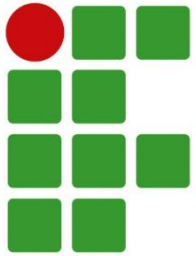


INSTITUTO FEDERAL
Goiás

Instituto Federal de Goiás
Câmpus Goiânia
Bacharelado em Sistemas de Informação
Disciplina: Programação Orientada a Objetos I

Classes x Objetos

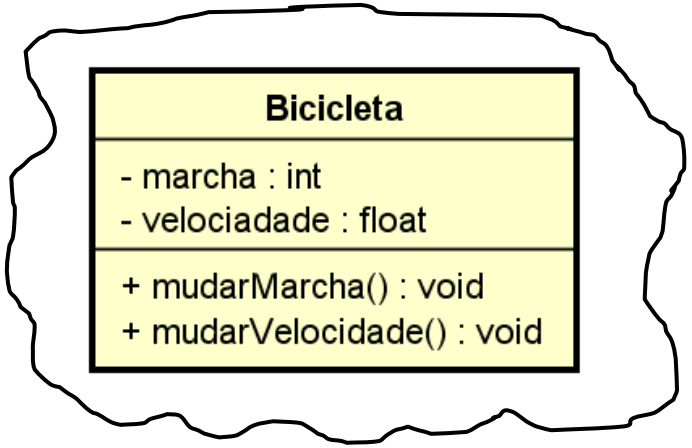
Prof. Ms. Renan Rodrigues de Oliveira
Goiânia - GO

Abstração

Construção de um modelo para representação de uma realidade, com foco nos aspectos essenciais.

Na POO, uma classe é uma abstração de entidades existentes no domínio do sistema de software.

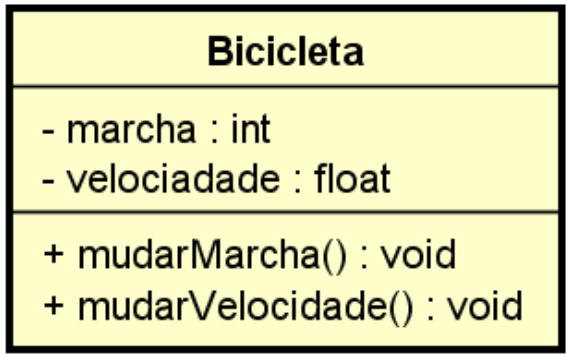
Realidade



Modelo

Classe

A unidade fundamental de programação em orientação a objetos é a classe.



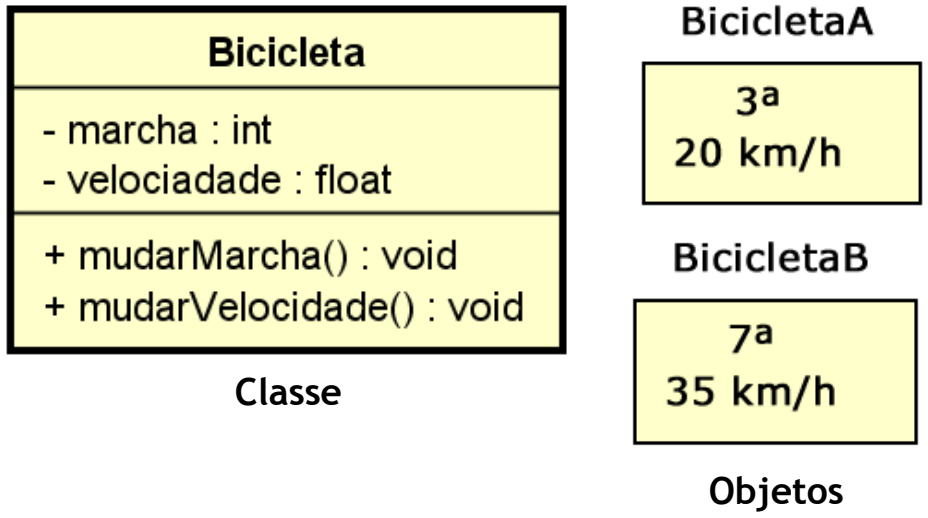
Classe

Classes contém:

- **Atributos**
  - São os dados (simples ou compostos) que caracterizam objetos daquela classe;
  - São armazenadas em variáveis;
  - Constituem o estado do objeto.
- **Operações**
  - São os métodos (procedimentos ou funções) que manipulam os dados.

