Aula 3

Agora sim é novidade

O que iremos aprender:

- Orientação à objeto básica;
- O que é uma classe:
  - Acessando um objeto de uma classe;
  - Métodos de uma classe;
  - Atributos de uma classe;

Classes - Conceitos

- As classes são projetos de um objeto (É preciso utilizar abstração):
  - Atributos(Propriedades);
  - Métodos (Comportamentos);

= **Classe**

= **Objeto**

Classes - Conceitos

- Uma classe geralmente representa um substantivo;
  - Uma pessoa, um lugar ou algo que seja “abstrato”.
- Características das classes:
  - Toda classe possui um nome;
  - Possuem visibilidade, exemplo: public, private, protected;
  - Possuem membros como: Características e Ações;
  - Para criar uma classe basta declarar a visibilidade + digitar a palavra reservada class + NomeDaClasse + abrir e fechar chaves { }.

Classes - Atributos

Atributos:

- Os atributos são as propriedades de um objeto;
- Conhecidos como variáveis ou campos;
- Definem o estado de um objeto;
- Esses valores podem sofrer alterações;

Classes - Métodos

Métodos:

- São ações ou procedimentos;
- Podem interagir e se comunicar com outros objetos;
- A execução dessas ações se dá através de mensagens;
- Solicita ao objeto que seja executada uma rotina;
- Boa prática: sempre usar verbos para os nomes dos métodos.

Modelando um sistema bancário

Orientação a Objetos : Criando um sistema bancário:

- Vamos supor que estamos criando um sistema para um banco;
- Inicialmente vamos modelar uma das classes mais importantes para nosso sistema que é a conta dos clientes;

Algumas perguntas sobre o modelo de nosso sistema :

- O que toda conta tem e é importante para nós?
- O que toda conta faz que podemos modelar inicialmente?
- Quais são as operações possíveis envolvendo uma conta bancária?

