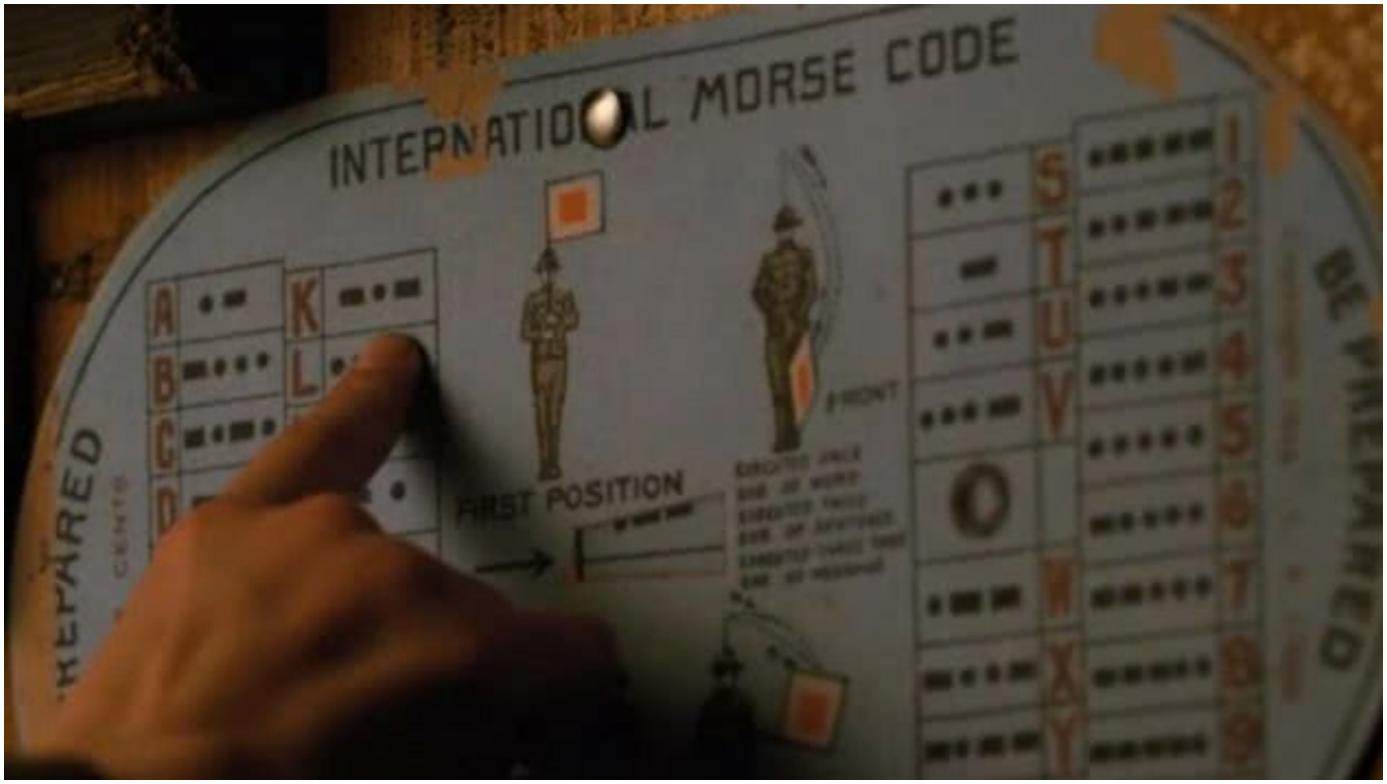
Instituto Superior de Engenharia do Porto

PROJETO III ESINF: CÓDIGO MORSE



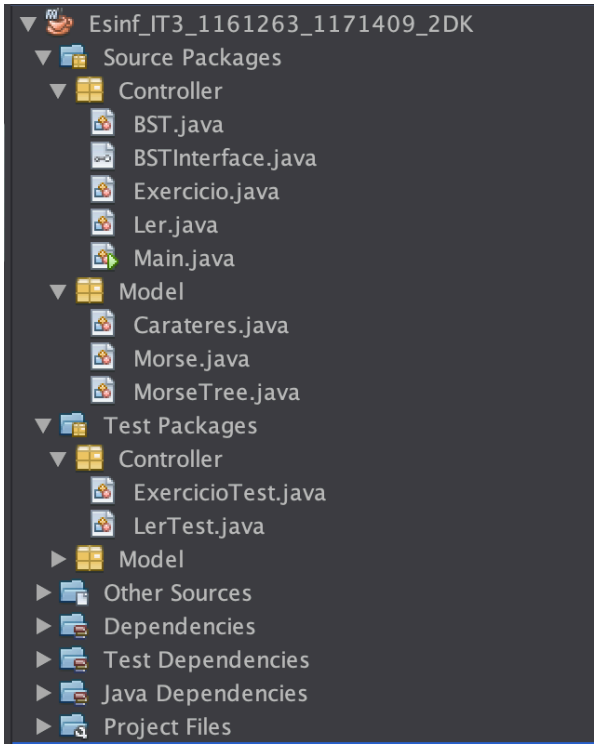
Trabalho realizado por:

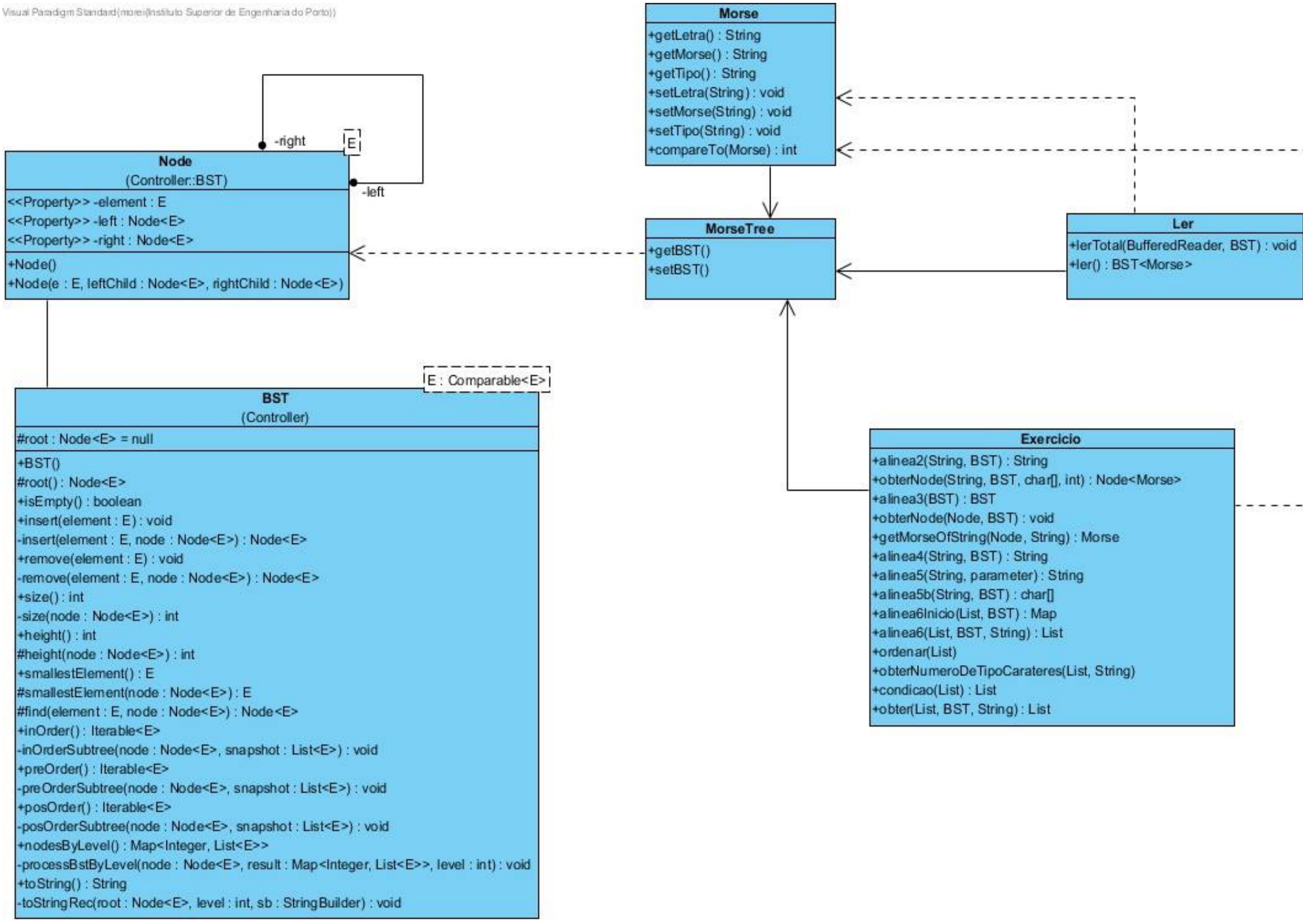
João Flores (1171409);
José Mota (1161263);

