

Programação Orientada a Objetos

Sobrecarga

Sobrecarga

- Definição

Métodos e construtores podem ter o mesmo nome desde que não sejam ambíguos.

- Exemplo:

Sobrecarga do construtor


```
public class Conta{
    private double saldo;
    Conta(){
        saldo = 500;
    }
    Conta(double saldoInicial){
        saldo = saldoInicial;
    }
    void deposito(double valor){
        saldo = saldo + valor;
    }
    void saque(double valor){
        saldo = saldo - valor;
    }
    double getSaldo(){
        return saldo;
    }
}
```

```
class Aplicacao{
    public static void main(String string[]){
        Conta conta = new Conta();
        Conta conta2 = new Conta(100);
        conta.deposito(100);
        System.out.println(conta.getSaldo());
        System.out.println(conta2.getSaldo());
    }
}
```

Sobrecarga

- Exemplo

Sobrecarga do construtor



```
public class Aluno {  
    private String matricula;  
    private double qualitativo;  
    public Aluno(){  
        this( "00INF000");  
    }  
    public Aluno(String matricula){  
        this.matricula = matricula;  
        qualitativo = 2.0;  
    }  
    public void registrarFalta(){  
        qualitativo -=0.1;  
    }  
    public double getQualitativo(){  
        return qualitativo;  
    }  
    public String getMatricula(){  
        return matricula;  
    }  
}
```

```
class Aplicacao{  
    public static void main(String args[]){  
        Aluno a = new Aluno();  
        Aluno b = new Aluno("14INF321");  
        a.registrarFalta();  
        a.registrarFalta();  
        System.out.println(a.getQualitativo());  
        System.out.println(b.getQualitativo());  
    }  
}
```

Sobrecarga

- Exemplo

Sobrecarga do construtor e do metodo *registrarFalta*.

```
public class Aluno {  
    private String matricula;  
    private double qualitativo;  
    public Aluno(){  
        this( "00INF000");  
    }  
    public Aluno(String matricula){  
        this.matricula = matricula;  
        qualitativo = 2.0;  
    }  
    public void registrarFalta(){  
        qualitativo -=0.1;  
    }  
    public void registrarFalta(int num){  
        qualitativo -=0.1*num;  
    }  
    public double getQualitativo(){  
        return qualitativo;  
    }  
    public String getMatricula(){  
        return matricula;  
    }  
}
```

```
class Aplicacao{  
    public static void main(String args[]){  
        Aluno a = new Aluno();  
        Aluno b = new Aluno("14INF321");  
        a.registrarFalta();  
        b.registrarFalta(4);  
        System.out.println(a.getQualitativo());  
        System.out.println(b.getQualitativo());  
    }  
}
```

Sobrecarga

- Método declaração

Assinatura

```
modificador tipoRetorno nomeMetodo(parâmetros){  
    <Corpo do método>  
}
```

- Assinatura: nome do método + tipos de parâmetros.
 - Obs.: o tipo de retorno não faz parte da assinatura do método, o mesmo ocorre com o nome dos parâmetros.

- Sobrecarga:

- Métodos com o **mesmo nome** mas de **assinaturas diferentes**(com a lista de parâmetros diferente).

Sobrecarga

- Exemplo

```
public class Aluno {  
    private String matricula;  
    private double qualitativo;  
    public Aluno(){  
        this( "00INF000");  
    }  
    public Aluno(String matricula){  
        this.matricula = matricula;  
        qualitativo = 2.0;  
    }  
    public void registrarFalta(){  
        qualitativo -=0.1;  
    }  
    public void registrarFalta(int num){  
        qualitativo -=0.1*num;  
    }  
    public double getQualitativo(){  
        return qualitativo;  
    }  
    public String getMatricula(){  
        return matricula;  
    }  
}
```

- Assinaturas:

▶ Aluno()

▶ Aluno(String)

▶ registrarFalta()

▶ registrarFalta(int)

Sobrecarga

```
public class Data {  
    private int dia;  
    private int mes;  
    private int ano;  
  
    public Data(){  
        dia = 1;  
        mes = 1;  
        ano = 1900;  
    }  
    public Data(int dia){  
        this.dia = dia;  
        this.mes = 1;  
        this.ano = 1900;  
    }  
    public Data(int dia, int mes){  
        this.dia = dia;  
        this.mes=mes;  
        this.ano = 1900;  
    }  
    public Data(int dia, int mes, int ano){  
        this.dia = dia;  
        this.mes=mes;  
        this.ano=ano;  
    }  
}
```

