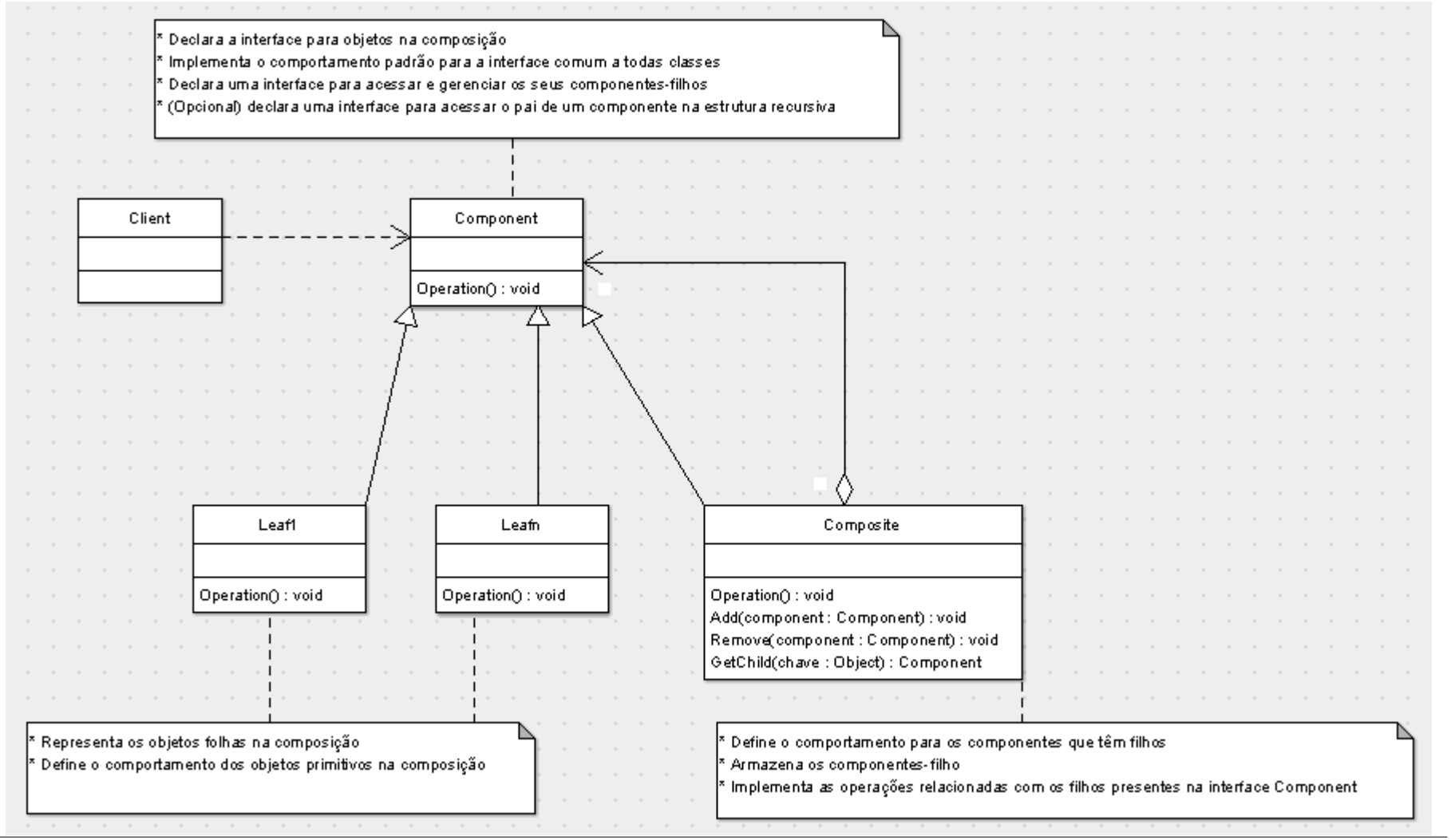
Composite

Instituto Federal do Espirito Santo – Campus Colatina
Disciplina: Projeto de Sistemas
Professor: Giovany Frossard Teixeira

