UNIVERSITI KEBANGSAAN MALAYSIA

The National University of Malaysia

TTTK1143: Rekabentuk Aturcara & Penyelesaian Masalah

Assignment - 2: Data Structure

PREPARED BY:

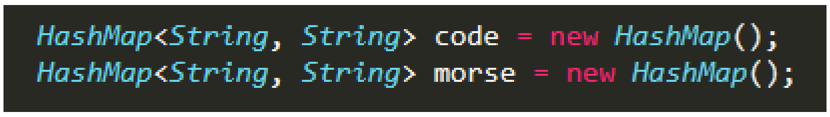
Name: MOHAMED SHAMEER ALI BIN A.M. ABDUL RAHMAN

Matric No: A173586

Contents

1.0 – Data Structure Used

1.1 – Hash Map



```
HashMap<String, String> code = new HashMap();
HashMap<String, String> morse = new HashMap();
```

The Hash Map used in the program

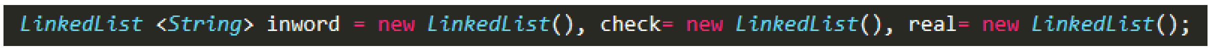
Two HashMap were used:

i. To store the Normal Character as Key and Morse code as Value.
ii. To store the Morse Code as key and Normal Character as Key.

Reason to use:

Hash Map is powerful. It is fast and easier to implement. We can search through it quickly and reduces the time complexity to nearly O(1) which means constant for larger sizes.

1.2 – LinkedList



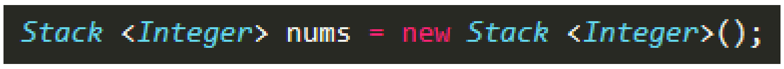
```
LinkedList <String> inword = new LinkedList(), check= new LinkedList(), real= new LinkedList();
```

These are some LinkedList used in this program.

Reason to use:

To store data easily and print it easily without having to deal with indexes and other things. LinkedList provide O(1) time complexity when entering data and List have O(n) which makes LinkedList quicker in larger translations.

1.3 – Stack



```
Stack <Integer> nums = new Stack <Integer>();
```

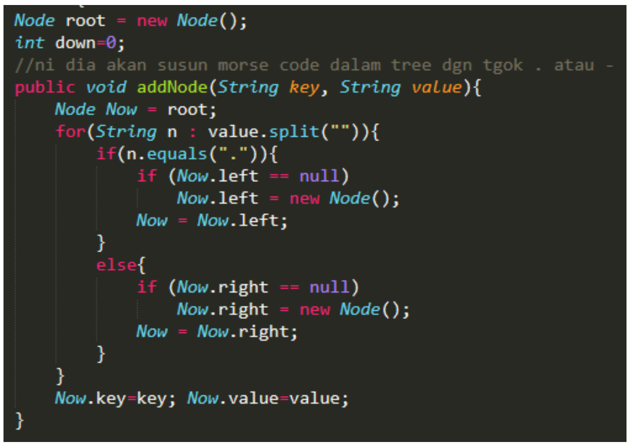
The only Stack used in the Program

Used to store temporary data.

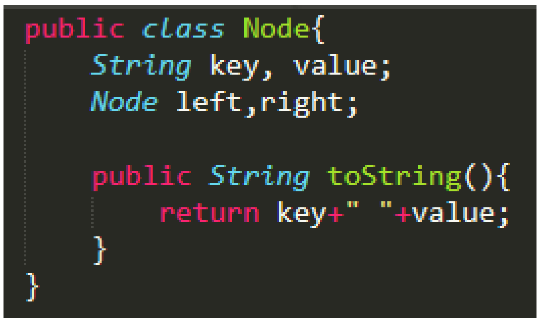
Reason to Use:

Easy to implement and doesn't take much space and also has a quick access.

1.4 – Tree



```java
Node root = new Node();
int down=0;
//ni dia akan susun morse code dalam tree dgn tgok . atau -
public void addNode(String key, String value){
    Node Now = root;
    for(String n : value.split("")){
        if(n.equals(".")){
            if (Now.left == null)
                Now.left = new Node();
            Now = Now.left;
        }
        else{
            if (Now.right == null)
                Now.right = new Node();
            Now = Now.right;
        }
    }
    Now.key=key; Now.value=value;
}
```



```java
public class Node{
    String key, value;
    Node left,right;

    public String toString(){
        return key+" "+value;
    }
}
```

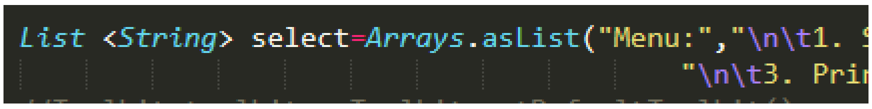
A Small part of the two Classes involves in Tree data structure in this program.

Used to Store the Morse code so that it is easier to be displayed in form of InOrder.

Reason to Use:

I use this because I can output the whole Morse code in form of InOrder and get the output as intended.

1.5 – List



```java
List <String> select=Arrays.asList("Menu:","\n\t1. 
                                    "\n\t3. Pri
```

The List used in the program

Used to store the menu.

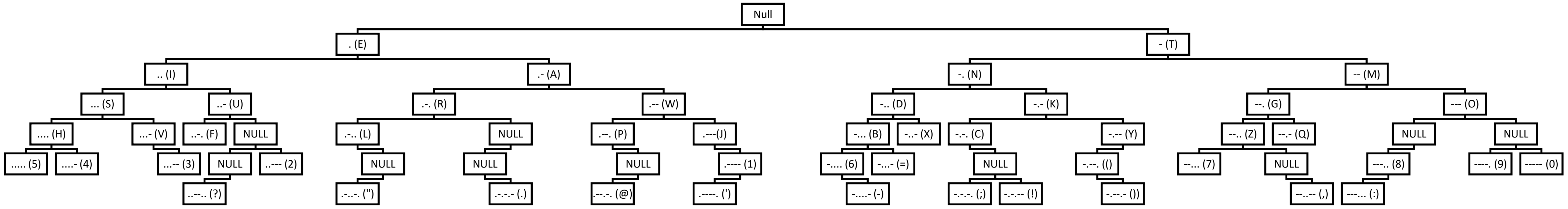
Reason to Use:

Simple and quick to use literation to illiterate through the list. And no further addition or removing is involved.

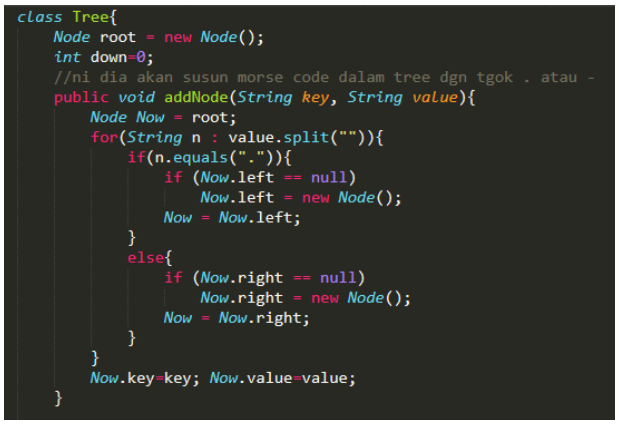
2.0 – Structure of Morse Code

This is the structure of the Morse Code Binary Tree after it has bee inserted into the program. The overall tree is balanced on both sides. The Nodes that are null are labeled NULL. **This is the Tree Structure used in this program.** It goes through the tree by referring its Morse code structure. If it's a "." It will go to the left node and if it's a "-" it will go to the right node and so on until the Morse code have been read.

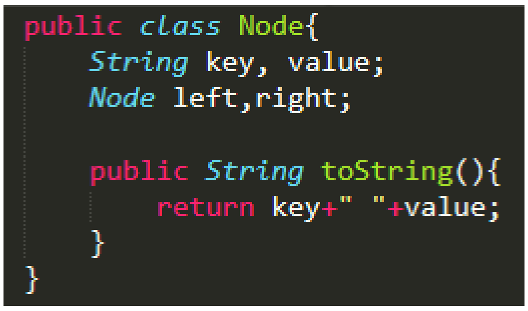
*__This Tree is used in the program__



The Structure of the Class for creating the tree in the program :



```java
class Tree{
    Node root = new Node();
    int down=0;
    //ni dia akan susun morse code dalam tree dgn tgok . atau -
    public void addNode(String key, String value){
        Node Now = root;
        for(String n : value.split("")){
            if(n.equals(".")){
                if (Now.left == null)
                    Now.left = new Node();
                Now = Now.left;
            }
            else{
                if (Now.right == null)
                    Now.right = new Node();
                Now = Now.right;
            }
        }
        Now.key=key; Now.value=value;
    }
```



```java
public class Node{
    String key, value;
    Node left,right;

    public String toString(){
        return key+" "+value;
    }
}
```

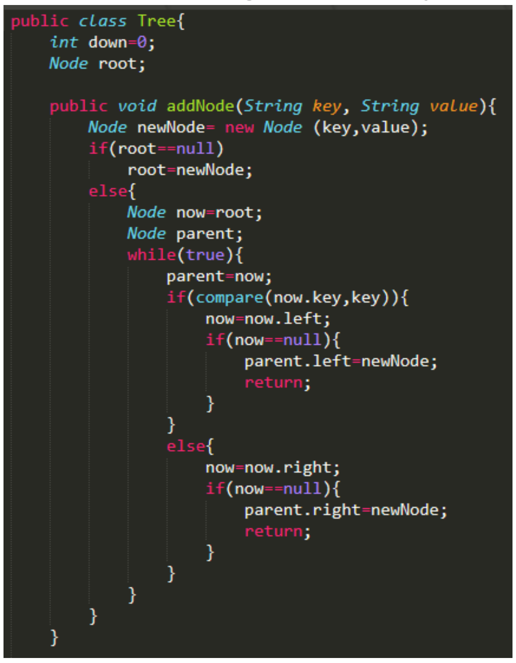
The tree code used in the program.