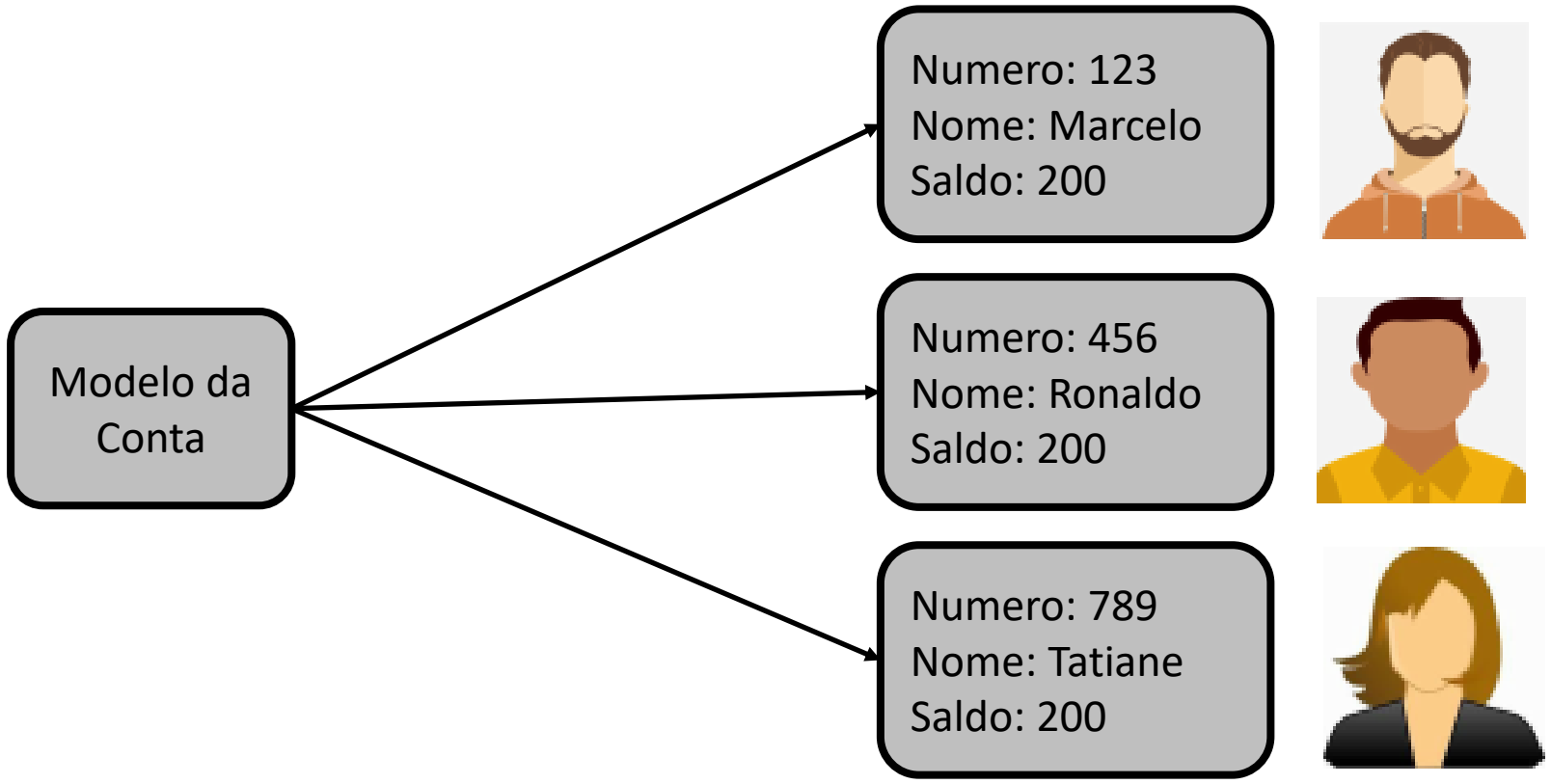
Modelando um sistema bancário

- O que toda conta tem e é importante para nós?
  - Número da conta;
  - Nome do titular da conta;
  - Saldo;

- O que toda conta faz?
  - Saque (de uma quantidade X);
  - Depósito (de uma quantidade X);
  - Imprime o nome do titular da conta;
  - Imprime o saldo atual;
  - Transfere quantias de uma conta para outra;

