public class Mensageiro {

public Mensagem prepararMensagem(String msg, String destino) {

Mensagem m = new Mensagem(msg);

if(destino.equals("interno"))

return m;

else if(destino.equals("externo")) {

Decorador d = new Criptografia(msg);

m.setTexto(d.getTexto());

d = new DireitosAutorais(m.getTexto());

m.setTexto(d.getTexto());

return m;

}

else

return m;

}

}

public class Mensagem {

private String texto;

public Mensagem(String texto) {

setTexto(texto);

}

public String getTexto() {

return texto;

}

public void setTexto(String texto) {

this.texto = texto;

}

}

public class Decorador extends Mensagem{

private Mensagem mensagem;

public Decorador(String texto) {

super(texto);

}

public Mensagem getMensagem() {

return mensagem;

}

public void setMensagem(Mensagem mensagem) {

this.mensagem = mensagem;

}

}

public class DireitosAutorais extends Decorador{

public DireitosAutorais(String texto) {

super(texto);

}

public String getTexto() {

setTexto(super.getTexto().concat("... [Direitos Reservados]."));

return super.getTexto();

}

}

public class Criptografia extends Decorador{

public Criptografia(String texto) {

super(texto);

}

public String getTexto() {

setTexto(super.getTexto().replace(" ", "\*"));

return super.getTexto();

}

}

public class ProgramaQ3 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

Mensageiro m = new Mensageiro();

System.out.print("Escreva seu texto: ");

String txt = input.nextLine();

System.out.println("Destino: interno ou externo");

String destino = input.nextLine();

System.out.println(m.prepararMensagem(txt, destino).getTexto());

input.close();

}

}

public interface ClasseComparavel {

public int comparacao(Object o);

}

public class ConversorData {

public static MinhaData dataToMData(String data) {

String[] dataSplit = data.split("/");

int[] dataInt = new int[3];

for(int i=0;i<dataInt.length;i++)

dataInt[i] = Integer.parseInt(dataSplit[i]);

MinhaData mData = new MinhaData(dataInt[0], dataInt[1], dataInt[2]);

return mData;

}

}

public class ProgramaQ4 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

MinhaData[] datas = new MinhaData[2];

for(int i=0;i<2;i++) {

System.out.println("Informe a "+(i+1)+"ª data: (DD/MM/YYYY)");

String data = input.nextLine();

datas[i] = ConversorData.dataToMData(data);

}

if(datas[0].comparacao(datas[1])==1) {

System.out.println("\n"+datas[1]);

System.out.println(datas[0]);

}

else {

System.out.println("\n"+datas[0]);

System.out.println(datas[1]);

}

input.close();

}

}

public class MinhaData implements ClasseComparavel{

private int dia;

private int mes;

private int ano;

public MinhaData(int dia, int mes, int ano) {

setDia(dia);

setMes(mes);

setAno(ano);

}

public String toString() {

return dia+"/"+mes+"/"+ano;

}

public int comparacao(Object o) {

MinhaData md = (MinhaData)o;

if(md.getAno()<ano)

return 1;

else if(md.getAno()==ano) {

if(md.getMes()<mes)

return 1;

else if(md.getMes()==mes) {

if(md.getDia()<dia)

return 1;

else if(md.getDia()==dia)

return 0;

return -1;

}

return -1;

}

return -1;

}

public int getDia() {

return dia;

}

public void setDia(int dia) {

this.dia = dia;

}

public int getMes() {

return mes;

}

public void setMes(int mes) {

this.mes = mes;

}

public int getAno() {

return ano;

}

public void setAno(int ano) {

this.ano = ano;

}

}

public abstract class Carro {

private String modelo;

private double preco;

public Carro(String modelo, double preco) {

setModelo(modelo);

setPreco(preco);

}

public String getModelo() {

return modelo;

}

public void setModelo(String modelo) {

this.modelo = modelo;

