

Aula 12 - Polimorfismo

1. Polimorfismo

Polimorfismo é o conceito de implementar métodos de diferentes formas.

Temos 2 tipos de Polimorfismo, são eles:

1.1. Override

Utilizado sempre com herança, onde temos um método implementado na super classe e podemos modificar seu comportamento na sub classe.

Exemplo:

Conta.java (Super Classe)

```
package Aula10;

public class Conta {
    private String agencia;
    private String numero;
    private double saldo;

    public String getAgencia() {
        return agencia;
    }

    public void setAgencia(String agencia) {
        this.agencia = agencia;
    }

    public String getNumero() {
        return numero;
    }

    public void setNumero(String numero) {
        this.numero = numero;
    }

    public double getSaldo() {
        return saldo;
    }

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

```

    }

    protected String exibirMensagem() {
        return "Bem vindo ao Internet Banking do Prof. Joseffe!";
    }

    Conta() {

    }

    Conta(String agencia, String numero, double salario) {
        this.agencia = agencia;
        this.numero = numero;
        this.saldo = salario;
    }

    public void Depositar(double valor) {
        this.saldo += valor;
    }
}

```

ContaCorrente.java (Sub Classe)

```

package Aula10;

public class ContaCorrente extends Conta{
    private double chequeEspecial;

    public double getChequeEspecial() {
        return chequeEspecial;
    }

    public void setChequeEspecial(double chequeEspecial) {
        this.chequeEspecial = chequeEspecial;
    }

    public ContaCorrente() {
    }

    public ContaCorrente(String agencia, String numero, double salario,
double chequeEspecial) {
        super(agencia, numero, salario);
    }
}

```

```

        this.chequeEspecial = chequeEspecial;
    }

    public void Depositar(double valor) {
        super.Depositar(valor);

        valor = valor - 0.10;
        this.setSaldo(valor);
    }
}

```

ContaPoupanca.java (Sub Classe)

```

package Aula10;

public class ContaPoupanca extends Conta{
    private double rentabilidade;

    public double getRentabilidade() {
        return rentabilidade;
    }

    public void setRentabilidade(double rentabilidade) {
        this.rentabilidade = rentabilidade;
    }

    public ContaPoupanca() {
    }

    public ContaPoupanca(String agencia, String numero, double salario,
double rentabilidade) {
        super(agencia, numero, salario);

        this.rentabilidade = rentabilidade;
    }

    public void Depositar(double valor) {
        super.Depositar(valor);

        valor = valor + 0.50;
        this.setSaldo(valor);
    }
}

```

Nesse exemplo, foi criado o método Depositar. O método Depositar foi criado na super classe e também nas 2 sub classes. Para que aconteça o override, o nome do método e sua assinatura devem ser idênticas.

Entretanto, na sua implementação (dentro do método) é que está a diferença. Perceba que nas sub classes chamamos a implementação do método da super classe utilizando a palavra “super” e logo após escrevemos a nossa implementação nesse método, alterando assim o seu comportamento original.

Vejamos o resultado no programa abaixo:

Programa

```
package Aula10;

import java.util.Scanner;

public class Programa {

    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);

        ContaCorrente cc = new ContaCorrente();

        cc.setAgencia("0001");
        cc.setNumero("14769");
        cc.Depositar(100);

        cc.setChequeEspecial(500);

        System.out.printf("Conta Corrente: Ag: %s, Num: %s, Saldo: %.2f, Chq Esp: %.2f", cc.getAgencia(), cc.getNumero(), cc.getSaldo(), cc.getChequeEspecial() );

        ContaPoupanca cp = new ContaPoupanca();

        cp.setAgencia("0002");
        cp.setNumero("32456");
        cp.Depositar(100);

        cp.setRentabilidade(2);
```

```
        System.out.printf("\n\nConta Poupança: Ag: %s, Num: %s,
Saldo: %.2f, Rent: %.2f", cp.getAgencia(), cp.getNumero(),
cp.getSaldo(), cp.getRentabilidade() );
    }
}
```

1.2. Overload

Utilizado sem herança, permite escrever 2 ou mais métodos com o mesmo nome, porém com assinaturas diferentes (com parâmetros diferentes).

Exemplo:

```
package Aula10;

public class Conta {
    private String agencia;
    private String numero;
    private double saldo;

    public String getAgencia() {
        return agencia;
    }

    public void setAgencia(String agencia) {
        this.agencia = agencia;
    }

    public String getNumero() {
        return numero;
    }

    public void setNumero(String numero) {
        this.numero = numero;
    }

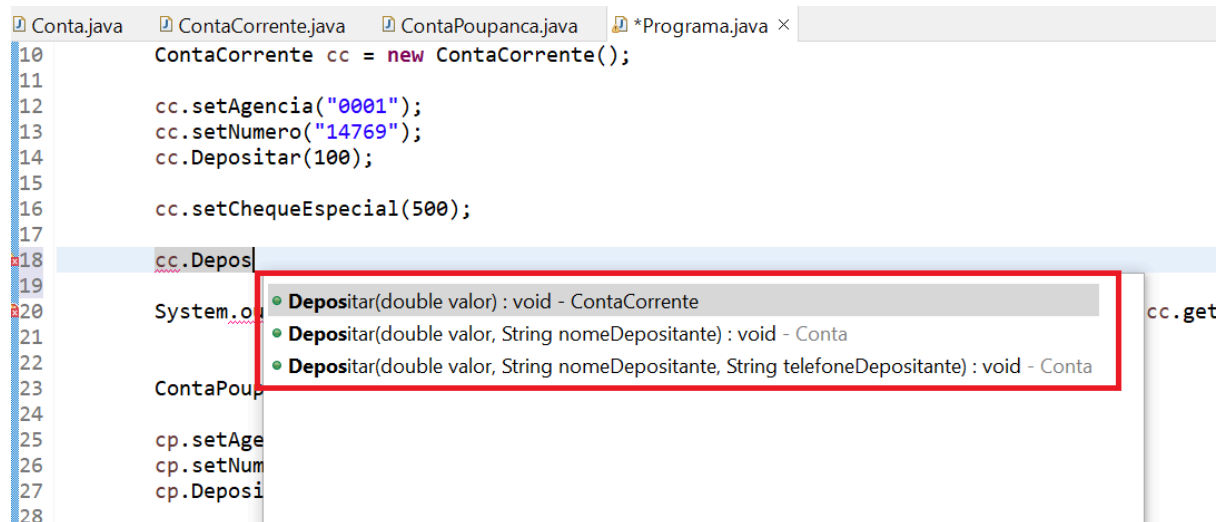
    public double getSaldo() {
        return saldo;
    }

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

```
protected String exibirMensagem() {  
    return "Bem vindo ao Internet Banking do Prof. Joseffe!";  
}  
  
Conta() {  
  
}  
  
Conta(String agencia, String numero, double salario) {  
    this.agencia = agencia;  
    this.numero = numero;  
    this.saldo = salario;  
}  
  
public void Depositar(double valor) {  
    this.saldo += valor;  
}  
  
public void Depositar(double valor, String nomeDepositante) {  
    this.saldo += valor;  
}  
  
public void Depositar(double valor, String nomeDepositante, String  
telefoneDepositante) {  
    this.saldo += valor;  
}  
}
```

Perceba que acima temos 3 métodos Depositar. Porém, todos com assinaturas diferentes.

Veja que na utilização desse método em nosso programa temos o seguinte resultado:



The screenshot shows an IDE with four tabs: Conta.java, ContaCorrente.java, ContaPoupanca.java, and *Programa.java. The main editor displays the following Java code:

```
10 ContaCorrente cc = new ContaCorrente();
11
12 cc.setAgencia("0001");
13 cc.setNumero("14769");
14 cc.Depositar(100);
15
16 cc.setChequeEspecial(500);
17
18 cc.Depos
19
20 System.out
21
22
23 ContaPoup
24
25 cp.setAge
26 cp.setNum
27 cp.Deposi
28
```

A red rectangle highlights the autocomplete popup for the `cc.Depos` call. The popup lists three methods:

- `Depositar(double valor) : void - ContaCorrente`
- `Depositar(double valor, String nomeDepositante) : void - Conta`
- `Depositar(double valor, String nomeDepositante, String telefoneDepositante) : void - Conta`

To the right of the popup, the text `cc.get` is visible.

Exercícios:

Vamos fazer um sistema de cadastro de clientes e sua respectiva conta bancária. O sistema deve permitir, inclusão de cliente, depósito na conta, saque na conta, exclusão de cliente e consulta de cliente e extrato.

O cliente deve escolher entre conta corrente e conta poupança no momento de se cadastrar. Utilize HashMap para armazenar os clientes e contas. Utilize Herança e Polimorfismo nos métodos da classe das contas.

Correção:

https://drive.google.com/drive/u/1/folders/1N3EKqdT_FdDEhXIOq__BZ2ESldhT3dIW