# Chapter V

*Implementation and Testing*

## 5.1 LinkList

### 5.1.1 addVector(Vector<String> nomor)

This is a method to add all data inside a Vector<String> to the LinkList.

```
public void addVector(Vector<String> nomor) {
    for(int i=0; i<nomor.size(); i++) {
        add(new Link(nomor.get(i)));
    }
}
```

### 5.1.2 createIndex()

This is a method to create index(s) of the LinkList. This method is called again (recursive) to create index of the index, to create multiple level index. The method will stop when the string used to identify the index is 3 characters or fewer (the first 4 numbers are used to identify prefix)

```
    public void createIndex() { //membuat 'index'
LinkedList, khusus untuk prefix
        index = new LinkList();
        if(first.nomor.length() < 4)
            return;

        curr = first;
        index.add(new Link(curr.nomor.substring(0,
curr.nomor.length()-1)));
        index.last.indexTo = curr;

        while(curr != null) {
            if(!curr.nomor.substring(0,
curr.nomor.length()-1).equals(index.last.nomor)) {
                index.add(new
Link(curr.nomor.substring(0, curr.nomor.length()-
1)));
                index.last.indexTo = curr;
            }
            curr = curr.next;
        }
       index.createIndex();
    }
```

### 5.1.3 get(String str, Link start)

This is a sequential searching method which will return the Link which contains the data matching with the search query (String str). The twist is, this method search Links starting from the "Link 'start'" to the 'right', instead from the first node. This method does **not** utilize indices, so with a huge number of data (of prefix), the search time will be longer.

```
public Link get(String str, Link start) {
        curr = start;
        while(curr != null) {
            System.out.println(curr.nomor + ":" +
str);
            if(curr.nomor.equals(str)) {
                return curr;
            }
            curr = curr.next;
        }
        System.out.println("Cannot find " + str);
        return null;
    }
```

### 5.1.4 get(String str)

It's the same with get(String str, Link start), except that it will search from the very first Link instead of the predetermined Link.

```
public Link get(String str) {
        return get(str, first);
    }
```

### 5.1.5 fastGet(String str)

This method is used as the 'vertical' traversal, where this method will be called recursively until it reaches the top of the index, then return the reference to the matching Link in the lower LinkList, subsequently until it returns the reference to the node with exact match.

```
public Link fastGet(String str) {
        Link node;
        if(index == null || index.isEmpty()) {
            node = first;
        }
        else {
            node = index.fastGet(str.substring(0,
str.length()-1));
        }
        Link result = get(str, node);
```

```
        if(result.indexTo == null) return result;
        else return result.indexTo;
    }
```

### 5.1.6 print()

This method is used to show the contents of a LinkList by iterating through the list.

```
 public void print() {
    curr = first;
        while(curr != null) {
            System.out.print(curr.nomor + " " +
curr.op);
            if(curr.indexTo != null)
System.out.print(" " + curr.indexTo.nomor);
            System.out.println();
            curr = curr.next;
        }
        System.out.println();
    }
```

### 5.1.7 printIndex()

This method is used to print the indices of the LinkList, starting from the top index to the bottom index.

```
  public void printIndex() {
        if(index != null && !index.isEmpty()) {
            index.printIndex();
        }
        print();
    }
```

### 5.1.8 size()

This method is used to return the number of Link of the LinkList.

```
  public int size() {
        if (first == null) return 0;
        int i=1;
        curr = first;
        while(curr != null) {
            i++;
            curr = curr.next;
        }
        return i;
    }
```

## 5.2 Optimizer

### 5.2.1 LoadFile()

This method is used to load all data from predetermined paths by calling all load methods consecutively. It must stay in a try...catch bracket because incorrect configuration or modified files may incur an unexpected exception.

```
public void loadFile() {
        try {
            loadOperator();
            loadPrefix();
            loadNumber();
            sortPrefix();
            loadSortedPrefix();
            assignPrefix();
            prefixList.printIndex();
    }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
```

### 5.2.2 generateRandom()

This method is used to generate all data (operators, prefixes, numbers, and costs) randomly, with specified criteria defined in a config file.

```
public void generateRandom() {
        loadSettings();
        generateOperator();
        assignCost();
        generatePrefix();
        generateNumber();
        sortPrefix();
        loadSortedPrefix();
        generatePrefixOp();
        assignPrefix();
    }
```

