# TypeScript – Exam Preparation – 11 June 2025

## Language Message Encoder

Your task is **implement** **the** **class** **LanguageMessageEncoder**, which allows for encoding and decoding secret messages using a compatible language set and a given cipher algorithm.

### Skeleton code

You are provided with a skeleton that comes with some files.

#### Contracts

In the **/contracts** folder of your skeleton you’ll be provided with some files, that the **LanguageMessageEncoder** class relies on or is meant to work with. You **MUST NOT add/remove or change any files** in the /contracts folder - during evaluation the **/contracts** folder in your project will be swapped with the default one from the skeleton, so **any changes you made in the folder WILL be lost**.

* **messageEncoder.ts** – contains the interface that all MessageEncoders should implement.
* **language.ts** – contains the interface for a Language, all language classes should implement this interface.
* **cipher.ts** – contains the interface for a Cipher, all cipher classes should implement this.
* **caesarCipher.ts** – an implementation for a Cipher that encodes messages by shifting each character forward in the alphabet, provided to help you understand the functionality of a **Cipher**.
* **lowercaseCharactersOnlyLanguage.ts** – an implementation of the Language class consisting of all lowercase English characters, provided to help you understand the functionality of a **Language** class.
* **partialMessageEncoder.ts** **–** an abstract class with some existing implementation for a MessageEncoder, the **LanguageMessageEncoder** class should inherit this one.

#### Additional Files

You are also provided with some additional files:

* **decorators.ts** - a decorators file that features the empty decorator functions linked to contracts classes.
* **languageMessageEncoder.ts** - the empty implementation of the **LanguageMessageEncoder** class
* **index.ts** - a file that you can use for testing, see the Examples section bellow

You are **free to add or modify any files** **outside of the /contracts** folder as you see fit.

#### Skeleton explanation

Aside from the interfaces in the skeleton, you are given the following implementations:

**LowercaseCharactersOnlyLanguage** – an implementation of the **Language** **interface**, a language consists of a:

1. **charset** – a private Set consisting of all string characters that are part of the language in increasing order, exposed using only a getter. The Set should be type limited to only allow characters that should be part of the charset and prevent adding characters in, that are not part of the language.
2. **isCompatibleToCharset** – a function that checks if a given secret message can be encoded/decoded with the provided charset. Each language may have different logic for whether a message is compatible, but for the **LowercaseCharactersOnlyLanguage**, the condition is that the secret message consist only of lowercase english characters.

**CaesarCipher** – an implementation of the generic **Cipher** **interface**, a cipher should have a generic parameter of the Language interface and should consist of a:

1. **language** – a private instance of the Language interface, exposed only through a getter.
2. **encode** – a public method that takes a secret message and returns its encoded version in the cipher’s language
3. **decode** – a public method that takes a secret message and returns its decoded version in the cipher’s language
4. A constructor that takes in a language and optionally any other parameters needed for the cipher.

### Tasks

You will be scored based on the following 3 tasks: Functionality, Structure and Decoration.

#### Functionality (25 points)

The **LanguageMessageEncoder** class should:

* Strip forbidden symbols from a message before it is passed for encoding, using the **PartialMessageEncoder** class’s **forbiddenSymbols** array**.**
* Keep track of the total number of sucessfully encoded and decoded characters.

Implement the following methods from the **MessageEncoder** interface:

1. **Method encodeMessage(message)** – validates and encodes a passed in secret message using the following logic:
   1. In case the passed value is not a string or has a length of 0 characters returns the string **"No message."**
   2. In case the message, after being stripped from forbidden symbols is not compatible with the language, returns the string **"Message not compatible."**
   3. Otherwise gets the message stripped from forbidden symbols, encodes it using the provided cipher and returns it.
2. **Method decodeMessage(message)** – validates and decodes a passed in secret message using the following logic:
   1. In case the passed value is not a string or has a length of 0 characters returns the string **"No message."**
   2. In case the message is not compatible with the language, returns the string **"Message not compatible."**
   3. Otherwise decodes the message using the provided cipher and returns it.
3. **Method** **totalProcessedCharacters(type)** – returns a message containing the total number of processed characters in the format **"Total processed characters count: <totalProcessedCharacters>"** where **totalProcessedCharacters** is based on the **type** parameter:
   1. **Type = "Encoded"** – The sum of the string length of all messages that were sucessfully encoded (after being stripped of forbidden symbols)
   2. **Type = "Decoded"** – The sum of the string length of all messages that were sucessfully decoded
   3. **Type = "Both"** – The sum of the above 2 cases (Encoded and Decoded)

#### Structure (50 Points)

Your **LanguageMessageEncoder** implementation should match the following structure requirements:

**NOTE:** These tasks will only award points if the underlying functionality also exists and works as intended.

1. The **LanguageMessageEncoder** should inheritthe **PartialMessageEncoder** abstract class and have method signatures compatible with the **MessageEncoder** **interface**. You are free to override any methods from **PartialMessageEncoder** as you see fit.
2. The **LanguageMessageEncoder** class should only allow to be initializedwith objects **compatible** with the **Language** and **Cipher** **interfaces**.
3. The **LanguageMessageEncoder** class should be a **generic** **class** accepting 2 generic parameters - a **Language** and a **Cipher** **compatible with that language**. Encoder instances instantiated with for instance the LowercaseCharactersOnlyLanguage, should also only take a cipher compatible with the LowercaseCharactersOnlyLanguage.
4. Ensure that any code and implementations written by you, do not use type assertions to the **any** type
5. Implement a new **DNACodeLanguage** class that implements the Language interface and uses the following functionality:
   1. Its charset consists and allows only the string characters for DNA bases (**A, C, G, T** in that order)
   2. Considers messages compatible if they also only consist of string characters for DNA bases (**A, C, G or T**)
6. All **literal** and **advanced types** in the **LanguageMessageEncoder** and implementations written by you should be replaced with equivalent **type aliases** for easier readability – **reuse interfaces where possible**. This task requires **DNACodeLanguage** to be implemented otherwise it rewards reduced points.
7. **All access modifiers** of properties and methods **in your code**, **that are not part of the interfaces or the provided implementations**, should use either the **private or protected** access modifiers. This task requires **DNACodeLanguage** to be implemented otherwise it rewards reduced points.

#### Decoration (25 Points)

You are tasked to change the functionality of 2 existing classes:

* **CaesarCipher -** Your task is to change the **offset** the cipher uses from 2 places to 3 places. You are **NOT** **allowed** to directly modify the files inside the **/contracts** folder.
* **PartialMessageEncoder** – Your task is to change the **forbidden symbols** that the message encoder removes to include **"(double quotes)** and **'(single quotes)** to the currently existing ones. You are **NOT allowed** to directly modify the files inside the **/contracts** folder.

**IMPORTANT:** The modifications should change the class definitions themselves – check Example 6 to see that the expected modifications are directly available on the **CaesarCipher** and **PartialMessageEncoder** class definitions.

* Modifying the functionality in a derived class like **LanguageMessageEncoder**, while the functionality remains unchanged in **PartialMessageEncoder will award NO points.**

### Examples

This is an example of how the **LanguageMessageEncoder** class is **intended to be used**:

* Examples 1-3 use the non-modified **CaesarCipher** with **offset of 2** and **original forbiddenSymbols**.
* Examples 4-5 shows the new **DNALanguage** class using the non-modified CaesarCipher with offset of 2 and original forbiddenSymbols.
* Example 6 uses the modified **CaesarCipher** with **offset of 3** and the **PartialMessageEncoder** with **modified** **forbiddenSymbols** that include single and double quotes.

|  |
| --- |
| **Example 1** |
| let lowercaseCharsLanguage = new LowercaseCharactersOnlyLanguage();  let caesarCipher = new CaesarCipher(lowercaseCharsLanguage);  let encoder = new LanguageMessageEncoder<LowercaseCharactersOnlyLanguage, CaesarCipher<LowercaseCharactersOnlyLanguage>>(lowercaseCharsLanguage, caesarCipher);  let encodedMessage = encoder.encodeMessage('the carthaginians are coming');  console.log(encodedMessage);  let decodedMessage = encoder.decodeMessage(encodedMessage);  console.log(decodedMessage);  console.log(encoder.totalProcessedCharacters('Both'));  let encodedMessage2 = encoder.encodeMessage('!abc\_');  console.log(encodedMessage2);  let decodedMessage2 = encoder.decodeMessage(encodedMessage2);  console.log(decodedMessage2);  console.log(encoder.totalProcessedCharacters('Encoded')); |
| **Output 1** |
| **vjgectvjcikpkcpuctgeqokpi**  **thecarthaginiansarecoming**  **Total processed characters count: 50**  **cde**  **abc**  **Total processed characters count: 28** |

|  |
| --- |
| **Example 2** |
| let lowercaseCharsLanguage = new LowercaseCharactersOnlyLanguage();  let caesarCipher = new CaesarCipher(lowercaseCharsLanguage);  let encoder = new LanguageMessageEncoder<LowercaseCharactersOnlyLanguage, CaesarCipher<LowercaseCharactersOnlyLanguage>>(lowercaseCharsLanguage, caesarCipher);  let encodedMessage = encoder.encodeMessage(undefined);  console.log(encodedMessage);  let decodedMessage = encoder.decodeMessage(true);  console.log(decodedMessage);  let encodedMessage2 = encoder.encodeMessage(undefined);  console.log(encodedMessage2);  let decodedMessage2 = encoder.decodeMessage(true);  console.log(decodedMessage2);  let encodedMessage3 = encoder.encodeMessage('1abc');  console.log(encodedMessage3);  let decodedMessage3 = encoder.decodeMessage('test"12"');  console.log(decodedMessage3); |
| **Output 2** |
| **No message.**  **No message.**  **No message.**  **No message.**  **Message not compatible.**  **Message not compatible.** |

