Exercício Extra

Exercício extra sobre Interfaces e Composição

Iremos simular um sistema de documentos para PessoaFisica e Juridica.

O sistema irá adicionar alguns tipos de pessoas, e ao chamar um método irá imprimir o documento referente ao seu tipo. Esse método não precisa imprimir o documento em si, basta uma mensagem dizendo que o documento é CNPJ ou CPF e a pessoa ou empresa que ele pertence.

Utilizaremos conceitos de Interfaces e Composição APENAS. Não usaremos Herança nem Classes Abstratas.

Algumas regrinhas para seguir:

- Deve-se usar **Composição** nas classes `PessoaFisica` e `PessoaJuridica`, logo, usa-se também **encapsulamento** nos métodos para setar e pegar `nome` e `idade` de `Pessoa`;
- O resultado do método `devolveDocumento()` será algo semelhante a isso: "Documento CNPJ da empresa: TriadWorks" ou "Documento CPF da pessoa: Handerson Frota";
- `Pessoa` **só pode ser instanciada** caso passe o valor de `nome` e `idade`.
- Ao instanciar qualquer uma das classes de `PessoaFisica` e/ou `PessoaJuridica`, `Pessoa` já deve ser inicializada e seus parâmetros também;

1) Crie 3 classes principais: `Pessoa`, `PessoaFisica`, `PessoaJuridica`:

   Pessoa.java

   ```
   String nome;
   Integer idade;
   ????? documento;
   ```

   Deve-se usar **Composição** nas classes `PessoaFisica` e `PessoaJuridica`, logo, usa-se também **encapsulamento** nos métodos para setar e pegar os dados de `Pessoa`.

2) Agora criar nossas classes que representarão um documento:

   Classes: `CNPJ` e `CPF`.

   Atributo:

   ```
   private String documento;
   ```

3) A classe `Principal` deve ser algo semelhante a isso:

```
public class Principal {

    public static void main(String[] args) {
        //instancias de PJ e PF


        pj.setNome("TriadWorks");
        pj.setIdade(7);
        //????


        pf.setNome("Handerson Frota");
        pf.setIdade(32);
        //???

        pj.imprimirDadosPessoa();
        pf.imprimirDadosPessoa();
    }

}
```

O resultado deverá ser algo assim:

```
CNPJ: Documento CNPJ: 08478999000187
Empresa: TriadWorks
CPF: Documento CPF: 65322878593
Pessoa: Handerson Frota
```

4) Desafio 1: Altere o método `getDocumentoComMascara` para que ele devolva realmente um numero formatado de um documento. Utilize a classe `MaskFormatter` para fazer isso.

   Máscaras:

```
CNPJ: ##.###.###/####-##
CPF: ###.###.###-##
```

   Resultado:

```
CNPJ: 08.478.999/0001-87
Empresa: TriadWorks
CPF: 653.228.785-93
Pessoa: Handerson Frota
```

5) Desafio 2: Altere a forma como instanciar um CNPJ ou CPF, que seja possível apenas passando o valor do documento.

Respostas dos Exercícios

9.4 - Exercícios

7. Classe `AppleTV` implementada:

```
public class AppleTV implements ControleRemoto {

    private int canal;
    private int volume;
    private boolean ligada;

    @Override
    public void volume(int volume) {
        this.volume = volume;
    }

    @Override
    public void mudarCanal(int canal) {
        this.canal = canal;
    }

    @Override
    public void ligar() {
        this.ligada = true;
    }

    @Override
    public void desligar() {
        this.ligada = false;
    }

    @Override
    public void status() {
        String mensagem = "Sua Apple TV está desligada!";
        if(this.ligada){
            mensagem = "Sua Apple TV está ligada, seja bem vindo!";
        }
        System.out.println(mensagem);
    }

}
```

14.7 - Exercícios

5. Qualquer classe que **seja um** `InputStream` serve alí. As que a própria API do Java traz consigo são encontradas na documentação da própria `InputStream` no campo *Direct Known Subclasses*.

   – AudioInputStream

- ByteArrayInputStream
- FileInputStream
- InputStream (do CORBA)
- ObjectInputStream
- PipedInputStream
- SequenceInputStream
- StringBufferInputStream
- **FilterInputStream**

Essa última, ainda tem suas filhas que, ainda que indiretamente, também são um `InputStream`. São elas: `BufferedInputStream`, `CheckedInputStream`, `CipherInputStream`, `DataInputStream`, `DeflaterInputStream`, `DigestInputStream`, `InflaterInputStream`, `LineNumberInputStream`, `ProgressMonitorInputStream`, `PushbackInputStream`.

15.6 - Exercícios

7. A mudança não irá alterar em nada o funcionamento do código. Essa é mais uma aplicação do **polimorfismo** e a vantagem que você ganha ao utilizar interface no lugar da implementação.