Objetos

Um programa orientado a objetos é composto por um conjunto de objetos que interagem entre si.



Objetos:

- São instâncias da classe;
- Objetos de software são conceitualmente similares a objetos do mundo real: eles consistem do estado e o comportamento relacionado;
- Um objeto armazena seu estado em campos (variáveis) e expõe seu comportamento através de métodos (funções);
- Objeto está para classe da mesma forma que variável está para tipo de dado.

Implementação de uma Classe em Java

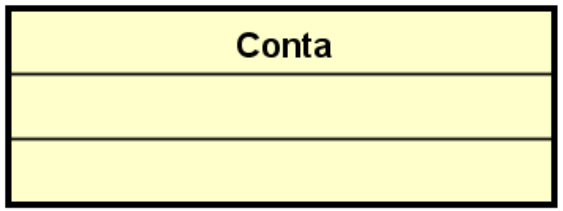
Exercício:

- Desenvolver uma classe Java chamada Conta com os seguintes atributos: numero, titular e saldo. A classe Conta deverá conter os seguintes métodos:

Método	Descrição
sacar()	Recebe o valor do saque e devolver um valor booleano indicando se existia saldo suficiente e a operação foi bem sucedida.
depositar()	Recebe o valor do depósito e incrementa o valor do saldo. O método deve devolver um valor booleano indicando se a operação foi bem sucedida.
imprimir()	Este método não retorna valor e deverá mostrar na tela todos os atributos da classe Conta. Para imprimir em Java, utilize o comando System.out.println().

Modelando a Classe Conta

Como modelar a classe Conta?



Modificadores de Acesso

Definem o escopo/visibilidade de um método/atributo.

Conta
- numero : int - titular : String - saldo : float
+ sacar(valor : float) : boolean + depositar(valor : int) : boolean + imprimir() : void

Marcações de acesso especificam a visibilidade dos atributos e operações.

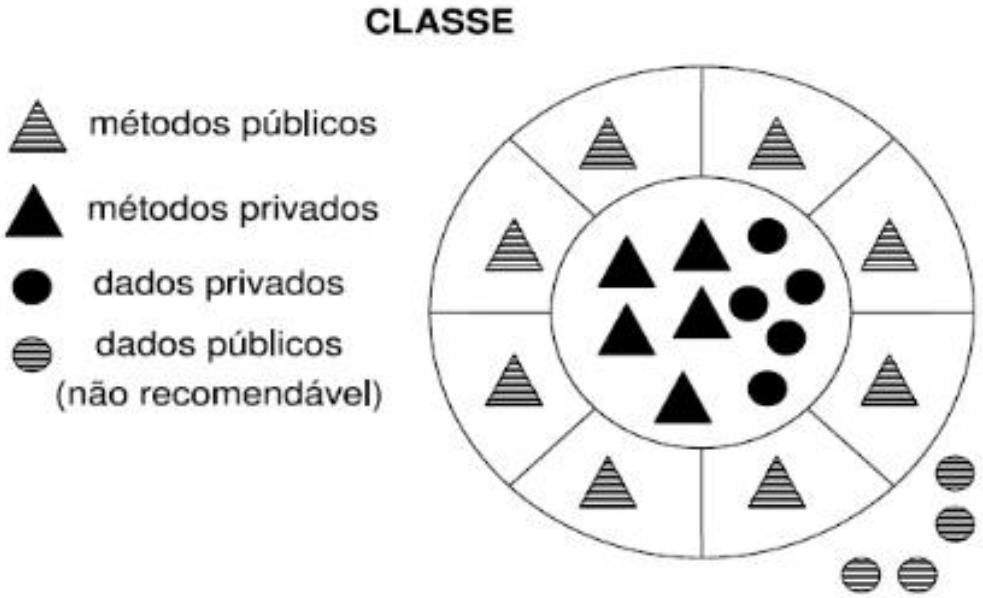
- **public**
  - Simbolizado por “+”
  - Indica que o atributo/método pode ser acessado por objetos de outras classes.
- **private**
  - Simbolizado por “-”
  - Protege o atributo do acesso externo, permitindo ao mesmo ser acessado somente por métodos daquela classe e pelo objeto instanciado;
- protected
  - Simbolizado por “#”
  - Permite o acesso a objetos de classes filhas mas protege do acesso de objetos que não fazem parte da hierarquia de classes.

A Ideia de Encapsulamento ...