2.1 – Further Exploration

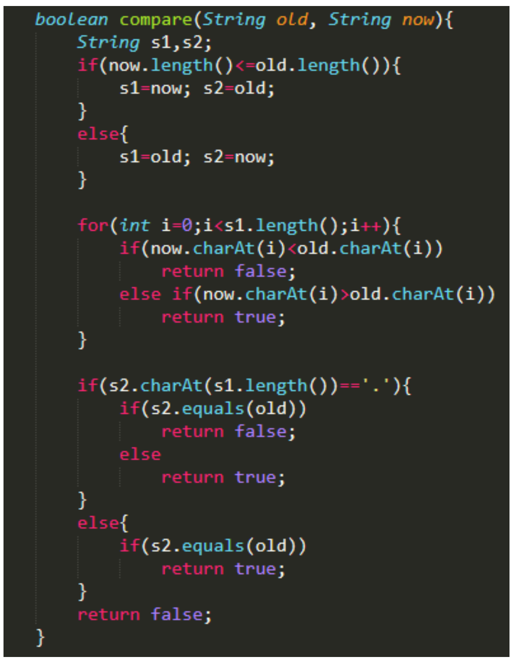
***This part is just for extra info. (You can just skip this part) ***

The tree that I created in my program is just fine and works perfectly but I try to explore other ways to do a Morse code tree.
One of the way was just sorting the Morse code in the tree by just comparing the first different character of two Morse code and sorting it away. This also gave me the exact output when I print it out using InOrder form.

This isn't in the program but it's just some code I tried and worked:



```java
public class Tree{
    int down=0;
    Node root;

    public void addNode(String key, String value){
        Node newNode= new Node (key,value);
        if(root==null)
            root=newNode;
        else{
            Node now=root;
            Node parent;
            while(true){
                parent=now;
                if(compare(now.key,key)){
                    now=now.left;
                    if(now==null){
                        parent.left=newNode;
                        return;
                    }
                }
                else{
                    now=now.right;
                    if(now==null){
                        parent.right=newNode;
                        return;
                    }
                }
            }
        }
    }
```



```java
boolean compare(String old, String now){
    String s1,s2;
    if(now.length()<=old.length()){
        s1=now; s2=old;
    }
    else{
        s1=old; s2=now;
    }

    for(int i=0;i<s1.length();i++){
        if(now.charAt(i)<old.charAt(i))
            return false;
        else if(now.charAt(i)>old.charAt(i))
            return true;
    }

    if(s2.charAt(s1.length())=='.'){
        if(s2.equals(old))
            return false;
        else
            return true;
    }
    else{
        if(s2.equals(old))
            return true;
    }
    return false;
}
```

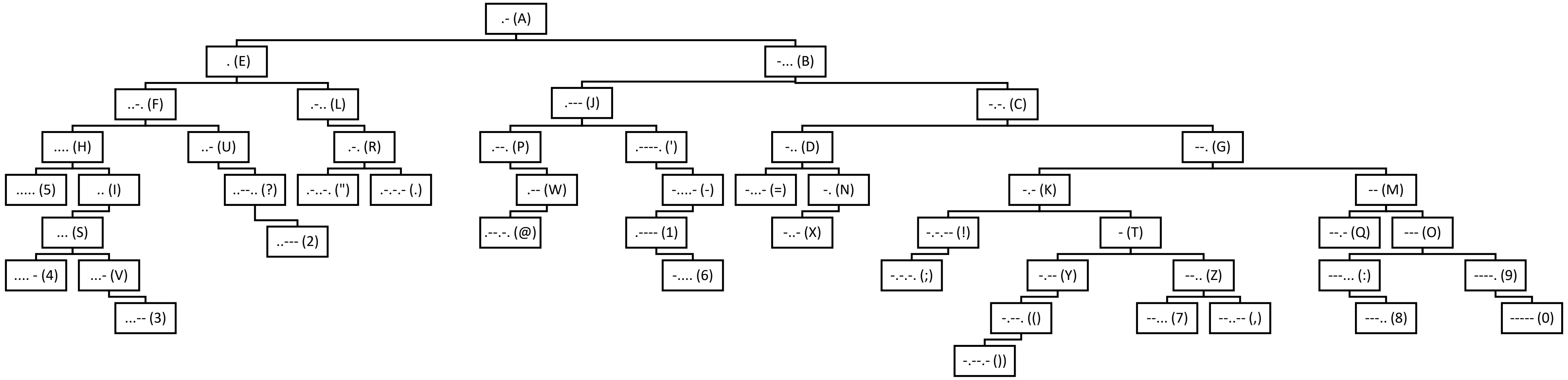
Just a different tree that I made but isn't used in the program

It node class is nearly the same but the part where I used the method compare() is where I manage to get the Morse code sorted inputting the Morse code and returning true or false according to which Morse code has the bigger value.

This is the tree I got form the extra coding I did. Even though it doesn't have null nodes but still this one took quite some time when it was first sorted in the tree. Because each Morse code will be needed to be checked with the parent of each node before becoming either it's left or right child.

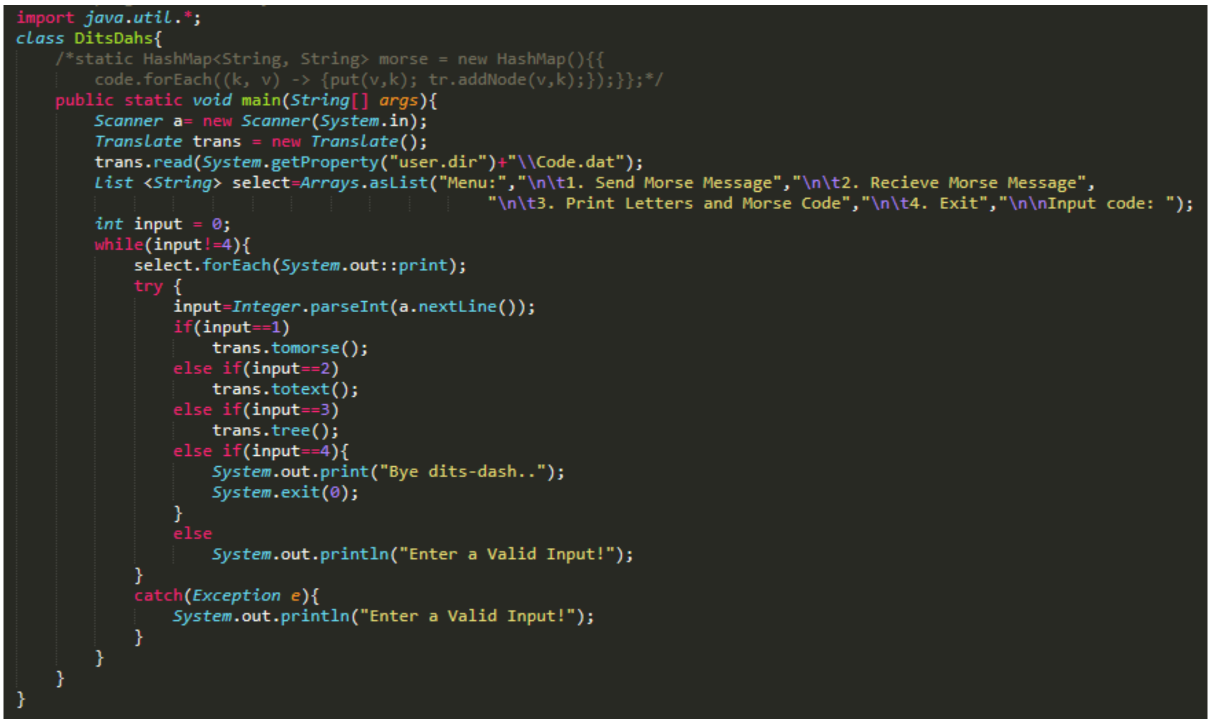
Another amazing thing about this tree is that its structure is affected by the order the Morse code is entered. In this case the Morse code for "A" was entered first than followed by the Morse code for "B" and so on. So, the Morse code for "A" became the root.

***This Tree is not included in the program.**



3.0 – Classes in Program

3.1 – DitsDash (Main Class)



```
import java.util.*;
class DitsDahs{
    /*static HashMap<String, String> morse = new HashMap(){{
        code.forEach((k, v) -> {put(v,k); tr.addNode(v,k);});}};*/
    public static void main(String[] args){
        Scanner a= new Scanner(System.in);
        Translate trans = new Translate();
        trans.read(System.getProperty("user.dir")+"\\Code.dat");
        List <String> select=Arrays.asList("Menu:","\n\t1. Send Morse Message","\n\t2. Recieve Morse Message",
                                            "\n\t3. Print Letters and Morse Code","\n\t4. Exit","\n\nInput code: ");
        int input = 0;
        while(input!=4){
            select.forEach(System.out::print);
            try {
                input=Integer.parseInt(a.nextLine());
                if(input==1)
                    trans.tomorse();
                else if(input==2)
                    trans.totext();
                else if(input==3)
                    trans.tree();
                else if(input==4){
                    System.out.print("Bye dits-dash..");
                    System.exit(0);
                }
                else
                    System.out.println("Enter a Valid Input!");
            }
            catch(Exception e){
                System.out.println("Enter a Valid Input!");
            }
        }
    }
}
```

