2 – Não apresentam, pois, todas as subclasses são exclusivas e não tem possibilidade de se transformar em outra classe, sendo assim não há necessidade de criar várias possibilidades de subclasse.

3 – Todas as heranças são completas e disjuntas. Completa, pois não existe outras subclasses. Disjunta, pois são classes exclusivas uma não pode sobrepor a outra.

4 – Quando o objeto é mandado por parâmetro e variável local não estruturada, há ganho de memória, pois assim que o método finaliza aquele objeto instanciado é deslocado da memória, nota-se que também que se ganha em encapsulamento, abaixando o acoplamento, porém, o desempenho cai.

5 –

public class ControleCliente {

public void adicionarCliente(Cliente c) {

...

}

}

public class ControleCliente {

public void adicionarCliente() {

Cliente c = new Cliente();

...

}

}

6 – Set, pois essa estrutura não permite que coloque objetos repetidos.

7 – public class Cliente {

List<Animal> animal = new LinkedList<> ();

}

public class Animal {

Set<Consulta> consulta = new HashSet<> ();

}

public class Conta {

Set<Exame> exame = new HashSet<> ();

}

public class Historico {

Set<Exame> exame = new HashSet<> ();

Set<Consulta> consulta = new HashSet<> ();

}

8 - A estrutura TreeSet compara os elementos e os ordena, os elementos são ordenados à medida que são adicionados na lista.

9 - public class Cliente {

TreeSet<Animal> animal = new TreeSet<Animal> ();

}

public class Animal {

TreeSet <Consulta> consulta = new TreeSet <Consulta> ();

}

public class Conta {

TreeSet <Exame> exame = new TreeSet <Exame> ();

}

public class Historico {

TreeSet <Exame> exame = new TreeSet <Exame> ();

TreeSet <Consulta> consulta = new TreeSet <Consulta> ();

}

16 - A classe conta necessita de dois métodos que estão implementados em classes distintas.

17 - class desconto {

public:

double valorDesconto;

double gerarDesconto(double);

};

class acrescimoHorario {

public:

double valorAcrescimo;

double AcrescimoHorario(double);

};

class conta : public desconto, public acrescimoHorario{

double gerarDesconto(double);

double AcrescimoHorario(double);

};

18 - As interfaces distintas consistem em métodos que serão utilizados pela a classe conta, sendo assim serão feitas duas realizações para assim a classe conta implementar o método concreto.

20 – A delegação possibilita contornar a limitação de algumas linguagens de programação no quesito de herança múltipla.

19 –

public interface Acrescentavel {

public double acrescimoHorario();

}

public interface Descontavel {

public double gerarDesconto();

}

public class Conta implements Descontavel, Acrescentavel {

private String horarioEmergencia;

private tipoPagamento formaPag;

private static double valorFinal;

private double valor;

private double valorDesconto;

private int valorAcrescimo;

@Override

public double gerarDesconto() {

…

}

@Override

public double acrescimoHorario() {

…

}

}

22 - Faça um quadro comparativo entre reuso por generalização, realização e delegação, apresentando no mínimo duas vantagens e duas desvantagens para cada um desses conceitos.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Generalização | Realização | Delegação |
| Vantagem | Desempenho. | Manutenibilidade. | Alto encapsulamento. |
| Vantagem | Acoplamento forte. | Revelar as operações de um objeto sem revelar a sua classe. | Dinâmica. |
| Desvantagem | Diminuição do encapsulamento dependendo de sua profundidade. | Obrigação de ter que implementar todos os métodos. | Fraco acoplamento. |
| Desvantagem | Heranças múltiplas causam maior complexidade do sistema. | Dificuldade na reutilização do código devido aos métodos serem abstratos. | Baixo desempenho. |