Esconder todos os membros de uma classe, além de esconder como funcionam as rotinas do nosso sistema.

Encapsular é **fundamental** para que seu sistema seja suscetível a mudanças: não precisaremos mudar uma regra de negócio em vários lugares, mas sim em apenas um único lugar, já que essa regra está **encapsulada**.

É conjunto de métodos públicos de uma classe, pois esta é a única maneira a qual você se comunica com objetos dessa classe.



Programando para a Interface e não para a Implementação

A implementação em si, o conteúdo dos métodos, não tem tanta importância para o usuário dessa classe, uma vez que ele só precisa saber o que cada método pretende fazer, e não como ele faz, pois isto pode mudar com o tempo.

Existem diversas analogias fáceis no mundo real:

- Quando você dirige um carro, o que te importa são os pedais e o volante (interface) e não o motor que você está usando (implementação). É claro que um motor diferente pode te dar melhores resultados, mas o que ele faz é o mesmo que um motor menos potente, a diferença está em como ele faz.
- Todos os celulares fazem a mesma coisa (interface), eles possuem maneiras (métodos) de discar, ligar, desligar, atender, etc. O que muda é como eles fazem (implementação).

Acessando ou Modificando Atributos?

Aplicando a ideia do encapsulamento, os atributos deveriam ser todos privados.

Muitas vezes, é necessário consultar e alterar o valor de um atributo a partir de qualquer lugar do sistema.

Mas, o que é melhor? Criar os dois métodos (um de leitura e outro de escrita) ou deixar o atributo público?

- Quando queremos consultar a quantidade de combustível de um automóvel, olhamos o painel ou abrimos o tanque de combustível?
- Quando queremos alterar o toque da campainha de um celular, utilizamos os menus do celular ou desmontamos o aparelho?

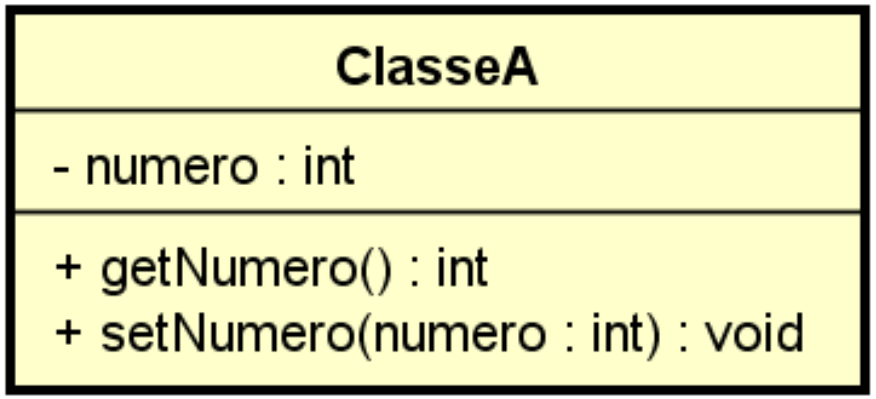
Acessar ou modificar as propriedades de um objeto manipulando diretamente os seus atributos é uma abordagem que normalmente gera problemas.

Getters e Setters

Para permitir o acesso aos atributos (já que eles são private) de uma maneira controlada, a prática mais comum é criar dois métodos:

- Um que retorna o valor;
- E outro que muda o valor.

O padrão para esses métodos é colocar a palavra get ou set antes do nome do atributo.



O padrão do método get não vale para variáveis do tipo boolean.

- Esses atributos são acessados via is e set.
- Exemplo: para verificar se uma lâmpada está acesa, seriam criados os métodos isLigado() e setLigado().

Implementando a Classe Conta

Conta
- numero : int - titular : String - saldo : float
+ sacar(valor : float) : boolean + depositar(valor : float) : boolean + imprimir() : void

Conta.java

```
1  public class Conta {
2
3      private int numero;
4      private String titular;
5      private float saldo;
6
7      public boolean sacar(float valor){...}
8      public boolean depositar(float valor){...}
9      public void imprimir(){}
10 }
```

Nome da Classe

Início da Classe

Fim da Classe

- Classes iniciam com a palavra reservada class;
- Convenciona-se utilizar o nome da classe com a 1ª letra em maiúscula;
- Uma "{" (abre chave) marca o início e "}" (fecha chave) marca o fim da classe;
- Normalmente cada classe é salva como um arquivo de mesmo nome da classe com extensão java (exemplo: Conta.java).

