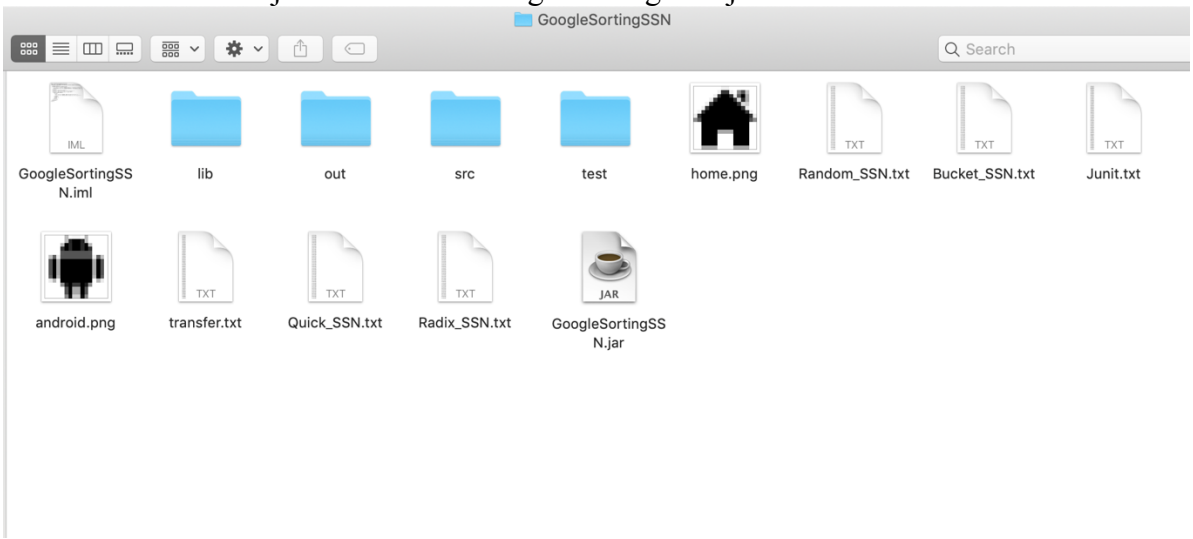
Programming Assignment 2:
Sorting Social Security Numbers

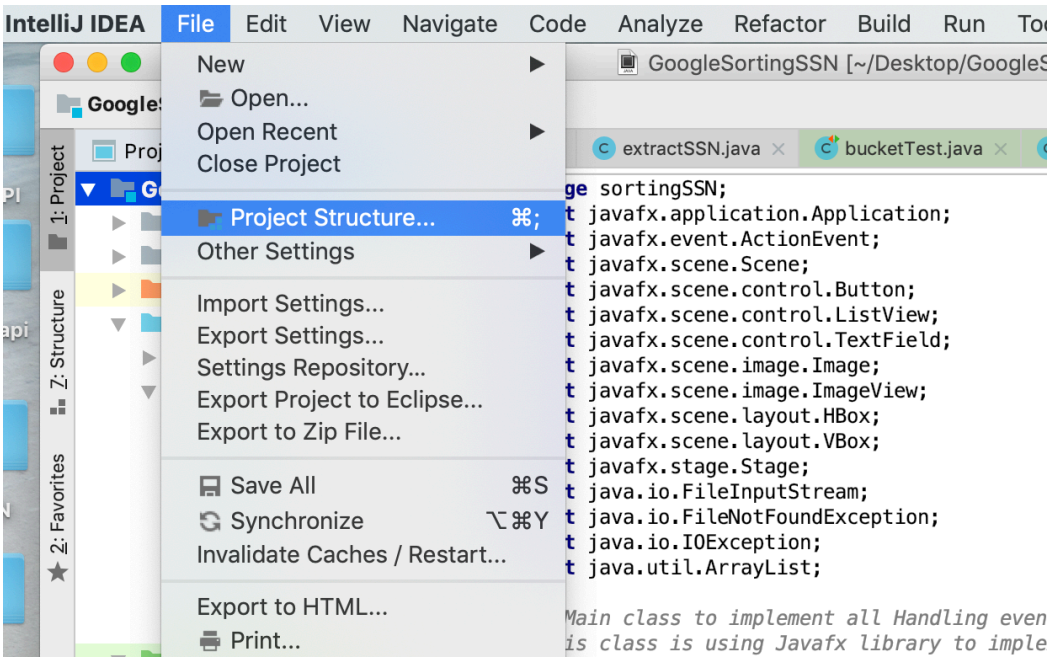
Yuxiao (Tom) Zheng
CS146-Section7

1. **unzip and install**

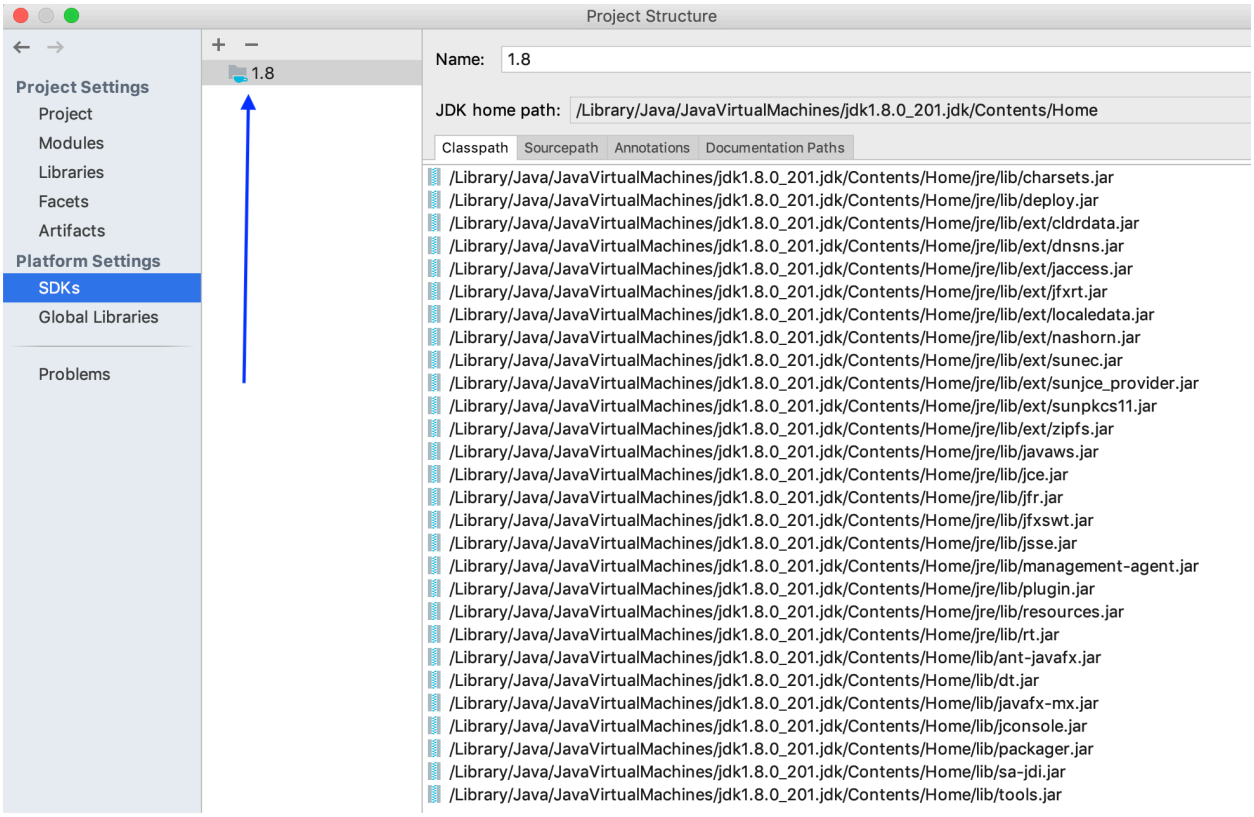
- Unzip the zip folder, there are three part, a **SRC** folder that contains all of the .java files, Quick_SSN.txt, Radix_SSN.txt, Bucket_SSN.txt, Random_SSN.txt, home.png, android.png, transfer.txt, Junit.txt, and a **TEST** folder contains Junit test .java files: bucketTest.java, quickTest.java, and radixTest.java
  a Programing Report which is Microsoft Word Document which you are reading it now and a runnable .jar file named “GoogleSortingSSN.jar”.