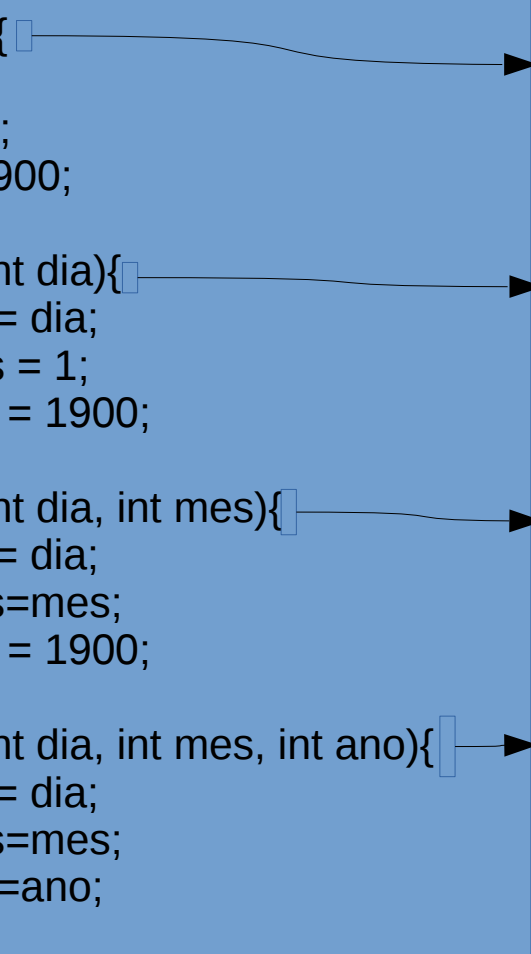


Diagram illustrating method overloading for the `Data` class. The class defines four constructors:

- `Data()`
- `Data(int)`
- `Data(int, int)`
- `Data(int, int, int)`

```
class Aplicacao{  
    public static void main(String args[]){  
        Data a = new Data();  
        Data b = new Data(12);  
        Data c = new Data(13,12);  
        Data d = new Data(03,02,2015);  
    }  
}
```

- Qual os valores armazenados nas datas referenciadas por *a*, *b*, *c* e *d*?

Sobrecarga

- Qual os valores armazenados nos objetos referenciados por *d* e *e*.

```
public class Data {  
    private int dia;  
    private int mes;  
    private int ano;  
  
    public Data(int ano, int mes, int dia){  
        this.dia = dia+1;  
        this.mes=mes+1;  
        this.ano = ano+1;  
    }  
    public Data(int dia, int mes, int ano){  
        this.dia = dia;  
        this.mes=mes;  
        this.ano=ano;  
    }  
}
```

```
class Aplicacao{  
    public static void main(String args[]){  
        Data d = new Data(03,02,2015);  
        Data e = new Data(2015,02,03);  
    }  
}
```


Sobrecarga

- Atenção os dois construtores possuem a mesmo assinatura. Isto é um erro.

```
public class Data {  
    private int dia;  
    private int mes;  
    private int ano;  
  
    public Data(int ano, int mes, int dia){  
        this.dia = dia+1;  
        this.mes=mes+1;  
        this.ano = ano+1;  
    }  
    public Data(int dia, int mes, int ano){  
        this.dia = dia;  
        this.mes=mes;  
        this.ano=ano;  
    }  
}
```

Diagram illustrating constructor overloading. Two constructors are shown, both with the signature `Data(int,int,int)`, which is incorrect for overloading.

Constructor 1: `Data(int ano, int mes, int dia)` (Signature: `Data(int,int,int)`)

Constructor 2: `Data(int dia, int mes, int ano)` (Signature: `Data(int,int,int)`)

Sobrecarga

```
public class Data {  
    private int dia;  
    private int mes;  
    private int ano;  
  
    public Data(){  
        dia = 1;  
        mes = 1;  
        ano = 1900;  
    }  
    public void setData(int dia){  
        this.dia = dia;  
        this.mes = 1;  
        this.ano = 1900;  
    }  
    public void setData(int dia, int mes){  
        this.dia = dia;  
        this.mes=mes;  
        this.ano = 1900;  
    }  
    public void setData(int dia, int mes, int ano){  
        this.dia = dia;  
        this.mes=mes;  
        this.ano=ano;  
    }  
}
```

```
class Aplicacao{  
    public static void main(String args[]){  
        Data a = new Data();  
        Data b = new Data();  
        Data c = new Data();  
        Data d = new Data();  
        a.setData(03);  
        b.setData(03,03);  
        c.setData(03,03,2013);  
    }  
}
```

Quais serão os valores das datas apontadas pelas referências *a*, *b*, *c* e *d*. ?