Implementando a Classe Conta

Conta.java

```
1  public class Conta {
2
3      private int numero;
4      private String titular;
5      private float saldo;
6
7      public boolean sacar(float valor){...}
8      public boolean depositar(float valor){...}
9      public void imprimir(){}
10 }
```

Tipo do Atributo

Atributos
(Variáveis de Instância)

Modificador de Acesso

Tipo de Retorno

Conta
- numero : int - titular : String - saldo : float
+ sacar(valor : float) : boolean + depositar(valor : float) : boolean + imprimir() : void

Operações
(Métodos de Instância)

- Atributos e métodos iniciam pelo modificador de acesso;
- Atributos obrigatoriamente possuem um tipo;
- Métodos devem indicar o tipo de retorno ou void quando não retornam nada;
- Métodos podem receber argumentos (parâmetros).

Implementando a Classe Conta

Parâmetro do método

Variável local

Variável de Instância

```
public boolean sacar(float valor) {
    boolean sucesso = false;

    if (saldo >= valor) {
        saldo -= valor;
        sucesso = true;
    }

    return sucesso;
}
```

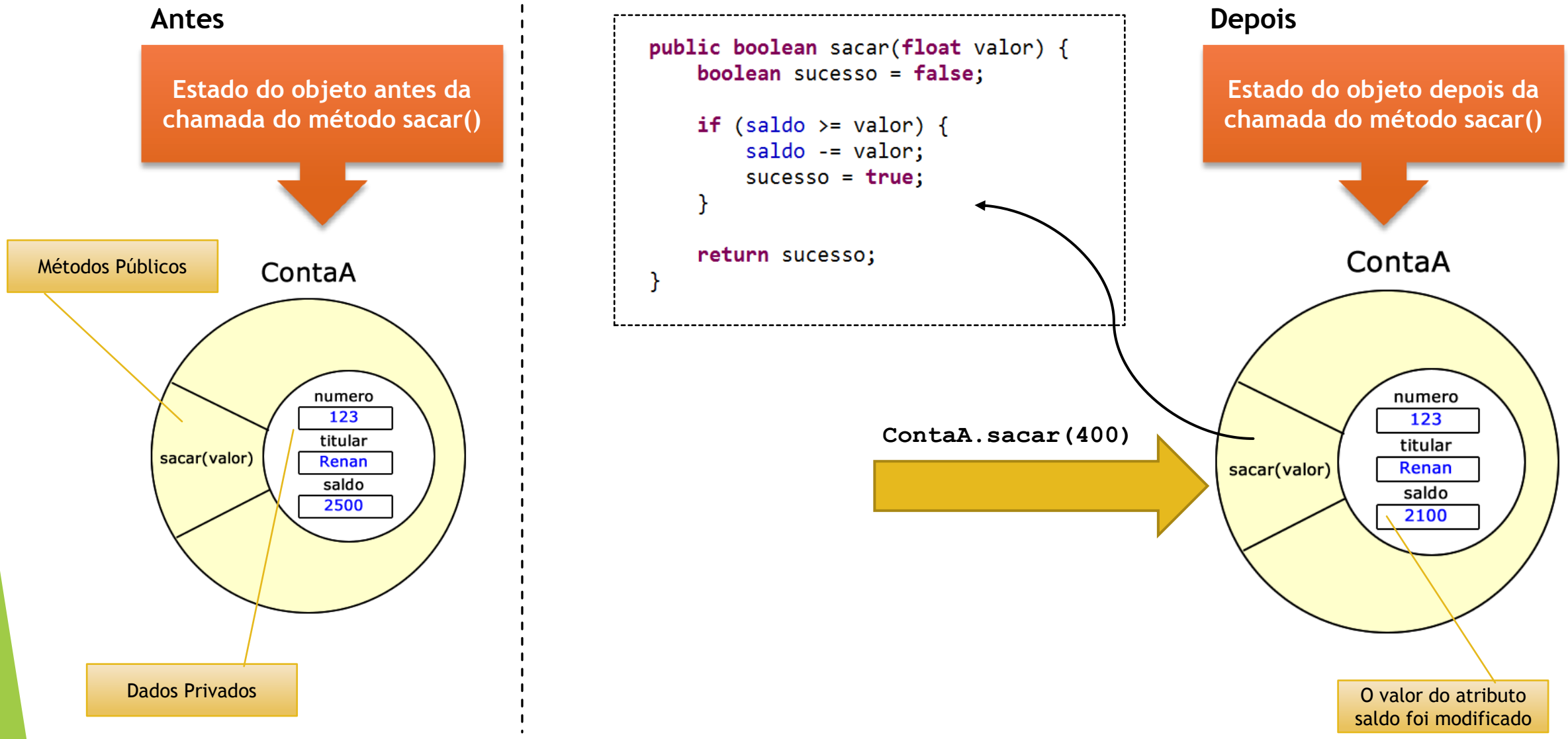
```
public boolean depositar(float valor) {
    boolean sucesso = false;

    if (valor >= 0) {
        saldo += valor;
        sucesso = true;
    }

    return sucesso;
}
```

```
public void imprimir(){
    System.out.println("------");
    System.out.println("Número da Conta: " + numero);
    System.out.println("Nome do Titular: " + titular);
    System.out.println("Saldo Atual: " + saldo);
}
```