### 5.2.3 loadSettings()

This method is used to read the config file and configures the settings (used for random). This method uses Properties class to simplify the configuration acquiring.

```
public void loadSettings() {
        System.out.println("Loading config file...");

        String filename = root + "config.ini";
```

```
        System.out.println(filename);
           try {
               FileInputStream input = new
FileInputStream(filename);

               Properties prop = new Properties();
               prop.load(input);

               opQty =
Integer.parseInt(prop.getProperty("opQty"));
               minIntCost =
Integer.parseInt(prop.getProperty("minIntCost"));
               maxIntCost =
Integer.parseInt(prop.getProperty("maxIntCost"));
               minExtCost =
Integer.parseInt(prop.getProperty("minExtCost"));
               maxExtCost =
Integer.parseInt(prop.getProperty("maxExtCost"));
               prefixQty =
Integer.parseInt(prop.getProperty("prefixQty"));
               numberQty =
Integer.parseInt(prop.getProperty("numberQty"));

               System.out.println("Finished loading
configuration...");
           }
           catch (FileNotFoundException e) {
               System.out.println("Config file not
found, using default settings ...");
           }
        catch (IOException e) {
               e.printStackTrace();
           }

    }
```

### 5.2.4 loadOperator() and saveOperator()

These methods are used to load data of operators and their price, and saving them to a file. When saving, if the file does not exist, it will create a new one.

```
    public void loadOperator() throws Exception {
        costOp = new Vector<String>();
        costInt = new Vector<String>();
        costExt = new Vector<String>();

        System.out.println("Loading operator
list...");

        String filename = root + "operator.txt";
     System.out.println(filename);
        try {
            BufferedReader br = new
BufferedReader(new FileReader(filename));
```

```
            String line = br.readLine();
            String[] str;

            while (line != null) {
                str = line.split(",");
                costOp.add(str[0].trim());
                costInt.add(str[1].trim());
                costExt.add(str[2].trim());
                line = br.readLine();
            }
        }
        catch(FileNotFoundException e) {
            System.out.println("File not found");
        }
    }

    public void saveOperator() {
        System.out.println("Saving operator
list...");

        String filename = root + "operator.txt";
        File file = new File(filename);

    try {
            if(!file.exists())
                file.createNewFile();

            BufferedWriter bw = new
BufferedWriter(new FileWriter(filename));

            for (int i = 0; i < costOp.size(); i++) {
            bw.append(costOp.get(i) + "," +
costInt.get(i) + "," + costExt.get(i) + "\n");
            }
            bw.close();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
```

### 5.2.5 loadPrefix() and savePrefix()

These methods are used to load data of prefixes and their own operator, and saving them to a file. When saving, if the file does not exist, it will create a new one.

```
    public void loadPrefix() throws Exception {
        prefix = new Vector<String>();
        prefixOp = new Vector<String>();

        System.out.println("Loading prefix list...");

        String filename = root + "prefix.txt";
```

```
        try {
            BufferedReader br = new
BufferedReader(new FileReader(filename));

            String line = br.readLine();
            String[] str;

            while (line != null) {
                str = line.split(",");
                prefix.add(str[0].trim());
                prefixOp.add(str[1].trim());
                line = br.readLine();
            }
        }
    catch(FileNotFoundException e) {
            System.out.println("File not found");
        }
    }

    public void savePrefix() {
        System.out.println("Saving prefix list...");

        String filename = root + "prefix.txt";
        File file = new File(filename);

        try {
            if(!file.exists())
                file.createNewFile();

            BufferedWriter bw = new
BufferedWriter(new FileWriter(filename));

            if(prefixList == null) return;
            prefixList.curr = prefixList.getFirst();

            while(prefixList.curr != null) {
                bw.append(prefixList.curr.nomor + ","
+ prefixList.curr.op + "\n");
                prefixList.curr =
prefixList.curr.next;
            }
            bw.close();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
```

### 5.2.6 loadNumber() and saveNumber()

These methods are used to load data of numbers, and saving them to a file. When saving, if the file does not exist, it will create a new one.

```
public void loadNumber() throws Exception {
```

```
        number = new Vector<String>();

        System.out.println("Loading number list...");

        String filename = root + "number.txt";
        try {
            BufferedReader br = new
BufferedReader(new FileReader(filename));

            String str = br.readLine();
            if(str == null) {
                System.out.println("File is empty");
                generateNumber();
        }
            while (str != null) {
                number.add(str.trim());
                str = br.readLine();
            }
        }
        catch (FileNotFoundException e) {
            System.out.println("File not found");
        }
    }

    public void saveNumber() {
        System.out.println("Saving number list...");

        String filename = root + "number.txt";
        File file = new File(filename);

        try {
            if(!file.exists())
                file.createNewFile();

            BufferedWriter bw = new
BufferedWriter(new FileWriter(filename));

            for (int i = 0; i < number.size(); i++) {
                bw.append(number.get(i) + "\n");
            }
            bw.close();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
```