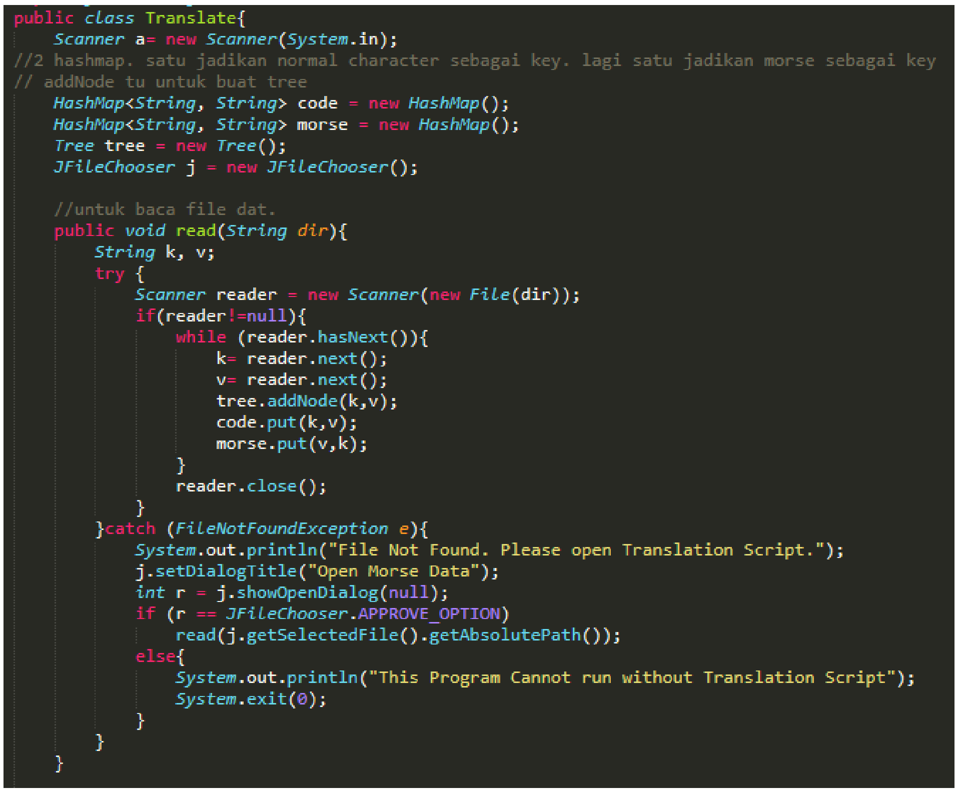
Main Method in DitsDash Clash

This is the main class where the **Program Should Be Run**. It contains the main method with the users input. It also sent the current directory this file is in that has the "Code.dat" file.

The part "trans.read()" is the methode in Translate class where the file is read.

3.2 – Translate Class

3.2.1 – read() method



```
public class Translate{
    Scanner a= new Scanner(System.in);
//2 hashmap. satu jadikan normal character sebagai key. lagi satu jadikan morse sebagai key
// addNode tu untuk buat tree
    HashMap<String, String> code = new HashMap();
    HashMap<String, String> morse = new HashMap();
    Tree tree = new Tree();
    JFileChooser j = new JFileChooser();

    //untuk baca file dat.
    public void read(String dir){
        String k, v;
        try {
            Scanner reader = new Scanner(new File(dir));
            if(reader!=null){
                while (reader.hasNext()){
                    k= reader.next();
                    v= reader.next();
                    tree.addNode(k,v);
                    code.put(k,v);
                    morse.put(v,k);
                }
                reader.close();
            }
        }catch (FileNotFoundException e){
            System.out.println("File Not Found. Please open Translation Script.");
            j.setDialogTitle("Open Morse Data");
            int r = j.showOpenDialog(null);
            if (r == JFileChooser.APPROVE_OPTION)
                read(j.getSelectedFile().getAbsolutePath());
            else{
                System.out.println("This Program Cannot run without Translation Script");
                System.exit(0);
            }
        }
    }
```

read() Method used in this program

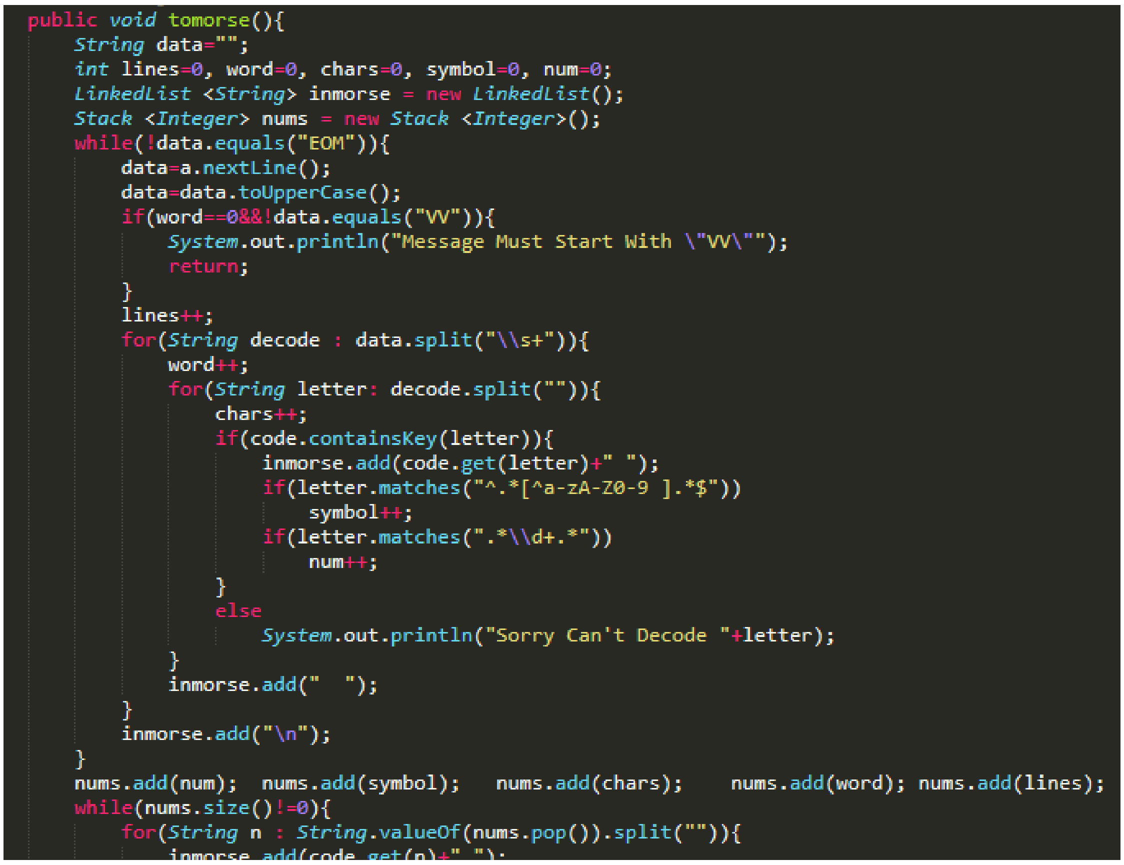
In this class the two hash map is used.

The read method reads the Code.dat file and puts the data into the two hash map and also into the tree.

The file reader is located inside the catch because if the **File is Not Found** it will open up a file chooser for the user to find and select the file.

3.2.2 – tomorse() method

This is the method used to translate the readable letters into Morse Code.



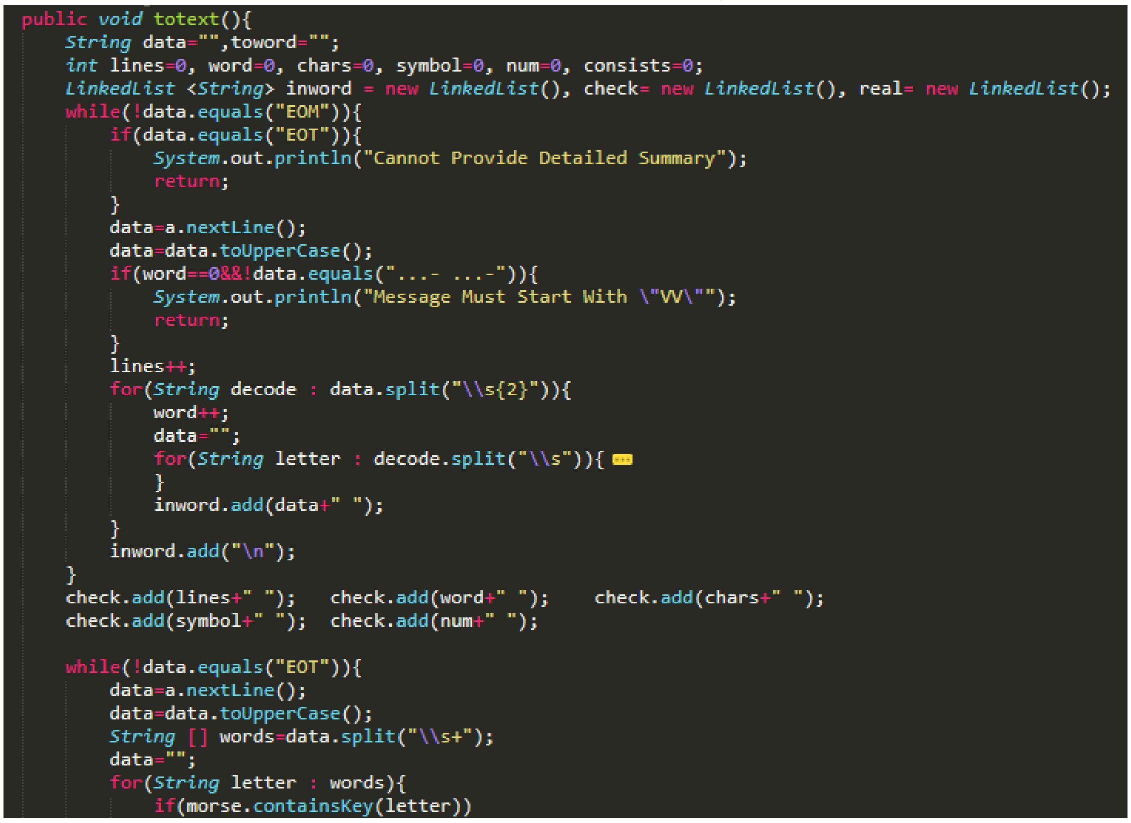
```
public void tomorse(){
    String data="";
    int lines=0, word=0, chars=0, symbol=0, num=0;
    LinkedList <String> inmorse = new LinkedList();
    Stack <Integer> nums = new Stack <Integer>();
    while(!data.equals("EOM")){
        data=a.nextLine();
        data=data.toUpperCase();
        if(word==0&&!data.equals("VV")){
            System.out.println("Message Must Start With \"VV\"");
            return;
        }
        lines++;
        for(String decode : data.split("\\s+")){
            word++;
            for(String letter: decode.split("")){
                chars++;
                if(code.containsKey(letter)){
                    inmorse.add(code.get(letter)+" ");
                    if(letter.matches("^.*[^a-zA-Z0-9 ].*$"))
                        symbol++;
                    if(letter.matches(".*\\d+.*"))
                        num++;
                }
                else
                    System.out.println("Sorry Can't Decode "+letter);
            }
            inmorse.add("  ");
        }
        inmorse.add("\n");
    }
    nums.add(num);  nums.add(symbol);   nums.add(chars);    nums.add(word); nums.add(lines);
    while(nums.size()!=0){
        for(String n : String.valueOf(nums.pop()).split("")){
            inmorse.add(code.get(n)+" ");
```

A Part of the tomorse() method

The input from the user is first split into words. And then into letters and then the rest of the translation process is carried on. While this is going on the numbers of letters, lines, words, numbers, and symbol are calculated along the way for the summary.

