Shira Pahmer 575285308
Maeven Fanebust 559816034

Question 1 (out of 2):

Part a: Basic design of Encoder-Decoder system

The following requirements pertain to a system responsible for encryption and decryption:

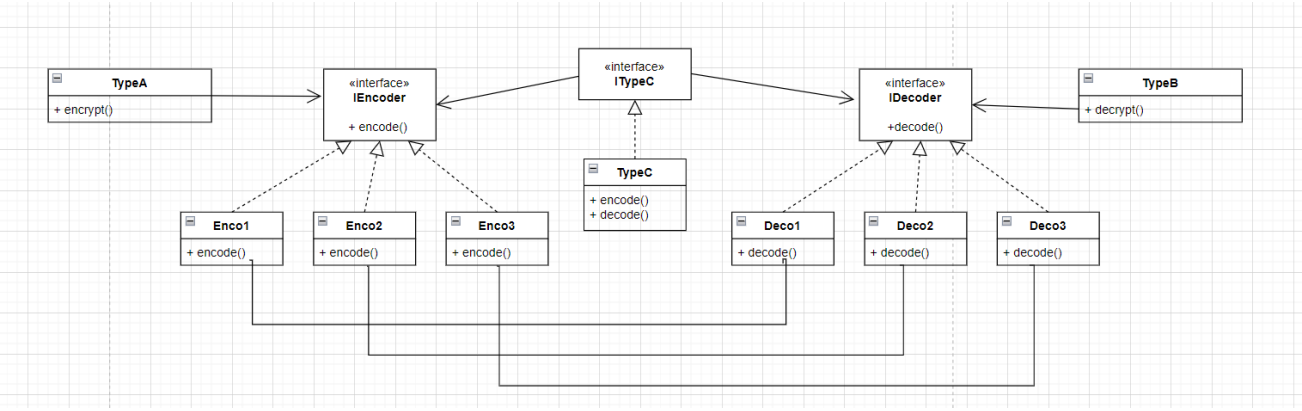
The system comprises encoders and decoders. Encoders have the *encode* operation and Decoders have the *decode* operation.

Within the system, there exist three distinct types, denoted as A, B, and C:

- Type A is dedicated to encryption tasks. It has the *encrypt* operation, for which it relies on Encoder-type objects and calls their *encode* operation.
- Type B is dedicated to decryption tasks. It has the *decrypt* operation, for which it relies on Decoder-type objects and calls their *decode* operation.
- Type C is multifaceted, capable of both encryption and decryption. This type requires objects of both Encoder and Decoder types, utilizing their respective *encode* and *decode* operations.

The system includes three types of encoders: Enco1, Enco2, and Enco3, each with corresponding counterparts in the decoder category: Deco1, Deco2, and Deco3.

Describe a design (using a class diagram) supporting the following requirements and complying with (not violating) SOLID principles:

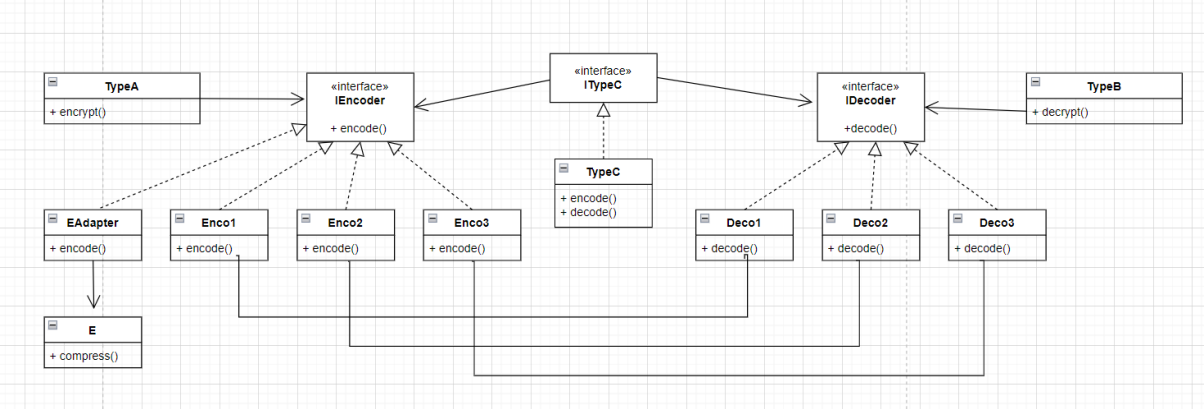


Part b: Introducing an external class into the system

Assume you find in the web a compiled *E* class, having an operation *compress*. We would like to use *E* as an additional *Encoder* (Where *compress* corresponds to *encode*). However, we are not allowed to change E's code or even recompile it. Draw a new diagram incorporating *E* into the system. Indicate the design pattern you used.

Clarification: Normally, each encoder requires a corresponding decoder. However, in this part your are not required to add the decoder.

