





Compartilhe com luciorocha @ professores.utfpr.edu.br


Nome:


RA:

Henrique,2369133 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >

Séfora, 2367777 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >

mabyilly_2368013 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >


daniel martins de carvalho/ gabriel takeshi abe <
 Cópia de Aula 31 - POCO4A - Exercícios propostos >

Vitor Luiz de Castro Viana_2503298 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >


João Pedro de Paula_2208458 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >.

Deivid da Silva Galvão 2408740 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >


João Vitor Yoshida, 2419904 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >

Filipe Augusto Parreira, 2320622 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >

João Pedro Cavani Meireles <  Cópia de Aula 31 - POCO4A - Exercícios propostos >


Thales Hasegawa, 2272075 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >


Angélica, 2047063 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >


Rafael Kendy, 2478544 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >


Guilherme Conceição Ramalho <  Cópia de Aula 31 - POCO4A - Exercícios propostos >


Lucas Viana, Victor Ramos, Carlos Eduardo, Luis Henrique


<  Cópia de Aula 31 - POCO4A - Exercícios propostos >

Matheus Hirata_2408848 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >

Thiago Cristovão, 2368072 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >

João Kawano, 2453223 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >

Wesley Zimmer 2369184 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >

Plinio 2279614 / Marcos Tadao 2374200<
 Cópia de Aula 31 - POCO4A - Exercícios propostos >

Raphael Uematsu, 2152363 <  Cópia de Aula 31 - POCO4A - Exercícios propostos >

Quiz 1:

<https://quizizz.com/admin/quiz/6112c285e1ed06001d21aee3/poo-diagramas-uml?createdByMe=true&searchLocale=>

Quiz 2:

https://quizizz.com/admin/quiz/6491a011413159001d47c967?source=quiz_share

1) Marque Verdadeiro ou Falso:

() O Padrão de Projeto Facade é utilizado como uma interface que mascara componentes complexos por trás de uma API (Application Programming Interface).

() O Padrão de Projeto Facade fornece uma interface genérica de contexto para uma funcionalidade mais específica.

() O Padrão de Projeto Facade serve como um ponto de entrada que favorece um código fracamente acoplado.

() No Padrão de Projeto Delegação uma classe delega a responsabilidade de implementação dos métodos para as subclasses, através de polimorfismo.

```
public abstract POO {
    public void imprimir();
}
public class Turma1 extends POO {
    public void imprimir () { System.out.println("TURMA1");
}
public class Turma2 extends POO {
    public void imprimir () { System.out.println("TURMA2");
}
```

() No Padrão de Projeto Delegação um objeto pode mudar de classe.

```

public interface POO {
    public void imprimir();
}
public class Turma1 implements POO {
    public void imprimir () { System.out.println("TURMA1");
}
public class Turma2 implements POO {
    public void imprimir () { System.out.println("TURMA2");
}
public class Principal {
    public Principal(){
        POO turma = new Turma1();
        turma.imprimir(); //TURMA1
        turma = new Turma2();
        turma.imprimir(); //TURMA2
    }
}

```

() No Padrão de Projeto Interface com Delegação, o objeto do tipo da Interface pode mudar de classe, desde que as classes implementem a mesma Interface.

() O Padrão de Projeto Iterator é usado para listar objetos de classes que implementam a mesma interface.

```

List <POO> lista = new ArrayList<>();
lista.add( new Turma1() );
lista.add( new Turma2() );

```

() O Padrão de Projeto Iterator é usado para listar objetos de classes que têm a mesma superclasse.

() O Padrão de Projeto Adapter utiliza polimorfismo.

() No Padrão de Projeto Adapter, um objeto do tipo da Interface pode mudar de classe e manter a mesma assinatura.

```

...
IPOO objeto = new Turma1(); //subclasse → superclasse
objeto.imprimir(); //TURMA1

```

```
objeto = new Turma2();
objeto.imprimir(); //TURMA2
```

() No Padrão de Projeto Adapter, um objeto do tipo da superclasse pode mudar de subclasse e manter a mesma assinatura.

() O Padrão de Projeto Singleton é utilizado para manter uma única instância de um objeto da classe.

```
private static POO turma;

private POO () { ... }
public static POO iniciar(){
    if ( turma == null )
        return new POO();
    else
        return null;
}
```

() No Padrão de Projeto Singleton não é utilizado polimorfismo.

() O Padrão de Projeto Visitor utiliza polimorfismo.