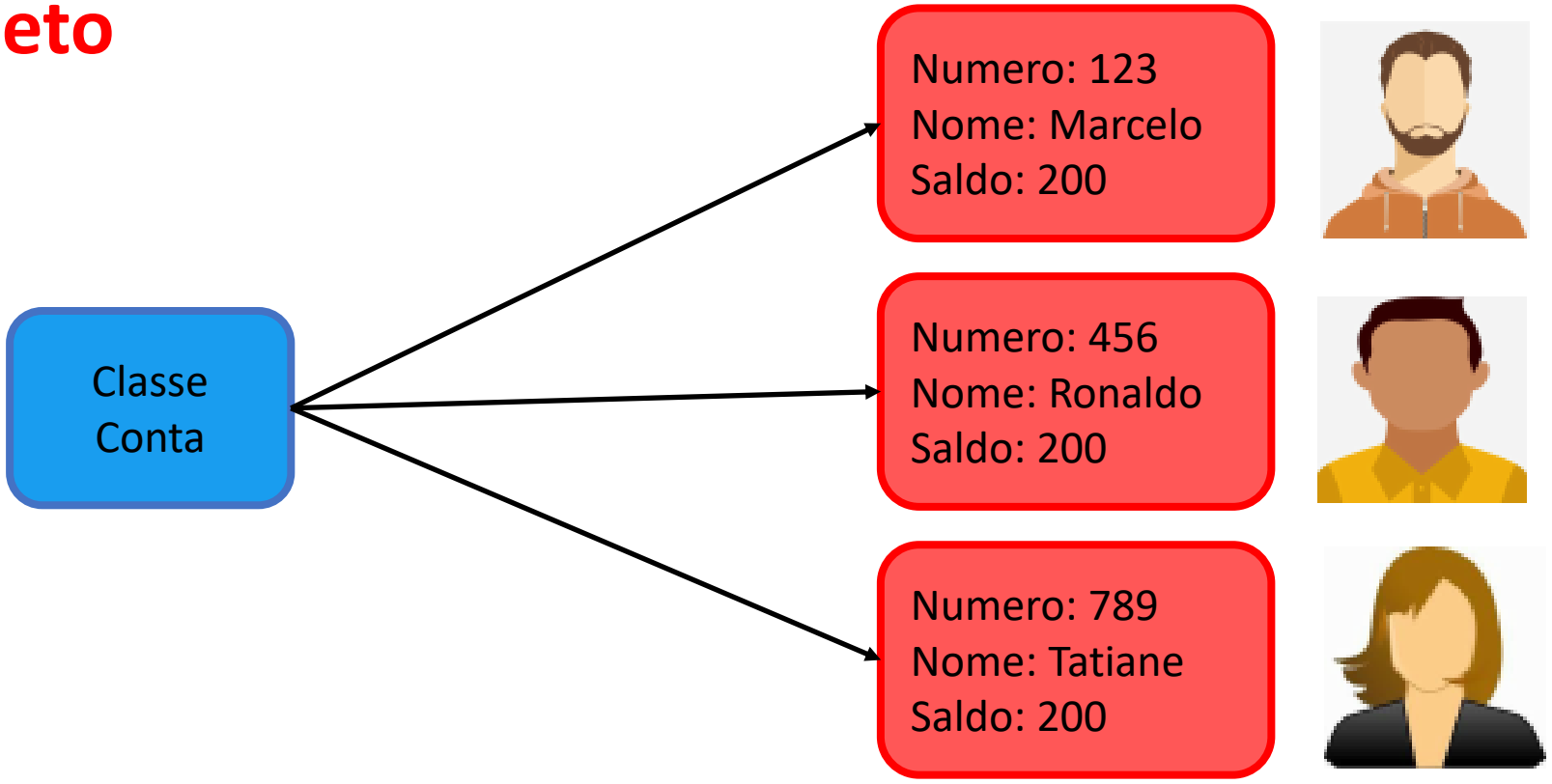
Modelando um sistema bancário

Com esse modelo podemos abrir várias contas diferentes



Modelando um sistema bancário

O modelo da conta, é o que chamamos de **Classe** e o que podemos construir partir desse modelo, damos o nome de **Objeto**



Implementando as classes - Atributos

Vamos começar apenas com o que uma Conta tem, e não o que ela faz. Transcrevendo o modelo anterior para um código em Java, temos a seguinte estrutura :

```
public class Conta {
   // o que uma conta tem
   int numero;
   String titular;
   double saldo;
   }
```

Construindo o objeto

Se quisermos acessar a classe que acabamos de criar em nosso código principal, precisamos fazer :

```
public class Principal {
   public static void main(String[] args) {
      Conta minhaConta = new Conta();
      }
}
```

Acessando um objeto

Através da variável minhaConta, podemos acessar o objeto recém criado para alterar seu titular, seu saldo, etc.

```
public class Principal {
   public static void main(String[] args) {
      Conta minhaConta = new Conta();
      minhaConta.titular = “Pedro”;
      minhaConta.saldo = 1000.0;
      minhaConta.numero = 1;
      System.out.println(“Saldo atual: ” +
      minhaConta.saldo);
      }
}
```

Métodos da classe - Características

Dentro da classe também declaramos quais as ações que ela realiza:

```
public class Conta {
   double saldo;
   // … outros atributos da classe …
   public void sacar(double valor) {
      double novoSaldo = this.saldo - valor;
      this.saldo = novoSaldo;
      }
}
```

Métodos da classe - Características

Podemos retornar ao cliente se ele pode fazer o saque ou não

```
public class Conta {
    // … outros métodos e atributos da classe …
    public boolean sacar(double valor) {
        if (this.saldo < valor) {
            return false;
        }
        else{
            double novoSaldo = this.saldo - valor;
            this.saldo = novoSaldo;
            return true;
        }
    }
}
```

Exercícios

Implemente o método transfere():

- Escreva o método transfere() que envia uma parte do saldo de uma conta X para uma conta Y;
- Requisitos : O método transfere() deve ser um método da classe Conta, não podendo ser um método externo, tem que receber 2 objetos do tipo Conta e realizar a transferência;
- Validação : Verifique se a transferência foi feita imprimindo os novos valores de saldo das contas X e Y na tela.

O método transfere()

Pode ser natural que em um primeiro momento, pensamos em implementar o método transfere da seguinte forma:

```
void transfere ( Conta origem, Conta destino, double valor ) {
    if (origem.saldo >= valor ) {
        origem.saldo = origem.saldo - valor;
        destino.saldo = destino.saldo + valor;
    }
    else {
        System.out.println(“Não há saldo suficiente para transferência”);
    }
}
```

O método transfere()

O problema deste tipo de implementação é que tranfere() deveria ser uma operação da própria classe Conta, então ela não deveria receber a classe remetente:

```
void transfere ( Conta origem, Conta destino, double valor ) {
    if (origem.saldo >= valor ) {
        origem.saldo = origem.saldo - valor;
        destino.saldo = destino.saldo + valor;
    }
    else {
        System.out.println(“Não há saldo suficiente para transferência”);
    }
}
```

Solução do exercício

```
boolean transfere(Conta destino, double valor) {
    boolean retirou = this.sacar(valor);
    if (retirou == false) {
        // não deu pra sacar!
        return false;
    }
    else {
        destino.depositar(valor);
        return true;
    }
}
```

Recap.

Já aprendemos:

- O que é Java
- Eclipse IDE
- Nosso primeiro código em Java : “Olá Mundo!”
- Variáveis primitivas
- Controle de fluxo
- Laços de repetição
- Orientação a objetos básica

Olhando para frente

O que iremos aprender agora:

- Modificadores de Acesso e Atributos de Classe:
  - Controlando o Acesso;
  - Encapsulamento;
  - Getters e Setters;
- Construtores:
  - Necessidade de um Construtor;
- Atributo de Classe:
  - O atributo static;

Zeraram a minha conta! E agora?

