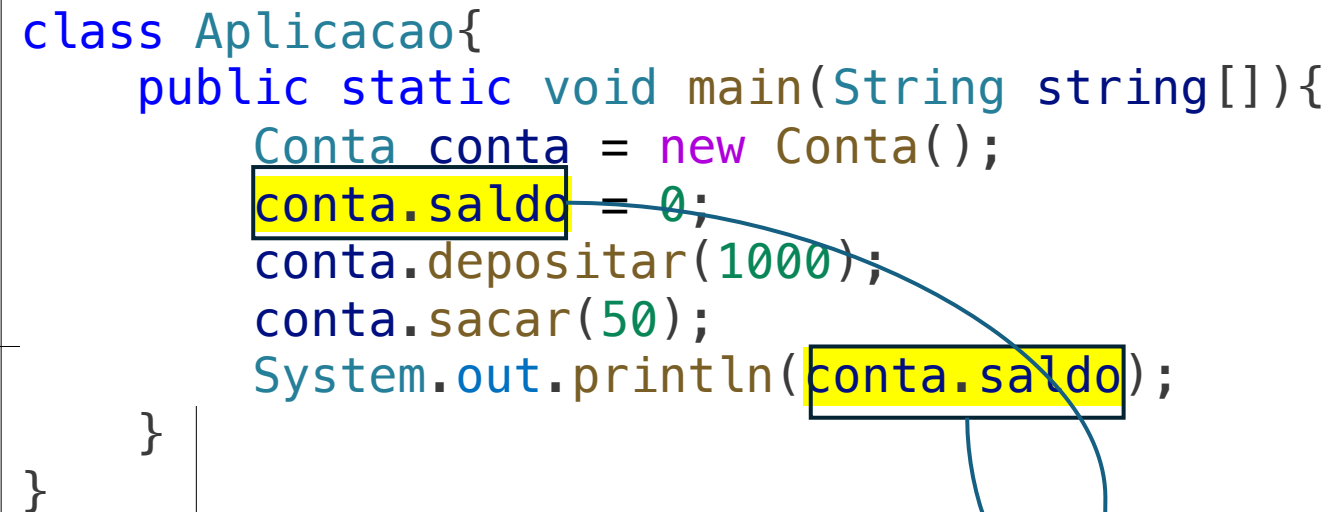


Programação Orientada a Objetos

Encapsulamento

```
public class Conta {  
    public String numero;  
    public double saldo;  
  
    public void depositar(double valor){  
        saldo = saldo + valor;  
    }  
  
    public void sacar(double valor){  
        saldo = saldo - valor;  
    }  
}
```

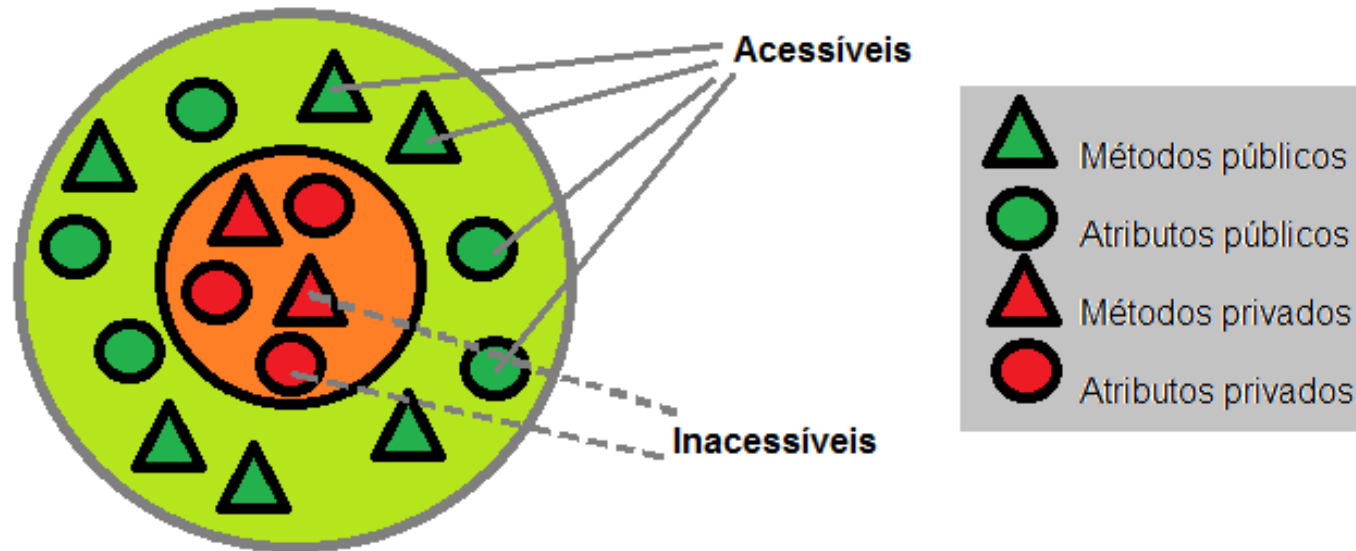
```
class Aplicacao{  
    public static void main(String string[]){  
        Conta conta = new Conta();  
        conta.saldo = 0;  
        conta.depositar(1000);  
        conta.sacar(50);  
        System.out.println(conta.saldo);  
    }  
}
```