() O Padrão de Projeto Visitor é similar a um filtro. Quando aplicado, o objeto do tipo da Interface pode mudar de classe com polimorfismo.

```
IFiltro turma = new Turma1();
turma.filtro( ... );

turma = new Turma2();
turma.filtro( ... );
```

() O Padrão de Projeto Observer só funciona se existir uma classe Observável.

Observador: objeto que recebe notificações. Relacionamento $1 \rightarrow N$
 Observável: objeto que envia notificações. Relacionamento $1 \rightarrow N$
 $N \times N$

() No Padrão de Projeto Observer, o objeto observador apenas receberá a notificação do objeto observável caso este notifique o objeto observador.

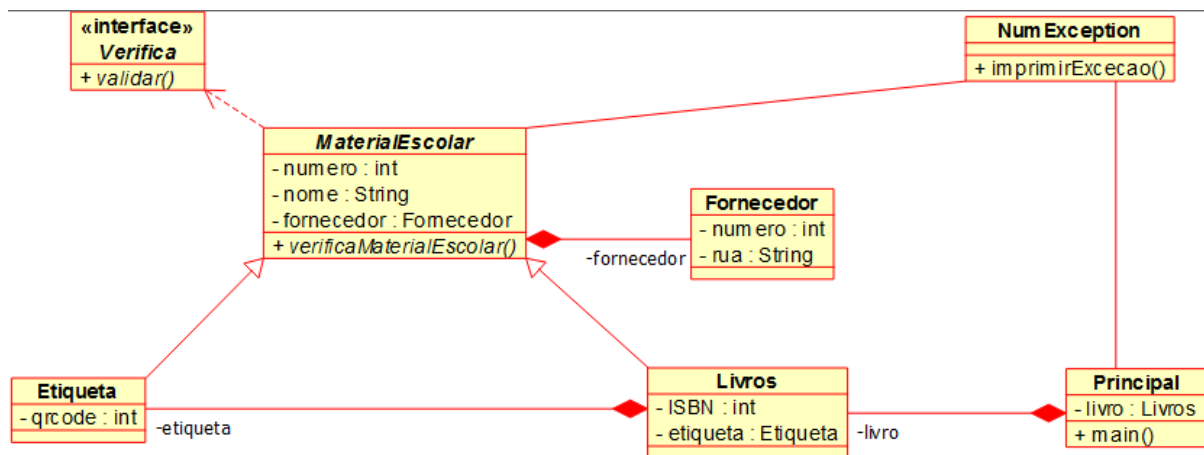
() O Padrão de Projeto Observer não utiliza polimorfismo.

() O Padrão de Projeto Template permite que a superclasse acesse métodos da subclasse.

```
public interface ISuperclasse {  
    public void imprimir();  
}  
  
public abstract class Superclasse implements ISuperclasse {  
    public Superclasse(){  
        this.imprimir();  
    }  
}  
  
public classe Subclasse extends Superclasse {  
    public Subclasse(){  
        super();  
    }  
    public void imprimir(){ System.out.println("SUBCLASSE"); }  
}
```

(V) No Padrão de Projeto Template, a superclasse deve ser abstrata e a subclasse deve implementar o método abstrato da superclasse.

2) Implemente o Diagrama de Classes a seguir, como solicitado:



1) (1,0 ponto) Métodos construtores: não serão desenvolvidos os métodos construtores. Dessa forma, a inicialização de cada atributo nas classes será feita logo na declaração deles, como segue:

- Os tipos numerais inicializam com zeros.
- Os tipos string inicializam com espaço em branco.
- Os objetos inicializam com o seu respectivo tipo.

2) (1,0 ponto) As classes **Fornecedor**, **Etiqueta** e **Livros** não poderão ser herdadas.

3) (1,0 ponto) A classe **NumException** é uma classe de exceção verificada. O método **imprimirExcecao()** exibirá "ERRO: numero negativo", caso o **numero** do **MaterialEscolar** fornecido na classe **Principal** seja negativo.

4) (1,0 ponto) A interface **Verifica** deverá ter o método **validar** para imprimir na tela se o atributo **numero** do **MaterialEscolar** é par ou ímpar.

5) A classe **MaterialEscolar** é abstrata e contém:

5.1) O método abstrato **verificaMaterialEscolar()** que fará o seguinte:

a) (0,5 ponto) Em **Etiqueta**: se o número da etiqueta está no intervalo fechado [10,50]. Se sim, imprimir "Etiqueta válida". Caso contrário, imprimir "Etiqueta inválida".

b) (0,5 ponto) Em **Livros**: se a quantidade de letras do nome do **livro** é maior que 20, imprimir "Nome inválido". Caso contrário, imprimir "Nome válido".

