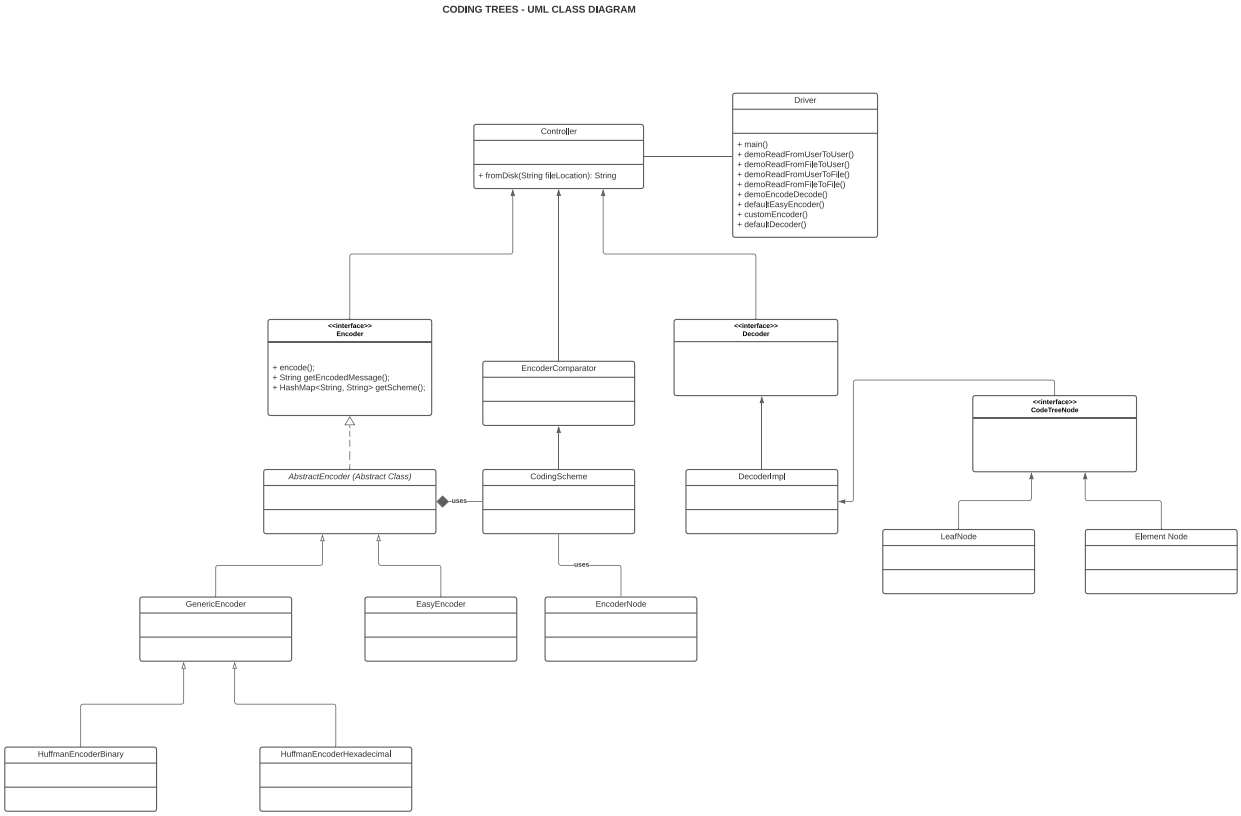
- **UML Class Diagram (Updated):**



- **Design Explanation:**

  - **Package Name:** enigma
  - **Encoder Interface:**
    - Consists of classes abstractencoder which implements the interface and this is where all the functions of an encoder resides.
    - The abstract class is extended by genericencoder and easy encoder.
    - The generic encoder is further extended by huffmanbinary and huffmanhexadeximal class.
    - The abstractencoder uses codingscheme class for generate the prefix coding dictionary for encoding.
    - The coding scheme class uses the encoder node class to represent the character – frequency relationship in a priority queue.
  - **Decoder Interface:**
    - This interface is implemented by the decoderimpl class.
    - This decoderimpl class uses the decoder tree under the hood to process the decoding.
    - The decoder tree is represented by the CodeTreeNode class.
    - It consists of 2 classes namely element node and leaf node.
  - **Controller Class:**
    - The controller class is where all the different types of encoding and decoding combinations happen.
  - **Driver Class:**

- The driver class consists of the main function that runs the program and can be played interactively by the user.

- **Testing Plan:**

  The classes to be tested are tested in separate test classes, to enable modularity in testing and also to cover all the functionalities of the respective classes.

  - **Test 1:** To check if the encoder object is correctly created.
  - **Test 2:** To check if the input text is correctly read from the user.
  - **Test 3:** To check if the input text is correctly read from the file.
  - **Test 4:** To check if the frequency dictionary is correctly created from the input message.
  - **Test 5:** To check if the priority queue is correctly populated based on increasing order of frequency and tie breaker is the lexical order between the characters.
  - **Test 6:** To check if the encoding is correctly happening. That is the characters are correctly encoded based on the given code dictionary.
  - **Test 7:** To check if the code dictionary is correctly generated in cases when it is not provided to us. Like Huffman encoding, and any other generic encoding cases where only code set is given.
  - **Test 8:** To check if the code dictionary can be saved to a file without an issue.
  - **Test 9:** To check if the dictionary is created as per the logic followed. That is, less frequent characters are pushed to bottom labyrinths of the tree and high frequency characters are in shallow nodes.
  - **Test 10:** To check if the Huffman binary encoding is correctly happening.
  - **Test 11:** To check if the Huffman hexadecimal encoding is correctly happening.
  - **Test 12:** To check if the Generic encoding is correctly happening.
  - **Test 13:** To check if the decoder is working perfectly. Given the encoded message and the prefix encoding dictionary.
  - **Test 14:** To check if the decoder tree is correctly generated.
  - **Test 15:** To check if the decoder tree is used correctly to decode the encoded message.
  - **Test 16:** To check if the decoder output is correctly saved to a file/printed to a screen.
  - **Test 17:** To check if the controller is correctly functioning.
  - **Test 18:** To check if the conversion of binary encoding to hexadecimal encoding working as expected.
  - **Test 19:** To check if the conversion of hexadecimal encoding to binary encoding working as expected.
  - **Test 17 – 30:** To check for other individual public methods of the classes Coding Scheme, Decoder utils, tree nodes (Element and Leaf) including a wider coverage of testing.

- **Assumptions Made:**
  - When encoding using a code dictionary that isn't having enough coverage (meaning it may don't have code for some characters that may appear in the input text).
  - The structure of prefix encoding while saving it to a text file (another assumption to save it as text file and not as json).

- Have used hashmap to bring a relationship between one node to another in the decoding tree structure.
- In decoder tree, the element node will not have any data and it will have only children (meaning link to other nodes) and the leaf node will have only data and no children.

- **Challenges Faced:**
  - The biggest challenge I faced is when choosing datastructures to represent trees, and when adding elements to the priority queue.
  - Use of relationships between classes were a challenge. That is, whether to use “Association” or “Composition” relationship between the encoder class and the coding scheme class.
  - The traversal of tree to print the reverse prefix encoding dictionary. That is code -> symbol.
  - Initially, thought of saving the prefix encoding dictionary as json and the parsing seems like an overhead for this assignment.