fusca.acelerar(150);
52   fusca.acelerar(100);
53
54   // alterando diretamente o valor do atributo
55   fusca.velocidade = 400;
56 }
```



# Modificadores de acesso: exemplo ideal

```
47 public static void main(String args[]){
48   CarroIdeal fusca = new CarroIdeal();
49
50   // alterando a velocidade atraves dos metodos do objeto
51   fusca.acelerar(150);// velocidade = 150
52   fusca.acelerar(100); // velocidade = 200
53
54   // alterando diretamente o valor do atributo
55   fusca.velocidade = 400; // ERRO ! nao ira' compilar
56 }
```



# Modificadores de acesso: exemplo ideal

```
47 public static void main(String args[]){
48   CarroIdeal fusca = new CarroIdeal();
49
50   // alterando a velocidade atraves dos metodos do objeto
51   fusca.acelerar(150);// velocidade = 150
52   fusca.acelerar(100); // velocidade = 200
53
54   // alterando diretamente o valor do atributo
55   fusca.velocidade = 400; // ERRO ! nao ira' compilar
56 }
```

Leia mais sobre em
https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html



# Sobrecarga de métodos

Uma classe pode ter **mais de um método com o mesmo nome**, porém com assinaturas diferentes

- Tipo de retorno
- Nome do método
- Lista de parâmetros

# Sobrecarga de métodos

```
57 public class Data{
58     private int dia, mes, ano;
59
60     public void alterarData(int d){
61         this.dia = d;
62     }
63     public void alterarData(int d, int m){
64         this.dia = d; this.mes = m;
65     }
66     public void alterarData(int d, int m, int a){
67         this.dia = d; this.mes = m; this.ano = a;
68     }
69 }
```

```
70 Data d = new Data();
71 d.alterarData(31);
72 d.alterarData(31,12);
73 d.alterarData(31,12,1969);
```



# Método construtor

# Valores iniciais de atributos

```
1  public class Pessoa{
2     private String nome;
3     private String cpf;
4     private int anoNasc;
5
6     public void imprimirDados(){
7      System.out.println("Nome: " + nome);
8      System.out.println("CPF: " + cpf);
9      System.out.println("Ano: " + anoNasc);
10    }
11 }// fim da classe
```

```
12 Pessoa p = new Pessoa();
13 p.imprimirDados();
```



# Valores iniciais de atributos

```
1  public class Pessoa{
2     private String nome;
3     private String cpf;
4     private int anoNasc;
5
6     public void imprimirDados(){
7      System.out.println("Nome: " + nome);
8      System.out.println("CPF: " + cpf);
9      System.out.println("Ano: " + anoNasc);
10    }
11 }// fim da classe
```

```
12 Pessoa p = new Pessoa();
13 p.imprimirDados();
```

- O que será impresso?



# Valores iniciais de atributos

```
1  public class Pessoa{
2     private String nome;
3     private String cpf;
4     private int anoNasc;
5
6     public void imprimirDados(){
7      System.out.println("Nome: " + nome);
8      System.out.println("CPF: " + cpf);
9      System.out.println("Ano: " + anoNasc);
10    }
11 }// fim da classe
```

```
12 Pessoa p = new Pessoa();
13 p.imprimirDados();
```

- O que será impresso?

```
14 Nome:
15 CPF:
16 Ano: 0
```

# Valores iniciais de atributos

- Em Java atributos de um objeto que não forem iniciados na criação deste objeto, receberão valores padrões
  - números ficam `0`
  - boolean com `false`
  - referências de objetos com `null`



# Valores iniciais de atributos

- Em Java atributos de um objeto que não forem iniciados na criação deste objeto, receberão valores padrões
  - números ficam `0`
  - boolean com `false`
  - referências de objetos com `null`

## Uma boa prática de programação

### Sempre inicie os atributos de forma explícita

```
18 Pessoa p = new Pessoa();
19
20 p.definirNome("Joao");
21 p.definirCPF("123.456.789-00");
22 p.definirAno(1950);
```



# Método construtor

Método para **atribuir valores aos atributos** na **criação de um objeto**

- Possui obrigatoriamente o mesmo nome da classe
- Não pode possuir tipo de retorno



# Método construtor

Método para **atribuir valores aos atributos** na **criação de um objeto**

- Possui obrigatoriamente o mesmo nome da classe
- Não pode possuir tipo de retorno

```
public class Pessoa{
    private String nome;
    private String cpf;
    private int anoNasc;

    // método construtor
    public Pessoa(){
        nome = "";
        cpf = "";
        anoNasc = 0;
    }
}// fim da classe
```



## Método construtor padrão

Método cuja de lista de parâmetros está vazia. Toda classe Java possui um construtor padrão vazio implícito.

- Uma classe pode conter métodos construtores **sobrecarregados**
- Ao instanciar um objeto o desenvolvedor indica qual construtor irá chamar



# Método construtor: exemplo

```
public class Pessoa{
   private String nome, cpf;
   private int anoNasc;

   // método construtor padrão
   public Pessoa(){
      nome = ""; cpf = ""; anoNasc = 0;
   }

   // método construtor com 1 parâmetro
   public Pessoa(String no){
      nome = no; cpf = ""; anoNasc = 0;
   }

   // método construtor com 3 parâmetros
   public Pessoa(String no, String c, int a){
      nome = no; cpf = c; anoNasc = a;
   }
}// fim da classe
```

# Invocando métodos construtores

```
Pessoa a = new Pessoa();
Pessoa b = new Pessoa("Maria");
Pessoa c = new Pessoa("Maria","123.456.789-00",1959);
```



# Exercícios

# Lista 03