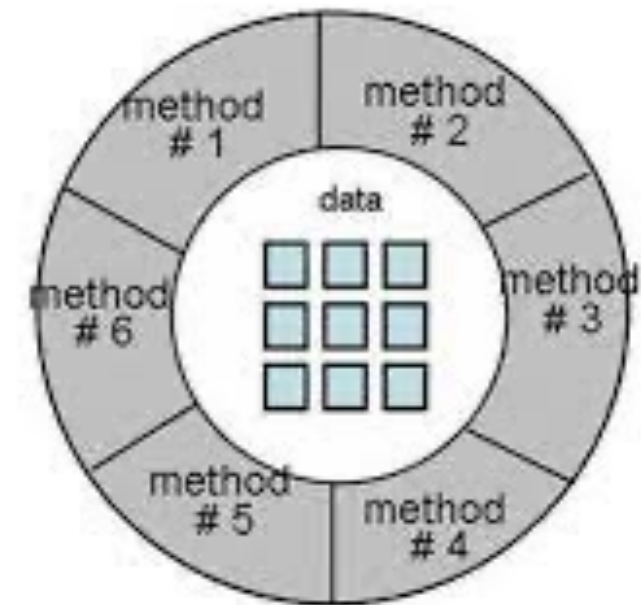
Acessando atributos

Não é um prática muito boa

Encapsulamento



Alguma informações da classe não pode ser acessadas diretamente fora delas



Acesso através de métodos

```
public class Conta {  
    public double saldo;  
  
    public void setSaldo(double valor){  
        saldo = valor;  
    }  
  
    public double getSaldo(){  
        return saldo;  
    }  
  
    public void depositar(double valor){  
        saldo = saldo + valor;  
    }  
  
    public void sacar(double valor){  
        saldo = saldo - valor;  
    }  
}
```

```
class Primeiro{  
    public static void main(String string[]){  
        Conta conta = new Conta();  
        conta.setSaldo(0);  
  
        conta.depositar(100);  
        conta.sacar(50);  
  
        System.out.println(conta.getSaldo());  
    }  
}
```

Atributos acessados através dos métodos **Get** e **Set**.

Entretanto....

```
public class Conta {  
    public double saldo;  
  
    public void setSaldo(double valor){  
        saldo = valor;  
    }  
  
    public double getSaldo(){  
        return saldo;  
    }  
  
    public void depositar(double valor){  
        saldo = saldo + valor;  
    }  
  
    public void sacar(double valor){  
        saldo = saldo - valor;  
    }  
}
```

```
class Aplicacao{  
    public static void main(String string[]){  
        Conta conta = new Conta();  
        conta.saldo = 0;  
        conta.depositar(1000);  
        conta.sacar(50);  
        System.out.println(conta.saldo);  
    }  
}
```

... o código ainda permite
acessar os atributos.