18 - Exercícios

1. Pessoa

```
public class Pessoa {

    private String nome;
    private Integer idade;
    private Documento documento;

    public Pessoa(String nome, Integer idade){
        this.nome = nome;
        this.idade = idade;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Integer getIdade() {
        return idade;
    }

    public void setIdade(Integer idade) {
        this.idade = idade;
    }

    public void setDocumento(Documento ocumento){
        this.documento = ocumento;
    }
```

```
    public Documento getDocumento(){
        return this.documento;
    }

}
```

Interface `Documento`:

```
public interface Documento {

    public String getDocumentoComMascara()

}
```

PessoaFisica

```
public class PessoaFisica {

    private Pessoa pessoa;

    public PessoaFisica(){
        pessoa = new Pessoa("", 0);
    }

    public void setNome(String nome){
        pessoa.setNome(nome);
    }

    public String getNome(){
        return pessoa.getNome();
    }

    public void setIdade(Integer idade){
        pessoa.setIdade(idade);
    }

    public Integer getIdade(){
        return pessoa.getIdade();
    }

    public void setDocumento(Documento documento) {
        pessoa.setDocumento(documento);
    }

    public Documento getDocumento() {
        return pessoa.getDocumento();
    }

    public void imprimirDadosPessoa() throws ParseException{
        System.out.println("CPF: " + pessoa.getDocumento().getDocumentoComMascara());
        System.out.println("Pessoa: " + pessoa.getNome());
    }

}
```

PessoaJuridica

```
public class PessoaJuridica {

    private Pessoa pessoa;

    public PessoaJuridica(){
        pessoa = new Pessoa("", 0);
    }

    public void setNome(String nome){
        pessoa.setNome(nome);
    }

    public String getNome(){
        return pessoa.getNome();
    }

    public void setIdade(Integer idade){
        pessoa.setIdade(idade);
    }

    public Integer getIdade(){
        return pessoa.getIdade();
    }

    public void setDocumento(Documento documento) {
        pessoa.setDocumento(documento);
    }

    public Documento getDocumento() {
        return pessoa.getDocumento();
    }

    public void imprimirDadosPessoa() throws ParseException{
        System.out.println("CNPJ: " + pessoa.getDocumento().getDocumentoComMascara());
        System.out.println("Empresa: " + pessoa.getNome());
    }

}
```

2. CNPJ

```
public class CNPJ implements Documento{

    private String documento;

    public String getDocumento() {
        return documento;
    }

    public void setDocumento(String documento) {
        this.documento = documento;
    }

    public String getDocumentoComMascara() throws ParseException {
        return "Documento CNPJ: " + this.documento;
    }

}
```

CNPJ

```
public class CPF implements Documento{

    private String documento;

    public String getDocumento() {
        return documento;
    }


    public void setDocumento(String documento) {
        this.documento = documento;
    }

    public String getDocumentoComMascara() throws ParseException {
        return "Documento CPF: " + this.documento;
    }

}
```

3.
```
public class Principal {

    public static void main(String[] args) {
        PessoaJuridica pj = new PessoaJuridica();
        PessoaFisica pf = new PessoaFisica();


        pj.setNome("TriadWorks");
        pj.setIdade(7);
        CNPJ documentoCNPJ = new CNPJ();
        documentoCNPJ.setDocumento("08478999000187");

        pj.setDocumento(documentoCNPJ);

        pf.setNome("Handerson Frota");
        pf.setIdade(32);
        CPF documentoCPF = new CPF();
        documentoCPF.setDocumento("65322878593");

        pf.setDocumento(documentoCPF);

        pj.imprimirDadosPessoa();
        pf.imprimirDadosPessoa();

    }

}
```

4. Alterar o método `getDocumentoComMascara` para formatar a máscara:

   CNPJ:

```
public String getDocumentoComMascara() throws ParseException {
    MaskFormatter mf = new MaskFormatter("##.###.###/####-##");
    mf.setValueContainsLiteralCharacters(false);
    return mf.valueToString(this.documento) ;
}
```

CPF:

```
public String getDocumentoComMascara() throws ParseException {
    MaskFormatter mf = new MaskFormatter("###.###.###-##");
    mf.setValueContainsLiteralCharacters(false);
    return mf.valueToString(this.documento) ;
}
```

Execute a classe com o método `main` novamente, **nenhuma** alteração precisa ser feita, apenas em **getDocumentoComMascara()**.

Resultado:

```
CNPJ: 08.478.999/0001-87
Empresa: TriadWorks
CPF: 653.228.785-93
Pessoa: Handerson Frota
```

Agora todo tipo de documento terá a capacidade de aplicar a máscara referente ao seu tipo.

5. Crie um construtor:

```
public CPF(String documento) {
    this.documento = documento;
}

public CNPJ(String documento){
    this.documento = documento;
}
```

Classe com método `main`:

```
public static void main(String[] args) throws ParseException {
    PessoaJuridica pj = new PessoaJuridica();
    PessoaFisica pf = new PessoaFisica();


    pj.setNome("TriadWorks");
    pj.setIdade(7);
    pj.setDocumento(new CNPJ("08478999000187"));

    pf.setNome("Handerson Frota");
    pf.setIdade(32);
    pf.setDocumento(new CPF("65322878593"));

    pj.imprimirDadosPessoa();
    pf.imprimirDadosPessoa();

}
```