Elementos Essenciais

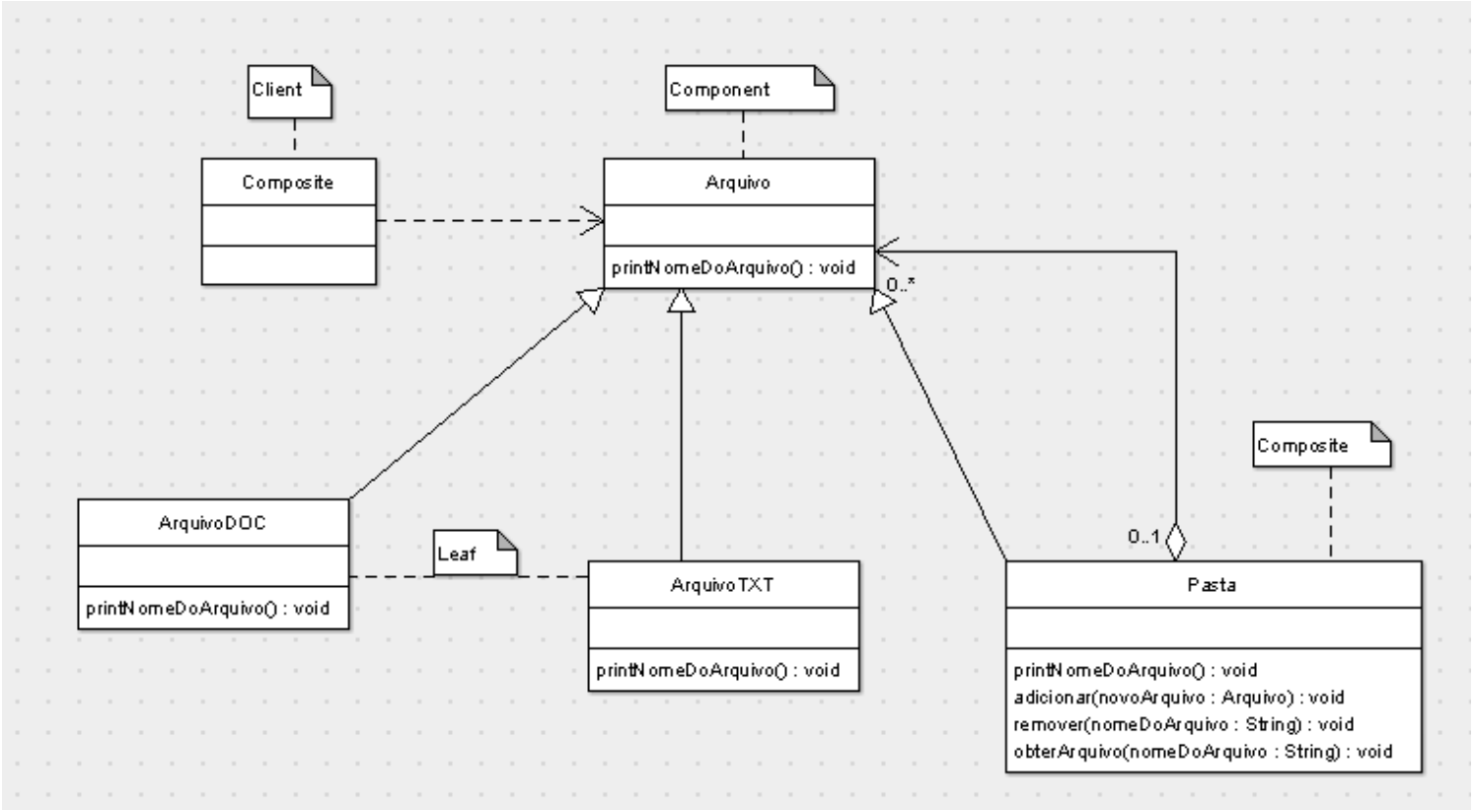
- **Nome:** *Composite*
- **Problema:** Existem problemas cuja composição de componentes é feita através de componentes mais simples, além disso, esses componentes podem ser agrupados e gerar novos componentes mais complexos (Ex: ponto, linha, retângulo). A priori uma linha pode ser tratada como um conjunto de pontos, um retângulo pode ser tratado através de um conjunto de linhas, entretanto, existem contextos onde esses elementos são tratados de forma idêntica.
- **Solução:** Compor objetos em estruturas de arvores para representarem hierarquias partes-todo. O Composite permite aos clientes tratarem de maneira uniforme objetos individuais e composições de objetos.
- **Consequências:** Facilidade no uso (uniformidade) de objetos/classes com característica recursiva (Uma pasta, por exemplo, possui arquivos e pastas dentro dela).

Padrão Composite



Problema

- Vamos tratar de um sistema para gerenciamento de arquivos e pastas. Sendo que pastas são arquivos que podem conter outros arquivos. Arquivos podem ser quaisquer tipos de arquivos (.DOC, .TXT, etc).



Códigos

```
public class Composite {
    public static void main(String[] args) {
        Arquivo arq1 = new ArquivoDOC("texto1");
        Arquivo arq2 = new ArquivoTXT("texto2");

        arq1.printNomeDoArquivo();
        arq2.printNomeDoArquivo();

        Arquivo pasta1 = new Pasta("Pasta1");
        Arquivo pasta2 = new Pasta("Pasta2");

        ( (Pasta)pasta2).adicionar(arq1);
        ( (Pasta)pasta2).adicionar(arq2);
        ( (Pasta)pasta2).printNomeDoArquivo();

        ( (Pasta)pasta1).adicionar(pasta2);
        ( (Pasta)pasta1).printNomeDoArquivo();
    }
}
```

```
run:
texto1.DOC
texto2.TXT
Pasta2
texto1.DOC
texto2.TXT
Pasta1
Pasta2
texto1.DOC
texto2.TXT
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