Imagine a seguinte situação :

```
public class Principal {
   public static void main(String[] args) {
      // declaração da conta e inicialização omitida
      ...
      minhaConta.deposita(1000);
      minhaConta.deposita(3000);
      minhaConta.saldo = 0;
      System.out.println("Saldo atual: " + minhaConta.saldo);
   }
}
```

Como impedir isso?

Adicionando o modificador de acesso private antes da declaração de saldo, no modelo da classe :

```
public class Conta {
    private double saldo;
    // ...
}
```

Modificadores de acesso

Adicionando o modificador de acesso private, a variável saldo só poderá ser modificada dentro do próprio objeto :

```
public class Conta {
    private double saldo;
            // ...
    }
    public class TestaAcessoDireto {
        public static void main(String[] args) {
        Conta minhaConta = new Conta();
        minhaConta.saldo = 1000;
    }
}
```

Exception in thread "main" java.lang.Error:
Unresolved compilation problem:
The field Conta.saldo is not visible at
TestaSaldo.main(TestaAcessoDireto.java:6)

Exercícios

- Implemente o método deposita() e faça o devido controle de acesso aos demais atributos da classe Conta;

- Devido ao controle de acesso restringir a visualização explícita da variável saldo, crie um método simples que retorne o valor do saldo para o usuário.

Exercícios

```
public class Conta {
    private String titular;
    private int numero;
    private double saldo;

    public void deposita(double valor){
        this.saldo = this.saldo + valor;
    }
    public double verificaSaldo(){
        return this.saldo;
    }
}
```

Getters e Setters

- Ferramentas para o controle de acesso de variáveis de um objeto;
- Comum na modelagem da classe, criar seus métodos de acessos;
- Esses métodos são denominados getters e setters;
- Getters (método que retorna o valor da variável);
- Setters (método que altera o valor da variável);
- A convenção é colocar get ou set + nome da variável, como nome do método;

```
public setSaldo(double saldo){
    this.saldo = saldo;
}
```

```
public double getSaldo(){
return this.saldo;
}
```

Getters e Setters

- Entretanto, é uma má prática ao criar uma classe, criar getters e setters para todos seus atributos. Só deve criar um getter ou setter se houver real necessidade;
- Exemplo: Não precisamos fazer um setter para o saldo, pois queremos atualizá-lo apenas pelos métodos **Deposita** e **Transfere** e **Sacar**;

```
public double verificaSaldo(){
    return this.saldo;
}
```

```
public deposita(double valor)
    this.saldo = this.saldo + valor;
}
```

Construtores

Construtores:

- O construtor de um objeto é um método especial;
- Inicializa seus atributos toda vez que é instanciado (inicializado);
- Quando é digitada a palavra reservada **new**, o objeto é construído;
- Existe o construtor default e o parametrizado para inicializar o objeto;
- O nome do construtor é sempre o mesmo nome da classe;

Construtores

```
public class Conta {
   // atributos da classe omitidos
   ...
   // construtor
   public Conta() {
         System.out.println("Construindo uma conta.");
   }
   // continuação da classe
}
```

Construtores

O interessante é que podemos passar um argumento para o construtor, assim podemos inicializar a classe com algum tipo de informação:

```
public class Conta {
   // atributos da classe omitidos
   // construtor
   public Conta(String titular) {
         this.titular = titular;
   }
   // ..
}
```

Construtores

Para que servem:

- Construtores podem servir para forçar objetos a inicializarem alguns valores em sua criação;
- Ou podem ser utilizados também para determinar ações que devem ser executadas quando um objeto é criado (como o caso do println no primeiro exemplo, ou forçar um depósito inicial);

Além disso:

- Uma classe pode ter mais de um construtor, e no momento do new, o construtor apropriado será escolhido;
- Os construtores serão diferenciados pela quantidade de parâmetros como veremos.

Classes – Sobrecarga de métodos

Sobrecarga:

- Fazer uso da declaração de métodos com o mesmo nome;
- Desde os métodos tenham parâmetros diferentes;

```
public int soma(int n1, int n2){}

public int soma(int n1, int n2, int n3){}

public double soma(double n1, int n2, Conta destino){}
```

Exercícios

- Utilize o exemplo a ContaCorrente
- Crie um construtor default com uma mensagem indicando que o objeto foi criado;
- Implemente a sobrecarga do construtores para preencher os atributos da conta: numero e nome;
- Implemente a sobrecarga do construtores para preencher os atributos da conta: numero, nome, sobrenome e saldo;
- Execute através da classe principal os diversos construtores e experimente os diferentes resultados;
- Utilizem modificadores de acesso para todos ao atributos da classe ContaCorrente;