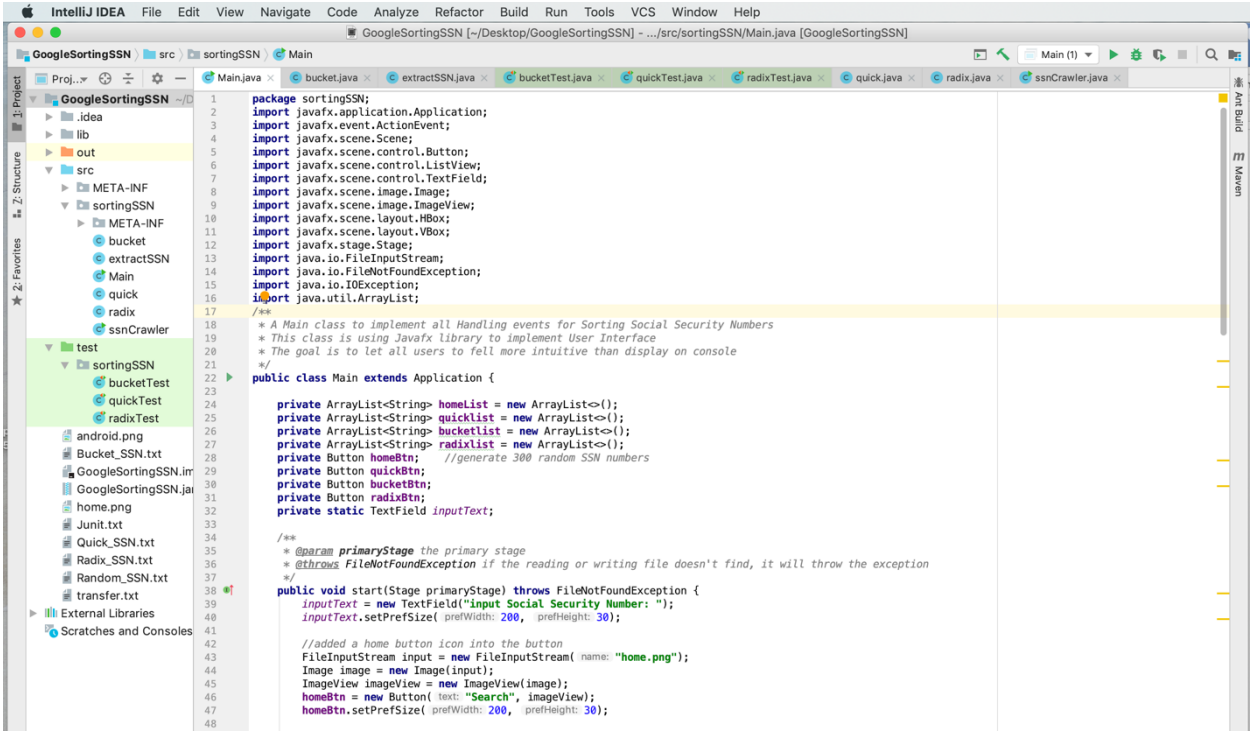
- Now the above are the all of required files including icon .png
- Still need to import jdk 8, because we using javaFX lirbary



1. Click File → Project Structure
2. Choose SDKs → click "+"
3. Pick the jdk 8 to import

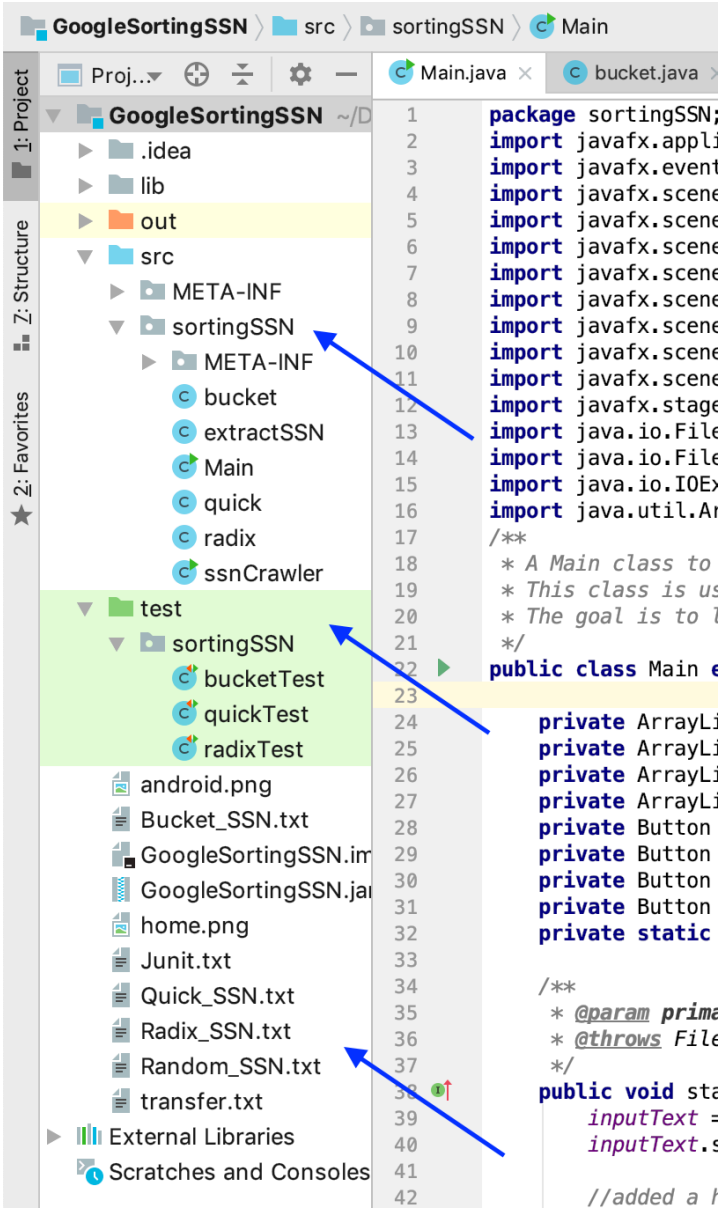


- Now, the whole program interface look like the following:



- If you have seen this program main interface, that means you have installed successfully! Now you can test all function and Junit test!