3.2.3 – totext() method

This is the method used to Translate the Morse code to the readable letters and also checks for the correctness of the data and its summary.



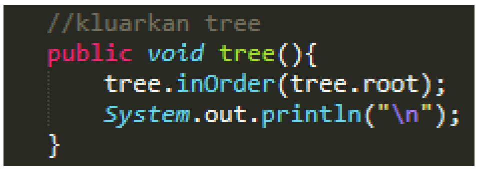
```java
public void totext(){
    String data="",toword="";
    int lines=0, word=0, chars=0, symbol=0, num=0, consists=0;
    LinkedList <String> inword = new LinkedList(), check= new LinkedList(), real= new LinkedList();
    while(!data.equals("EOM")){
        if(data.equals("EOT")){
            System.out.println("Cannot Provide Detailed Summary");
            return;
        }
        data=a.nextLine();
        data=data.toUpperCase();
        if(word==0&&!data.equals("...- ...-")){
            System.out.println("Message Must Start With \"VV\"");
            return;
        }
        lines++;
        for(String decode : data.split("\\s{2}")){
            word++;
            data="";
            for(String letter : decode.split("\\s")){
            }
            inword.add(data+" ");
        }
        inword.add("\n");
    }
    check.add(lines+" ");   check.add(word+" ");    check.add(chars+" ");
    check.add(symbol+" ");  check.add(num+" ");

    while(!data.equals("EOT")){
        data=a.nextLine();
        data=data.toUpperCase();
        String [] words=data.split("\\s+");
        data="";
        for(String letter : words){
            if(morse.containsKey(letter))
```

A Part of the tomorse() method (The second for loop is compresses to screen capture most of the coding).

The user input in broken down into words by splitting double spaces only and then into letters by splitting spaces between the Morse code entered. The Morse code is then translated back into readable letters with also the summary is checked if its valid or not.

3.2.4 – tree() method

The method used to print out the tree in the form intended and in this case in InOrder form.



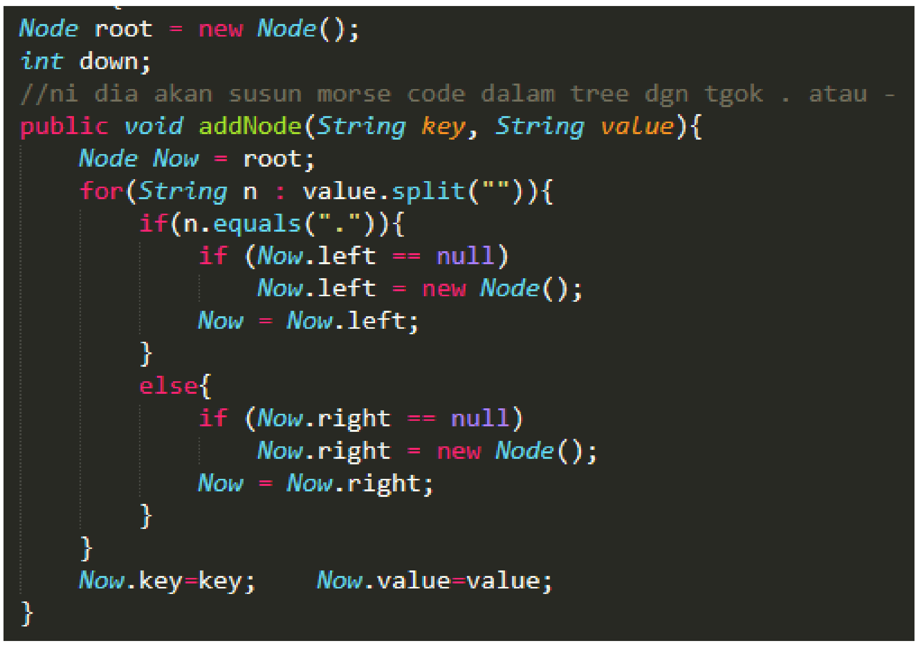
```java
//kluarkan tree
public void tree(){
    tree.inOrder(tree.root);
    System.out.println("\n");
}
```

This method to call the Inorder print method.

3.3 – Tree Class

3.3.1 – addNode () method

Method used to insert Morse code into tree and sort it by referring its Morse code structure.



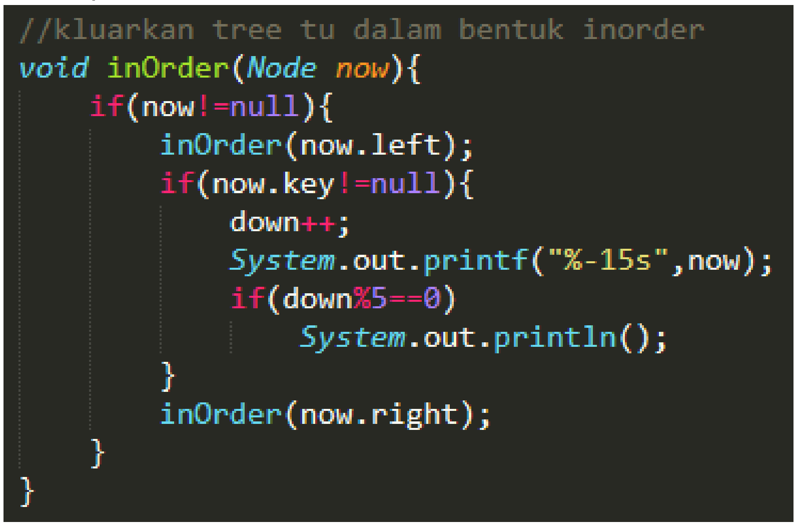
```java
Node root = new Node();
int down;
//ni dia akan susun morse code dalam tree dgn tgok . atau -
public void addNode(String key, String value){
    Node Now = root;
    for(String n : value.split("")){
        if(n.equals(".")){
            if (Now.left == null)
                Now.left = new Node();
            Now = Now.left;
        }
        else{
            if (Now.right == null)
                Now.right = new Node();
            Now = Now.right;
        }
    }
    Now.key=key;    Now.value=value;
}
```

addNode() Method in this program

The for Each is used to read every character of the Morse code entered and if it’s a “.” It will be sent to the left branch and if “-“ it will go to the right branch and it will do this till al the character of a Morse code is read. It will also create null node along the way.

3.3.2 – inOrder () method

The method to print out the whole tree in inorder form.



```
//kluarkan tree tu dalam bentuk inorder
void inOrder(Node now){
    if(now!=null){
        inOrder(now.left);
        if(now.key!=null){
            down++;
            System.out.printf("%-15s",now);
            if(down%5==0)
                System.out.println();
        }
        inOrder(now.right);
    }
}
```

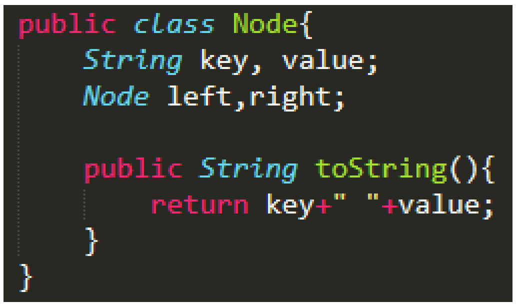
inOrder() Method in this program

It will go through the tree form the first node entered which is the root. And go to the most left and print out the node. If it's a null node it wont print it out.

The "%-15s" specifier is used to give equal space between the printed-out node. If we use /t or spaces sometime if the Morse code is longer it won't print it out nicely.

3.4 – Node Class

Holds Node information and variables.



