POO - PROGRAMAÇÃO ORIENTADA A OBJETOS

Interfaces, herança múltipla e classes seladas
Object a superclasse de toda classe Java

Rodrigo R Silva

Instituto Federal de Educação, Ciência e Tecnologia Sul-Rio-Grandense
Campus Bagé

TADS
Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

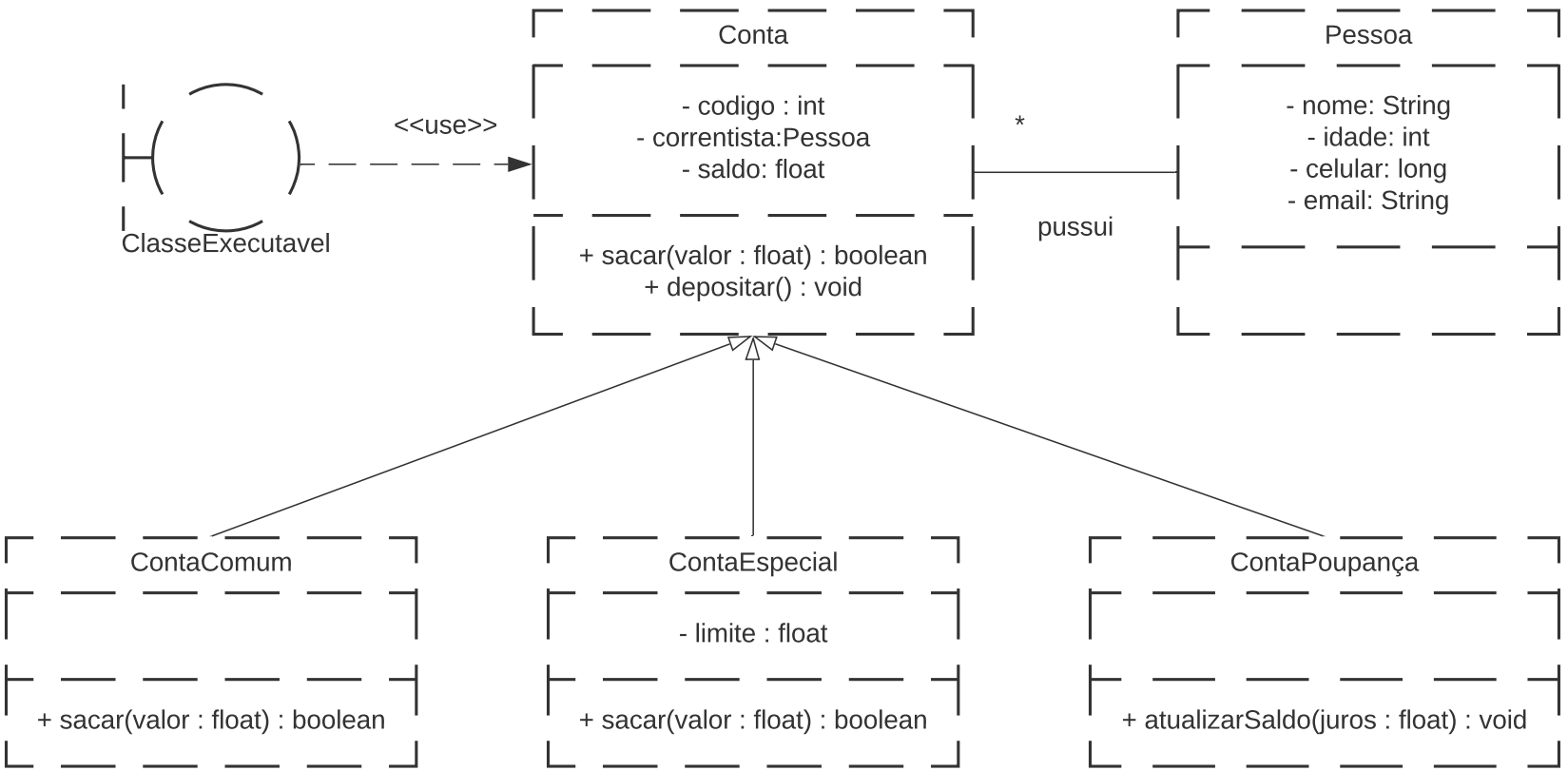
Nesta Aula Veremos...