```
public abstract class Arquivo {
    protected String nomeDoArquivo;
    public Arquivo(String nomeDoArquivo) {
        this.nomeDoArquivo = nomeDoArquivo;
    }
    public String getNomeDoArquivo() {
        return nomeDoArquivo;
    }
    public  void printNomeDoArquivo() {
        System.out.println(this.nomeDoArquivo);
    }
}
```

```
public class ArquivoDOC extends Arquivo{
    public ArquivoDOC(String nomeDoArquivo) {
        super(nomeDoArquivo);
    }
    @Override
    public void printNomeDoArquivo() {
        System.out.println(this.nomeDoArquivo + ".DOC");
    }
}
```

Códigos Classe Pasta

```java
public class Pasta extends Arquivo {
    protected ArrayList<Arquivo> arquivos;
    public Pasta(String nomeDoArquivo) {
        super(nomeDoArquivo);
        this.arquivos = new ArrayList<>();
    }
    @Override
    public void printNomeDoArquivo() {
        System.out.println(this.nomeDoArquivo);
        arquivos.forEach((arquivo) -> {
            arquivo.printNomeDoArquivo();
        });
    }
    public void adicionar(Arquivo novoArquivo) {
        this.arquivos.add(novoArquivo);
    }
    public void remover(String nomeDoArquivo) throws Exception {
        for (Arquivo arquivo : arquivos) {
            if (arquivo.getNomeDoArquivo().equals(nomeDoArquivo)) {
                this.arquivos.remove(arquivo);
                return;
            }
        }
        throw new Exception("Não existe este arquivo");
    }
    public Arquivo obterArquivo(String nomeDoArquivo) throws Exception {
        for (Arquivo arquivo : arquivos) {
            if (arquivo.getNomeDoArquivo().equals(nomeDoArquivo)) {
                return arquivo;
            }
        }
        throw new Exception("Não existe este arquivo");
    }
}
```

Vantagens

O cliente pode tratar estruturas compostas e objetos individuais de maneira uniforme

Torna mais fácil acrescentar novas espécies de componentes

Desvantagem

- Perceba que precisamos utilizar um cast para fazer a chamada aos métodos da **Pasta**. Caso o objeto não fosse uma **Pasta** de fato teríamos problemas.
- Mas esse problema só ocorre porque não definimos os métodos de **adicionar**, **remover** e **obterArquivo** na classe **Arquivo**. De fato esses métodos não tem funcionalidade lá, entretanto se colocarmos esses métodos apenas com disparos a Exceptions, a assinatura ficará definida em **Arquivo** e não precisaremos fazer o cast. Essa abordagem implica também em colocarmos um try-catch na classe **Composite** (que contem o main).

Nova classe Arquivo

```
public abstract class Arquivo {
    protected String nomeDoArquivo;
    public Arquivo(String nomeDoArquivo) {
        this.nomeDoArquivo = nomeDoArquivo;
    }
    public String getNomeDoArquivo() {
        return nomeDoArquivo;
    }
    public  void printNomeDoArquivo() {
        System.out.println(this.nomeDoArquivo);
    }

    public void adicionar(Arquivo novoArquivo) throws Exception{
        throw new Exception("Não pode inserir arquivos em: "
                + this.nomeDoArquivo + " - Não é uma pasta");
    }
    public void remover(String nomeDoArquivo) throws Exception {
        throw new Exception("Não pode remover arquivos em: "
                + this.nomeDoArquivo + " -Não é uma pasta");
    }
    public Arquivo obterArquivo(String nomeDoArquivo) throws Exception {
        throw new Exception("Não pode pesquisar arquivos em: "
                + this.nomeDoArquivo + " - Não é uma pasta");
    }
}
```

Novo main()

```
run:
texto1.DOC
texto2.TXT
Pasta2
texto1.DOC
texto2.TXT
Pasta1
Pasta2
texto1.DOC
texto2.TXT
ERRO: Não pode inserir arquivos em: texto1 - Não é uma pasta
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

```
public class Composite {
    public static void main(String[] args) {
        try {
            Arquivo arq1 = new ArquivoDOC("texto1");
            Arquivo arq2 = new ArquivoTXT("texto2");

            arq1.printNomeDoArquivo();
            arq2.printNomeDoArquivo();

            Arquivo pasta1 = new Pasta("Pasta1");
            Arquivo pasta2 = new Pasta("Pasta2");
            pasta2.adicionar(arq1);
            pasta2.adicionar(arq2);
            pasta2.printNomeDoArquivo();

            pasta1.adicionar(pasta2);
            pasta1.printNomeDoArquivo();

            arq1.adicionar(arq2);

        } catch (Exception erro) {
            System.out.println("ERRO: " + erro.getMessage());
        }
    }
}
```

Dúvidas?