O Código Morse é um sistema de transmissão de informação baseada em séries intermitentes de sons, luzes, traços ou cliques representando letras, números e sinais de pontuação[1]. Desenvolvido por Samuel Morse em 1835 para ser aplicado no telegrafo, é utilizado ainda nos dias de hoje tanto em radioamadorismo como em sistemas de rádio orientação para navios e aeronaves. Um exemplo é o sistema de radiofarol não direcional (em inglês Non-Directional Beacon). Em comparação com outros sistemas de comunicação modernos, o código morse mantém a vantagem da facilidade de descodificação e codificação por humanos, exemplo disso é a popular sequência SOS:

...___...

Para este trabalho foram criadas as classes presentes na figura seguinte, afim de resolver questões como por exemplo, descodificar e codificar uma palavra, encontrar constituintes do código morse comum entre duas letras.





Ler (Alinea1) :

Metodo para ler o ficheiro e criar a bst, começamos por instanciar o uma bst no qual seu elemento é um objeto do tipo morse que contem o char, o tipo e a carater correspondente.Após obter a bst colocamos na root um elemento node inicial com “Start” para facil perceção de qual é a root. Criamos um File com o nome do ficheiro disponivel para criar a bst, um buffered reader para puder ler o ficheiro como um scanner, mas este tipo poder obter certos carateres usa-se “UTF8”.

No metodo lerTotal o enquanto o reader tiver linhas cria-se um vetor de string que separa a linha por espaços, cria um morse com a ordem respetiva dos elementos do vetor de modo a estar em concordancia com o construtor e de seguida insere na bst. Este insert tem como auxílio para introduzir pela ordem correta, conforme o morse, o compareTo que direciona para a esquerda se for menor e para a direita se for maior.

```java
public class Ler extends BST<Morse> {

    /**
     * file para ler a colocar na bst
     */
    private static final String file_name = "morse_v3.csv";

    public static BST<Morse> ler() throws FileNotFoundException, UnsupportedEncodingException, IOException {
        MorseTree mt = new MorseTree();
        BST<Morse> bst = mt.getBst();
        bst.insert(new Morse("Start", "Start", "Start"));
        File fileDir = new File(file_name);
        BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream(fileDir), "UTF8"));
        lerTotal(in, bst);
        in.close();
        return bst;
    }

    public static void lerTotal(BufferedReader in, BST<Morse> bst) throws IOException {
        String str;
        while ((str = in.readLine()) != null) {
            String tmp[] = str.split(" ");
            Morse m = new Morse(tmp[0], tmp[1], tmp[2]);
            bst.insert(m);
        }
    }
```

```
public static BST<Morse> ler() throws FileNotFoundException,
UnsupportedEncodingException, IOException {                    6*O(1) + 2*O(n) = O(n)
    MorseTree mt = new MorseTree();                                          O(1)
    BST<Morse> bst = mt.getBst();                                            O(1)
    bst.insert(new Morse("Start", " ", null));                               O(n)
    File fileDir = new File(file_name);                                      O(1)
    BufferedReader in=new BufferedReader(new InputStreamReader(new
FileInputStream(fileDir), "UTF8"));                                          O(1)
    lerTotal(in, bst);                                                       O(n)
    in.close();                                                              O(1)
    return bst;                                                              O(1)
  }

  public static void lerTotal(BufferedReader in, BST<Morse> bst) throws IOException {
  O(1)+2*O(n) = O(n)
    String str;                                                              O(1)
    while ((str = in.readLine()) != null) {                                    2*O(n)
      String tmp[] = str.split(" ");                                         O(n)
      Morse m = new Morse(tmp[0], tmp[1], tmp[2]);                           O(1)
      bst.insert(m);                                                         O(n)
    }
  }
```

Alinea 2:

Descodificar uma sequência de um código morse para a correspondente representação alfanumérica. Para tal é recebida uma string que representa varias letras em codigo morse separado por espaços. É criada uma string vazia para adicionar cada letra descodificada . Criamos um vetor com todos os elementos da string separada por espaços percoremos todas as posições do vetor das letras, é colocado num vetor todos os chars da string do for each, é chamado o metodo que retorna a letra e adiciona a string que vai ser returnada.

