

# Implementação de Herança em Java

Exercício

**Vinicius Takeo Friedrich Kuwaki**

Universidade do Estado de Santa Catarina

# Seções

Exemplo

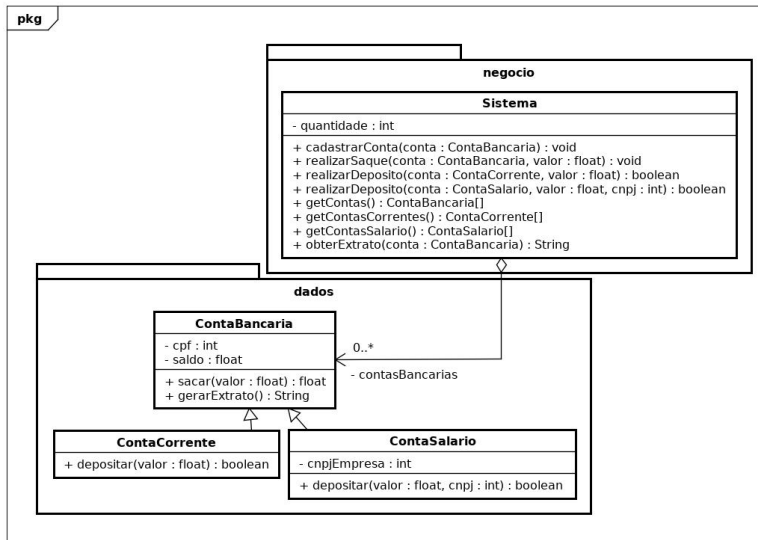
Resolução

Exercício

## Exemplo

O diagrama de classes a seguir representa um sistema bancário, onde existem contas bancárias de dois tipos: correntes e salário. Observe o diagrama:

# Exemplo - Diagrama



## Exemplo

- A classe ContaBancaria possui dois métodos: sacar() e gerarExtrato();
- O primeiro deles: sacar(), recebe um valor e subtrai do saldo;
- Já o segundo: gerarExtrato(), retorna uma String contendo o estado atual da conta, no caso, o saldo.
- Já as extensões dessa classe: ContaCorrente e ContaSalario, possuem métodos para realizar o depósito;
- Na ContaCorrente, basta passar o valor como parâmetro para realizar o depósito no método depositar();
- Já na ContaSalario é necessário passar o valor e o cnpj da empresa que está realizando o depósito.
- A classe ContaSalario também reescreve o método gerarExtrato(), adicionando o cnpj da empresa;

## Exemplo

Já a classe Sistema possui os seguintes métodos:

- `cadastrarConta()`: que recebe uma `ContaBancaria` (seja ela do tipo corrente ou do tipo salário) e adiciona ao array do tipo `ContaBancaria`;
- `realizarSaque()`: que dado uma conta e um valor, retira esse valor da conta utilizando o próprio método da conta;
- Dois métodos para realizar depósitos específicos para cada tipo de conta bancária;
- E métodos que retornam as contas bancárias do sistema:
  - `getContas()`: retorna todas as contas do array;
  - `getContasCorrentes()` e `getContasSalario()` realiza um filtro no array e retorna apenas o tipo de conta especificada no nome do método.

Também será criado uma classe Main para realizar a interface via console com o usuário.

# Seções

Exemplo

Resolução

Exercício

- Primeiro vamos criar as classes pertencentes ao pacote de dados;
- No caso as classes:
  - ContaBancaria (super classe);
  - ContaCorrente;
  - ContaSalario;



# Classe ContaBancaria

- Ela pertence ao pacote de dados, logo precisamos da declaração do package:

```
package dados;
```

- Ela possuirá dois atributos:
  - cpf: do tipo inteiro e **privado**, isto é, só a própria super classe poderá acessar;
  - saldo: do tipo float e **protected**, isto é, qualquer classe que a extender poderá acessar esse atributo.
- Os métodos getters e setters não serão apresentados aqui.

```
public class ContaBancaria {  
  
    private int cpf;  
    protected float saldo;  
  
    public ContaBancaria() {  
        this.saldo = 0;  
    }  
}
```

# Classe ContaBancaria

- O método **sacar()** recebe um valor e subtrai do saldo:

```
public float sacar(float valor) {  
    saldo -= valor;  
    return valor;  
}
```