- Note: the left directory path should look like the following:

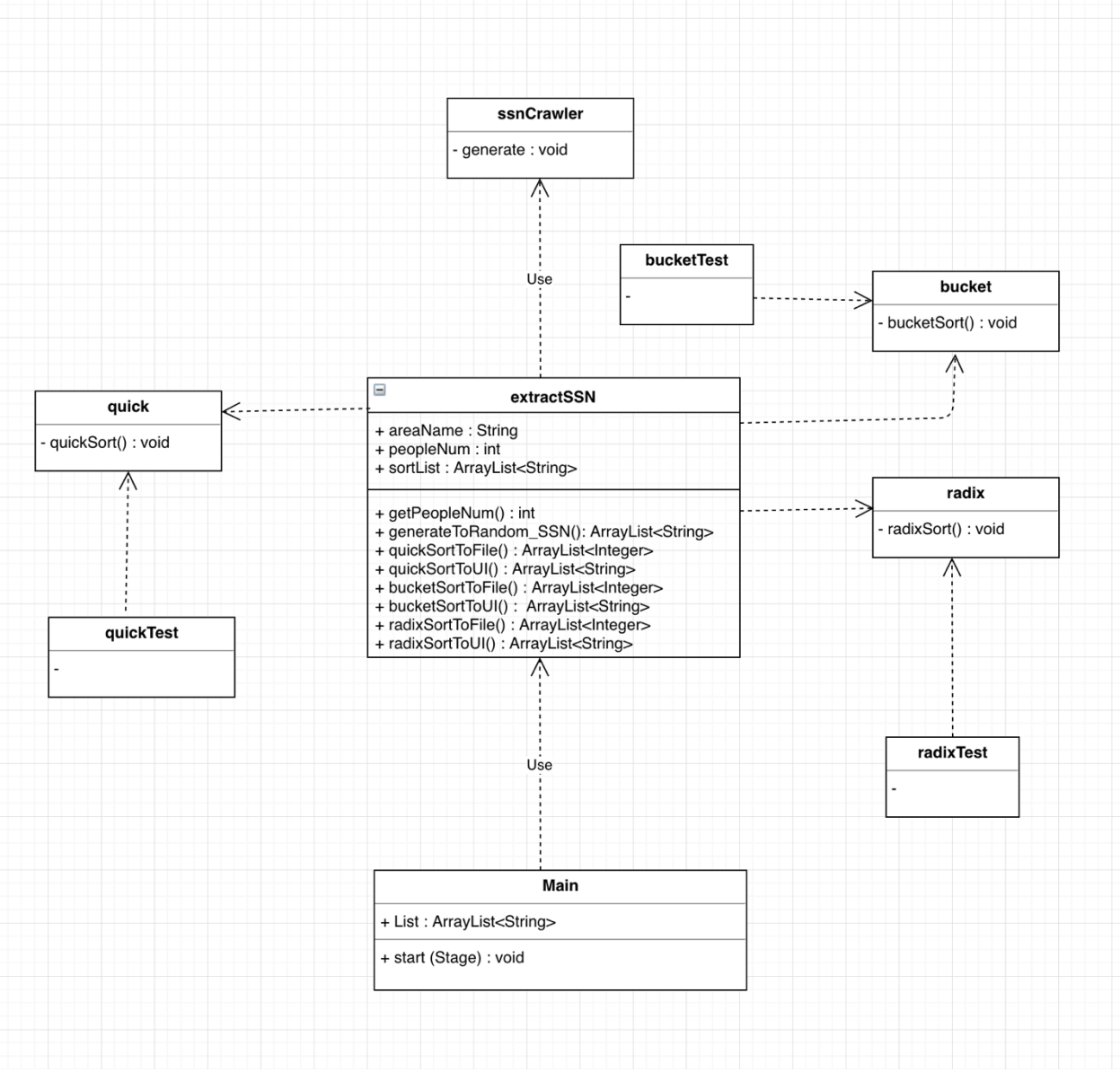


- Warning: Don't move the **.txt file** and **.png file** to the other folder, or you won't see the icon display on the button in the **User Interface**, or you won't see the **SSN** numbers into the **.txt** files.

- Note: the further installation for **Junit Test**, you will see it in **Part 8 :**
  Running Time Big-**O** (Junit -Test)

2. **Programming Design (deeply encapsulation)**

- **UML** diagram:



- Create a **ssnCrawler** class which has a method named [void generate()], this method will generate 300 random SSN numbers.

- Create a class **extractSSN** which has some necessary attributes:

```
/* Area name */
private String necs = "[Northeast Coast States] : ";
private String scs = "[South Coast States] :  ";
private String ms = "[Middle States] :  ";
private String ms = "[Middle States] :  ";
private String nwcs = "[Northwest Coast States] : ";
private String wcs = "[West Coast States]  :  ";
/* Area people number */
public static int necsPeople;
public static int scsPeople;
public static int msPeople;
public static int nwcsPeople;
public static int wcsPeople;
/* after quickSort, bucketSort, and radixSort, all sorted data will into the
required ArrayList */
private ArrayList<String> strlist = new ArrayList<>();
private ArrayList<String> quickList = new ArrayList<>();
private ArrayList<String> bucketList = new ArrayList<>();
private ArrayList<String> radixList = new ArrayList<>();
/* Access Method: get different area people's number */
public int getNecsPeople(){
    return necsPeople;
}
public int getScsPeople(){
    return scsPeople;
}
public int getMsPeople(){
    return msPeople;
}
public int getNwcsPeople(){
    return nwcsPeople;
}
public int getWcsPeople(){
    return wcsPeople;
}
```

```
 */
public class extractSSN {
    /* Area name */
    private String necs = "[Northeast Coast States] : ";
    private String scs = "[South Coast States] :  ";
    private String ms = "[Middle States] :  ";
    private String nwcs = "[Northwest Coast States] : ";
    private String wcs = "[West Coast States]  :  ";
    /* Area people number */
    public static int necsPeople;
    public static int scsPeople;
    public static int msPeople;
    public static int nwcsPeople;
    public static int wcsPeople;
    /* after quickSort, bucketSort, and radixSort, all sorted data will into the required ArrayList */
    private ArrayList<String> strlist = new ArrayList<>();
    private ArrayList<String> quickList = new ArrayList<>();
    private ArrayList<String> bucketList = new ArrayList<>();
    private ArrayList<String> radixList = new ArrayList<>();

    /* Access Method: get different area people's number */
    public int getNecsPeople() { return necsPeople; }
    public int getScsPeople() { return scsPeople; }
    public int getMsPeople() { return msPeople; }
    public int getNwcsPeople() { return nwcsPeople; }
    public int getWcsPeople() { return wcsPeople; }
```