Exercício

- Crie uma classe Funcionario para uma aplicação que armazena a quantidade de horas trabalhadas por um funcionário em um mês. Nesta classe um funcionário registra o ponto(informar a quantidade de horas trabalhadas no dia) de duas forma, ele insere a hora de chega e a hora de saida ou ele informa o total de horas trabalhada.
 - Obs: para criar o método registrarPonto será necessário utilizar sobrecarga.
- Para testar sua classe crie um objeto do tipo funcionário, registre o ponto do funcionário nas duas formas indicadas e exiba o total de horas trabalhadas pelo funcionário.

Resposta

```
public class Funcionario {  
    private int totalHoras= 0;  
  
    public void registrarPonto(int chegada, int saida){  
        double totalDia = saida – chegada;  
        totalHoras += totalDia;  
    }  
    public void registrarPonto(int totalDia){  
        totalHoras += totalDia;  
    }  
    public int getTotalHoras(int dia, int mes, int ano){  
        return totalHoras;  
    }  
}
```

```
class Aplicacao{  
    public static void main(String args[]){  
        Funcionario a = new Funcionario();  
        a.registrarPonto(10,16);  
        a.registrarPonto(4);  
        System.out.println(a.getTotalHoras());  
    }  
}
```

Sobrecarga

- Sobrecarga no reaproveitamento de método.

```
public class Data {  
    private int dia;  
    private int mes;  
    private int ano;  
    public Data(){  
        this(1,1,1900);  
    }  
    public Data(int dia){  
        this(dia,1,1900);  
    }  
    public Data(int dia, int mes){  
        this(dia,mes,1900);  
    }  
    public Data(int dia, int mes, int ano){  
        this.dia = dia;  
        this.mes=mes;  
        this.ano=ano;  
    }  
}
```

```
public class Data {  
    private int dia;  
    private int mes;  
    private int ano;  
    public Data(){  
        this(1);  
    }  
    public Data(int dia){  
        this(dia,1);  
    }  
    public Data(int dia, int mes){  
        this(dia,mes,1900);  
    }  
    public Data(int dia, int mes, int ano){  
        this.dia = dia;  
        this.mes=mes;  
        this.ano=ano;  
    }  
}
```

Exercício

- Altere a classe Aluno para que os métodos/construtores sobrecarregados sejam reaproveitados.

```
public class Aluno {  
    private String matricula;  
    private double qualitativo;  
    public Aluno(){  
        this.matricula= "00INF000";  
        qualitativo = 2.0;  
    }  
    public Aluno(String matricula){  
        this.matricula = matricula;  
        qualitativo = 2.0;  
    }  
    public void registrarFalta(){  
        qualitativo -=0.1;  
    }  
    public void registrarFalta(int num){  
        qualitativo -=0.1*num;  
    }  
    public double getQualitativo(){  
        return qualitativo;  
    }  
    public String getMatricula(){  
        return matricula;  
    }  
}
```

Resposta

- Altere a classe Aluno para que os métodos/construtores sobrecarregados sejam reaproveitados.

```
public class Aluno {  
    private String matricula;  
    private double qualitativo;  
    public Aluno(){  
        this.matricula= "00INF000";  
        qualitativo = 2.0;  
    }  
    public Aluno(String matricula){  
        this.matricula = matricula;  
        qualitativo = 2.0;  
    }  
    public void registrarFalta(){  
        qualitativo -=0.1;  
    }  
    public void registrarFalta(int num){  
        qualitativo -=0.1*num;  
    }  
    public double getQualitativo(){  
        return qualitativo;  
    }  
    public String getMatricula(){  
        return matricula;  
    }  
}
```

```
public class Aluno {  
    private String matricula;  
    private double qualitativo;  
    public Aluno(){  
        this( "00INF000");  
    }  
    public Aluno(String matricula){  
        this.matricula = matricula;  
        qualitativo = 2.0;  
    }  
    public void registrarFalta(){  
        registrarFalta(1);  
    }  
    public void registrarFalta(int num){  
        qualitativo -=0.1*num;  
    }  
    public double getQualitativo(){  
        return qualitativo;  
    }  
    public String getMatricula(){  
        return matricula;  
    }  
}
```

Exercicio

- Altere a classe Funcionario para que o método sobrecarregado seja reproveitado na sua implementação.