- Já o método **gerarExtrato()** retorna o saldo:

```
public String gerarExtrato() {  
    return "Saldo dispon vel: R$" + this.saldo;  
}
```

- E o método **toString()** apenas retorna o cpf da conta:

```
public String toString() {  
    return "CPF: " + this.cpf;  
}
```

# Classe ContaCorrente

- Agora iremos implementar a classe ContaCorrente;
- Também pertencente ao pacote dados, ela não possui nenhum atributo além dos atributos da sua **superclasse**;
- Para estender uma superclasse é necessário utilizar a palavra reservada **extends**;
- No construtor dela é necessário invocar o método **super()**;
- Esse método deve ser sempre o primeiro a ser invocado dentro do construtor de classes estendidas!

```
package dados;  
  
public class ContaCorrente extends ContaBancaria {  
  
    public ContaCorrente() {  
        super();  
    }  
}
```

## Classe ContaCorrente

- Será também implementado o método **depositar()** que recebe um valor e soma ao atributo saldo, retornando true, para confirmar que a operação foi realizada com sucesso:

```
public boolean depositar(float valor) {  
    this.saldo += valor;  
    return true;  
}
```

- O método **gerarExtrato()** será sobrescrito, para exibir o tipo de conta:
- Utilizaremos o método gerarExtrato() da superclasse para sobreescreve-lo:

```
public String gerarExtrato() {  
    return "Conta Corrente: \n" + "CPF: " + this.getCpf() + "\n" + super.  
    gerarExtrato();  
}
```

- A mesma coisa é realizada no método toString():

```
public String toString() {  
    return "Conta Corrente: \n" + super.toString();  
}
```

## Classe ContaSalario

- Agora iremos implementar a última classe do pacote de dados;
- A classe ContaSalario possui um atributo `cpnjEmpresa`, que representa a empresa que pode depositar nessa conta;
- O construtor dessa classe também precisa invocar o método `super()`;
- Além de ser necessário novamente a palavra reservada `extends`:

```
package dados;  
  
public class ContaSalario extends ContaBancaria {  
    private int cpnjEmpresa;  
  
    public ContaSalario() {  
        super();  
    }  
}
```

## Classe ContaSalario

- O método de depositar dessa classe recebe dois parâmetros: o valor a ser depositado e o cnpj do depositante;
- O valor só é acrescentado a conta caso os cnpj's forem iguais;
- O método vai retornar true caso tenha sido depositado e false caso contrário

```
public boolean depositar(float valor, int cnpjEmpresa) {  
    if (cnpjEmpresa == this.cnpjEmpresa) {  
        this.saldo += valor;  
        return true;  
    }  
  
    return false;  
}
```

- Essa classe também sobreescreve o método gerar extrato, exibindo também o cnpj da empresa:

```
@Override  
public String gerarExtrato() {  
    return "Conta Salario: \n" + "CNPJ da Empresa: " + this.cnpjEmpresa + "\n" +  
    super.gerarExtrato();  
}
```

- Além de também sobrescrever o método toString():

```
public String toString() {  
    return "Conta Salario: \n" + super.toString() + "\n" + "CNPJ: " + this.  
    cpnjEmpresa;  
}
```

## Pacote Negócio

- Agora iremos implementar as funcionalidades do sistema;
- Teremos uma classe que irá administrar tudo: a classe Sistema;



# Classe Sistema

- Iremos declarar o pacote negocio e importar as classes que utilizaremos:

```
package negocio;  
  
import dados.ContaBancaria;  
import dados.ContaCorrente;  
import dados.ContaSalario;
```

- A classe Sistema terá um array de ContaBancaria e um atributo que controla a quantidade de contas nesse array;
- As instancias das classes serão mantidas através de **polimorfismo** nesse array;
- Isto é, um array da superclasse armazena tanto instâncias da superclasse como instâncias de classes filhas;
- Nesse sistema só haverá instâncias das classes filhas (ContaCorrente e ContaSalario);

```
public class Sistema {  
  
    private ContaBancaria[] contaBancarias = new ContaBancaria[100];  
    private int quantidade = 0;
```