5.2) (1,0 ponto) O método **setNumero**: se o valor for positivo, atribuirá este valor à variável de instância **numero** do MaterialEscolar. Caso contrário, disparará uma exceção **NumException**.

6) A partir da classe **Principal**, deve-se instanciar um objeto do tipo **Livros**.

6.1) (2,0 pontos) Entrada dos dados: os valores serão passados como parâmetros por meio dos métodos mutadores das classes, obrigatoriamente na ordem a seguir e apenas para os seguintes atributos:

ENTRADA DE DADOS	
Classes	Atributos a serem instanciados
Livros	1) numero <i>(Na classe Principal, ao tentar utilizar o método mutador para o numero, se este disparar uma NumException, seu catch deverá invocar o método imprimirExcecao())</i>
	2) ISBN
	3) rua
Etiqueta (trata-se do atributo "etiqueta" que está na Classe Livros especificada acima)	4) qrcode
	5) nome

6.2) (2,0 pontos) Saída de dados: a partir das entradas anteriores, as saídas serão por impressão na tela, obrigatoriamente na ordem a seguir e apenas dos seguintes dados:

SAÍDA DE DADOS	
Classes	Dados a serem impressos na tela
Livros	1) numero
	2) Se numero é par ou ímpar
	3) ISBN
	4) rua
	5) numero do Endereco
Etiqueta (trata-se do atributo "etiqueta" que está na Classe Livros especificada acima)	6) qrcode da Etiqueta
	7) Se o qrcode da etiqueta é valida ou não
	8) Nome da etiqueta
	9) Se o nome da etiqueta é valido ou não
	10) Quais atributos não foram fornecidos a partir das entradas da tabela anterior.

```
package aula31;

import java.util.Scanner;

public class Principal {
```



```

private Livros livro;

public class NumException extends Exception{
    public void imprimirExcecao(){
        System.out.println("ERRO: numero negativo");
    }
}

public class Fornecedor{
    private int numero=0;
    private String rua="";
    public void setRua(String rua){
        this.rua=rua;
    }
}

public interface Verifica{
    public abstract void validar();
}
//se tiver em italico e abstract
public abstract class MaterialEscolar implements Verifica{
    private int numero=0;
    private String nome="";
    private Fornecedor fornecedor= new Fornecedor();

    public abstract void verificaMaterialEscolar();

    public void validar(){
        if(this.numero%2==0)
            System.out.println("PAR");
        else
            System.out.println("IMPAR");
    }
    public int getNumero(){
        return this.numero;
    }
    public void setNumero(int numero){
        if(numero> 0)
            this.numero=numero;
        else
            try{
                throw new NumException();
            }catch(NumException e){
                e.imprimirExcecao();
            }
    }
    public String getNome(){
        return this.nome;
    }
}

```

```

    public Fornecedor getFornecedor(){
        return this.fornecedor;
    }
}

public final class Livros extends MaterialEscolar{
    private int ISBN=0;
    private Etiqueta etiqueta= new Etiqueta();
    /* public Livros(int ISBN, Etiqueta etiqueta){
        this.ISBN = ISBN;
        this.etiqueta=etiqueta;
    }*/

    public void verificaMaterialEscolar() {
        if(getNome().length()>20)
            System.out.println("Nome inválido");
        else
            System.out.println("Nome válido");
    }

    public void setISBN(int numero){
        this.ISBN= ISBN;
    }
    public void validar() {

    }
}

public final class Etiqueta extends MaterialEscolar{
    private int qrcode=0;
    /*public Etiqueta(int qrcode){
        this.qrcode=qrcode;
    }*/

    public void verificaMaterialEscolar() {
        if(getNumero()>=10 && getNumero()<=50)
            System.out.println("Etiqueta válida");
        else
            System.out.println("Etiqueta inválida");
    }

    public void validar() {

    }
}
public Principal(){

```

```
/*Etiqueta etiqueta = new Etiqueta(111);
livro=new Livros(123456,etiqueta);*/
int numero =0;
try{
    Scanner leitor= new Scanner(System.in);
    numero=leitor.nextInt();
    if(numero< 0)
        throw new NumException();
    livro = new Livros();
    livro.setNumero(numero);
    livro.setISBN(numero);
    livro.getFornecedor().setRua("Rua 123");

    leitor.close();
}catch(NumException e){
    e.imprimirExcecao();
}
}

public static void main(String[] args) {
    Principal aula=new Principal();
}

}
```