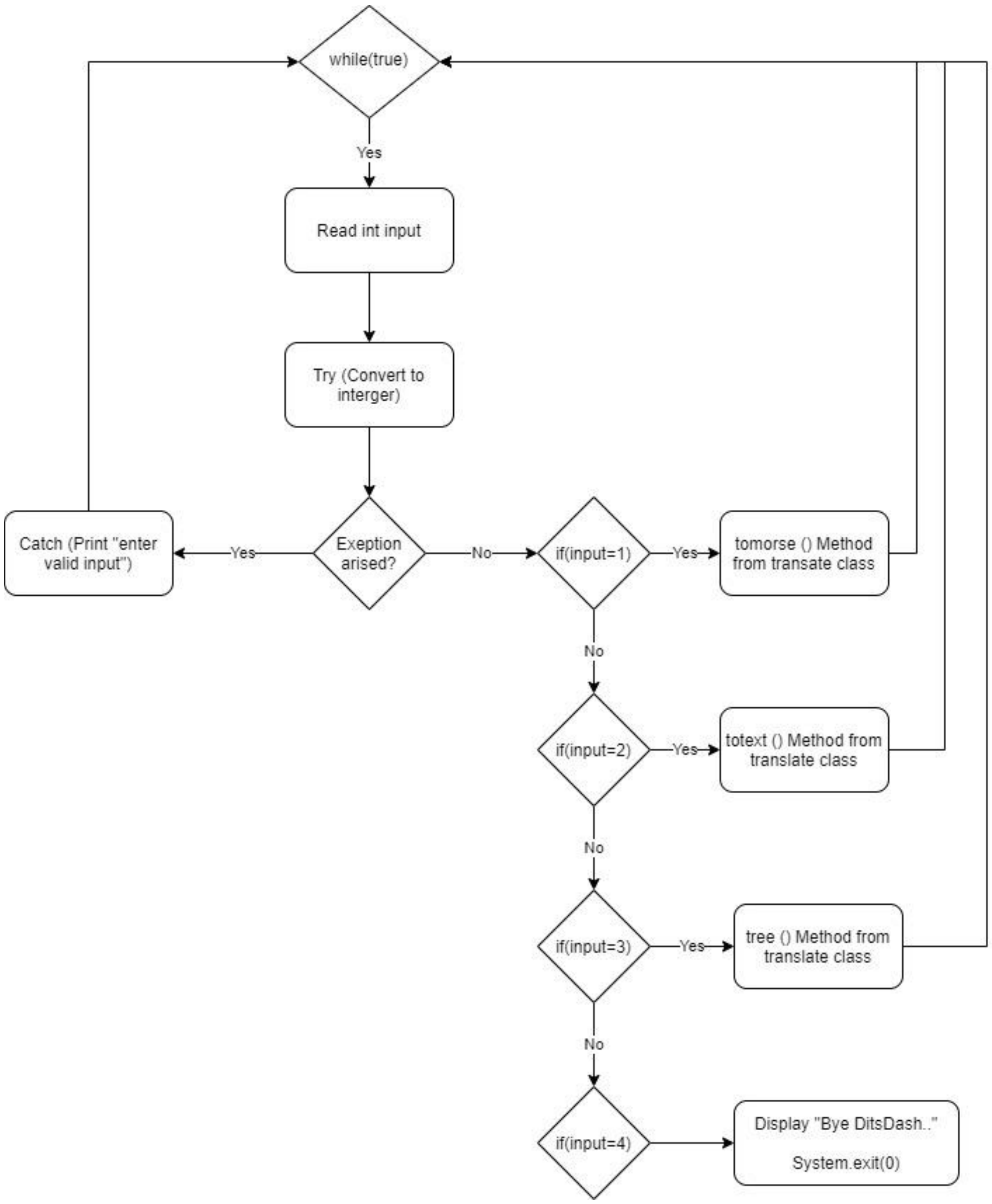
```
public class Node{
    String key, value;
    Node left,right;

    public String toString(){
        return key+" "+value;
    }
}
```

The Node Class

This class have a toString () method that is used to print out the Node information which is the letter and Morse code. It also has the Child Node life and right for each node.

4.0 – Algorithms

4.1 – Main Method


while(true)
Yes
Read int input
Try (Convert to interger)
Exeption arised?
Yes
Catch (Print "enter valid input")
No
if(input=1)
Yes
tomorse () Method from transate class
No
if(input=2)
Yes
totext () Method from translate class
No
if(input=3)
Yes
tree () Method from translate class
No
if(input=4)
Display "Bye DitsDash.."
System.exit(0)

4.2 – Encode Method

This whole process involved in the encoding part:

```
Start
    String data = “”
    Int lines, word, symbol,numbers, chars
    while (data != “EOM”)
        Read data
        if (word==0 && !data==”VV”)
            Display “Must start with VV”
            Return;
        lines++
        for (decode : data.split(“\\s+”))
            word++
            for (letter : decode.split(“”)
                chars++
                if (the letter is in the HashMap)
                    Get the letters Morse code and add to list
                    if (letter have symbol)
                        symbol++
                    if (letter have number)
                        number++
                else
                    Display “Can’t decode” + letter
            End
            Add converted letter into inmorse list
        End
        Add all the counter into stack
        while (stack != empty)
            Split the numbers into one digit
            Convert each into Morse code using HashMap
            Add to list
        End while
        Display inmorse List
    End while
End
```

4.3 – Decode

This whole process involved in the decoding part:

```
Start
    String data = “”
    Int lines, word,number,symols,chars
    while (data != “EOM”)
        if(data == “EOT”)
            display “cant give summary”
            return
        read data
        if(word=0 && data !==”…- …-“)
            display “ must start with “…- …-“
            return
        lines++
        for(decode : data.split(“\\s{2}”)
            word++
            data=””
            for(letter : decode.split(“\\s”)
                chars++
                if(morse code in hash map)
                    Check hash map for morse code and
                    convert
                    If(converted letter is symbol)
                        Symbol++
                    If(converted letter is number)
                        Number++
                    Data=data+converted letter
                Else
                    Display “can’t decode” + letter
            End
            Add data (converted text) into inword list
        End
        Add counters in check list
    End while
```

```
While (data != "EOT")
        Read data
        Word = data.split("\\s+")
        Data=""
        For(letter : word )
                If(the letter is in the hash map)
                        Convert the letter to the corresponding number)
                        Data = data + converted morse code
                Else
                        Display "Can't decode" + letter
                        Return
        End
        Add data (numbers) to inword list
        If (data != "EOT"
                Add data to real list
End while

Display inword list
Display check list
If (real list == check list)
        Display "Result: Consistent summary"
Else
        Display "Result: Inconsistent summary"
```

4.4 – Display Tree

The algorithm for the generation of tree and displaying it

4.4.1 - Making the tree

The letters and Morse code are read through the file and is passed to the addNode function. The key is the letter and value are the Morse code.

```
Start
    Node root = new Node ()
    AddNode Function (key, value)
        Node Now = root
        For (n : value.split(“”)
            If (n == “.”)
                If (Now.left  == null)
                    Now.left = new Node ()
                Now= Now.left
            Else
                If (Now.right == null)
                    Now.right = new Node ()
                Now = Now.right
        End
        Now.key=key
        Now.value=value
    End function
End
```

What this does is that it splits the Morse code entered into it and it illiterate through it. If it found a “.”, it will go to the left node and if it found a “-“, it will go to the right node and so on until the whole structure of the Morse code is read. And then it will continue with the other Morse codes until all of it has entered into the program.

4.4.2 - Displaying The tree

It is displayed in an inOrder form. The root of the tree is passed into the InOrder function.

```
Start
    Inorder(Node now)
        If(now != null)
            Inorder(now.left) (Go to the left most child node of the
            current node)
            If(the current nodes key is not null)
                Display the node (it will display with the key and
                value)
            InOrder(now.right) (Go to the right most child node of the
            current node)
    End
End
```

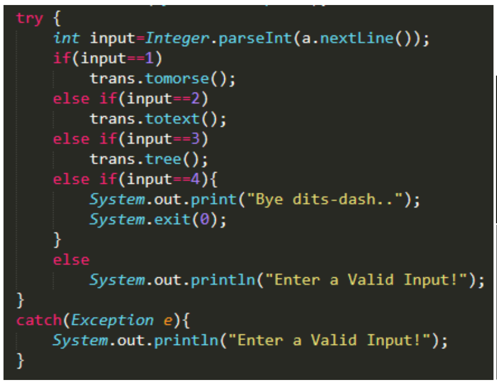
It goes to the left most node in the tree and it will print out the node if it’s null and then go to its own parent and print it and then to the right most node of that parent. This who process will continue till the whole tree is printed out.

5.0 – Assumptions Made

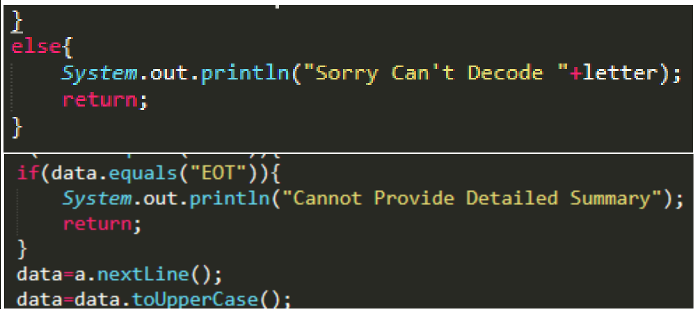
5.1 – Incorrect User Input

Some users might enter wrong input at places that they need to enter other things. Like the input code for selecting menu, user might enter symbols, letters, or any other numbers. So, the best way is to take the input as string and try to convert it to integer in try and catch. If the input s not as expected than it will throw a message but won't make the program crash.

Some users also might enter the input in a wrong order than expected so I look for that to. For example, like user enter EOT before EOM in the Decode part. And if the user enters letters, or symbol that is not in the translation script like "[", "]", "$', "&", "^" and many others. The program should handle all this type of problems.



```
try {
    int input=Integer.parseInt(a.nextLine());
    if(input==1)
        trans.tomorse();
    else if(input==2)
        trans.totext();
    else if(input==3)
        trans.tree();
    else if(input==4){
        System.out.print("Bye dits-dash..");
        System.exit(0);
    }
    else
        System.out.println("Enter a Valid Input!");
}
catch(Exception e){
    System.out.println("Enter a Valid Input!");
}
```



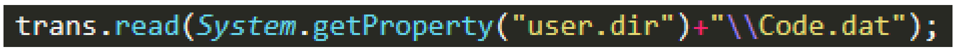
```
}
else{
    System.out.println("Sorry Can't Decode "+letter);
    return;
}
```

```
if(data.equals("EOT")){
    System.out.println("Cannot Provide Detailed Summary");
    return;
}
data=a.nextLine();
data=data.toUpperCase();
```