## Classe Sistema

- O primeiro método a ser criado é método de cadastrarConta;
- Esse método recebe um objeto do tipo ContaBancaria e adiciona ao array;
- Como Java é uma linguagem que implementa polimorfismo, tanto faz qual é o tipo de conta que será enviada como parâmetro. Seja uma instância de uma ContaCorrente ou de uma ContaSalario, ambas serão tratadas como ContaBancaria e adicionadas ao array.

```
public void cadastrarConta(ContaBancaria conta) {  
    if (quantidade < 100) {  
        this.contaBancarias[quantidade] = conta;  
        quantidade++;  
    }  
}
```

- A cada inserção de uma nova conta no sistema, o atributo **quantidade** é incrementado em 1 para controlar o topo da pilha;

- O método de realizar saque nos retornará o extrato;
- E receberá a conta e o valor a ser retirado:

```
public String realizarSaque(ContaBancaria conta, float valor) {  
    conta.sacar(valor);  
    return this.obterExtrato(conta);  
}
```

- Observe que o método utiliza do método **sacar()** do objeto **conta** para manter o encapsulamento;

## Classe Sistema

- Agora iremos implementar os métodos de depositar;
- Para ContaCorrente precisamos apenas da conta e do valor;

```
public boolean realizarDeposito(ContaCorrente conta, float valor) {  
    return conta.depositar(valor);  
}
```

- Já para ContaSalario precisamos do cnpj também;

```
public boolean realizarDeposito(ContaSalario conta, float valor, int cnpj) {  
    return conta.depositar(valor, cnpj);  
}
```

- Note que em ambos o retorno é do tipo booleano, pois os métodos de depositar das duas classes são do tipo booleano;

# Classe Sistema

- Agora iremos implementar os métodos que retornam apenas instâncias de tipos específicos de contas;
- Começaremos pelas instâncias de ContaCorrente;

```
public ContaCorrente [] getContasCorrentes () {
```

- Esse método retorna um array de ContaCorrente;
- Primeiro precisamos de uma variável para contar quantos objetos do array principal da classe Sistema (atributo contaBancarias) são instâncias da classe ContaCorrente:

```
public ContaCorrente [] getContasCorrentes () {  
    int max = 0;
```

- Agora precisamos percorrer todo o array contendo as contas e para cada instância de ContaCorrente iremos incrementar essa variável;
- Utilizaremos do operador `instanceof`;

```
for (int i = 0; i < quantidade; i++) {  
    if (contaBancarias[i] instanceof ContaCorrente) {  
        max++;  
    }  
}
```

- Agora iremos alocar um array de ContaCorrente a partir dessa variável:

```
ContaCorrente[] contas = new ContaCorrente[max];
```

## Classe Sistema

- Após alocado, precisamos preencher esse array;
- Iremos criar uma variável para controlar o topo do array que iremos controlar:

```
int qnt = 0;
```

- Agora iremos novamente percorrer o array da classe Sistema (que contém todos os tipos de contas misturadas) e iremos colocar no array recém criado apenas as instâncias da classe ContaCorrente:
- Porém, como esse array é do tipo ContaCorrente e as instâncias do array da classe Sistema é do tipo ContaBancaria, precisamos fazer um Casting:

```
for (int i = 0; i < quantidade; i++) {  
    if (contaBancarias[i] instanceof ContaCorrente) {  
        contas[qnt] = (ContaCorrente) (contaBancarias[i]);  
        qnt++;  
    }  
}
```

- Por fim basta retornar o array:

```
return contas;
```

- Para o método getContasSalario() é a mesma lógica, só que aplicada a classe ContaSalario, observe o código:



# Classe Sistema

```
public ContaSalario[] getContaSalarios() {  
    int max = 0;  
  
    for (int i = 0; i < quantidade; i++) {  
        if (contaBancarias[i] instanceof ContaSalario) {  
            max++;  
        }  
    }  
  
    ContaSalario[] contas = new ContaSalario[max];  
  
    int qnt = 0;  
  
    for (int i = 0; i < quantidade; i++) {  
        if (contaBancarias[i] instanceof ContaSalario) {  
            contas[qnt] = (ContaSalario) (contaBancarias[i]);  
            qnt++;  
        }  
    }  
  
    return contas;  
}
```

- Agora iremos criar Um método get para o atributo quantidade, caso alguma classe queira saber a quantidade atual de objetos no array de contas:

```
public int getQuantidadeContas() {  
    return this.quantidade;  
}
```