### 5.2.7 generateOperator()

Used to create random operator name (from A to Z, .i.e. operator "A", "B", and so on).

```
  public void generateOperator() { //membuat nama
operator secara random
```

```
        costOp = new Vector<String>();

        for(int i=0; i<opQty; i++) {
            costOp.add("" + (char)(i + 'A'));
        }

        printVector(costOp);
    }
```

### 5.2.8 assignCost()

Used to generate random text cost.

```
  public void assignCost() { //memasukkan biaya
telepon untuk sesama (INTernal) dan antaroperator
(EXTernal) secara random
        costInt = new Vector<String>();
        costExt = new Vector<String>();

        for(int i=0; i<opQty; i++) {
            costInt.add("" + (minIntCost +
r.nextInt(maxIntCost - minIntCost)));
            costExt.add("" + (minExtCost +
r.nextInt(maxExtCost - minExtCost)));
        }

        printVector(costInt);
        printVector(costExt);
    }
```

### 5.2.9 generatePrefix()

Used to generate random prefixes. It will always generate "08xxx". The `"%03d"` means it will always generate a string of 3 characters, i.e. if the generated number is 1, then it will generate "001" instead of "1".

```
  public void generatePrefix() { //membuat prefix
nomor secara random
        prefix = new Vector<String>();

        for(int i=0; i<prefixQty; i++) {
            prefix.add("08" + String.format("%03d",
r.nextInt(1000)));
        }

        printVector(prefix);
    }
```

### 5.2.10 generateNumber(int limit) and generateNumber()

Used to generate random numbers, based on available prefixes. The number of random numbers generated is equal to 'limit' parameter. Its generateNumber() counterpart will generate number equal to the specified in config file.

```
  public void generateNumber() { //membuat nomor-
nomor secara acak berdasarkan prefix yang sudah
ditetapkan
        number = new Vector<String>();

        generateNumber(numberQty);
    }

    public void generateNumber(int limit) { //membuat
nomor-nomor secara acak berdasarkan prefix yang sudah
ditetapkan
        for(int i=0; i<limit; i++) {

number.add(prefix.get(r.nextInt(prefix.size()-1)) +
String.format("%03d", r.nextInt(1000)));
        }

        printVector(number);
    }

    public void printVector(Vector<String> v) {
//mencetak isi dari sebuah Vector
        for(int i=0; i<v.size(); i++) {
        System.out.println(v.get(i));
        }
        System.out.println("------------");
    }
```

### 5.2.11 printVector(Vector<String> v)

Used to print all the contents of a Vector<String>.

```
    public void printVector(Vector<String> v) {
//mencetak isi dari sebuah Vector
        for(int i=0; i<v.size(); i++) {
            System.out.println(v.get(i));
        }
        System.out.println("------------");
    }
```

### 5.2.12 sortPrefix()

This method uses method sort() from the class Collection to sort the prefix.

```
public void sortPrefix() {
        sortedPrefix = prefix;
        Collections.sort(sortedPrefix);
    }
```

### 5.2.13 loadSortedPrefix()

Used to create the LinkList of the sorted prefix.

```
    public void loadSortedPrefix() {
        prefixList = new LinkList();
        prefixList.addVector(sortedPrefix);
        prefixList.createIndex();
        //prefixList.printIndex();
    }
```

### 5.2.14 generatePrefixOp()

Used to assign random operators to the each sorted prefix, i.e. "08576" will correspond to "Tri".

```
    public void generatePrefixOp() { //mengatur
operator untuk tiap prefix secara random
        prefixOp = new Vector<String>();

        for(int i=0; i<prefixQty; i++) {

prefixOp.add(costOp.get(r.nextInt(opQty)));
        }

        printVector(prefixOp);
    }
```

### 5.2.15 assignPrefix()

Used to match all random operators from generatePrefixOp() to the LinkList.

```
    public void assignPrefix() {
        prefixList.curr = prefixList.getFirst();

        for(int i=0; i<prefixOp.size(); i++) {
            prefixList.curr.op = prefixOp.get(i);
            prefixList.curr = prefixList.curr.next;
        }
    }
```