Introdução

Contextualizando...

- Tomando como exemplo a aplicação bancária que estamos utilizando como estudo de caso.
- Notem a estrutura de hierarquia de classes.
- Três classes especializadas a partir da classe Conta, são elas:
- ContaPoupança
- ContaComum
- ContaEspecial

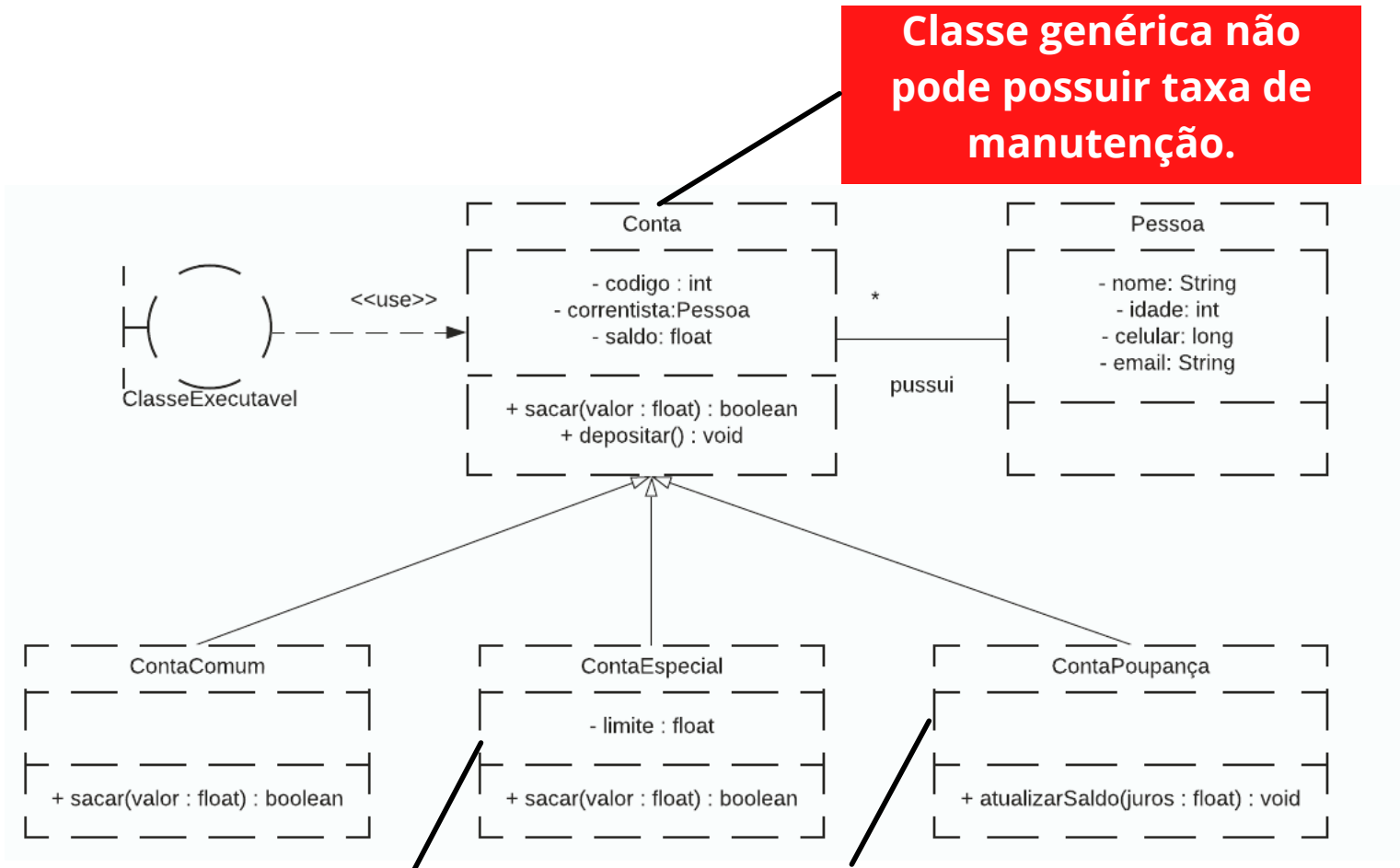
Modelo de Classes



Considerações sobre o modelo

- Notem que em uma aplicação bancária algumas especializações possuem comportamentos que não são comuns a todas especializações do modelo.
- Ex: nem todas as contas do banco possuem uma taxa de manutenção.
- Conta salário possui, assim como, conta especial e conta investimento.
- Já conta poupança não possui.
- Esse comportamento não pode ser definido como um método na classe mais genérica do modelo.