Modificando o Estado do Objeto



Métodos Construtores

Método especial chamado automaticamente pelo ambiente de execução quando um objeto é criado.

Apesar de parecer, um construtor não é um método.

Conta
- numero : int - titular : String - saldo : float
+ Conta(numero : int, titular : String, saldo : float) + Conta(numero : int, titular : String) + sacar(valor : float) : boolean + depositar(valor : float) : boolean + imprimir() : void

Por padrão o Java já cria esse construtor sem parâmetros (construtor default) para todas as Classes.

- Na declaração do Objeto, o comando new é o responsável pela chamado do construtor;
- Pode ser utilizado para receber argumentos, podendo inicializar o estado do objeto durante a sua construção;
- Um construtor tem sempre o mesmo nome da classe a qual pertence;
- Uma classe pode possuir mais de um construtor.

Métodos Construtores com Assinaturas Diferentes

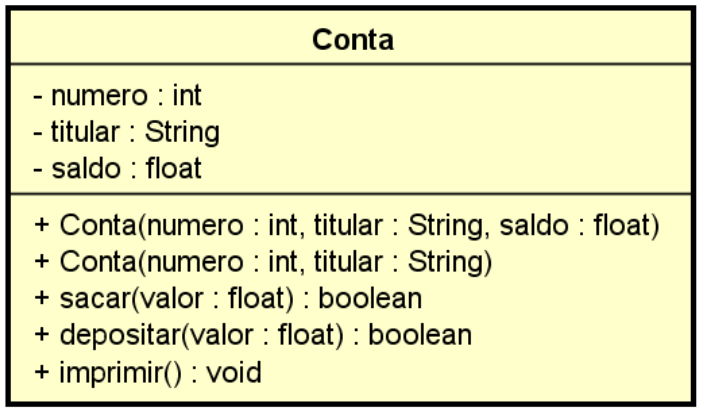
```java
public Conta(int numero, String titular, float saldo) {
    this.numero = numero;
    this.titular = titular;
    this.saldo = saldo;
}

public Conta(int numero, String titular) {
    this.numero = numero;
    this.titular = titular;
}
```

Este construtor não inicializar a variável de instância saldo. Como ela é do tipo primitivo float, será automaticamente inicializada com o valor 0 (zero).

As variáveis de instância são inicializadas automaticamente com o valor padrão do seu tipo.

Neste caso, this resolve a ambiguidade de nomes (parâmetros x atributos)



É usual criar construtores que recebem diversos argumentos para não obrigar o usuário de uma classe chamar diversos métodos do tipo "set".

Para a classe Conta, isto é desejável!

Código Completo em Java da Classe Conta

```java
public class Conta {

    private int numero;
    private String titular;
    private float saldo;

    public Conta(int numero, String titular, float saldo) {
        this.numero = numero;
        this.titular = titular;
        this.saldo = saldo;
    }

    public Conta(int numero, String titular) {
        this.numero = numero;
        this.titular = titular;
    }

    public boolean sacar(float valor) {
        boolean sucesso = false;

        if (saldo >= valor) {
            saldo -= valor;
            sucesso = true;
        }

        return sucesso;
    }

    public boolean depositar(float valor) {
        boolean sucesso = false;

        if (valor >= 0) {
            saldo += valor;
            sucesso = true;
        }

        return sucesso;
    }

    public void imprimir(){
        System.out.println("------");
        System.out.println("Número da Conta: " + numero);
        System.out.println("Nome do Titular: " + titular);
        System.out.println("Saldo Atual: " + saldo);
    }

    public int getNumero() {
        return numero;
    }
    public String getTitular() {
        return titular;
    }
    public float getSaldo() {
        return saldo;
    }
}
```