Modificadores de acesso

- Controlam o acesso aos atributos e métodos.
 - **public**: permite que todos seja acesso por qualquer classe.
 - **private**: só tem acesso dentro de própria classe

```
public class Conta {  
    private double saldo;  
  
    public void setSaldo(double valor){  
        saldo = valor;  
    }  
  
    public double getSaldo(){  
        return saldo;  
    }  
  
    public void depositar(double valor){  
        saldo = saldo + valor;  
    }  
  
    public void sacar(double valor){  
        saldo = saldo - valor;  
    }  
}
```

```
public class Conta {  
    private double saldo;  
  
    public void setSaldo(double valor){  
        saldo = valor;  
    }  
  
    public double getSaldo(){  
        return saldo;  
    }  
  
    public void depositar(double valor){  
        saldo = saldo + valor;  
    }  
  
    public void sacar(double valor){  
        saldo = saldo - valor;  
    }  
}
```

```
class Primeiro{  
    public static void main(String string[]){  
        Conta conta = new Conta();  
        conta.setSaldo(0);  
  
        conta.depositar(100);  
        conta.sacar(50);  
  
        System.out.println(conta.getSaldo());  
    }  
}
```



```
public class Conta {  
    private double saldo;  
  
    public void setSaldo(double valor){  
        saldo = valor;  
    }  
  
    public double getSaldo(){  
        return saldo;  
    }  
  
    public void depositar(double valor){  
        saldo = saldo + valor;  
    }  
  
    public void sacar(double valor){  
        saldo = saldo - valor;  
    }  
}
```

```
class Aplicacao{  
    public static void main(String string[]){  
        Conta conta = new Conta();  
        conta.saldo = 0;  
        conta.depositar(1000);  
        conta.sacar(50);  
        System.out.println(conta.saldo);  
    }  
}
```

Agora, ao tentar acessar saldo, irá gerar um erro

Atividade:

Utilizando encapsulamento, crie uma classe *Aluno* que tenha nome e média, os métodos de acesso a esses atributos e um método que retorna a situação do aluno (Aprovado ou Reprovado). Por último, crie um aluno, atribua um nome e uma média a ele e exiba a sua situação.

Aluno

- nome
- media

- + setNome(valor)
- + getNome: String
- + setMedia(valor)
- + getMedia(): double
- + calcularSituacao(): String

Atividade

Utilizando encapsulamento crie uma classe *Lampada* que tenha um estado (acesa ou apagada), os métodos de acesso a esse atributo e os métodos de acender e apagar a lâmpada. Depois, crie uma lâmpada; coloque seu estado (através do método set); altere seu estado (apagar e acender); e exiba o estado final.

Lampada

- estado

+ setEstado(valor)

+ getEstado: Tipo

+ acender()

+apagar()

Atividade:

Utilizando encapsulamento, crie uma classe Retângulo que tenha base e altura e método que retorna o valor de sua área. Por fim, crie um retângulo, atribua valores a sua base e altura e exiba a sua área

Retangulo

- base
- altura

+ setBase(valor)
+ setAltura(valor)
+ calcularArea(): Tipo

Atividade

Utilizando encapsulamento, crie uma classe *Funcionario*, que tenha nome, salário e um método para calcular e retornar o imposto de renda e outro para calcular e retornar o INSS, sendo o imposto 15% o valor do salário e o INSS 10% o valor do salário. Por fim, crie um funcionário atribua o seu salário e exiba seu imposto e INSS.

Funcionario

- salario
- nome

- + setNome(valor)
- + getNome: Tipo
- + setSalario(valor)
- + getSalario():Tipo
- + calcularIR()
- + calcularINSS()

```
public class Conta {  
    private double saldo;  
  
    public void setSaldo(double valor){  
        saldo = valor;  
    }  
  
    public double getSaldo(){  
        return saldo;  
    }  
  
    public void depositar(double valor){  
        saldo = saldo + valor;  
    }  
  
    public void sacar(double valor){  
        saldo = saldo - valor;  
    }  
}
```

```
class Primeiro{  
    public static void main(String string[]){  
        Conta conta = new Conta();  
        conta.setSaldo(0);  
  
        conta.depositar(100);  
        conta.sacar(50);  
  
        System.out.println(conta.getSaldo());  
    }  
}
```

Se podemos colocar o valor do saldo diretamente para que serve o saque e depósito?

O ideal seria que toda conta iniciassem com o saldo 0 (zero) e seu valor fosse alterado somente por saque ou depósito.