The above 4 parts:

1. displaying the area name on the **User Interface**
2. will count the total numbers of different area's people
3. provides a data structure ArrayList, which can store 300 sorted SSN numbers into my arrayList
4. return the changed people number of different area

- Inside **extractSSN** class, I will create seven method to help implement different functioning:

```
1. public ArrayList<String> generateToRandom_SSN()
2. public ArrayList<Integer> quickSortToFile()
3. public ArrayList<String> quickSortToUI()
4. public ArrayList<Integer> bucketSortToFile()
5. public ArrayList<String> bucketSortToUI()
6. public ArrayList<Integer> radixSortToFile()
7. public ArrayList<String> radixSortToUI()
```

Originally, this seven methods'codes will write into the Main class; they help Main class implement different functioning.

Now, from method() 1 to 7, I only write either one sentence or two sentences inside **try()** clase of different button event in the **Main** class to implement the user's desired functioning : this way offers the **highest possible degeree of encapsulation;** see the **codes** of 7 methods encapsulating in extractSSN class. I will further explain in Part 4, **"4. Classes/Subrountines/function call"**.

- Create a class **quick** which provide a method called **quickSort()** to help sort 300 **SSN** numbers

```java
public class quick {

    //Create a new arrayList in order to add ssn numbers
    private ArrayList<Integer> arrlist;


    /**
     * The following procedure implements quickSort
     * @param arr the given arrayList
     * @param p the left position
     * @param r the right position
     */
    public void quickSort(ArrayList<Integer>arr, int p, int r){
        if(p < r){
            int q = Partition(arr, p, r);
            quickSort(arr, p,  r: q-1);
            quickSort(arr,  p: q+1, r);
        }
    }
```

- Create a class **bucket** which provide a method called **bucketSort()** to help sort 300 **SSN** numbers

```java
public class bucket{
    /**
     * Convert a Integer arrayList to a double array[]
     * @param intarr the given arrayList with Integer elements
     * @return a double array[]
     */
    public double[] convertArr(ArrayList<Integer> intarr){
        double [] darr = new double[intarr.size()];
        for(int i=0; i<intarr.size(); i++){
            darr[i] =  (intarr.get(i))/ 1000000000.0;
        }
        return darr;
    }

    /**
     * BuckSort method to sort a double array
     * @param arr the required arrayList will order by calling bucketSort()
     * @return the new ArrayList with calling buckSort()
     */
    public void buckSort(double[] arr){
```

- Create a class **radix** which provide a method called **raidxSort()** to help sort 300 **SSN** numbers

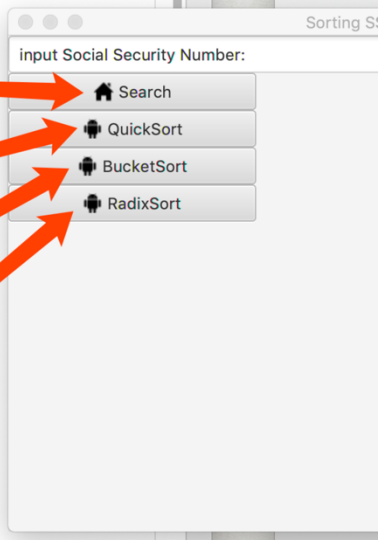
```
public class radix {

    /**
     * @param arr the required arrayList to convert to a integer array
     * @return a new integer array
     */
    public int[] convertToArr(ArrayList<Integer> arr){
        int [] iarr = new int[arr.size()];
        for(int i=0; i<arr.size(); i++){
            iarr[i] =  arr.get(i);
        }
        return iarr;
    }
    /**
     * Get the maximum from this array
     * @param arr the required array will be sorted
     * @param n the length of this array
     * @return a maximum
     */
    public int getMax(int arr[], int n){
        int max = arr[0];
        for(int i=1; i<n; i++){
            if(arr[i] > max){
                max = arr[i];
            }
        }
        return max;
    }
```

- Create a **Main** class with 4 arrayLists and 4 Buttons :

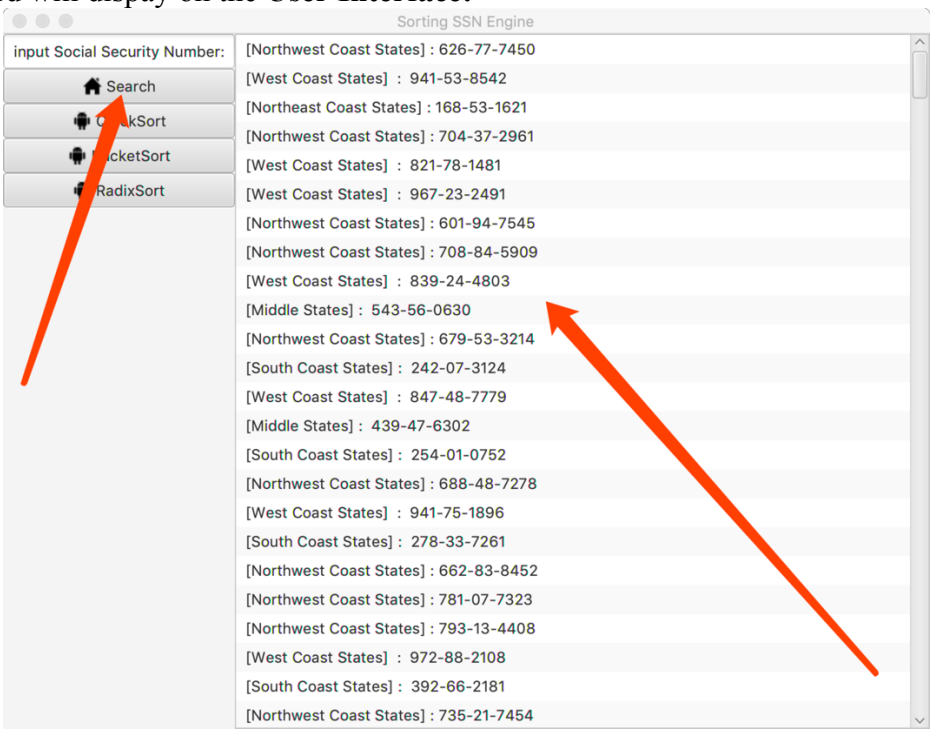
```
private ArrayList<String> homeList = new ArrayList<>();
private ArrayList<String> quicklist = new ArrayList<>();
private ArrayList<String> bucketlist = new ArrayList<>();
private ArrayList<String> radixlist = new ArrayList<>();
private Button homeBtn;    //generate 300 random SSN numbers
private Button quickBtn;
private Button bucketBtn;
private Button radixBtn;
```