Conta
- numero : int - titular : String - saldo : float
+ Conta(numero : int, titular : String, saldo : float) + Conta(numero : int, titular : String) + sacar(valor : float) : boolean + depositar(valor : float) : boolean + imprimir() : void

Implementando o Método Main

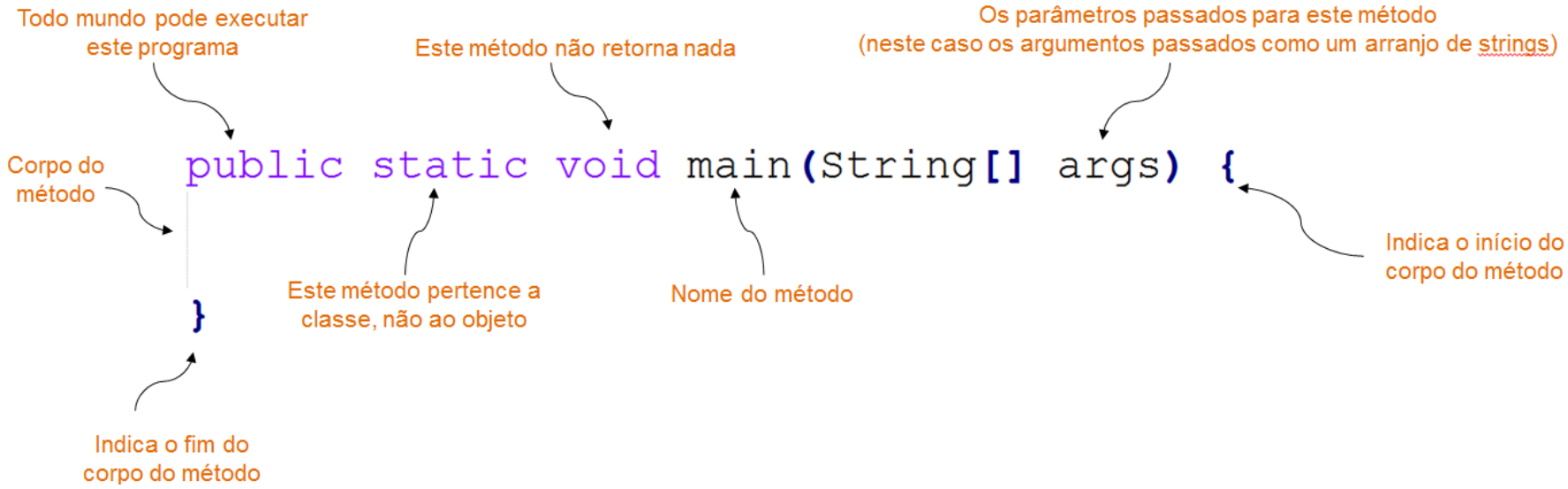
```
public static void main(String[] args) {

    Conta c = new Conta(123,"Renan Rodrigues", 2500);
    c.imprimir();

    c.depositar(100);
    c.sacar(800);
    c.sacar(5000);
    c.depositar(-100);
    c.depositar(300);
    c.imprimir();

}
```

O ponto de partida de todo aplicativo Java.



Testando a Classe Conta

```
public static void main(String[] args) {

    Conta c = new Conta(123,"Renan Rodrigues", 2500);
    c.imprimir();

    c.depositar(100);
    c.sacar(800);
    c.sacar(5000);
    c.depositar(-100);
    c.depositar(300);
    c.imprimir();

}
```

Saída do Programa

```
------
Número da Conta: 123
Nome do Titular: Renan Rodrigues
Saldo Atual: 2500.0
------
Número da Conta: 123
Nome do Titular: Renan Rodrigues
Saldo Atual: 2100.0
```

Métodos (e atributos) são acessados através do operador "." (ponto), precedido pelo nome do objeto (variável) e seguido do nome do método;

O operador new:

- Aloca memória para o novo objeto (a quantidade necessária é obtida a partir da definição da classe);
- Chama o construtor da classe para inicializar o estado do novo objeto;
- Retorna uma referência (um endereço de memória para o objeto recém criado);