Construtor

- Método executado quando um objeto é criado.
- Possui o nome **igual** ao da classe e não tem tipo de retorno

```
public class Conta {  
    private double saldo;  
  
    public Conta(){  
  
    }  
  
    public void setSaldo(double valor){  
        saldo = valor;  
    }  
  
    public double getSaldo(){  
        return saldo;  
    }  
  
    public void depositar(double valor){  
        saldo = saldo + valor;  
    }  
  
    public void sacar(double valor){  
        saldo = saldo - valor;  
    }  
}
```

```
public class Conta {
    private double saldo;

    public Conta(){
        saldo = 0;
    }

    public void setSaldo(double valor){
        saldo = valor;
    }

    public double getSaldo(){
        return saldo;
    }

    public void depositar(double valor){
        saldo = saldo + valor;
    }

    public void sacar(double valor){
        saldo = saldo - valor;
    }
}
```

```
class Primeiro{
    public static void main(String string[]){
        Conta conta = new Conta();
        conta.setSaldo(0);

        conta.depositar(100);
        conta.sacar(50);

        System.out.println(conta.getSaldo());
    }
}
```


Atividade

Crie o construtor da classe *Lampada* e coloque para o seu estado inicial ser igual a apagada.

Lampada

- estado

- + Lampada()
- + setEstado(valor)
- + getEstado: Tipo
- + aceder()
- + apagar()

Inicialização

- Alguns tipos possuem valores padrões iniciais:
 - Quando não inicializado ele assume um valor pré-definido pela linguagem. Isso ocorre com tipos primitivos e String.

```
class Conta{  
    double saldo;  
  
    void deposito(double  
valor){  
        saldo = saldo +  
valor;  
    }  
  
    void saque(double valor){  
        saldo = saldo -  
valor;  
    }  
}
```

Por padrão o valor do saldo será 0 na criação de todas os objetos.

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

Inicialização

- Valores padrões da linguagem.

Tipo	Valor
boolean	false
byte	0
short	0
int	0
long	0
char	\u0000
float	0.0f
double	0.0d
String	""

EXERCÍCIO

- Crie um classe que defina os objetos do tipo aluno os quais possuem nome, matrícula e qualitativo. Por padrão todo aluno possui 2 pontos de qualitativos o quais são decrementados em 0,1 pontos a cada falta, seja por ausencia ou pela não entrega de um trabalho.
 - Obs: o aluno deve possuir um método `informarFalta` que decrementa o qualitativo do aluno em 0,1 ponto.
- Por fim, crie um aluno com o nome João de matrícula 14INF123 e atribua duas faltas ao aluno criado, por fim, informe sua matrícula e seus qualitativos.

Resposta

```
class Aluno {  
    String nome;  
    String matricula;  
    double qualitativo = 2.0;  
  
    void informarFalta(){  
        qualitativo -=0.1;  
    }  
}
```

```
class Aplicacao{  
    public static void main(String args[]){  
        Aluno a = new Aluno();  
        a.matricula = "Paula";  
        a.matricula = "14INF123";  
        a.informarFalta();  
        a.informarFalta();  
        System.out.println(a.qualitativo);  
    }  
}
```

```
class Aluno {  
    String nome;  
    String matricula;  
    double qualitativo = 2.0;  
    void informarFalta(){  
        qualitativo -=0.1;  
    }  
    double
```

```
class Aplicacao{  
    public static void main(String args[]){  
        Aluno a = new Aluno();  
        a.matricula = "Paula";  
        a.matricula = "14INF123";  
        a.informarFalta();  
        a.informarFalta();  
    }
```

Inicialização

- Outra forma de inicialização dos campos dos objetos é através do construtor.

– Construtor

- Rotina executada no momento da criação de um objeto.

– No momento da utilização do operador *new*.

- Exemplo: `Conta c = new Conta();`
(Neste momento)

Construtor

Construtor da classe
conta.

- Declaração:
 - Semelhante a um método.
 - Possui :
 - parâmetro
 - Nome.
 - Mas não possui o tipos de retorno.
 - Seu nome deve obrigatoriamente ser igual ao da

```
class Conta{  
    double saldo;  
    Conta(){  
        saldo = 500;  
    }  
    void deposito(double  
valor){  
        saldo = saldo +  
valor;  
    }  
    void saque(double valor){  
        saldo = saldo -  
valor;  
    }  
    double retornarSaldo(){  
        return saldo;  
    }  
}
```


Exemplo

Ativando o construtor da classe.

```
class Conta{
    double saldo;
    Conta(){
        saldo = 500;
        System.out.println("Criando
objeto);
    }
    void deposito(double valor){
        saldo = saldo + valor
    }
    void saque(double valor){
        saldo = saldo - valor;
    }
    double retornarSaldo(){
        return saldo;
    }
}
```

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

        System.out.println(conta.retornarSaldo());