- E um método que retorna todos os tipos de contas sem realizar um filtro nela, isto é, o get do array de contas;

```
public ContaBancaria[] getContaBancarias() {  
    return contaBancarias;  
}
```

- E por fim criaremos um método para obter extratos;
- Já que cada classe filha implementou o seu `gerarExtrato()`, basta retornar a chamada desse método;
- Através de polimorfismo, cada extensão da superclasse irá retornar a chamada do seu método próprio **`gerarExtrato()`**.

```
public String obterExtrato(ContaBancaria conta) {  
    return conta.gerarExtrato();  
}
```

## Pacote Apresentação

- Esse pacote fará a interface via console com o usuário;
- Refletindo as funcionalidades expressas na classe Sistema;
- Para isso criaremos uma classe Principal contendo um método main().

# Classe Principal

- Primeiro iremos declarar o pacote ao qual a classe pertence e as classes que serão importadas:

```
package apresentacao;  
  
import java.util.Scanner;  
  
import dados.ContaBancaria;  
import dados.ContaCorrente;  
import dados.ContaSalario;  
import negocio.Sistema;
```

- A classe Principal terá dois atributos estáticos: uma instancia da classe Sistema e uma da classe Scanner;

```
public class Principal {  
  
    private static Sistema sistema = new Sistema();  
    private static Scanner s = new Scanner(System.in);  
  
}
```

Agora iremos construir os métodos que serão chamados pelo método main:

## Classe Principal

- Criaremos dois métodos para instanciar objetos do tipo ContaCorrente e ContaSalario:
- Ambos os métodos serão estáticos e retornarão objetos do tipo especificado no nome do método:
- Faremos primeiro o método novaContaCorrente():
- Esse método instancia um novo objeto do tipo ContaCorrente e requisita ao usuário as informações referentes a conta;
- Logo após, seta os atributos e retorna o objeto:

```
private static ContaCorrente novaContaCorrente() {  
    ContaCorrente conta = new ContaCorrente();  
  
    System.out.println("Digite o cpf:");  
    conta.setCpf(s.nextInt());  
  
    return conta;  
}
```

# Classe Principal

- O método `novaContaSalario()` segue o mesmo principio só que aplicado a classe `ContaSalario`:

```
private static ContaSalario novaContaSalario() {  
    ContaSalario conta = new ContaSalario();  
  
    System.out.println("Digite o cpf:");  
    conta.setCpf(s.nextInt());  
  
    System.out.println("Digite o cnpj da empresa:");  
    conta.setCnpjEmpresa(s.nextInt());  
  
    return conta;  
}
```

- Agora iremos criar o método que será chamado mais tarde pelo método main para cadastrar contas;
- Esse método requisita ao usuário qual o tipo de conta que ele deseja cadastrar;
- E chama um dos métodos que foi descrito anteriormente;
- Enviando o retorno da chamada do método para o sistema realizar o cadastro;



# Classe Principal

```
private static void cadastrarConta() {  
  
    System.out.println("Digite o tipo de conta que deseja cadastrar:");  
    System.out.println("1 - Conta Corrente");  
    System.out.println("2 - Conta Salario");  
  
    int escolha = s.nextInt();  
  
    switch (escolha) {  
        case 1:  
            sistema.cadastrarConta(novaContaCorrente());  
            break;  
        case 2:  
            sistema.cadastrarConta(novaContaSalario());  
            break;  
        default:  
            System.out.println("Escolha inv lida!");  
            break;  
    }  
}
```

## Classe Principal

- Agora vamos criar o método que irá exibir ao usuário as contas que existem no sistema;
- Esse método requisita ao sistema as contas existentes e utiliza do método toString() de cada objeto para exibir via console as informações:

```
private static void exibirContas() {  
    for (int i = 0; i < sistema.getQuantidadeContas(); i++) {  
        System.out.println("Conta " + i + ":\n" + sistema.getContaBancarias()[i].  
            toString() + "\n");  
    }  
}
```

- Como não é feito nenhum casting nos objetos, o método toString() utilizado é o da superclasse ContaBancaria!