- Every button will add a .png file on the button, then the **User Interface** will show a button with the image (icon) :

```java
/**
 * @param primaryStage the primary stage
 * @throws FileNotFoundException if the reading or writing file doesn't find, it will throw the exception
 */
public void start(Stage primaryStage) throws FileNotFoundException {
    inputText = new TextField("input Social Security Number: ");
    inputText.setPrefSize( prefWidth: 200,  prefHeight: 30);

    //added a home button icon into the button
    FileInputStream input = new FileInputStream( name: "home.png");
    Image image = new Image(input);
    ImageView imageView = new ImageView(image);
    homeBtn = new Button( text: "Search", imageView);
    homeBtn.setPrefSize( prefWidth: 200,  prefHeight: 30);

    //added a quickSort button icon into the button
    FileInputStream input1 = new FileInputStream( name: "android.png");
    Image image1 = new Image(input1);
    ImageView imageView1 = new ImageView(image1);
    quickBtn = new Button( text: "QuickSort", imageView1);
    quickBtn.setPrefSize( prefWidth: 200,  prefHeight: 30);

    //added a bucketSort button icon into the button
    FileInputStream input2 = new FileInputStream( name: "android.png");
    Image image2 = new Image(input2);
    ImageView imageView2 = new ImageView(image2);
    bucketBtn = new Button( text: "BucketSort", imageView2);
    bucketBtn.setPrefSize( prefWidth: 200,  prefHeight: 30);

    //added a radixSort button icon into the button
    FileInputStream input3 = new FileInputStream( name: "android.png");
    Image image3 = new Image(input3);
    ImageView imageView3 = new ImageView(image3);
    radixBtn = new Button( text: "RadixSort", imageView3);
    radixBtn.setPrefSize( prefWidth: 200,  prefHeight: 30);
```



- Inside **Main** class, there are 4 **arrayList<String>** to hold the sorted **SSN** numbers, this step's purpose is to one by one call the **ListView<String>** to add every record, finally all record will dispay on the **User Interface:**



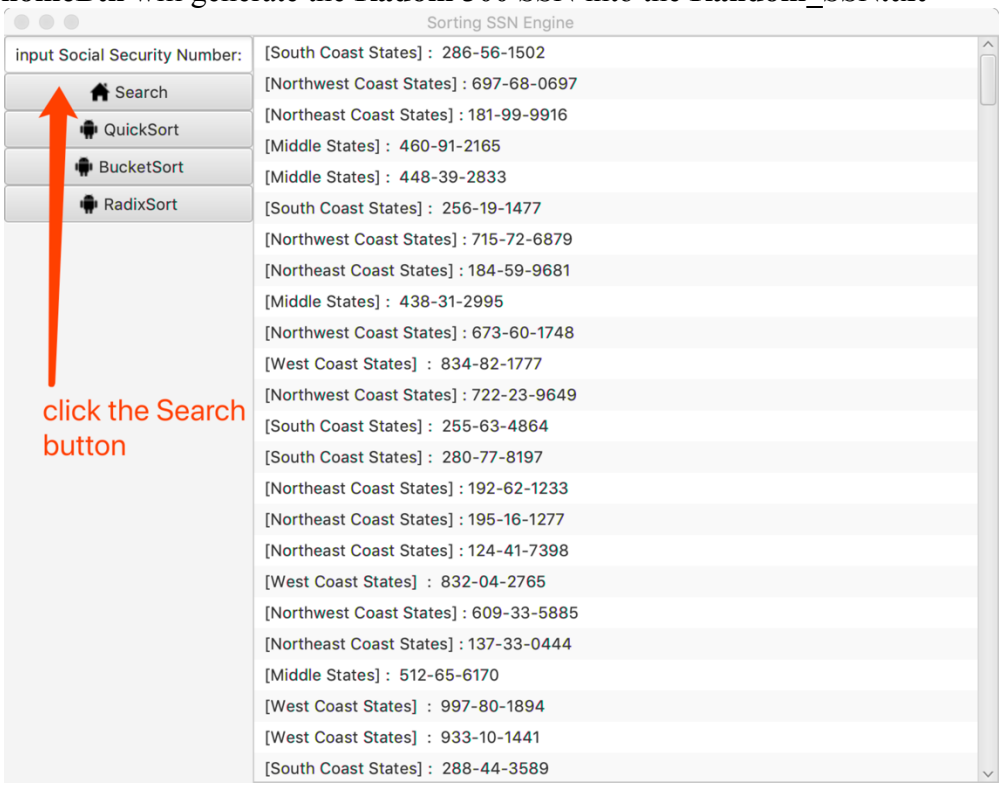
- In the Main class, there are 4 **button event** that will implment different functioning;
- A primary method **start(Stage ..)** to execute the main task.

```
public void start(Stage primaryStage) throws FileNotFoundException { … }
```

  - four different **button** to implement different desired functioning:

```
homeBtn.setOnAction((ActionEvent event)-> { … }
```