        System.out.println(conta2.retornarSaldo());
    }
}
```

Construtor

Construtor com
parâmetro.

- Com argumentos:
 - Da mesma forma que os métodos os construtores possuem parâmetros:
 - Parâmetros
 - São informações passadas para o construtor no momento de sua execução.
 - Declaração
 - tipo nomeParametro
 - Atenção:
 - Construtores com parâmetros devem obrigatoriamente serem ativados com seus parâmetros:
 - Exemplo:
 - `Conta novaConta = new`

```
class Conta{  
    double saldo;  
    Conta(double  
saldoinicial){  
        saldo =  
saldoinicial;  
    }  
    void deposito(double  
valor){  
        saldo = saldo +  
valor;  
    }  
    void saque(double valor){  
        saldo = saldo -  
valor;  
    }  
    double retornarSaldo(){  
        return saldo;  
    }  
}
```

Exemplo

```
class Conta{
    double saldo;
    Conta(double
saldoinicial){
        saldo =
saldoinicial;
    }
    void deposito(double
valor){
        saldo = saldo +
valor;
    }
    void saque(double valor){
        saldo = saldo -
valor;
    }
    double retornarSaldo(){
        return saldo;
    }
}
```

Erro. Não informou
o parâmetro.

```
class Aplicacao{
    public static void main(String string[]){
        Conta conta = new Conta(500);
        Conta conta2 = new Conta(100);
        Conta conta3 = new Conta();
        conta2.deposito(100);

        System.out.println(conta.retornarSaldo());

        System.out.println(conta2.retornarSaldo());
    }
}
```

Construtores

- Qual a finalidade dos construtores
 - Informar valores para os objetos criados
 - Obrigar o usuário de uma classe a passar argumentos para o objeto durante o seu processo de criação.
- Exercício:
 - Altere o exercício anterior para que, a matrícula e o nome dos alunos devam obrigatoriamente serem informados no momento da criação do objeto. Crie mais um

Resposta

```
class Aluno {
    String nome;
    String matricula;
    double qualitativo;

    Aluno(String nomeA, String
matriculaA){
        nome= nomeA;
        matricula = matriculaA
        qualitativo = 2.0;
    }
    void informarFalta(){
        qualitativo -=0.1;
    }
    double retornarQualitativo(){
        return qualitativo;
    }
}
```

```
class Aplicacao{
    public static void main(String args[]){
        Aluno a = new
        Aluno("Paula","14INF123");
        Aluno b = new
        Aluno("Joao","14INF321");
        a.informarFalta();
        a.informarFalta();

        System.out.println(a.retornarQualitativo());

        System.out.println(b.retornarQualitativo());
    }
}
```

Construtor Padrão

- Construtor padrão.
 - Construtor sem argumento que está implícito em todas as classe
 - Exemplo
 - Mesmo sem declarar o construtor, estamos acionando o construtor de aluno

```
class Aluno {  
    String nome;  
    String matricula;  
    double qualitativo = 2.0;  
    void informarFalta(){  
        qualitativo -=0.1;  
    }  
}
```

```
class Aplicacao{  
    public static void main(String  
args[]){  
        Aluno a = new Aluno();  
        a.matricula = "Paula";  
        a.matricula = "14INF123";  
        a.informarFalta();  
        a.informarFalta();  
  
        System.out.println(a.qualitativo);  
    }  
}
```

Construtor Padrão

- É como se:

```
class Aluno {  
    String nome;  
    String matricula;  
    double qualitativo = 2.0;  
    void informarFalta(){  
        qualitativo -= 0.1;  
    }  
    double  
    retornarQualitativo(){  
        return qualitativo;  
    }  
}
```

=

```
class Aluno {  
    String nome;  
    String matricula;  
    double qualitativo = 2.0;  
  
    Aluno(){  
  
    }  
    void informarFalta(){  
        qualitativo -= 0.1;  
    }  
    double  
    retornarQualitativo(){  
        return qualitativo;  
    }  
}
```


Construtor Padrão

- A partir do momento que outro construtor é criado, o construtor padrão deixa de existir

```
class Aluno {  
    String nome;  
    String matricula;  
    double qualitativo = 2.0;  
  
    Aluno(String n, String m){  
        nome=n;  
        matricula =m;  
    }  
    void informarFalta(){  
        qualitativo -=0.1;  
    }  
    double  
    retornarQualitativo(){  
        return qualitativo;  
    }  
}
```

Nesta classe não existe mais
o construtor padrão

Erro

