OBS: As questões cuja resposta não tem referência, são de elaboração própria

1.1. Por ser uma forma mais eficaz de se reutilizar código, que mantém a linearidade e a concisão, permitindo melhor manutenção.

1.2. A Programação Orientada a Objetos (POO) surgiu no final da década de 1960, com a linguagem Simula-68. Alan Kay, um cientista da computação americano, é considerado um dos principais responsáveis pela criação da POO.

REFERENCIA:https://materialpublic.imd.ufrn.br/curso/disciplina/1/8/1/3#:~:text=Orientada%20a%20Objetos-,Como%20Tudo%20Come%C3%A7ou,mensagens%20para%20constru%C3%A7%C3%A3o%20de%20programas

1.3. A modularidade, a forma como os dados e funções do código são tratados.

2. Estruturando o código em blocos menores com funcionalidades limitadas que se integram entre si,
tornando a manutenção e documentação mais fáceis.

3. Na linguagem de programação C, que segue o paradigma imperativo/estruturado, a reutilização de código é
alcançada principalmente por meio de funções e módulos. Ao dividir o código em funções bem definidas e
organizadas em arquivos separados, podemos reutilizar e manter o código de forma mais eficiente.

REFERENCIA:https://www.cs.rochester.edu/u/ferguson/csc/c/tutorial/reusable/index.html?utm_source=chatgpt.com

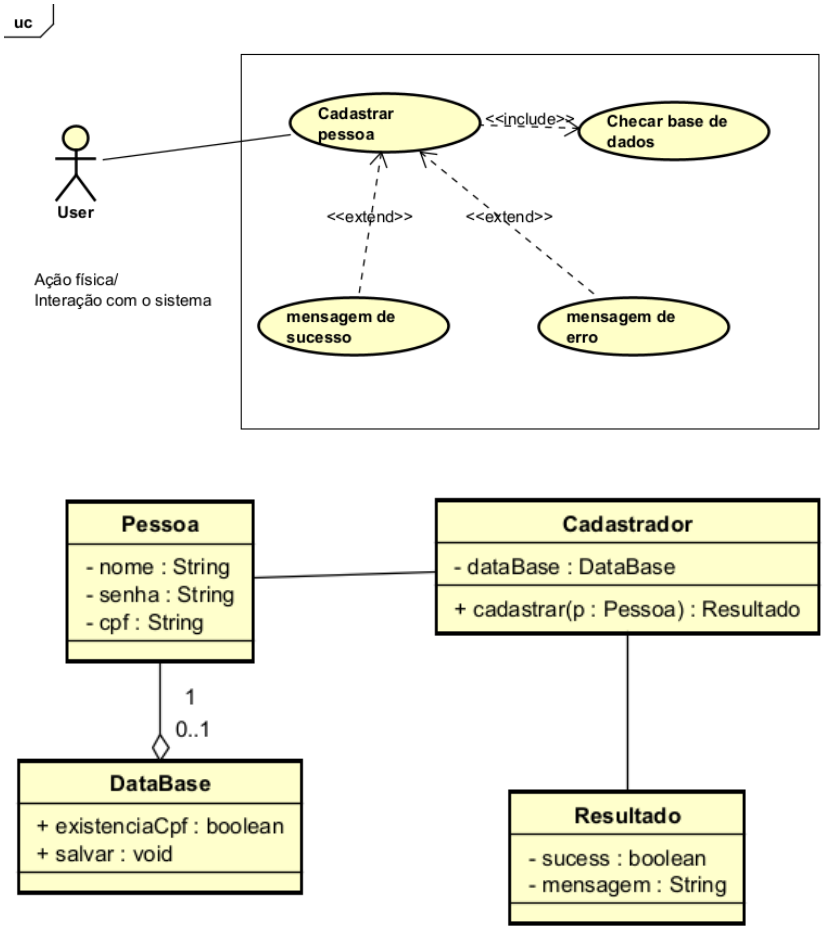
4. A struct é uma estrutura de dados que agrupa variáveis,
enquanto a classe é um tipo de referência que tem métodos próprios.

REFERENCIA:
https://www.alura.com.br/artigos/c-sharp-entendendo-classes-e-structs#:~:text=As%20structs%2C%20muito%20comuns%20na,%C3%A9%20um%20tipo%20de%20refer%C3%AAncia.

5.1. Para diferenciar o que será transformado em código do que será apenas uma interação com o sistema, deve-se identificar se a ação envolve lógica de negócio, processamento ou armazenamento de dados, nesse caso, é código. Interações simples, como entrada de dados ou navegação entre telas, geralmente são apenas interface. Ferramentas como casos de uso e histórias de usuário ajudam nessa separação.

5.2. Uma demanda que será transformada em código deve ser apresentada por meio de histórias de usuário com objetivos e critérios de aceitação bem definidos. Além disso, o uso de protótipos, diagramas e ferramentas de gestão ágil facilita a compreensão da equipe sobre o que deve ser implementado.

5.3.



7.1. Não, um objeto, ainda que “vazio” ocuparia mais espaço que uma variável de tipos primitivos, pois, o compilador, dependendo da linguagem, aloca certa quantidade de memória.

7.2. Java usa o heap para armazenar objetos, e há overhead do objeto (geralmente 8 a 16 bytes por objeto para metadados). Já para variáveis de tipos primitivos ele apenas aloca a quantidade necessária para manter um dado daquele tipo.

REFERENCIA:https://www.reddit.com/r/java/comments/1grag6v/java_24_to_reduce_object_header_size_and_save

8. Acesso direto é quando o compilador sabe exatamente onde o valor está armazenado na memória. Acesso indireto, quando o acesso é feito por “ponteiros” que guardam o local de acesso em outra variável. Em C, utiliza-se uma variável para armazenar a referência de outra, tornando assim o acesso indireto. Caso contrário utiliza-se o nome da variável, acessando assim, diretamente.

Java não usa ponteiros explícitos; em vez disso, trabalha com referências seguras a objetos. A liberação da memória é feita pelo Garbage Collector, sem necessidade de free() como em C.
REFERÊNCIA:https://www.digitalocean.com/community/tutorials/java-heap-space-vs-stack-memory

9. Método construtor é aquele chamado para a inicialização do objeto, ele tem o mesmo nome da classe e existe mesmo se não for declarado. Sim, é possível existir mais de um método construtor, pra isso é necessário que a assinatura dele seja diferente.

10.1. Ele possui o mesmo nome que a classe a qual ele pertence.

10.2. Não necessariamente, apenas se houver a necessidade de atribuição no momento da criação.

11. Static diz respeito a um atributo ou comportamento que pertence a classe, ou seja, não depende de dados específicos para realizar uma ação ou conter um valor. Já final refere-se a um valor ou método imutável, em todo caso, constante.

12. Significa que o método ou atributo pertence a classe e não a um objeto específico. Por exemplo, um comportamento que se estende a todas as instâncias dessa classe ou um atributo comum a todas elas.

13. I. Erro: valor1 não foi declarada como static; sendo assim não é possível acessá-la sem referenciar um objeto do tipo Classe.

II. 0

III. Erro: valor3 foi declarada mas não foi inicializada.

IV. Erro: instância1 foi declarada mas não inicializada.

V. Imprime o toString default para a instância2

VI. Erro: instância1 foi declarada mas não inicializada.

VII. Acho que seria 0.

VIII. Erro: a valor3 é local e não pertence a classe.

IX. 0

X. 0

XI. Erro: a valor3 é local e não pertence a classe ou ao objeto.