homeBtn will generate the **Radom 300 SSN** into the **Random_SSN.txt**



`quickBtn.setOnAction((ActionEvent event)->{ … }`

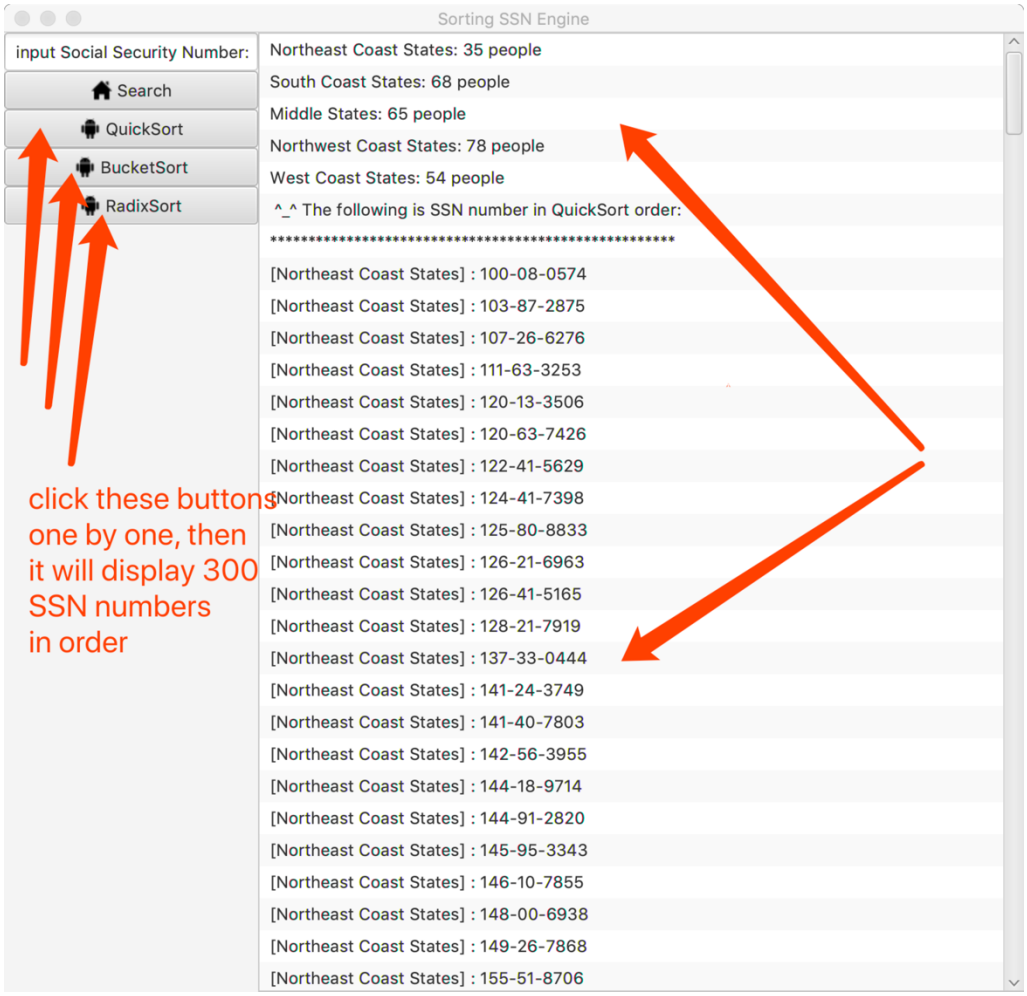
`quickBtn will display the` **300 SSN** on the **User Interface**, and it will write **300 SSN** into the **Quick_SSN.txt**

`bucketBtn.setOnAction((ActionEvent event)->{ … }`

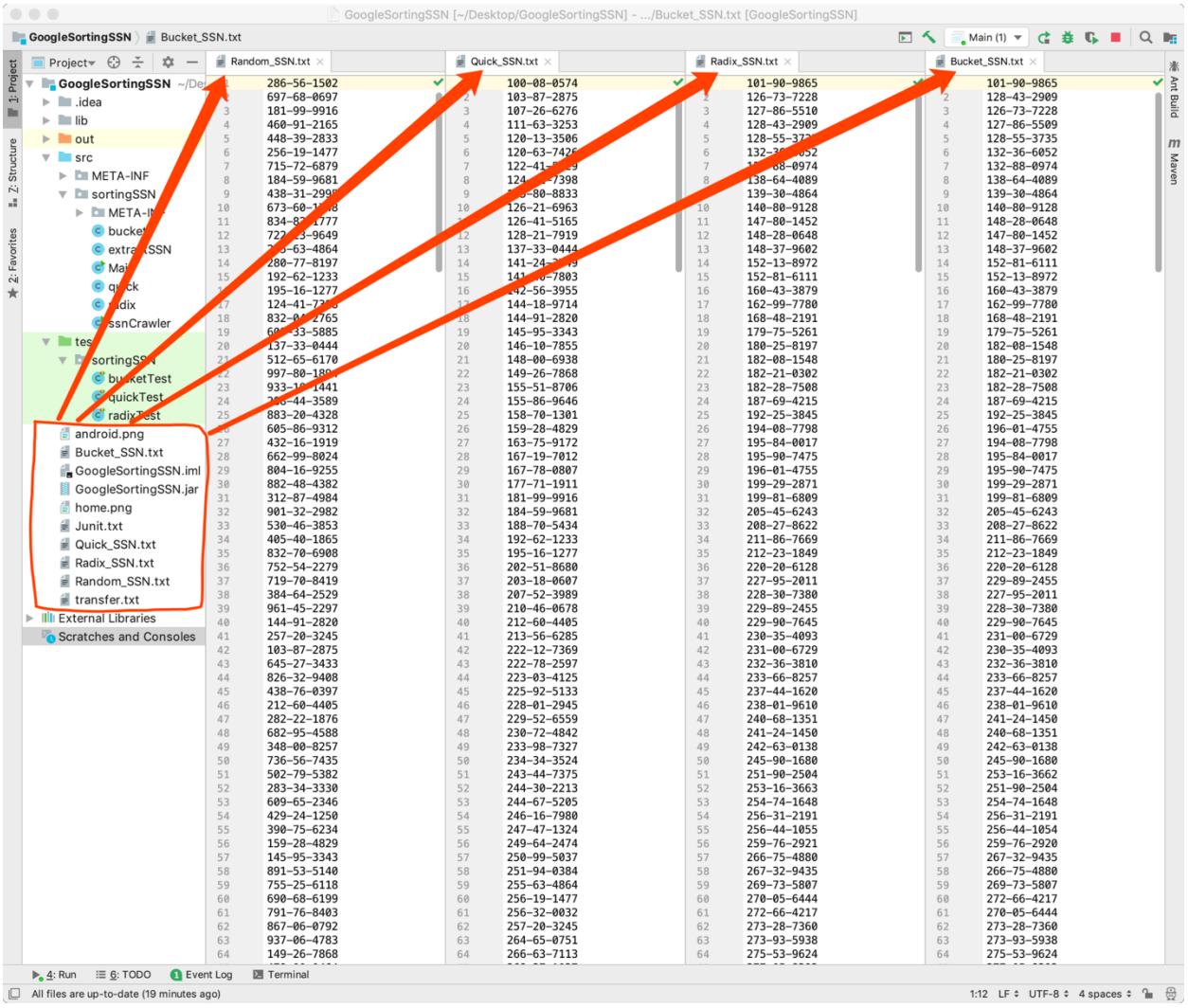
`bucketBtn will display the` **300 SSN** on the **User Interface**, and it will write **300 SSN** into the **Quick_SSN.txt**

`radixBtn.setOnAction((ActionEvent event)->{ … }`

`radixBtn will display the` **300 SSN** on the **User Interface**, and it will write **300 SSN** into the **Radix_SSN.txt**



- The following shows that these **button event** can also write all SSN data into the required .txt files.