**Note:** This example is meant to check correct typization and constraints, the comments in the output are not the result of runtime execution, but errors that TS should flag on the red text in the input.

|  |
| --- |
| **Example 3** |
| let lowercaseCharsLanguage = new LowercaseCharactersOnlyLanguage();  let caesarCipher = new CaesarCipher(lowercaseCharsLanguage);  lowercaseCharsLanguage.charset.add('A')  let encoder = new LanguageMessageEncoder<LowercaseCharactersOnlyLanguage, CaesarCipher<Language>>(lowercaseCharsLanguage, caesarCipher);  encoder.totalProcessedCharacters('age'); |
| **Output 3** |
| **//TypeScript Error: "Argument of type 'А' is not assignable to type**  **//TypeScript Error: CaesarCipher<Language> does not satisfy the constraint**  **//TypeScript Error: Argument of type '"age"' is not assignable to parameter of type '"Encoded" | "Decoded" | "Both"'** |

|  |
| --- |
| **Example 4** |
| let dnaCharsLanguage = new DNACodeLanguage();  let caesarCipher = new CaesarCipher(dnaCharsLanguage);  let encoder = new LanguageMessageEncoder<DNACodeLanguage, CaesarCipher<DNACodeLanguage>>(dnaCharsLanguage, caesarCipher);  let encodedMessage = encoder.encodeMessage('ACGT');  console.log(encodedMessage);  let decodedMessage = encoder.decodeMessage(encodedMessage);  console.log(decodedMessage);  let encodedMessage2 = encoder.encodeMessage('GAGCCCTCA');  console.log(encodedMessage2);  let decodedMessage2 = encoder.decodeMessage(encodedMessage2);  console.log(decodedMessage2);  let decodedMessage3 = encoder.decodeMessage('AGGCAT');  console.log(decodedMessage3);  let decodedMessage4 = encoder.decodeMessage('DACG');  console.log(decodedMessage4);  console.log(encoder.totalProcessedCharacters('Both')); |
| **Output 4** |
| **GTAC**  **ACGT**  **AGATTTCTG**  **GAGCCCTCA**  **GAATGC**  **Message not compatible.**  **Total processed characters count: 32** |

**Note:** This example is meant to check correct typization and constraints, the comments in the output are not the result of runtime execution, but errors that TS should flag on the red text in the input.

|  |
| --- |
| **Example 5** |
| let dnaCharsLanguage = new DNACodeLanguage();  dnaCharsLanguage.charset.add('B'); |
| **Output 5** |
| **//TypeScript Error: "Argument of type 'B' is not assignable to type** |

|  |
| --- |
| **Example 6** |
| let lowercaseCharsLanguage = new LowercaseCharactersOnlyLanguage();  let caesarCipher = new CaesarCipher(lowercaseCharsLanguage);  let encoder = new LanguageMessageEncoder<LowercaseCharactersOnlyLanguage, CaesarCipher<LowercaseCharactersOnlyLanguage>>(lowercaseCharsLanguage, caesarCipher);  let encodedMessage = encoder.encodeMessage('there is no "spoon"');  console.log(encodedMessage);  let decodedMessage = encoder.decodeMessage(encodedMessage);  console.log(decodedMessage);  let encodedMessage2 = encoder.encodeMessage("in cryptography, a 'cipher' is an algorithm for performing encryption or decryption - a series of well-defined steps that can be followed as a procedure.");  console.log(encodedMessage2);  let decodedMessage2 = encoder.decodeMessage(encodedMessage2);  console.log(decodedMessage2);  console.log(encoder.totalProcessedCharacters('Both'));  let encodedMessage3 = encoder.encodeMessage('\_test;b1c2');  console.log(encodedMessage3);  let encodedMessage4 = encoder.encodeMessage('\_test;b-c');  console.log(encodedMessage4);  let decodedMessage3 = encoder.decodeMessage('"hello" he said');  console.log(decodedMessage3);  console.log(encoder.totalProcessedCharacters('Decoded'));  console.log(PartialMessageEncoder.forbiddenSymbols); |
| **Output 6** |
| **wkhuhlvqrvsrrq**  **thereisnospoon**  **lqfubswrjudskbdflskhulvdqdojrulwkpirushuiruplqjhqfubswlrqrughfubswlrqdvhulhvrizhooghilqhgvwhsvwkdwfdqehiroorzhgdvdsurfhgxuh**  **incryptographyacipherisanalgorithmforperformingencryptionordecryptionaseriesofwelldefinedstepsthatcanbefollowedasaprocedure**  **Total processed characters count: 274**  **Message not compatible.**  **whvwef**  **Message not compatible.**  **Total processed characters count: 137**  **(10) ['\_', ',', '.', '!', '?', '-', ';', ' ', '"', "'"]** |