}

public double getPreco() {

return preco;

}

public void setPreco(double preco) {

this.preco = preco;

}

}

public class CarroNovo extends Carro{

private double percentualAcres;

public CarroNovo(String modelo, double preco) {

super(modelo, preco);

setPercentualAcres(super.getPreco()\*0.10);

}

public String toString() {

NumberFormat dinheiro = NumberFormat.getCurrencyInstance(new Locale("pt","BR"));

String acres = "Preço mais acréscimo: "+dinheiro.format(getPreco());

return "Carro novo, "+super.getModelo()+", "+dinheiro.format(super.getPreco())+"\n"+acres;

}

public double getPreco() {

return super.getPreco()+getPercentualAcres();

}

public double getPercentualAcres() {

return percentualAcres;

}

public void setPercentualAcres(double percentualAcres) {

this.percentualAcres = percentualAcres;

}

}

public class CarroUsado extends Carro{

private double percentualDesc;

public CarroUsado(String modelo, double preco) {

super(modelo, preco);

setPercentualDesc(super.getPreco()\*0.10);

}

public String toString() {

NumberFormat dinheiro = NumberFormat.getCurrencyInstance(new Locale("pt","BR"));

String desc = "Preço mais desconto: "+dinheiro.format(getPreco());

return "Carro usado, "+super.getModelo()+", "+dinheiro.format(super.getPreco())+"\n"+desc;

}

public double getPreco() {

return super.getPreco()-getPercentualDesc();

}

public double getPercentualDesc() {

return percentualDesc;

}

public void setPercentualDesc(double percentualDesc) {

this.percentualDesc = percentualDesc;

}

}

public class ProgramaQ5 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

Carro c;

System.out.print("Informe o preço do carro: ");

double preco = Double.parseDouble(input.nextLine());

System.out.print("Informe o modelo do carro: ");

String modelo = input.nextLine();

System.out.print("1-Carro Novo, 2-Carro Usado: ");

String op = input.nextLine();

if(op.equals("1"))

c = new CarroNovo(modelo, preco);

else

c = new CarroUsado(modelo, preco);

System.out.println(c);

input.close();

}

}

6 - Não, Por que interfaces não podem ter construtores, e também numa interface não se pode definir campos, pois o mesmo é uma implementação de um atributo objeto, a interface também não permite construtores, pois num construtor temos as instruções usadas para inicializar campos.

7 - Fonte de inspiração: <https://www.youtube.com/watch?v=SXTd3qei8aQ> (Java - POO - Interfaces x Classe Abstrata)

Uma das principais razões para usar interfaces em vez de herança direta é a flexibilidade que elas oferecem. Uma interface define um conjunto de métodos que uma classe deve implementar, mas não especifica como esses métodos devem ser implementados. Isso permite que as classes que implementam a interface tenham diferentes comportamentos e ainda assim possam ser usadas de maneira intercambiável.

Além disso, o uso de interfaces promove a separação de preocupações e a modularidade do código. Isso torna mais fácil para os desenvolvedores entenderem como as diferentes partes do sistema interagem e permite que o código seja modificado sem afetar outras partes do sistema.

Usando interfaces, podemos criar uma hierarquia de tipos que podem ser usados ​​de maneira polimórfica.

Em resumo, o uso de interfaces em vez de herança direta promove a flexibilidade, modularidade e separação de preocupações. Essas são vantagens importantes que podem levar a um código mais limpo, robusto e fácil de manter.

8 - Em C ++, esses problemas são chamados de ambiguidades: o compilador gera um erro se não puder determinar qual método usar. O desenvolvedor deve então redefinir o método na classe derivada ou usar o operador de resolução de escopo (: :) para especificar o método a ser usado.

9 - I) a. II) a. III) d.

10 - a) Alica b) BedoinCa c) Abedoin

d) Cadaco e) Sasa f) Sasa

g) Sasa h) Alica i) Abedoin