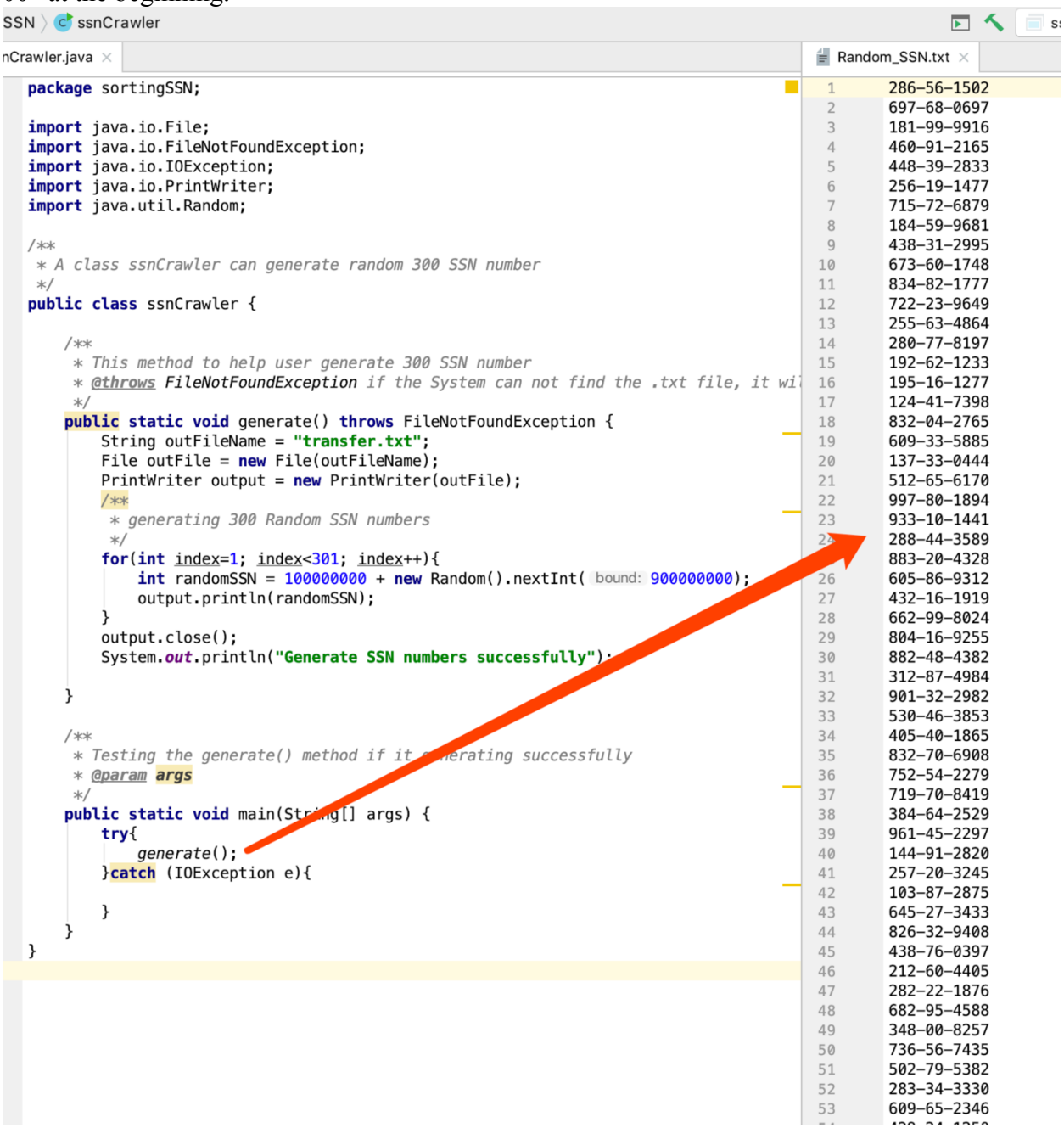


3. **Implementation**

ssnCrawler class has a method **void generate()** , which generate 300 SSN numbers.

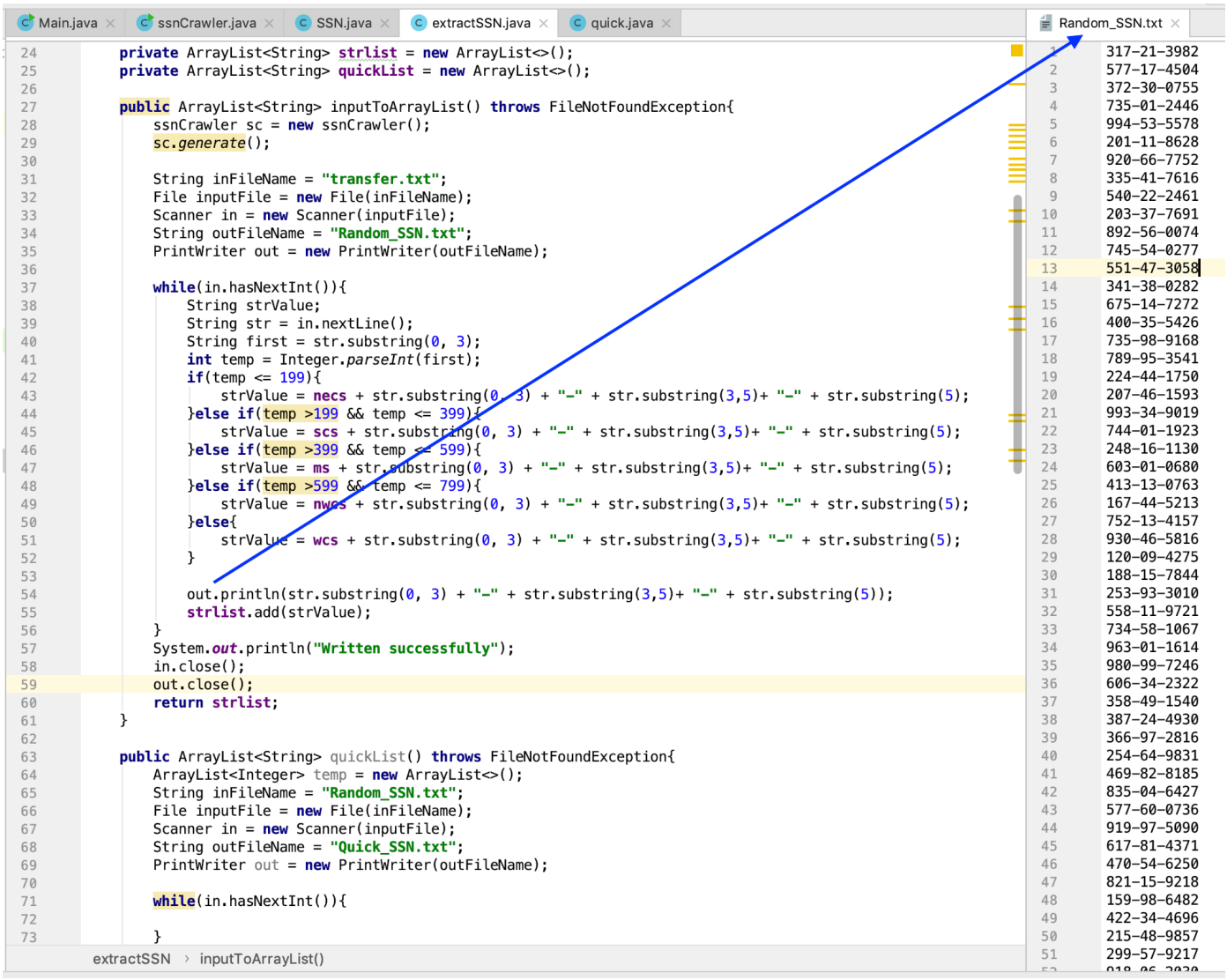
```
int randomSSN = 100000000 + new Random().nextInt(900000000);
```