## Classe Principal

- Agora iremos criar um método para o usuário escolher uma conta;
- Utilizaremos novamente do método descrito anteriormente;
- Esse método irá exibir ao usuário todas as contas do sistema;
- E requisitar a ele que escolha uma:

```
private static ContaBancaria escolherContaBancaria() {  
    exibirContas();  
    System.out.println("Escolha uma conta:");  
  
    int conta = s.nextInt();  
  
    if (conta < sistema.getQuantidadeContas()) {  
        return sistema.getContaBancarias()[conta];  
    }  
  
    return null;  
}
```

## Classe Principal

- A partir desse método, criaremos o método de realizar saque;
- O usuário escolhe uma conta e informa o valor a ser retirado da conta;
- O sistema então nos retorna o extrato que será então exibido via console;

```
private static void realizarSaque() {  
    ContaBancaria conta = escolherContaBancaria();  
  
    if (conta != null) {  
        System.out.println("Digite o valor a ser sacado:");  
        int valor = s.nextInt();  
  
        System.out.println(sistema.realizarSaque(conta, valor));  
    }  
}
```

## Classe Principal

- Também utilizando do método de escolher uma conta iremos implementar o método de realizar um depósito;
- Como no sistema existem dois métodos distintos para essa funcionalidade, precisaremos nos ater a isso;
- Primeiro é necessário que o usuário escolha uma conta para realizar o depósito;
- Caso essa conta exista poderemos começar a implementar o depósito:

```
private static void realizarDeposito() {  
    ContaBancaria conta = escolherContaBancaria();  
    if (conta != null) {
```

## Classe Principal

- Precisamos agora verificar que tipo de instancia a conta é: se ela é uma ContaCorrente ou ContaSalario;
- Utilizaremos novamente o operador `instanceof`:

```
if (conta instanceof ContaCorrente) {
```

```
} else {
```

- Para cada caso precisaremos realizar um Casting no objeto, para podermos passar o objeto para o sistema;
- Primeiro será apresentado a estrutura de seleção referente ao caso da conta corrente:

# Classe Principal

```
if (conta instanceof ContaCorrente) {  
    System.out.println("Digite um valor a ser depositado:");  
    int valor = s.nextInt();  
  
    sistema.realizarDeposito((ContaCorrente) (conta), valor);  
    System.out.println("Deposito realizado com sucesso!");  
    System.out.println(sistema.obterExtrato((ContaCorrente) (conta)));  
}  
else {
```

- Para o caso contrário temos que tratar nosso retorno booleano;
- Como para conta corrente, o resultado é sempre true, não nos importamos tanto;
- Mas na conta salário há a possibilidade de retorno do tipo false;
- Para isso iremos exibir uma mensagem informando ao usuário que houve uma falha no depósito;

# Classe Principal

- Agora para o caso contrário:

```
    } else {  
        System.out.println("Digite um valor a ser depositado:");  
        int valor = s.nextInt();  
  
        System.out.println("Digite o cnpj da empresa que est depositando:");  
        int cnpj = s.nextInt();  
  
        if (sistema.realizarDeposito((ContaSalario) (conta), valor, cnpj)) {  
            System.out.println("Deposito realizado com sucesso!");  
            System.out.println(sistema.obterExtrato((ContaSalario) (conta)));  
        } else {  
            System.out.println("Falha ao depositar!");  
        }  
    }  
}
```



## Classe Principal

- Agora iremos implementar o método de mostrar extratos;
- Como toda a lógica desse método está na camada de negócio, basta requisitar ao usuário a conta e exibir no console a chamada do método da classe Sistema:

```
private static void mostrarExtrato() {  
    ContaBancaria conta = escolherContaBancaria();  
    if (conta != null) {  
        System.out.println(sistema.obterExtrato(conta));  
    }  
}
```

## Classe Principal

- Agora que todos os métodos foram implementados, vamos implementar a main;
- Para isso criaremos um método para exibir um menu com as funcionalidades disponíveis;

```
public static void imprimeMenu() {  
  
    System.out.println("Escolha uma opção:");  
    System.out.println("0 - Sair");  
    System.out.println("1 - Cadastrar Conta");  
    System.out.println("2 - Realizar Saque");  
    System.out.println("3 - Realizar Depósito");  
    System.out.println("4 - Exibir Extrato");  
  
}
```

- E por fim o método main:

```
public static void main(String[] args) {  
  
    int opcao = -1;
```