Para descodificar a letra é passada a string morse, o node que vai ter o elemento morse igual à string morse o vetor de char e um inteiro usado para avançar no vetor de char.

Neste metodo é começado por verificar se o node é nulo se for retorna null (caso base), compara-se tambem se a string corresponde ao morse do node atual, dado que este vai aproximando do node igual à string letra. Caso o carater seja um ponto o metodo é novamente chamando com um node a esquerda e incrementa o inteiro i, caso não seja usa o node à direita.

Este metodo necessita de outro caso base que termina quando a string da palavra é igual ao elemento do node.

```java
/**
 * metodo para traduzir de morse para string = letra+ letra
 *
 * @param morse string a traduzir
 * @param bst a arvore binaria
 * @return
 */
public static String alinea2(String morse, BST<Morse> bst) {
    String fim = "";
    int i = 0;
    String morseVec[] = morse.split(" ");
    for (String m : morseVec) {
        char[] c = m.toCharArray();
        fim += obterNode(m, bst.root(), c, i).getElement().getLetra();
    }
    return fim;
}

/**
 * metodo para obter os nodes conforme o morse
 *
 * @param m
 * @param node node atual
 * @param c arrray de char codigo morse para uma respetiva letra
 * @param i contador
 * @return
 */
private static Node<Morse> obterNode(String m, Node<Morse> node, char[] c, int i) {
    if (node == null) {
        return null;
    }
    if (m.compareTo(node.getElement().getMorse()) == 0) {
        return node;
    }
    if (i < c.length) {
        if (c[i] == '.') {
            return obterNode(m, node.getLeft(), c, ++i);
        }
        if (c[i] == '_') {
            return obterNode(m, node.getRight(), c, ++i);
        } else {
            Morse m1 = new Morse(" ", " ", " ");
            return new Node(m1, null, null);
        }
    }
    Morse m1 = new Morse("Erro", "Erro", "Erro");
    return new Node(m1, null, null);
}
```

```
public static String alinea2(String morse, BST<Morse> bst) {                    3*O(1)+O(n)^2= O(n)^2

        String fim = "";                                                        O(1)
        int i = 0;                                                              O(1)
         String morseVec[] = morse.split(" ");                                  O(n)
        for (String m : morseVec) {                                             O(n)* (2*O(n))=O(n)^2
                char[] c = m.toCharArray();                                     O(n)
                fim += obterNode(m, bst.root(), c, i).getElement().getLetra();  O(n)
        }
        return fim;                                                             O(1)
}

private static Node<Morse>obterNode(String m, Node<Morse> node, char[] c, int i) {
                                                                                4*O(1)+O(n) = O(n)
    if (node == null) {                                                         O(1)
      return null;                                                              O(1)
    }
    if (m.compareTo(node.getElement().getMorse()) == 0) {                       O(1)
      return node;                                                              O(1)
    }
    if (i < c.length) {                                                         O(n)
      if (c[i] == '.') {                                                        O(n)
        return obterNode(m, node.getLeft(), c, ++i);                            O(n)
      }if (c[i] == '_'){                                                        O(n)
        return obterNode(m, node.getRight(), c, ++i);                           O(n)
      }
      Else                                                                      2* O(1)
        Morse m1 = new Morse(" ", " ", " ");                                    O(1)
        return new Node(m1, null, null);                                        O(1)
    }
        Morse m1 = new Morse("Erro", "Erro", "Erro");                           O(1)
        return new Node(m1, null, null);                                        O(1)
  }
```