```
class Aplicacao{  
    public static void main(String args[]){  
        Aluno a = new Aluno();  
        a.informarFalta();  
        a.informarFalta();  
  
        System.out.println(a.retornarQualitativo());  
    }  
}
```

- Podemos ter vários construtores em uma classe

- Eles devem ser diferentes em número e tipos de parâmetros.
- No momento da criação do objeto escolhemos qual construtor queremos utilizar

- Exemplo:

- Conta a = new Conta();

- Conta b = new Conta(100);

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

Exemplo

```
class Conta{
    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 retornarSaldo(){
```

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

        System.out.println(conta.retornarSaldo());

        System.out.println(conta2.retornarSaldo());
    }
}
```

EXERCÍCIO

- Altere o exercício anterior para que a classe aluno possua 2 construtores. Um sem argumento, que coloca o valores “Aluno” e “00INF000” para nome e matrícula, e outro que receba o nome do aluno e a matrícula como parâmetros. Crie dois alunos, um com cada um dos construtores.
 - Em ambos os construtores os qualitativos devem ser inicializados com 2 pts.

```
class Aluno {
    String nome;
    String matricula;
    double qualitativo;
    Aluno(){
        nome= "Aluno";
        matricula = "00INF000";
        qualitativo = 2.0;
    }
    Aluno(String nomeA, String
matriculaA){
        nome= nomeA;
        matricula = matriculaA;
        qualitativo = 2.0;
    }
    void informarFalta(){
        qualitativo -=0.1;
    }
}
```

```
class Aplicacao{
    public static void main(String args[]){
        Aluno a = new Aluno();
        Aluno b = new
Aluno("Joao","14INF321");
        a.informarFalta();
        a.informarFalta();

        System.out.println(a.retornarQualitativo());

        System.out.println(b.retornarQualitativo());
    }
}
```

Construtor Reaproveitamento

Reaproveitamento

- Podemos reaproveitar o código de um construtor em outro construtor.

– Isso é feito pela palavra ***this***.

- Substitui o nome do construtor reaproveitado.
- Deve ser o primeiro comando utilizado no construtor.

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

Detalhes

```
class Conta{  
    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 retornarSaldo(){  
        return saldo;  
    }  
}
```

```
class Conta{  
    double saldo;  
    Conta(){  
        this(500);  
    }  
    Conta(double saldoinicial){  
        saldo = saldoinicial;  
    }  
    void deposito(double  
valor){  
        saldo = saldo +  
valor;  
    }  
    void saque(double valor){  
        saldo = saldo -  
valor;  
    }  
    double informarSaldo(){  
        return saldo;  
    }  
}
```


Details

```
class Conta{
    double saldo;
    Conta(){
        System.out.println("Sem
argumento");
        this(500);
    }
    Conta(double saldoInicial){
        saldo = saldoInicial;
    }
    void deposito(double valor){
        saldo = saldo + valor;
    }
    void saque(double valor){
        saldo = saldo - valor;
    }
    double informarSaldo(){
        return saldo;
    }
}
```

```
class Conta{
    double saldo;
    Conta(){
        this(500);
        System.out.println("Sem
argumento");
    }
    Conta(double saldoInicial){
        saldo = saldoInicial;
    }
    void deposito(double valor){
        saldo = saldo + valor;
    }
    void saque(double valor){
        saldo = saldo - valor;
    }
    double informarSaldo(){
        return saldo;
    }
}
```

EXERCÍCIO

- Altere o exercício anterior para que o construtor sem argumento reaproveite o construtor com argumento da classe Aluno.

Resposta

```
class Aluno {
    String nome;
    String matricula;
    double qualitativo;
    Aluno(){
        nome= "Aluno";
        matricula = "00INF000";
        qualitativo = 2.0;
    }
    Aluno(String nomeA, String
matriculaA){
        nome= nomeA;
        matricula = matriculaA;
        qualitativo = 2.0;
    }
    void informarFalta(){
        qualitativo -=0.1;
    }
    double retornarQualitativo(){
        return qualitativo;
    }
}
```

```
class Aluno {
    String nome;
    String matricula;
    double qualitativo;

    Aluno(){
        this("Aluno","00INF00");
    }

    Aluno(String nomeA, String matriculaA){
        nome= nomeA;
        matricula = matriculaA;
        qualitativo = 2.0;
    }
    void informarFalta(){
        qualitativo -=0.1;
    }
    double retornarQualitativo(){
        return qualitativo;
    }
}
```