Construtor, modificadores de acesso, sobrecarga de métodos e atributos estáticos

CST em Análise e Desenvolvimento de Sistemas

Prof. Emerson Ribeiro de Mello

mello@ifsc.edu.br

INSTITUTO FEDERAL
Santa Catarina
Campus São José

Licenciamento

Slides licenciados sob Creative Commons “Atribuição 4.0 Internacional”

Sumário

1. Palavra reservada: `this`
2. Sobrecarga de métodos
3. Método construtor
4. Modificador `final`
5. Membros estáticos

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 para atributos e métodos

- **private**
  - Poderão ser acessados somente por métodos da própria classe
- **public**
  - Poderão ser acessados por métodos de qualquer classe

> ⚠ **Atenção**
>
> Os atributos de uma classe **não devem ser acessados diretamente** por outras classes, pois isso **quebra o encapsulamento**. Crie métodos públicos para acessar ou alterar os valores dos atributos

Modele uma classe para representar um Carro em um jogo

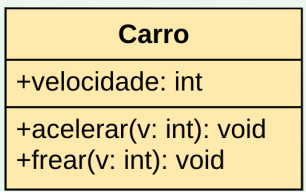
- O carro só se movimenta para frente e a velocidade máxima é 200 km/h
- Para aumentar a velocidade deve-se indicar o incremento que se deseja fazer na velocidade atual
- Para frear o carro deve-se indicar o decremento que se deseja fazer na velocidade atual

Modele uma classe para representar um Carro em um jogo

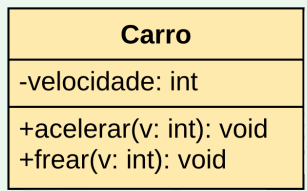
- O carro só se movimenta para frente e a velocidade máxima é 200 km/h
- Para aumentar a velocidade deve-se indicar o incremento que se deseja fazer na velocidade atual
- Para frear o carro deve-se indicar o decremento que se deseja fazer na velocidade atual

Qual modelagem seria mais adequada?

(a)



(b)



Modificadores de acesso: exemplo não ideal

```
public class CarroNaoIdeal{
  // atributos
  public int velocidade = 0;

  // metodos
  public void acelerar(int v){
    // o carro só pode atingir 200km/h
    if ((velocidade + v) <= 200){
      velocidade += v;
    }else{
      velocidade = 200;
    }
  }

  public void frear(int v){
    //....
  }
}
```

Modificadores de acesso: exemplo não ideal

```
public static void main(String args[]){
  CarroNaoIdeal fusca = new CarroNaoIdeal();

  // alterando a velocidade atraves dos metodos do objeto
  fusca.acelerar(150);
  fusca.acelerar(100);

  // alterando diretamente o valor do atributo
  fusca.velocidade = 400;
}
```

Modificadores de acesso: exemplo não ideal

```
public static void main(String args[]){
  CarroNaoIdeal fusca = new CarroNaoIdeal();

  // alterando a velocidade atraves dos metodos do objeto
  fusca.acelerar(150);// velocidade = 150
  fusca.acelerar(100); // velocidade = 200

  // alterando diretamente o valor do atributo
  fusca.velocidade = 400; // velocidade = 400
}
```

Modificadores de acesso: exemplo ideal

```
public class CarroIdeal{
  // atributos
  private int velocidade = 0;

  // metodos
  public void acelerar(int v){
    // o carro só pode atingir 200km/h
    if ((velocidade + v) <= 200){
      velocidade += v;
    }else{
      velocidade = 200;
    }
  }

  public void frear(int v){
    //....
  }
}
```

Modificadores de acesso: exemplo ideal

```
public static void main(String args[]){
  CarroIdeal fusca = new CarroIdeal();

  // alterando a velocidade atraves dos metodos do objeto
  fusca.acelerar(150);
  fusca.acelerar(100);

  // alterando diretamente o valor do atributo
  fusca.velocidade = 400;
}
```

Modificadores de acesso: exemplo ideal

```
public static void main(String args[]){
  CarroIdeal fusca = new CarroIdeal();

  // alterando a velocidade atraves dos metodos do objeto
  fusca.acelerar(150);// velocidade = 150
  fusca.acelerar(100); // velocidade = 200

  // alterando diretamente o valor do atributo
  fusca.velocidade = 400; // ERRO ! nao ira' compilar
}
```

Modificadores de acesso: exemplo ideal

```
public static void main(String args[]){
  CarroIdeal fusca = new CarroIdeal();

  // alterando a velocidade atraves dos metodos do objeto
  fusca.acelerar(150);// velocidade = 150
  fusca.acelerar(100); // velocidade = 200

  // alterando diretamente o valor do atributo
  fusca.velocidade = 400; // ERRO ! nao ira' compilar
}
```

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

Palavra reservada: `this`

Palavra reservada: `this`

- `this` é uma referência para o próprio objeto e pode ser usado para garantir que estará referenciando os membros de uma classe e não uma variável local

```
public class Pessoa{
    private String nome;
    private int idade;

    public void setNome(String nome){
        this.nome = nome; // o uso do this é obrigatório
    }

    public void setIdade(int i){
        // o uso do this não é obrigatório
        idade = i;
    }
}
```