Alinea 3

Método para construir uma nova BST para codificação morse apenas das letras, utilizando a BST criada na alínea1(ler). Para este método, criamos uma nova BST, BST esta que apenas irá conter letras. Primeiramente, criamos o primeiro node, que será o início (“START”). De seguida, através do método obterNode (método recursivo) percorrer node a node a árvore, e vamos comparar com o tipo de cada node a fim de saber se pertence a “LETTER” ou não. Caso seja, adicionamos esse node á nova arvore criada no método principal (alinea3), caso contrário passamos aos nodes seguintes. Por fim retornamos a arvore criada, contendo todas as letras de “ A a Z”.

```java
/**
 * metodo para obter uma bst que so contenha morse do tipo carater
 * @param bst bst com todos os elementos
 * @return
 */
public static BST<Morse> alinea3(BST<Morse> bst) {
    MorseTree mt = new MorseTree();
    BST<Morse> bstLetter = mt.getBst();
    bstLetter.insert(new Morse("START", "START", "START"));
    obterNode(bst.root, bstLetter);
    return bstLetter;
}

/**
 *
 * @param node
 * @param bstLetter
 */
private static void obterNode(Node<Morse> node, BST<Morse> bstLetter) {
    if (node == null) {
        return;
    }
    if (node.getElement().getTipo().equalsIgnoreCase("Letter")) {
        bstLetter.insert(node.getElement());
    }
    obterNode(node.getLeft(), bstLetter);
    obterNode(node.getRight(), bstLetter);
}
```

```
public static BST<Morse> alinea3(BST<Morse> bst) {                    O(n)
    MorseTree mt = new MorseTree();                                   O(1)
    BST<Morse> bstLetter = mt.getBst();                               O(1)
    bstLetter.insert(new Morse("START", "START", "START"));           O(1)
    obterNode(bst.root, bstLetter);                                   O(n)
    return bstLetter;                                                 O(1)
}
private static void obterNode(Node<Morse> node, BST<Morse> bstLetter) {  O(n)
    if (node == null) {                                               O(1)
        return;                                                       O(1)
    }
    if (node.getElement().getTipo().equalsIgnoreCase("Letter")) {     O(1)
        bstLetter.insert(node.getElement());                          O(1)
    }
    obterNode(node.getLeft(), bstLetter);                             O(n)
    obterNode(node.getRight(), bstLetter);                            O(n)
```

Alínea 4:

Método para codificação de uma String passada por parâmetro. Para tal, decompomos a String a fim de passar cada careter que a constitui para um array de Chars. Através de um ciclo for, para cada letra(char) iremos buscar o morse (node) correspondente. Para tal convertemos o caracter em string e chamamos o método de procura getMorseOfString. Este método recebe como parâmetro a root(a origem) e o careter convertido para String. Neste método, vamos percorrer a BST partindo da root, iremos procurar o node que contem a String desejada. Caso seja encontrada damos retorno do node correspondente a letra, caso contrario, iremos verificar se o node da esquerda não está a null : caso se verifique avançámos para esse node se não avançámos para o da direita, este procedimento repete-se até que se encontre o node correspondente(recursividade). Após se obter o node, vamos buscar o código morse correspondente e adicionamos esse código a uma string inicialmente vazia. Por fim retornamos a string que contem o código morse correspondente

```java
/**
 *
 * @param word- String para a qual se vai efetuar a codificacao
 * @param bstLetter- bst criada na alinea 3 que vai conter todas as letras de "A a Z"
 * @return String contendo o codigo morse da String introduzida
 */
public static String alinea4(String word, BST<Morse> bstLetter) {
    String morseCode = "";
    char[] caracteres_palavra = word.toCharArray();

    for (char c : caracteres_palavra) {
        String charToLetter= String.valueOf(c);
        Morse letter=getMorseOfString(bstLetter.root(), charToLetter);
        if(letter!= null){
            morseCode += letter.getMorse()+" ";
        }
    }
    return morseCode;
}


 /**
 *
 * @param root
 * @param string
 * @return
 */
private static Morse getMorseOfString(Node<Morse> root, String string) {
    Morse left = null;
    Morse right = null;
    if (root != null) {
        if (root.getElement().getLetra().equals(string)) {
            return root.getElement();
        }
        if (root.getLeft() != null) {
            left = getMorseOfString(root.getLeft(), string);
        }
        if (root.getRight() != null) {
            right = getMorseOfString(root.getRight(), string);
        }
        if (left != null) {
            return left;
        } else if (right != null) {
            return right;
        }
    }
    return null;
}
```

```
public static String alinea4(String word, BST<Morse> bstLetter) {                    O(n^3)
    String morseCode = "";                                                           O(1)
    char[] caracteres_palavra = word.toCharArray();                                  O(n)

    for (char c : caracteres_palavra) {                                   O(n)
      String charToLetter= String.valueOf(c);                             O(1)
      Morse letter=getMorseOfString(bstLetter.root(), charToLetter);      O(n)      O(n2)
      if(letter!= null){                                                  O(1)
        morseCode += letter.getMorse()+" ";                               O(1)
      }
    }
    return morseCode;                                                                O(1)
  }


private static Morse getMorseOfString(Node<Morse> root, String string) {    O(n)
    Morse left = null;                                          O(1)
    Morse right = null;                                         O(1)
    if (root != null) {                                         O(1)
      if (root.getElement().getLetra().equals(string)) {        O(1)
        return root.getElement();                               O(1)
      }
      if (root.getLeft() != null) {                             O(1)
        left = getMorseOfString(root.getLeft(), string);        O(n)
      }
      if (root.getRight() != null) {                            O(1)
        right = getMorseOfString(root.getRight(), string);      O(n)
      }
      if (left != null) {                                       O(1)
        return left;                                            O(1)
      } else if (right != null) {                               O(1)
```

```
            return right;                                   O(1)
        }
    }
    return null;                                            O(1)
}
```