String

- O tipo String, diferente dos tipos básicos(int, double, char, float..) é definido na classe String e como todo objeto possui métodos.
 - Métodos auxiliam no tratamento dos objetos do tipo String.
- Exemplo de métodos:
 - charAt(int) – Retorna o char do índice passado como argumento.
 - String a = “Stepheson”;
 - a.charAt(1); //Retorna o char ‘t’

String

- Alguns métodos úteis:
 - `charAt(int index)`: retorna o char de um índice da string
 - `compareTo(String anotherString)`: comparação lexicográfica entre duas Strings.
 - `contains(CharSequence s)`: verifica se a String contém uma sequência de caracteres.
 - `endsWith(String suffix)`: verifica se a String termina com o sufixo informado.
 - `startsWith(String prefix)`: verifica se a String começa com o prefixo informado.
 - `indexOf(int ch)`: retorna o índice da primeira ocorrência de um caractere na String.
 - `lastIndexOf(int ch)`: retorna a última ocorrência de um caractere na String.
 - `substring(int beginIndex)`: retorna uma substring do índice indicado até o final.
 - `substring(int beginIndex, int endIndex)`: retorna uma substring do índice indicada até segunda posição informada -1.
 - `toLowerCase()`: transforma a String em caixa baixa.
 - `toUpperCase()`: transforma a String em caixa alta.

String

- Exemplo:

```
String nome = "Stephenson Galvão";  
System.out.println(nome.charAt(1));  
System.out.println(nome.toUpperCase());  
System.out.println(nome.indexOf('e'));  
System.out.println(nome.indexOf(' '));  
System.out.println(nome.substring(11));  
System.out.println(nome.substring(0,10));  
System.out.println(nome.contains("son"));
```

```
t  
STEPHENSON GALVÃO  
2  
10  
Galvão  
Stephenson  
true
```

Exercício

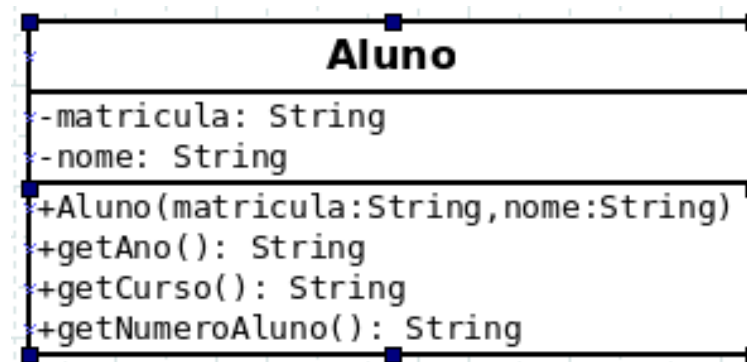
- Crie um classe para representar os alunos de uma aplicação os quais são formados por nome e matrícula(String), sendo esta ultima formato abaixo.
 - Adicione à classe metodos que retorne a parte da matricula que informe o ano de ingresso do curso, o prefixo do curso e o número do aluno.

2014INF003

Ano de ingresso ←

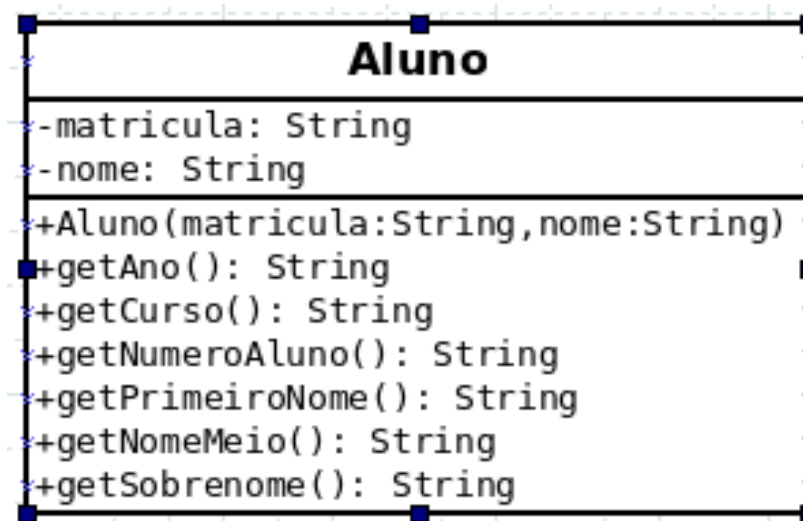
Curso ←

Numero do aluno na turma ←



Exercício

- Na classe do exercício anterior adicione métodos que retorne o primeiro nome do aluno, seus nomes do meio e seu sobrenome.



Conversão de tipo

- Cada tipo possui um biblioteca de funções de apoio
 - tipo int tem a biblioteca Integer
 - tipo double tem a biblioteca Double
 - tipo boolean tem a biblioteca Boolean.
- Cada biblioteca existe um método que converte um tipo em outro.
 - Principal é o `valueOf()`.

```
String numero = "01";  
int a = Integer.valueOf(numero);  
double b = Double.valueOf(numero);
```

Exercício

- Altere a classe aluno para que o método *getAno* retorne o ano do ingresso no tipo int e não no tipo String.