Representação gráfica

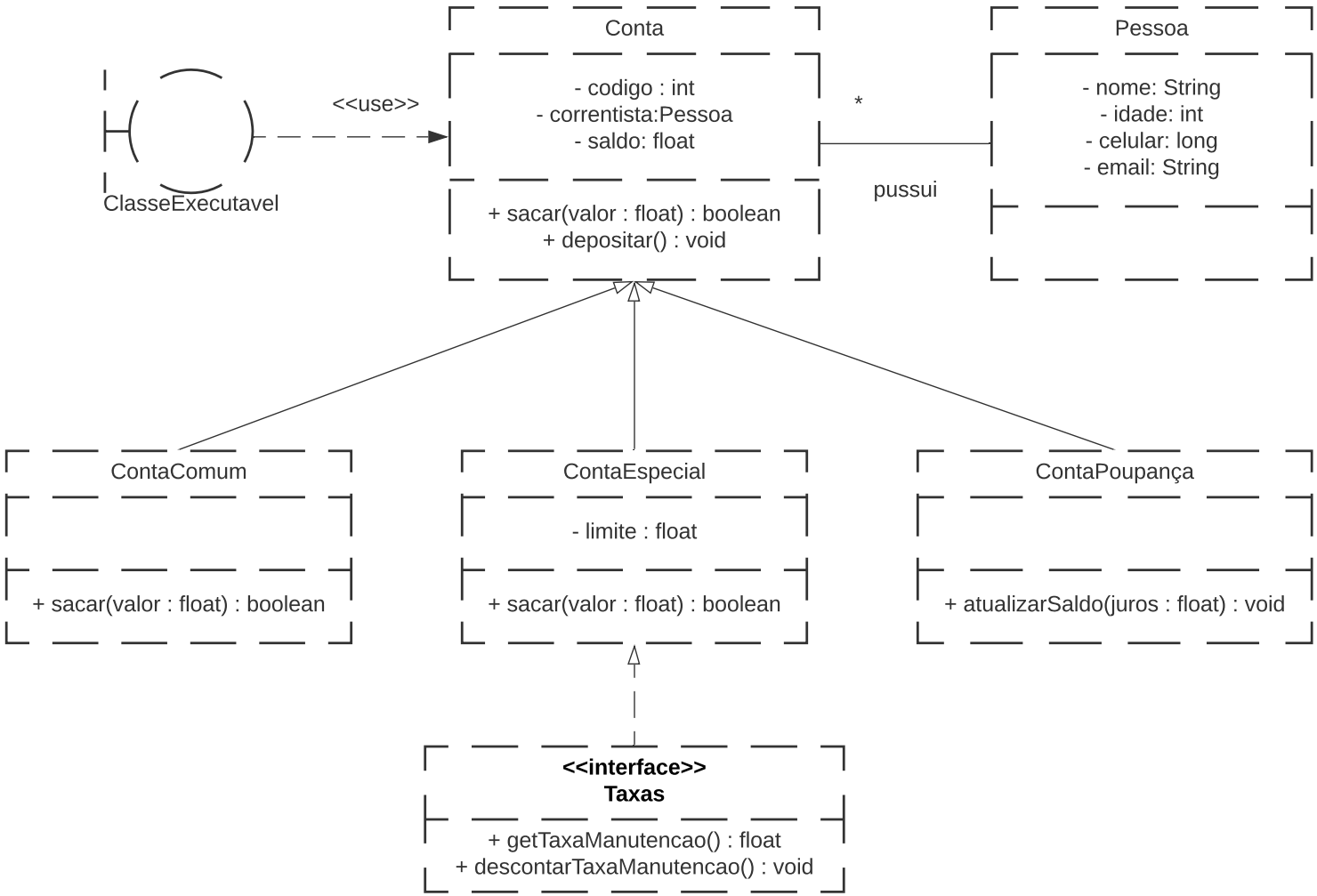


Interface

O conceito de Interface

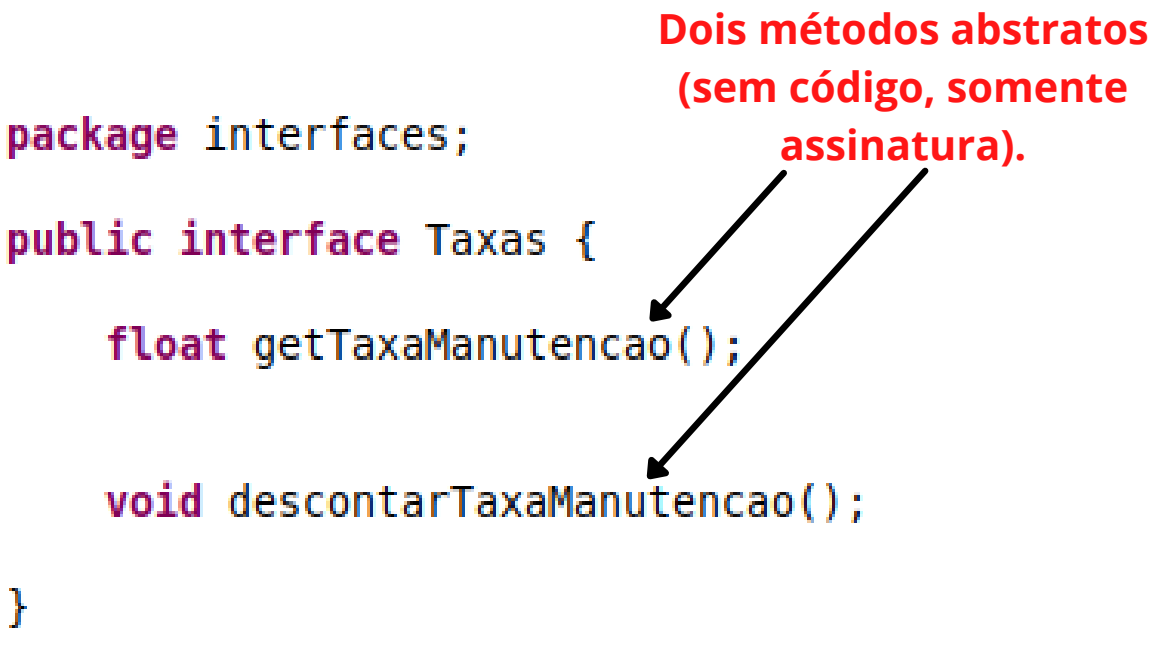
- Uma estrutura que define um conjunto de comportamentos que podem ser implementados por determinadas classes.
- Somente as classes do modelo que precisam devem **implementar a interface**.
- Ex: desconto de taxa de manutenção da conta.

Modelo de Classes



Definindo uma interface em Java

- A definição de uma Interface é semelhante a definição de uma classe.
- Usamos a palavra reservada interface na declaração da mesma.
- Vejamos um exemplo:



Herança múltipla

- O conceito de interface permite a implementação de herança múltipla em Java.
- Visto que uma classe pode herdar de uma outra classe, mas de várias outras interfaces.
- Não há limite de implementação de interfaces por classes Java.