Some example of handling incorrect user input

5.2 – File Problem

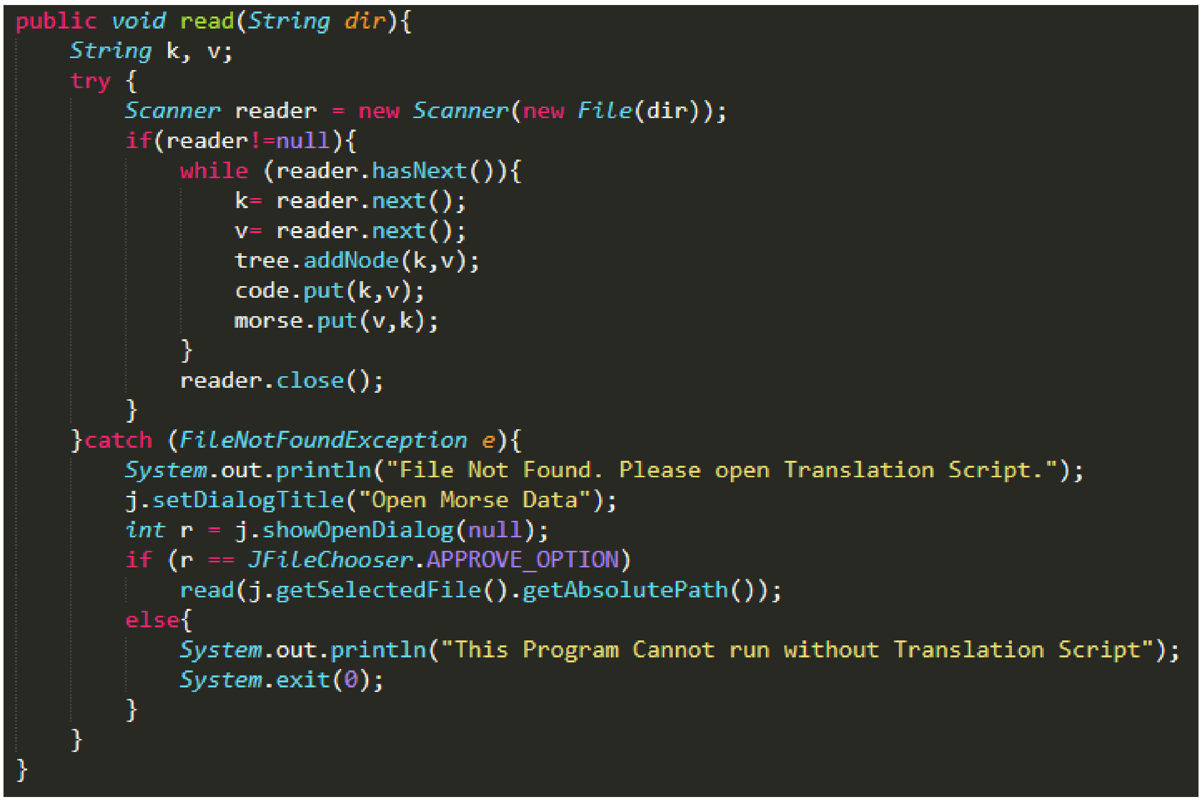
Sometimes we may have file corrupted or the directory of the file is missing. So, to deal with this I set the directory of the file as the current working directory of the java file and then search for the file name.



```
trans.read(System.getProperty("user.dir")+"\\Code.dat");
```

Here I sent the directory to the trans class. It will search for Code.dat at the same directory the java file is.

But sometimes, we might also have problem in the directory. What if the program cannot find the file? So, for this I did a try and catch where the catch will open a file chooser for the user to choose the file manually.

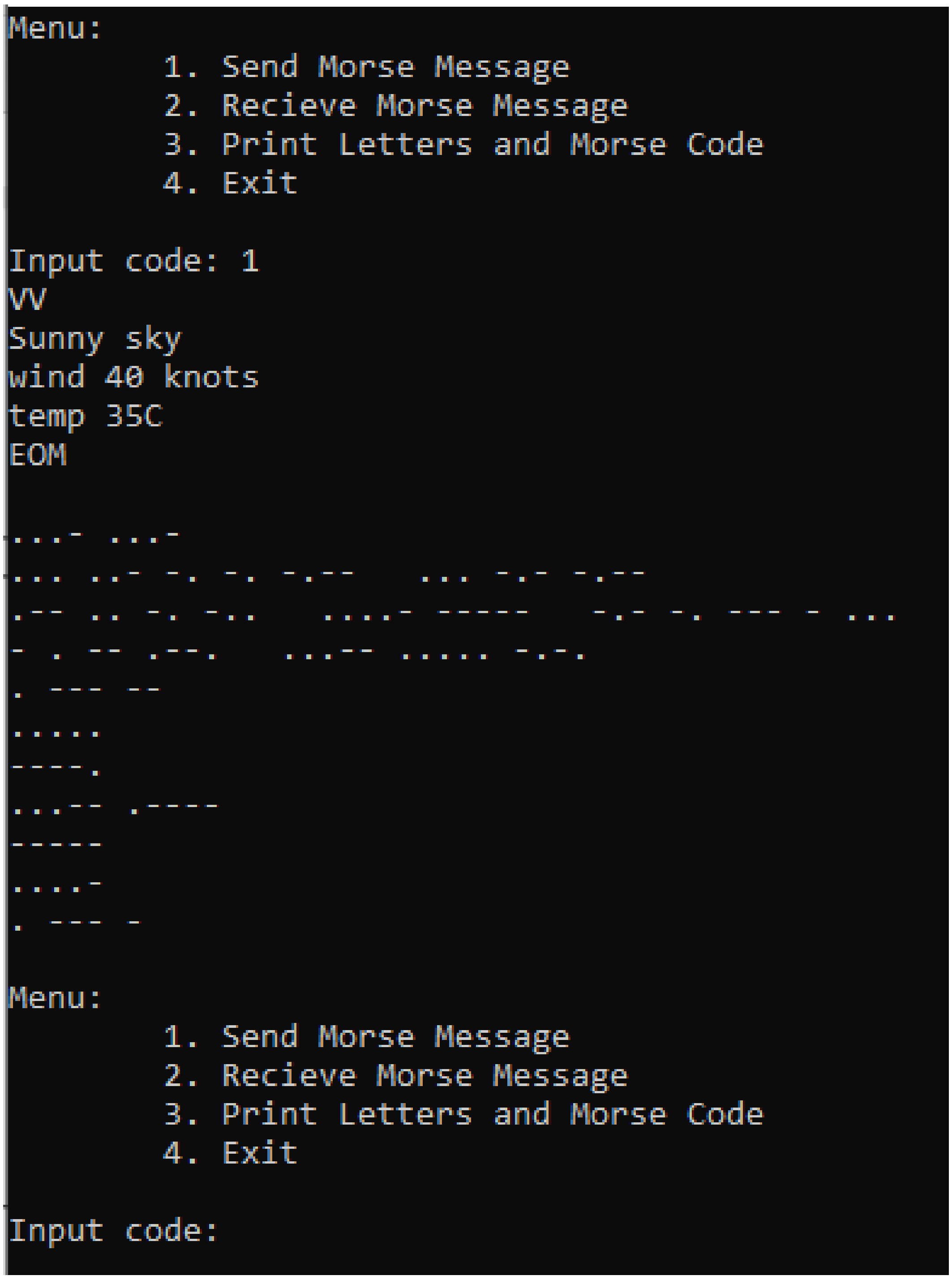


```
public void read(String dir){
    String k, v;
    try {
        Scanner reader = new Scanner(new File(dir));
        if(reader!=null){
            while (reader.hasNext()){
                k= reader.next();
                v= reader.next();
                tree.addNode(k,v);
                code.put(k,v);
                morse.put(v,k);
            }
            reader.close();
        }
    }catch (FileNotFoundException e){
        System.out.println("File Not Found. Please open Translation Script.");
        j.setDialogTitle("Open Morse Data");
        int r = j.showOpenDialog(null);
        if (r == JFileChooser.APPROVE_OPTION)
            read(j.getSelectedFile().getAbsolutePath());
        else{
            System.out.println("This Program Cannot run without Translation Script");
            System.exit(0);
        }
    }
}
```

Here, the method receives the directory of the file but if it can't open it, it will open the File Chooser.

6.0 – Input / Output

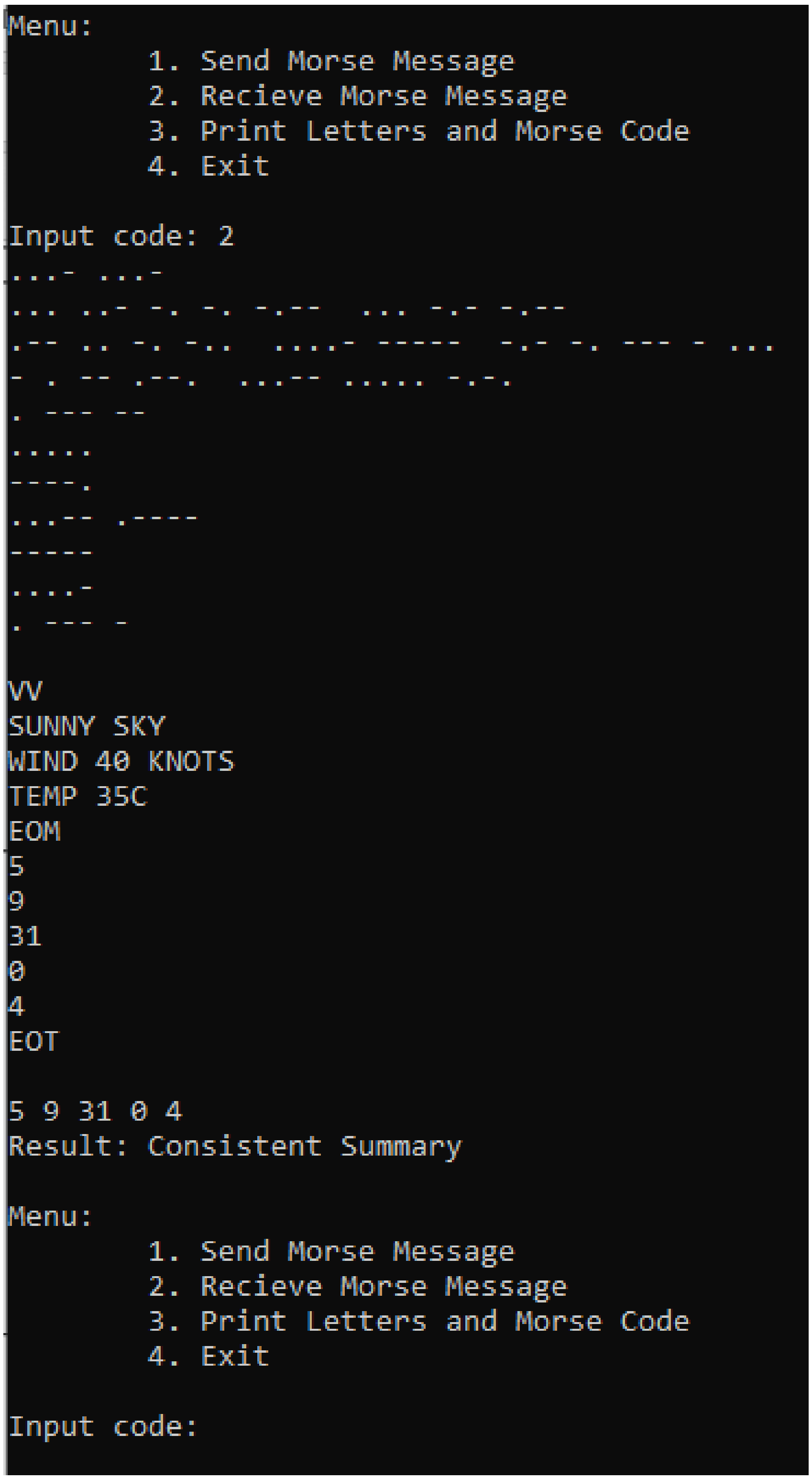
6.1 – Send Morse Message



```
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: 1
VV
Sunny sky
wind 40 knots
temp 35C
EOM

...- ...-
... ..- -. -. -.--   ... -.- -.--
.-- .. -. -..   ....- -----   -.- -. --- - ...
- . -- .--.   ...-- ..... -.-.
. --- --
.....
----.
...-- .----
-----
....-
. --- -

Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code:
```

