Conclusões:

1) Classes e objetos
    a. Modelagem de um sistema a partir da definição das entidades do mundo real cujos dados serão manipulados pelo sistema computacional
        i. Diminuição do GAP entre a modelagem e a programação – diminuição do GAP entre Tipo abstrato de dados e o tipo de dados.
            1. Cada entidade do mundo real é candidata a ser uma classe no sistema computacional
            2. Os objetos são então instâncias da classe que tratam os dados das entidades
2) Trabalhamos fundamentos da Programação Orientada a Objetos:
    a. Uma classe por arquivo
    b. Toda classe inicia com Letra Maiúscula
    c. O arquivo tem o mesmo nome da classe
    d. A classe passa a ser um Tipo de Dados -> permite criar variáveis de trabalho para referenciar objetos da classe
    e. A variável de trabalho armazena o endereço onde o objeto está armazenado.
    f. O método da classe atua sobre o objeto que está no endereço armazenado na variável de trabalho.
    g. Copiar o conteúdo de uma variável de trabalho implica na nova variável "apontar" para o mesmo endereço da outra variável e, consequentemente, acessar por meio dos métodos o mesmo conteúdo. Mudar algum atributo por meio de uma dessas variáveis irá determinar a mudança do valor dos dados acessados pela outra variável.
    h. Quando instanciamos uma classe (criamos um objeto da classe por meio do comando new) é executado o método construtor da classe.
        i. O método construtor tem a função de inicializar os atributos do objeto instanciado.
        ii. Na POO, o método construtor tem o mesmo nome da classe e não apresenta valor de retorno.
        iii. Pode-se ter mais de um método construtor que permitam inicializar os valores dos atributos de forma diferenciada.
            1. Isso acontece graças ao Polimorfismo
    i. Polimorfismo consiste em definir duas ou mais funções/rotinas que desempenham uma tarefa que pode ser realizada de mais de uma maneira.
        i. <tipo de acesso> <valor de retorno> SELETOR(Lista de Parâmetros) (ASSINATURA)
        ii. No polimorfismo, o <valor de retorno> e o SELETOR são fixos, enquanto cada método possui a sua própria lista de parâmetros.
        iii. Quando um método é implementado na classe filha com a mesma ASSINATURA, o método da classe filha é invocado no lugar do método da classe mãe – Sobrecarga – Rescrita.
    j. Herança é a possibilidade de se criar uma classe a partir do código de uma outra classe.
        i. Classe mãe ou Super possui um código base
        ii. Classe filha é criada herdando o código base da Classe mãe e, inclui ou modifica o código recebido.

iii. Quando um objeto da classe filha é instanciado a partir do Método construtor Default, é executado primeiro o método construtor da Super Classe e após o método construtor da classe filha.
iv. Quando um objeto da classe filha é instanciado por qualquer um dos métodos construtores definidos para a classe filha, em primeiro lugar é executado o método construtor da Super Classe.
    1. Por causa disso, é importante que a definição dos métodos construtores padrão (default) sejam de tal forma gerais que as classes derivadas possam utilizar esses valores como os valores iniciais dos atributos herdados.
    2. Nas classes derivadas, o método construtor geralmente inicializa os valores dos atributos herdados da Super Classe que demandam valores iniciais específicos para a classe derivada.

k. Encapsulamento trata de um conjunto de ações que definem como os atributos de um objeto serão acessados.
    i. What are public, private and protected in object oriented programming?
        1. They are access modifiers and help us implement Encapsulation (or information hiding). They tell the compiler which other classes should have access to the field or method being defined.
            *a. **private** - Only the current class will have access to the field or method.*
            *b. **protected** - Only the current class and subclasses (and sometimes also same-package classes) of this class will have access to the field or method.*
            c. ***public*** *- Any class can refer to the field or call the method.*

l. **Por que herança é uma coisa boa?**
    i. Porque permite que sejam desenvolvidas Super Classes que serão usadas à exaustão para criar classes derivadas:
        1. Super Classe Pessoa: Classes Derivadas: Motorista, Aluno, Professor etc.
        2. A Super Classe contém os atributos e métodos genéricos para todas as Classes Derivadas
        3. Classes Derivadas incluem atributos e métodos específicos para atender as demandas das entidades que elas modelam.
        4. Se uma atualização de atributo ou método pode ser realizada na Super Classe, todas as classes derivadas passam a ter essa alteração a partir da recompilação de todo o código.

m. **Por que herança está caindo em desuso?**
    i. Porque causa forte acoplamento de código entre a Super Classe e as Classes Derivadas.
    ii. Passa a haver uma dificuldade maior de manutenção do código, principalmente quando começarem a surgir demandas de alteração em atributos e métodos nas Classes Derivadas, específicas de uma

classe derivada sobre os atributos e métodos que foram herdados da Super Classe.

iii. Herança pode ser o mal da Orientação a Objetos: https://objectpascalprogramming.com/posts/heranca-pode-ser-o-mal-da-orientacao-a-objetos-parte-1/

iv.