Classe implementando interface

```
public class ContaEspecial extends Conta implements Taxas{
    private float limite;
    public ContaEspecial(){ }

    @Override
    public float getTaxaManutencao(){
        return 15.00f;
    }

    @Override
    public void descontarTaxaManutencao(){
        this.setSaldo(this.getSaldo() - this.getTaxaManutencao());
    }
}
```

TADS Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Sealed

Classes e Interfaces seladas

- O lançamento do Java SE 15 em setembro de 2020 apresenta as classes seladas (Sealed Class), JEP 360 como sendo um recurso novo.
- Uma classe selada é uma classe ou interface que restringe quais outras classes ou interfaces podem estendê-la.
- As classes seladas também são úteis para criar hierarquias seguras, desacoplando a acessibilidade da extensibilidade.
- As classes seladas trabalham junto com registros e correspondência de padrões para oferecer suporte a uma forma de programação centrada nos dados.

Classes e Interfaces seladas

- Uma classe ou interface pode ser declarada como **sealed**, o que significa que apenas um conjunto específico de classes ou interfaces pode estendê-la diretamente.
- A lista **permits** significa que apenas as classes ou interfaces ali declaradas podem implementar a classe ou interface.
- A selagem pode ser considerada uma generalização do **final**.

Classe implementando sealed

selead tornar a classe Conta e selada.

permits informa a lista de classes que podem implementar ou extender a classe Conta.

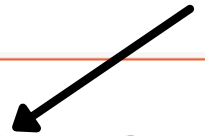
```
package classes.java;

public abstract sealed class Conta permits ContaEspecial, ContaComum{

    protected int codigo;
    protected Pessoa correntista;
    protected float saldo;
    protected static int numeroContas;

    public final int SACAR = 1;
    public final int DEPOSITAR = 0;

    public Conta() {
        incrementarContas();
```

TADS Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Erro na classe não permitida

Classe ContaPoupanca não pode mais extender a classe Conta.



```java
package classes.java;

public class ContaPoupanca extends Conta{

    public ContaPoupanca() {
        super();
    }

    public ContaPoupanca(int numero, Pessoa correntista, float saldo) {
        super(numero, correntista, saldo);
    }

    public void atualizarSaldo(float juros) {
        this.setSaldo(this.getSaldo() + this.getSaldo() * juros / 100);
    }
     @Override
        public boolean sacar(float valor) {
            if(this.saldo - valor >= 0){
                this.saldo -= valor;
                return true;
            }
            return false;
        }
}
```

TADS Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Object

Object

- Java define uma classe que forma a base de todos os modelos de classes da linguagem. Desde classes da própria API Java, como classes definidas por programadores.
- Esta classe é denominada **Object** e é uma classe implícita. Portanto, não é necessário estender Object quando definimos uma nova classe.

Object como generalização

- Portanto, toda classe em Java é um subtipo de Object.
- Dessa forma, qualquer objeto de qualquer classe pode ser referenciado como Object.
- Vejamos um exemplo:
  ContaEspecial contaEspecial = new ContaEspecial();
  Object objetoGenerico = contaEspecial;

Coerção (Cast) de objetos Java

- Quando um objeto de uma determinada classe é referenciado como Object, devemos realizar uma **ação de coerção** para referenciarmos o objeto em sua classe original.
- Ex:
  ContaEspecial especial1 = new ContaEspecial();
  Object obj = especial1;
  ...
  ContaEspecial especial2 = (ContaEspecial) obj;

TADS Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Coerção (Cast) em primitivos

- Já em tipos de dados primitivos a coerção realiza uma **conversão de valores**.
- Ex:
  long valorQualquer = 9483762182727L;
  int outroValor = (int) valorQualquer;

Métodos da classe Object

- A classe object define alguns métodos. Podemos reescrever seus comportamentos em nossas classes.
- Alguns destes métodos são úteis e podemos implementar algumas funcionalidades em nossas classes. São eles:
- toString();
- equals();
- hashCode();

O método toString()

- O método toString() retorna uma representação do objeto na forma textual.
- Geralmente, o valor de um ou mais atributos definidos na classe.
- Ex: vamos reescrever o comportamento de toString() na classe Pessoa.

```
@Override
public String toString() {
    return "Correntista: nome = " + nome + ", celular = " + celular;
}
```