6.2 – Receive Morse Message

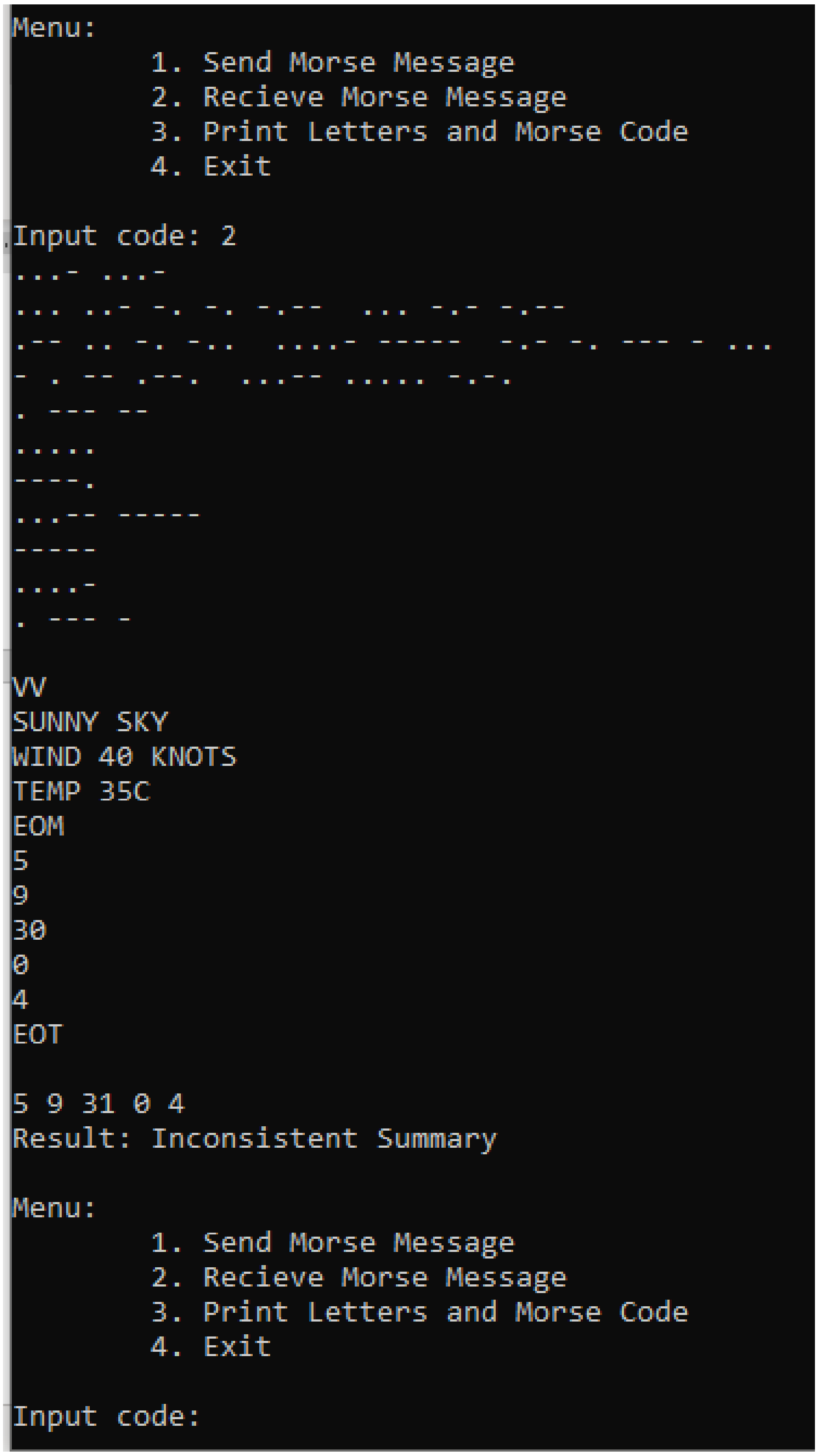
6.2.1 – Consistent Summary



```
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: 2
...- ...-
... ..- -. -. -.--  ... -.- -.--
.-- .. -. -..  ....- -----  -.- -. --- - ...
- . -- .--.  ...-- ..... -.-.
. --- --
.....
----.
...-- .----
-----
....-
. --- -

VV
SUNNY SKY
WIND 40 KNOTS
TEMP 35C
EOM
5
9
31
0
4
EOT

5 9 31 0 4
Result: Consistent Summary

Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code:
```

If the summary is the same

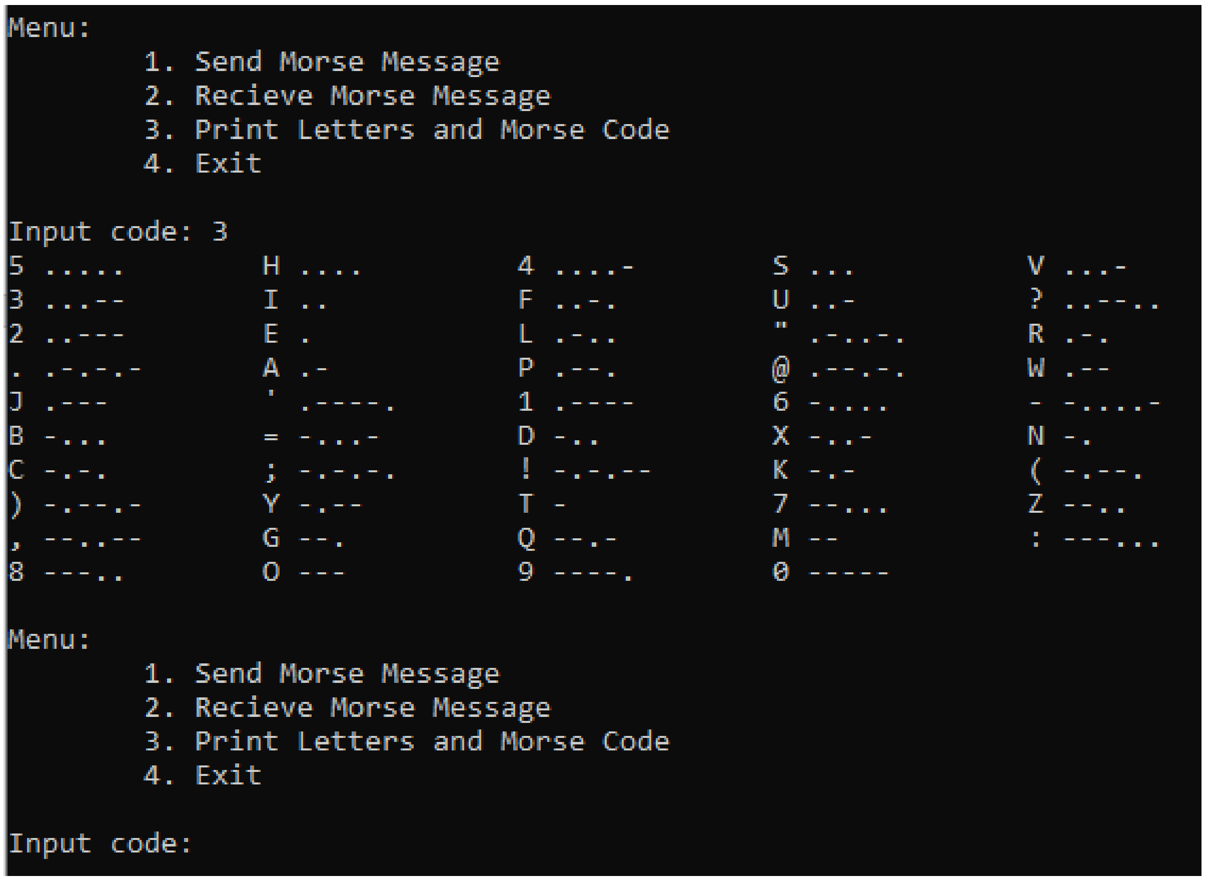
6.2.2 – Inconsistent



```
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: 2
...- ...-
... ..- -. -. -.--  ... -.- -.--
.-- .. -. -..  ....- -----  -.- -. --- - ...
- . -- .--.  ...-- ..... -.-.
. --- --
.....
----.
...-- -----
-----
....-
. --- -

VV
SUNNY SKY
WIND 40 KNOTS
TEMP 35C
EOM
5
9
30
0
4
EOT

5 9 31 0 4
Result: Inconsistent Summary

Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code:
```