### 5.2.16 findCost(String nomor, String op)

Used to find the cost of SMS/text from a specified number (nomor) to a particular operator (op). If the fastMode is toggled

on, then the search will be done with fastGet() method instead of get() method.

```
    public int findCost(String nomor, String op) {
        String nomorPrefix = nomor.substring(0, 4);
        String nomorOp;

        if(fastMode) nomorOp =
prefixList.fastGet(nomorPrefix).op;
        else nomorOp =
prefixList.get(nomorPrefix).op;

        int indexPrefix = costOp.indexOf(nomorOp);

        if(nomorOp.equals(op))
            return
Integer.parseInt(costInt.get(indexPrefix));
        else
            return
Integer.parseInt(costExt.get(indexPrefix));
    }
```

### 5.2.17 findTotalCost(String op)

Used to find the total cost needed to text all numbers in the specified list with a number from operator specified (op).

```
    public int findTotalCost(String op) {
        int totalCost =0;

        Enumeration vEnum = number.elements();
        while(vEnum.hasMoreElements()) {
            String nomorEnum =
vEnum.nextElement().toString();
            //System.out.println(nomorEnum);
            totalCost += findCost(nomorEnum, op);
        }
        return totalCost;
    }
```

### 5.2.18 optimize(boolean fastMode) and optimize()

Used to iterate through all available operators, and determine which one has the least total cost. The method may be called in fast mode (input **true** as the parameter) or in normal mode (input **false** as the parameter). The mode is defaulted as **true** (using optimize() instead of optimize(boolean)).

```
    public void optimize(boolean fastMode) {
        int min = 99999999;
        String op = "";
```

```
        this.fastMode = fastMode;

        Enumeration vEnum = costOp.elements();
        while(vEnum.hasMoreElements()) {
            String nomorOp =
vEnum.nextElement().toString();

            //System.out.println(nomorOp);

            int totalCost = findTotalCost(nomorOp);

            if(min > totalCost) {
                min = totalCost;
                op = nomorOp;
            }
        }
        System.out.println("Paling murah pakai
operator " + op + " cuma " + min);
    }
```

## 5.3 GUI

### 5.3.1 optimize(boolean fastMode)

It functions the same as the method optimize() in Optimizer. But instead showing the output in terminal, it will show it in the output area. The RowSorter sorts the result based on the total cost, from the lowest (optimal) cost to the highest cost.

```
   public void optimize(boolean fastMode) {
        int min = 99999999;
        int i=0;
        String op = "";

        Enumeration vEnum = o.costOp.elements();

        while(vEnum.hasMoreElements()) {
            String nomorOp =
vEnum.nextElement().toString();

         System.out.println(nomorOp);

            int totalCost = o.findTotalCost(nomorOp);

            try {
                calcModel.setValueAt(nomorOp, i, 0);
                calcModel.setValueAt(totalCost, i,
1);
            }
            catch (Exception e) {
                calcModel.addRow(new String[]
{ nomorOp, "" + totalCost });
            }

            if(min > totalCost) {
```

```
                min = totalCost;
                op = nomorOp;
            }
            i++;
        }

        calcTable.setAutoCreateRowSorter(true);
        RowSorter sorter = calcTable.getRowSorter();
        sorter.setSortKeys(Arrays.asList(new
RowSorter.SortKey(1, SortOrder.ASCENDING));

        print("Paling murah pakai operator " + op);
    }
```

### 5.3.2 UpdateTable()

This method creates a new TableModel which synchronizes with the vectors in Optimizer. After the TableModel is changed, the method fireTableDataChanged() will force the GUI to redraw the contents of the table.

```
   public void updateTable() {
        DefaultTableModel numberModel = new
DefaultTableModel();
        numberModel.addColumn("Number", o.number);
        numberTable.setModel(numberModel);
        numberModel.fireTableDataChanged();

        DefaultTableModel prefixModel = new
DefaultTableModel();
        prefixModel.addColumn("Prefix",
o.sortedPrefix);
        prefixModel.addColumn("Operator",
o.prefixOp);
        prefixTable.setModel(prefixModel);
        prefixModel.fireTableDataChanged();

        DefaultTableModel opModel = new
DefaultTableModel();
        opModel.addColumn("Operator", o.costOp);
        opModel.addColumn("Sesama", o.costInt);
        opModel.addColumn("Antar", o.costExt);
        opTable.setModel(opModel);
        opModel.fireTableDataChanged();
    }
```