Referenciando toString()

- Quando nossas aplicações referenciam a instância de uma classe, será chamado o método toString().
- Isso pode ser utilizado em instruções que mostram um objeto na tela.
- Ex: vamos mostrar na tela o objeto do tipo Pessoa.

```
Pessoa correntista = new Pessoa("Ciclano",33,999887766,"ciclano@gmail.com");
System.out.println(correntista);
```

Saída padrão de toString()

- Se uma classe não reescrever o comportamento do método toString(), será implementado o comportamento padrão definido na classe Object.
- Esse comportamento mostra o nome da classe, o símbolo @ e um número hexadecimal.
- Ex:
  classes.java.Pessoa@7e774085

O método equals()

- Este método deve ser utilizado para implementarmos em nossas classes um comportamento de comparação.
- Tal comportamento visa verificar se dois objetos da mesma classe são iguais.
- Isso é muito útil nas nossas aplicações, pois podemos controlar se o usuário está tentando criar dois objetos iguais.
- O critério de igualdade é definido pelo programador. Geralmente, pela combinação dos valores de atributos.

Exemplo de equals()

- Vejamos como implementar o comportamento do método equals() na classe Pessoa.
- Vamos comparar dois objetos do tipo Pessoa através do atributo número do CPF. Visto que duas pessoa não podem possuir um mesmo número.

```
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Pessoa other = (Pessoa) obj;
    return celular == other.celular && cpf == other.cpf && Objects.equals(email, other.email)
            && idade == other.idade && Objects.equals(nome, other.nome);
}
```

TADS Tecnologia em Análise e Desenvolvimento de Sistemas

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

Comparando dois objetos

- Veja a comparação de dois objetos.

```
Pessoa correntista = new Pessoa("Ciclano",22233344455l,33,999887766,"ciclano@gmail.com");
Pessoa correntista1 = new Pessoa("Beltrano",22233344455l,22,999112233,"beltrano@gmail.com");

if(correntista.equals(correntista1)) {
    System.out.println("Pessoas iguais =(");
}else {
    System.out.println("Pessoas diferentes");
    }

System.out.println("-----------------------------------");
System.out.println(correntista);
System.out.println("-----------------------------------");
System.out.println(correntista1);
```

O método hascode()

- O **hashCode** é uma ferramenta da JVM usada para montar a tabela de hash de modo correto.
- Tabela Hash [também conhecida como Tabela de Dispersão ou Tabela de Espalhamento] é uma tabela onde as informações são armazenadas conforme um **numero hash** calculado com base nas propriedades da informação. Isso permite que seja muito rápido recuperar uma informação na tabela.

Tempo de busca

- Imagine uma tabela com as informações de todos os pacientes de um hospital. Se fosse buscar um paciente em especial iria demorar um tempo (O(n) numa busca linear ou O(log N) para buscas binarias) o que pode ser extremamente ruim em uma situação real onde existe um volume de dados gigantescos. Usando uma tabela hash a busca reduz seu tempo de busca (O(1)) para qualquer situação, bastando apenas o calculo do valor hash na entrada e na saída da informação.

Exemplo de implementação do hascode

```java
package classe.executavel;

import classes.java.Telefone;

public class ClasseExecutavelTelefone {

    public static void main(String[] args) {


        Telefone tel1 = new Telefone();
        tel1.setMarca("Motorola");
        tel1.setModelo("Razor");
        tel1.setImei(12345);

        Telefone tel2 = new Telefone();
        tel2.setMarca("Motorola");
        tel2.setModelo("Razor");
        tel2.setImei(123456);

        System.out.println(tel1);
        System.out.println(tel1.hashCode());
        System.out.println(tel2);
        System.out.println(tel2.hashCode());

        System.out.println(tel1.equals(tel2));
```

```
<terminated> ClasseExecutavelTe
classes.java.Telefone@3058
12376
classes.java.Telefone@1e25f
123487
false
```

hash gerado conforme sobreescrita do método.

OBRIGADO!