Alinea 5:

Metodo que retorna o conjunto de símbolos de código morse comum entre duas letras passadas como parâmetro. Para tal através do método alinea5b convertemos a palavra em código morse, para tal utilizamos o método da alínea 4 para codificar a String e convertemos esta mesma para char e coloca-se num array de char. De seguida, através de um ciclo for onde o critério de paragem é o tamanho do maior vetor de char(size obtido no método SIze) e adicionamos a string o caracter em comum as duas strings e retornamos essa string.

```java
/**
 *
 * @param m1-letra para encontrar morse comum
 * @param m2-letra para encontrar morse comum
 * @return  simblo do codigo morse em comum entre as duas letras
 */
public static String alinea5(String m1, String m2) throws UnsupportedEncodingException, IOException {
    String common = "";
    BST<Morse> bst = Ler.ler();
    char[] morseToCharLetter1 = alinea5b(m1, bst);
    char[] morseToCharLetter2 = alinea5b(m1, bst);

    for (int i = 0; i< Size(morseToCharLetter1,morseToCharLetter2); i++) {
        if (morseToCharLetter1[i] == morseToCharLetter2[i]) {
            common = common + morseToCharLetter1[i] + "";
        }
    }
    return common;
}


/**
 * @param m String a ser codificada
 * @param bst Arvore binaria contendo todos os pontos
 * @return  vetor contendo os simbolos da palavra em codigo morse
 **/
private static char[] alinea5b(String m, BST<Morse> bst) {
    return (alinea4(m, bst).toCharArray());
}

/**
 *
 * @param morseToCharLetter1 vetor que contem o codigo morse correspondente a primeira string
 * @param morseToCharLetter2 vetor que contem o codigo morse correspondente a segunda string
 * @return  o tamanho do vetor que contem um size maior
 */
private static int Size(char[] morseToCharLetter1,char[] morseToCharLetter2){
    int size;
    if (morseToCharLetter1.length > morseToCharLetter2.length) {
        size = morseToCharLetter1.length;
    } else {
        size = morseToCharLetter2.length;
    }
    return size;
}
```

```
 public static String alinea5(String m1, String m2) throws UnsupportedEncodingException,
IOException {                                                                    O(n^3)
    String common = "";                                                          O(1)
    BST<Morse> bst = Ler.ler();                                                  O(n)
    char[] morseToCharLetter1 = alinea5b(m1, bst);                               O(n^3)
    char[] morseToCharLetter2 = alinea5b(m1, bst);                               O(n^3)
    for (int i = 0; i< Size(morseToCharLetter1,morseToCharLetter2); i++) {       O(n)
      if (morseToCharLetter1[i] == morseToCharLetter2[i]) {                      O(1)
        common = common + morseToCharLetter1[i] + "";                            O(1)
      }
    }
    return common;                                                               O(1)
  }
  private static char[] alinea5b(String m, BST<Morse> bst) {      O(n^3)
    return (alinea4(m, bst).toCharArray());                       O(n^3)
}
  private static int Size(char[] morseToCharLetter1,char[] morseToCharLetter2){     O(1)
    int size;                                                                  O(1)
    if (morseToCharLetter1.length > morseToCharLetter2.length) {               O(1)
      size = morseToCharLetter1.length;                                        O(1)
    } else {
      size = morseToCharLetter2.length;                                        O(1)
    }
    return size;                                                               O(1)
  }
```

ALInea 6:

Para a alinea 6 filtrar e ordenar uma lista de palavras em código morse por total de ocorrências de tipo de classe de símbolos que a compõem. Para tal é necessario receber uma lista com palavras em morse para a necessidade é criado um mapa que vai conter as varias listas ordenadas que contenham pelo menos um ocurencia de um certo tipo as varias listas vão estar contidas nas keys dos tipos existentes, os morses são traduzidos como na alinea 2. Apos obter os varios elementos adiciona-os a uma string inicialmente vazia e com um metodo que verifica se o o tipo no elemento é igual ao que se pretende se for incrementa-se um a uma variavel inicialmente iniciada o zero. Uma necessidade pedida no enunciado é que a lista esteja ordenada logo temos um metodo que ordena comforme o numero de ocurencias, de seguida retiramos os elementos que têm o contador a zero.

```java
public static Map<String, List<Pair<Integer, String>>> alinea6Inicio(List<String> listaPalavrasEmMorse, BST<Morse> bst) {
    Map<String, List<Pair<Integer, String>>> map = new TreeMap<>();
    List<Pair<Integer, String>> letter = alinea6(listaPalavrasEmMorse, "Letter", bst);
    List<Pair<Integer, String>> number = alinea6(listaPalavrasEmMorse, "Number", bst);
    List<Pair<Integer, String>> punctuation = alinea6(listaPalavrasEmMorse, "Punctuation", bst);
    List<Pair<Integer, String>> nonEnglish = alinea6(listaPalavrasEmMorse, "Non-English", bst);
    map.put("Letter", letter);
    map.put("Number", number);
    map.put("Punctuation", punctuation);
    map.put("Non-English", nonEnglish);
    System.out.println("");
    return map;
}

/**
 *
 * @param listaPalavrasEmMorse
 * @param tipo
 * @param bst
 * @return
 */
public static List<Pair<Integer, String>> alinea6(List<String> listaPalavrasEmMorse, String tipo, BST<Morse> bst) {
    List<Pair<Integer, String>> letrasOcurencia = obter(listaPalavrasEmMorse, bst, tipo);
    ordernar(letrasOcurencia);
    List<Pair<Integer, String>> a = condicao(letrasOcurencia);
    return a;
}

/**
 *
 * @param listaPalavrasEmMorse
 * @param bst
 * @param tipo
 * @return
 */
public static List<Pair<Integer, String>> obter(List<String> listaPalavrasEmMorse, BST<Morse> bst, String tipo) {
    List<Pair<Integer, String>> letrasOcurencia = new ArrayList<>();
    for (String s : listaPalavrasEmMorse) {
        String morseVec[] = s.split(" ");
        List<Node<Morse>> listaParcial = new ArrayList<>();
        for (String m : morseVec) {
            char[] c = m.toCharArray();
            Node<Morse> node = obterNode(m, bst.root(), c, 0);
            listaParcial.add(node);
        }
        Pair<Integer, String> pair = obterNumeroDeTipoCarateres(listaParcial, tipo);
        letrasOcurencia.add(pair);
    }
    return letrasOcurencia;
}
```

```java
    /**
     *
     * @param letrasOcurencia
     */
    private static void ordernar(List<Pair<Integer, String>> letrasOcurencia) {
        for (int i = 0; i < letrasOcurencia.size() - 1; i++) {
            for (int j = i + 1; j < letrasOcurencia.size(); j++) {
                if (letrasOcurencia.get(i).getKey() < letrasOcurencia.get(j).getKey()) {
                    Pair<Integer, String> pair = letrasOcurencia.get(i);
                    letrasOcurencia.set(i, letrasOcurencia.get(j));
                    letrasOcurencia.set(j, pair);
                }
            }
        }
    }

    /**
     *
     * @param listaParcial
     * @param tipo
     * @return
     */
    public static Pair<Integer, String> obterNumeroDeTipoCarateres(List<Node<Morse>> listaParcial, String tipo) {
        int i = 0;
        String inicial = "";
        for (Node<Morse> n : listaParcial) {
            if (n.getElement().getTipo().equals(tipo)) {
                i++;
            }
            inicial += n.getElement().getLetra();
        }
        Pair<Integer, String> pair = new Pair<>(i, inicial);
        return pair;
    }

    /**
     *
     * @param letrasOcurencia
     * @return
     */
    public static List<Pair<Integer, String>> condicao(List<Pair<Integer, String>> letrasOcurencia) {
        List<Pair<Integer, String>> nova = new ArrayList<>();
        for (int i = 0; i < letrasOcurencia.size(); i++) {
            if (letrasOcurencia.get(i).getKey() != 0) {
                nova.add(letrasOcurencia.get(i));
            }
        }
        return nova;
    }
}
```

```
public static List<Pair<Integer, String>> condicao(List<Pair<Integer, String>> letrasOcurencia) {
                                                                    2*O(1) + O(n) = O(n)