Palavra reservada: `this`: Invocar outros métodos da classe

```
public class Retangulo{
    private int x, y;
    private int largura, altura;

    public Retangulo(){
        this(0,0,10,10); // invocando outro o construtor da classe
    }

    public Retangulo(int x, int y, int l, int a){
        this.setX(x); // invocando outro método da classe
        this.setY(y); // invocando outro método da classe
        this.largura = l;
        this.altura = a;
    }

    public void setX(int x){
        this.x = (x >= 0) ? x : 0;
    }
    //...
}
```

Sobrecarga de métodos

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

```
public class Data{
    private int dia, mes, ano;

    public void alterarData(int d){
        this.dia = d;
    }
    public void alterarData(int d, int m){
        this.dia = d; this.mes = m;
    }
    public void alterarData(int d, int m, int a){
        this.dia = d; this.mes = m; this.ano = a;
    }
}
```

```
Data d = new Data();
d.alterarData(31);
d.alterarData(31,12);
d.alterarData(31,12,1969);
```

Método construtor

Valores iniciais de variáveis em Java

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

- Variáveis locais devem ser inicializadas antes de serem usadas
- Atributos de um objeto que não forem iniciados na criação deste objeto, receberão valores padrões

Tipo	Valor padrão
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false
referências a objetos	null

Criando um objeto e atribuindo valores aos atributos

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

   public String toString(){
       return "Nome: " + nome + "CPF: " +
                cpf + "Ano: " + anoNasc;
   }
}// fim da classe
```

```
Pessoa p = new Pessoa();

// atribuindo valores aos atributos
p.definirNome("Joao");
p.definirCPF("123.456.789-00");
p.definirAno(1950);

// invoca o método toString()
System.out.println(p);
```

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
- Pode ser sobrecarregado
- Se não for definido, a classe terá um construtor padrão vazio implícito (sem parâmetros)

Método construtor

Exemplo de classe com 3 construtores sobrecarregados

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

    // método construtor padrão. Não possui parâmetros
    public Pessoa(){
        this.nome = ""; this.cpf = ""; this.anoNasc = 0;
    }

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

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

Método construtor

Exemplo de invocação de métodos construtores

```
// invocando o método construtor padrão
Pessoa a = new Pessoa();

// invocando o método construtor com 1 parâmetro
Pessoa b = new Pessoa("Maria");

// invocando o método construtor com 3 parâmetros
Pessoa c = new Pessoa("Maria","123.456.789-00",1959);
```

Modificador `final`

Modificador `final`

- Pode ser usado em atributos, métodos ou em variáveis locais
- Uma variável após ter recebido um valor, esse não poderá ser alterado
- Métodos não poderão ser sobrescritos (conceito de herança)
- Por convenção, nomes de constantes são escritos em letras maiúsculas

Modificador final

Exemplo de uso em atributos

```
public class Celular{
    // constante com valor definido na declaração da variável
    private final String FREQUENCIA = "1800";

    // constante com valor definido no construtor
    private final int SERIAL;

    public Celular(int s){
        // atribuição de valor a constante no construtor e que não poderá ser alterado
        this.SERIAL = s;
    }
}
```

Membros estáticos

Membros não estáticos

Atributos não estáticos

Cada instância da classe terá uma cópia distinta desse atributo

```
public class Celular{
    private int total;

    public Celular(){
        this.total++;
    }

    public int getTotal(){
        return this.total;
    }
}
```

```
Celular b = new Celular();
Celular c = new Celular();
int r = b.getTotal();
int t = c.getTotal();
```

- Qual será o valor de `r` e `t`?

Membros não estáticos

Atributos não estáticos

Cada instância da classe terá uma cópia distinta desse atributo

```
public class Celular{
    private int total;

    public Celular(){
        this.total++;
    }

    public int getTotal(){
        return this.total;
    }
}
```

```
Celular b = new Celular();
Celular c = new Celular();
int r = b.getTotal();
int t = c.getTotal();
```

- Qual será o valor de `r` e `t`?
  - r = 1
  - t = 1

Membros de classe estáticos

Membro de classe que pertença a classe e não a uma instância específica da classe

- Método estático não pode acessar membros não estáticos

```
public class Celular{
  private static int total;

  public Celular(){
    total++;
  }

  public static int getTotal(){
    return total;
    }
}
```

```
Celular b = new Celular();
Celular c = new Celular();
int r = b.getTotal();
int t = c.getTotal();
int h = Celular.getTotal();
System.out.print(r+","+t+","+h);
```

- Qual será o valor de `r`, `t` e `h`?

Membros de classe estáticos

Membro de classe que pertença a classe e não a uma instância específica da classe

- Método estático não pode acessar membros não estáticos

```
public class Celular{
  private static int total;

  public Celular(){
    total++;
  }

  public static int getTotal(){
    return total;
    }
}
```

```
Celular b = new Celular();
Celular c = new Celular();
int r = b.getTotal();
int t = c.getTotal();
int h = Celular.getTotal();
System.out.print(r+","+t+","+h);
```

- Qual será o valor de `r`, `t` e `h`?
  - r = 2
  - t = 2
  - h = 2

Membros de classe estáticos

```
public class Celular {
    private static int total = 0;
    private int creditos = 0;

    public Celular(int creditos){
        total++; this.creditos = creditos;
    }
    // OK
    public int getCreditos() { // não estático
        return creditos; // não estático
    }
    // OK
    public int getTotal(){ //não estático
        return total; // estático
    }
    // ERRO DE SINTAXE
    public static int retornaCreditos(){ //estático
        return creditos; // não estático
    }
}
```