If the summary wasn't the same

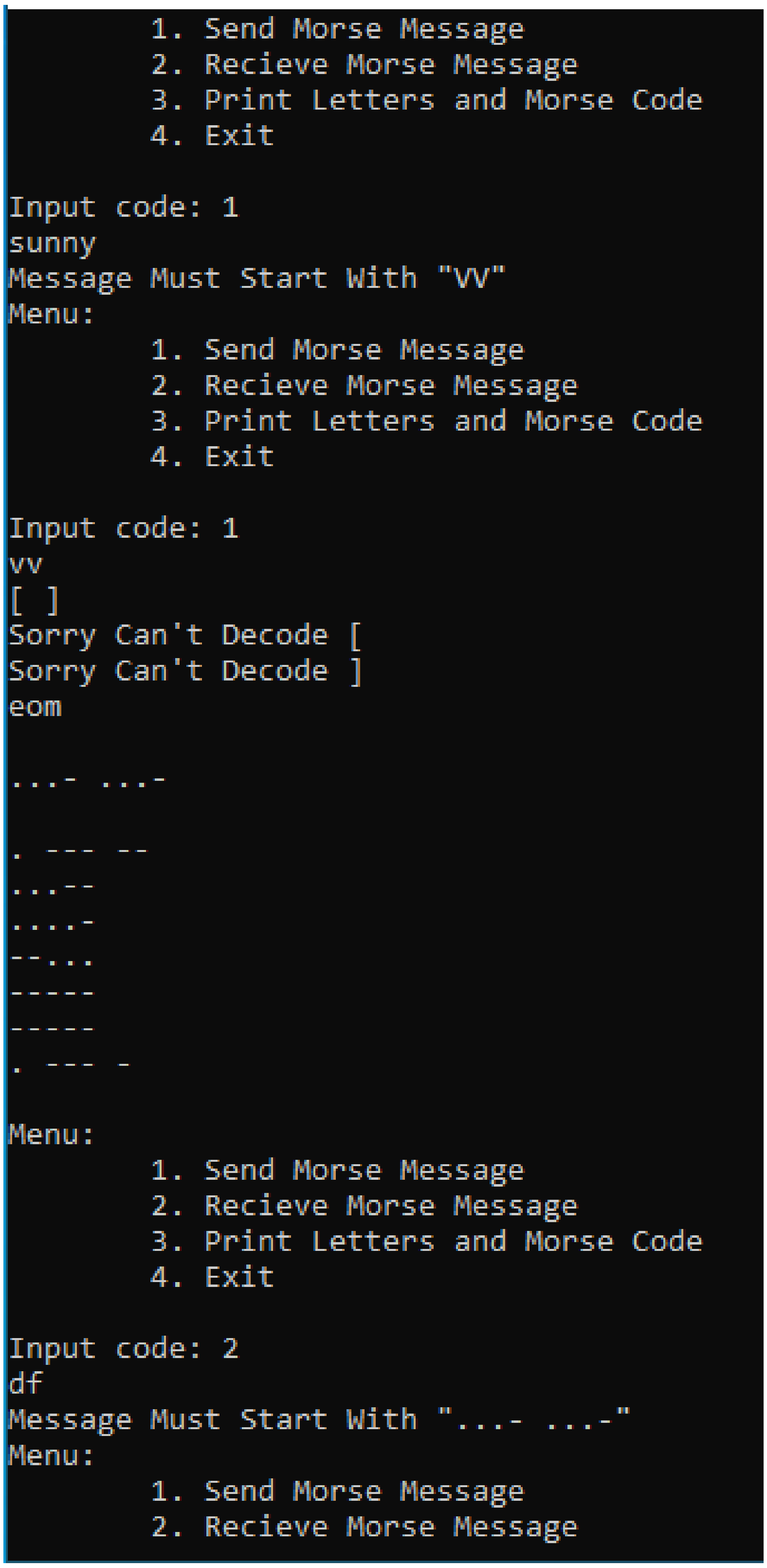
6.3 – Print Letters and Morse Code



```
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: 3
5 .....        H ....         4 ....-        S ...          V ...-
3 ...--        I ..           F ..-.         U ..-          ? ..--..
2 ..---        E .            L .-..         " .-..-.       R .-.
. .-.-.-       A .-           P .--.         @ .--.-.       W .--
J .---         ' .----.       1 .----        6 -....        - -....-
B -...         = -...-        D -..          X -..-         N -.
C -.-.         ; -.-.-.       ! -.-.--       K -.-          ( -.--.
) -.--.-       Y -.--         T -            7 --...        Z --..
, --..--       G --.          Q --.-         M --           : ---...
8 ---..        O ---          9 ----.        0 -----

Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code:
```

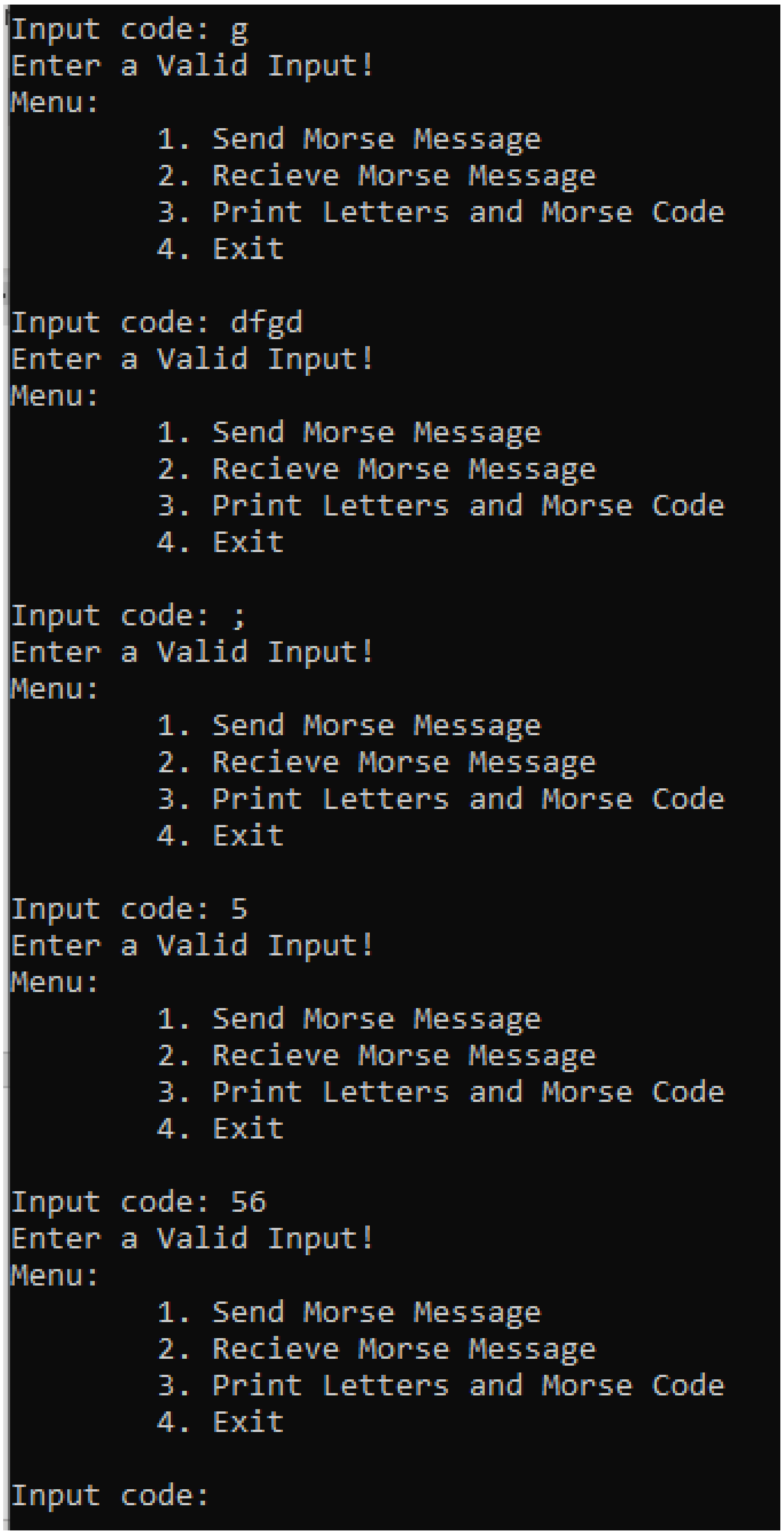
6.4 – Incorrect input

If the user starts the encoding part without VV/ …- …- or enters letters and symbols that is not on the translation file.



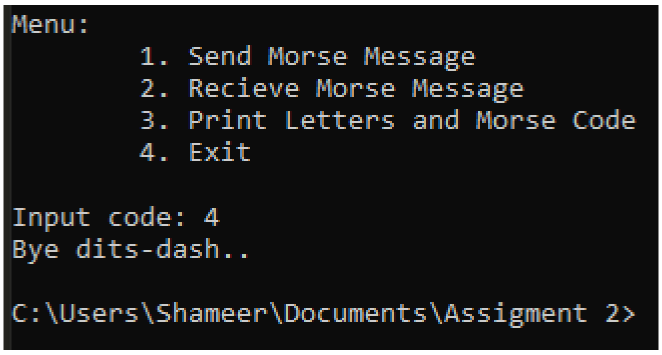
```
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: 1
sunny
Message Must Start With "VV"
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: 1
vv
[ ]
Sorry Can't Decode [
Sorry Can't Decode ]
eom

...- ...-

. --- --
...--
....-
--...
-----
-----
. --- -

Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: 2
df
Message Must Start With "...- ...-"
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
```

If the user enters number, symbols or letter that is not on the menu at input code part:



```
Input code: g
Enter a Valid Input!
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: dfgd
Enter a Valid Input!
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: ;
Enter a Valid Input!
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: 5
Enter a Valid Input!
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: 56
Enter a Valid Input!
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code:
```

6.5 – Exit



```
Menu:
        1. Send Morse Message
        2. Recieve Morse Message
        3. Print Letters and Morse Code
        4. Exit

Input code: 4
Bye dits-dash..

C:\Users\Shameer\Documents\Assigment 2>
```

End

That’s all for this documentation. The program has a total of 4 java files, DitsDash.java, Tree.java, Node.java, Translate.java. The folder has all its java file and their class. The DitsDash java file/class should be run to use this program.

It also has a Code.dat file which contains the translation script for the program to work.

This program is made by me, Shameer Ali and I hope it runs without any problem.

Email: a173586@siswa.ukm.edu.my
Whsapp: 01140448922 - Shameer