    List<Pair<Integer, String>> nova = new ArrayList<>();                    O(1)
    for (int i = 0; i < letrasOcurencia.size(); i++) {                       O(n)
      if (letrasOcurencia.get(i).getKey() != 0) {                            O(1)
        nova.add(letrasOcurencia.get(i));                                    O(1)
      }
    }
    return nova;                                                             O(1)
  }
```

```
public static Pair<Integer, String> obterNumeroDeTipoCarateres(List<Node<Morse>>
listaParcial, String tipo) {                                  4*O(1) + O(n) = O(n)
    int i = 0;                                                               O(1)
    String inicial = "";                                                     O(1)
    for (Node<Morse> n : listaParcial) {                                     O(n)
      if (n.getElement().getTipo().equals(tipo)) {                           O(1)
        i++;
      }
      inicial += n.getElement().getLetra();                                  O(1)
    }
    Pair<Integer, String> pair = new Pair<>(i, inicial);                     O(1)
    return pair;                                                             O(1)
  }
```

```
private static void ordernar(List<Pair<Integer, String>> letrasOcurencia) {
                                                          O(n)^2 + 3*O(1)= O(n)^2
    for (int i = 0; i < letrasOcurencia.size() - 1; i++) {                   O(n)
      for (int j = i + 1; j < letrasOcurencia.size(); j++) {                 O(n)
        if (letrasOcurencia.get(i).getKey() < letrasOcurencia.get(j).getKey()) {   3*O(1)
          Pair<Integer, String> pair = letrasOcurencia.get(i);
          letrasOcurencia.set(i, letrasOcurencia.get(j));
          letrasOcurencia.set(j, pair);
        }
      }
    }
  }
```

```
public static List<Pair<Integer, String>> obter(List<String> listaPalavrasEmMorse, BST<Morse>
bst, String tipo) {                                          4*O(1)+ O(n)^3 = O(n)^3
    List<Pair<Integer, String>> letrasOcurencia = new ArrayList<>();          O(1)
    for (String s : listaPalavrasEmMorse) {
                                                  (O(n)*(2*O(n))*(2*O(n))=O(n)^3
      String morseVec[] = s.split(" ");                                       O(n)
      List<Node<Morse>> listaParcial = new ArrayList<>();                     O(1)
      for (String m : morseVec) {                                             O(n)
        char[] c = m.toCharArray();                                           O(n)
        Node<Morse> node = obterNode(m, bst.root(), c, 0);                    O(n)
        listaParcial.add(node);                                               O(1)
      }
      Pair<Integer, String> pair = obterNumeroDeTipoCarateres(listaParcial, tipo);    O(1)
      letrasOcurencia.add(pair);                                              O(1)
    }
    return letrasOcurencia;                                                   O(1)
  }

public static List<Pair<Integer, String>> alinea6(List<String> listaPalavrasEmMorse, String tipo,
BST<Morse> bst) {                                    O(n)^3 +O(n)^2+O(n)+O(1) = O(n)^3
    List<Pair<Integer, String>> letrasOcurencia = obter(listaPalavrasEmMorse, bst, tipo);    O(n)^3
    ordernar(letrasOcurencia);                                                O(n)^2
    List<Pair<Integer, String>>a=condicao(letrasOcurencia);                   O(n)
    return a;                                                                 O(1)
  }
public static Map<String, List<Pair<Integer, String>>> alinea6Inicio(List<String>
listaPalavrasEmMorse, BST<Morse> bst) {                      6*O(1)+ 3*O(n)^3= O(n)^3
    Map<String, List<Pair<Integer, String>>> map = new TreeMap<>();          O(1)
    List<Pair<Integer, String>> letter = alinea6(listaPalavrasEmMorse, "Letter", bst);    O(n)^3
   List<Pair<Integer, String>> number = alinea6(listaPalavrasEmMorse, "Number", bst);     O(n)^3
   List<Pair<Integer, String>> punctuation = alinea6(listaPalavrasEmMorse, "Punctuation", bst);  O(n)^3
   List<Pair<Integer, String>> nonEnglish = alinea6(listaPalavrasEmMorse, "Non-English", bst);   O(n)^3
    map.put("Letter", letter);                                                O(1)
    map.put("Number", number);                                                O(1)
    map.put("Punctuation", punctuation);                                      O(1)
    map.put("Non-English", nonEnglish);                                       O(1)
    return map;}                                                              O(1)
```