We use the object adapter pattern



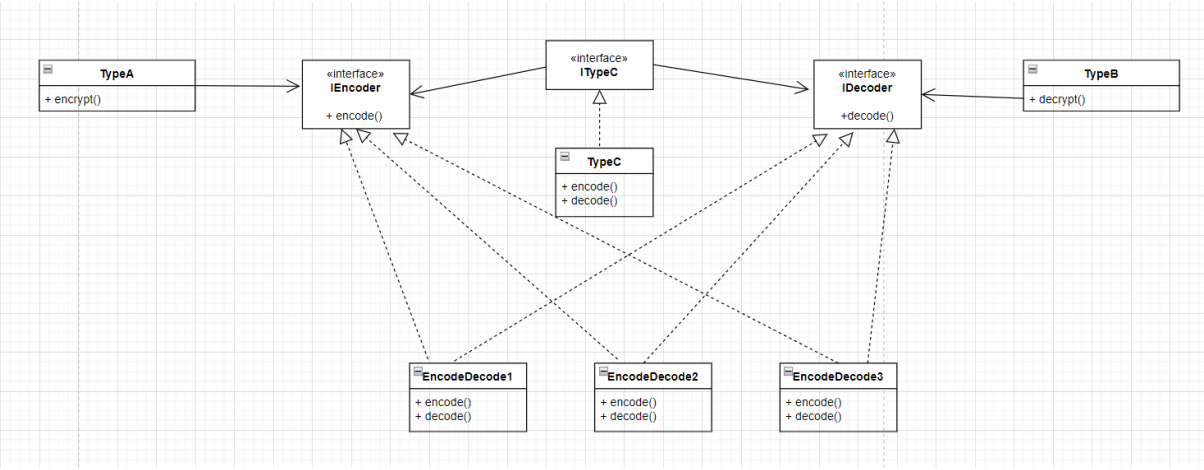
Part c: Adding Drinkers

(Ignore part b) As each specific Encoder operation is based on a particular encoding algorithm, and its corresponding decoder must mirror the encoding process, it is apparent that segregating the encode and decode operations into two distinct classes (Encoder and Decoder) violates one of the SOLID principles. Specify which SOLID principle is violated.

Therefore, we decided to amalgamate each pair of EncoderX and DecoderX classes into a single class named EncoderDecoderX. For instance, instead of having separate classes for a specific encoder type like Enco1 and its corresponding decoder Deco1, we will now have a single class called EncoDeco1. This consolidated class will be capable of executing both encode and decode operations.

Update the design (draw a new diagram), so that it supports the requirements and adheres to the SOLID principle. Specify the principles you have considered during this process.

Keeping the encoder and decoder classes segregated is a violation of the dependancy inversion program because the encoders and decoders have a mutual dependancy that way. By combining them into one class we avoid that issue. In the new diagram we make sure that DIP is not violated and also avoiding any ISP issues.



Question 2:

Describe a design (using a class diagram) supporting the following requirements and complying with (not violating) SOLID principles:

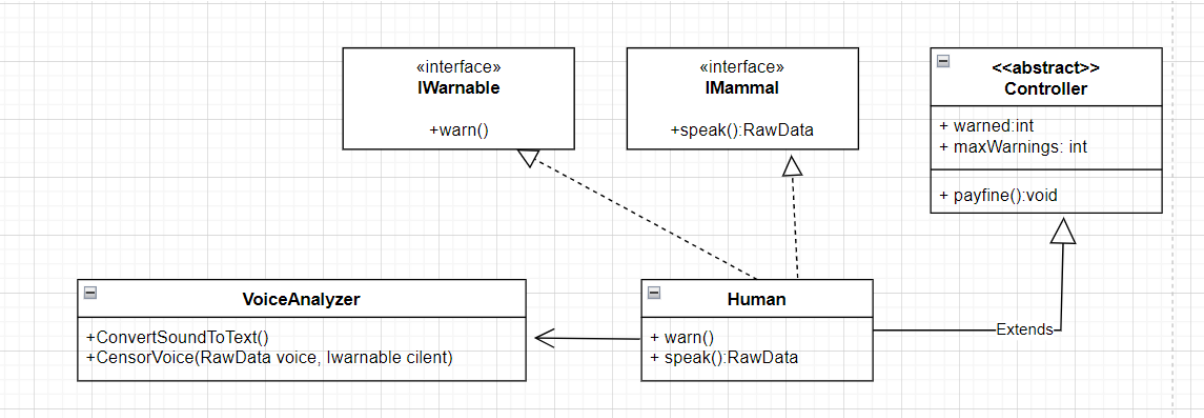
The system represents *Mammal*s. Each *Mammal* has a *speak* operation, generating *RawData* representing the *Mammal*'s voice.

A *Human* is a type of *Mammal*. Every invocation of *speak* by a *human* should be followed by a voice-analysis process. If the *Human*'s voice contains certain words (The list of words may change from one execution to another), the *Human* will be *warn*ed. A *Human* with 3 warnings executes a function *payFine*.

Apart from *Human*, there are additional *Mammal*s (e.g., *Dog*), but *Human* is the only *Mammal* that can be *warn*ed. There may be other non-*Mammal* *warn*-able classes that generate voice (e.g., *Parrot*, *Gramophone*)

In your design incorporate the following functions (You should decide in what class they belong):

- List<String> *convertSoundToText* (*RawData* voice): returns a list of words that the voice contains.
- void *CensorVoice*(*RawData* voice, ? client): Analyzes the voice and if it contains certain words, *warn*s the client (You should decide what should be the client's type)
- void *warn*(): Updates the object that it has been *warn*ed.



Draw a class diagram that meets the requirements and complies (does not violate) the SOLID principles. Note that you must place the described functions into classes in the diagram (based on the SRP principle).

Guiding questions (there is no need to include the answers in the submission. The purpose of the questions is to help you define the design):

- Which of the functions can be reused in a variety of applications beyond the application described?
- Can the described process be defined as a series of actions belonging to separate responsibilities?
- What are the abstractions that fit the description and what are the concrete modules (that have implemented content)?