# Classe Principal

```
while (opcao != 0) {  
    imprimeMenu();  
    opcao = s.nextInt();  
  
    switch (opcao) {  
        case 0:  
            break;  
        case 1:  
            cadastrarConta();  
            break;  
        case 2:  
            realizarSaque();  
            break;  
        case 3:  
            realizarDeposito();  
            break;  
        case 4:  
            mostrarExtrato();  
            break;  
    }  
}  
}
```

- Os código-fonte desse exemplo estão disponíveis em:  
[github.com/takeofriedrich/monitoriapoo](https://github.com/takeofriedrich/monitoriapoo) nas aulas práticas;
- Fique atento pois os métodos não estão implementados seguindo a mesma ordem dos slides!
- Não é necessário enviar o exemplo no Moodle, apenas o exercício que será descrito nos próximos slides:

# Seções

Exemplo

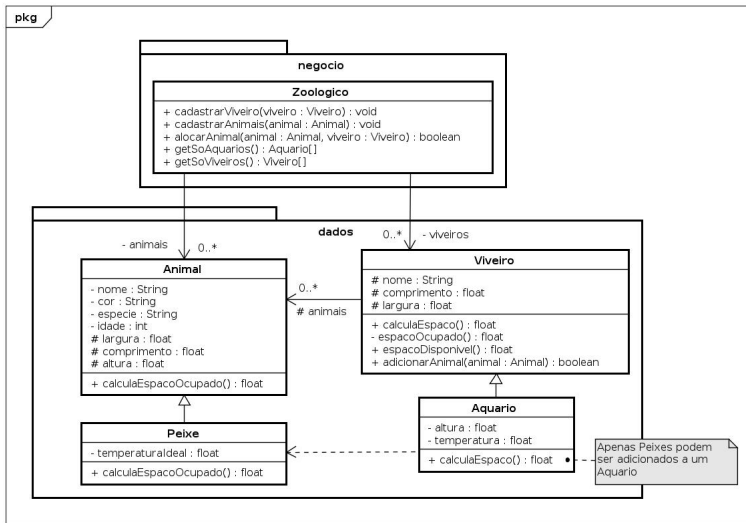
Resolução

Exercício

## Exercício

Implemente em Java um sistema que administre um Zoológico de acordo com o Diagrama de Classes UML a seguir:

# Exercício




## Exercício

- Esse Zoológico possui viveiros, dos quais abrigam diversos animais;
- Entretanto apenas os aquários podem abrigar peixes;
- Já os viveiros comuns podem abrigar animais comuns
- Exemplos de instancias das classes:  
Animal: Zebra  
Peixe: Peixe Espada
- Os animais apenas podem ser alocados em um viveiro caso a área disponível no viveiro seja maior que 70% da área do animal. Para o aquário deve ser considerado o volume;
- Caso já existam animais no viveiro, é necessário subtrair a área de todos os animais já presentes no viveiro para calcular o espaço disponível. Para o aquário deve ser considerado o volume;
- Os animais aquáticos possuem uma restrição de temperatura. Caso a temperatura do aquário esteja 3 graus maior ou menor que a temperatura ideal do animal, ele não pode ser colocado no aquário.



## Exercício

- Além das classes e funcionalidades expressas no diagrama, implemente uma interface de com o usuário via console que o permita utilizar todas as funcionalidades apresentadas no pacote de negócios.
- Na interface com o usuário, ao exibir os viveiros existentes no Zoologico, caso o viveiro não contenha nenhum animal, o sistema deve exibir uma mensagem de viveiro vazio, caso contrário devem ser exibido os animais dentro do viveiro com suas respectivas informações: nome, cor, espécie e temperatura ideal (caso seja um peixe);

 KUWAKI, V. T. F. Modelo de slides udesc lattex. In: . [S.l.]: Disponível em: <<https://github.com/takeofriedrich/slidesUdescLattex>>. Acesso em: 24 jan. 2020.

Duvidas:  
Vinicius Takeo Friedrich Kuwaki  
[vinicius.kuwaki@edu.udesc.br](mailto:vinicius.kuwaki@edu.udesc.br)  
[github.com/takeofriedrich](https://github.com/takeofriedrich)



**UDESC**  
UNIVERSIDADE  
DO ESTADO DE  
SANTA CATARINA