this statement will generate a 9-place number, from 0 ~ 999-99-9999, **Note:** it **will not** generate with “000” at the beginning.



Implement extractSSN class's functioning:

1. `public ArrayList<String> generateToRandom_SSN()`

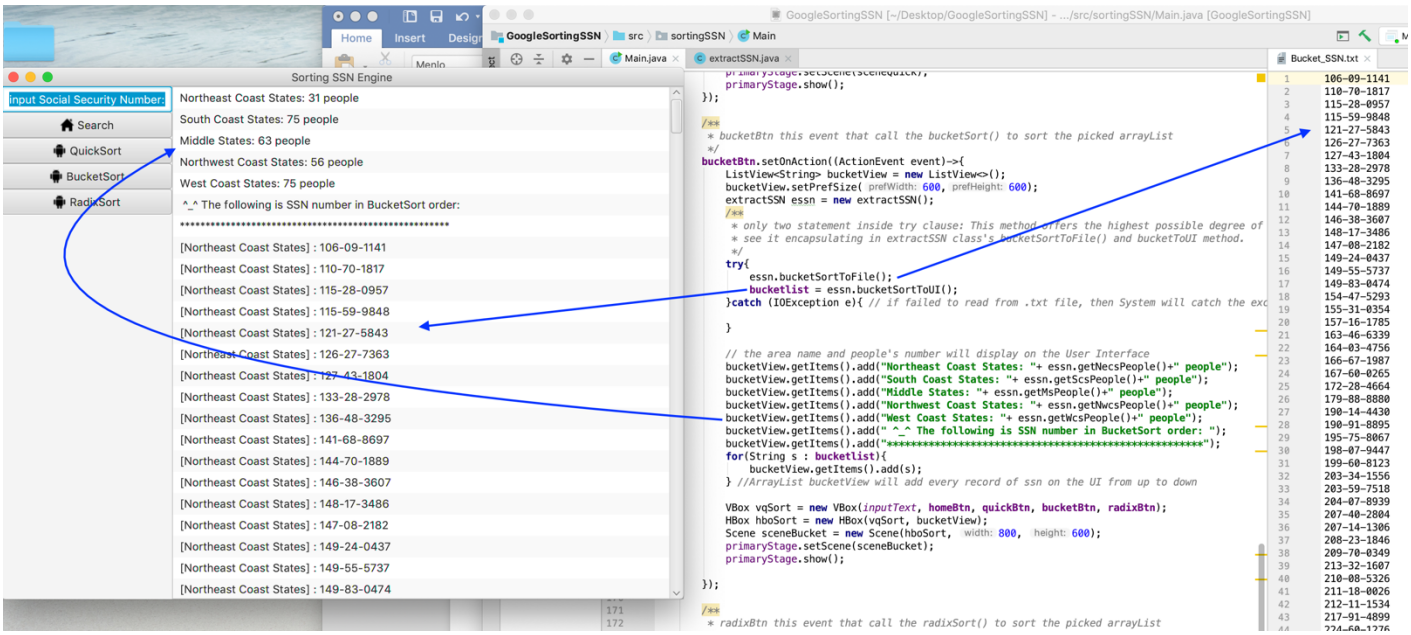



```java
private ArrayList<String> strlist = new ArrayList<>();
private ArrayList<String> quickList = new ArrayList<>();

public ArrayList<String> inputToArrayList() throws FileNotFoundException{
    ssnCrawler sc = new ssnCrawler();
    sc.generate();

    String inFileName = "transfer.txt";
    File inputFile = new File(inFileName);
    Scanner in = new Scanner(inputFile);
    String outFileName = "Random_SSN.txt";
    PrintWriter out = new PrintWriter(outFileName);

    while(in.hasNextInt()){
        String strValue;
        String str = in.nextLine();
        String first = str.substring(0, 3);
        int temp = Integer.parseInt(first);
        if(temp <= 199){
            strValue = necs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5);
        }else if(temp >199 && temp <= 399){
            strValue = scs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5);
        }else if(temp >399 && temp <= 599){
            strValue = ms + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5);
        }else if(temp >599 && temp <= 799){
            strValue = nwcs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5);
        }else{
            strValue = wcs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5);
        }

        out.println(str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5));
        strlist.add(strValue);
    }
    System.out.println("Written successfully");
    in.close();
    out.close();
    return strlist;
}

public ArrayList<String> quickList() throws FileNotFoundException{
    ArrayList<Integer> temp = new ArrayList<>();
    String inFileName = "Random_SSN.txt";
    File inputFile = new File(inFileName);
    Scanner in = new Scanner(inputFile);
    String outFileName = "Quick_SSN.txt";
    PrintWriter out = new PrintWriter(outFileName);

    while(in.hasNextInt()){

    }
```

```
317-21-3982
577-17-4504
372-30-0755
735-01-2446
994-53-5578
201-11-8628
920-66-7752
335-41-7616
540-22-2461
203-37-7691
892-56-0074
745-54-0277
551-47-3058
341-38-0282
675-14-7272
400-35-5426
735-98-9168
789-95-3541
224-44-1750
207-46-1593
993-34-9019
744-01-1923
248-16-1130
603-01-0680
413-13-0763
167-44-5213
752-13-4157
930-46-5816
120-09-4275
188-15-7844
253-93-3010
558-11-9721
734-58-1067
963-01-1614
980-99-7246
606-34-2322
358-49-1540
387-24-4930
366-97-2816
254-64-9831
469-82-8185
835-04-6427
577-60-0736
919-97-5090
617-81-4371
470-54-6250
821-15-9218
159-98-6482
422-34-4696
215-48-9857
299-57-9217
```

2. `public ArrayList<Integer> quickSortToFile()`
   `public ArrayList<String> quickSortToUI()`

- This two methods either write **300 SSN** to the **.txt** file or display 300 SSN on the **User Interface**, see the result in the following:

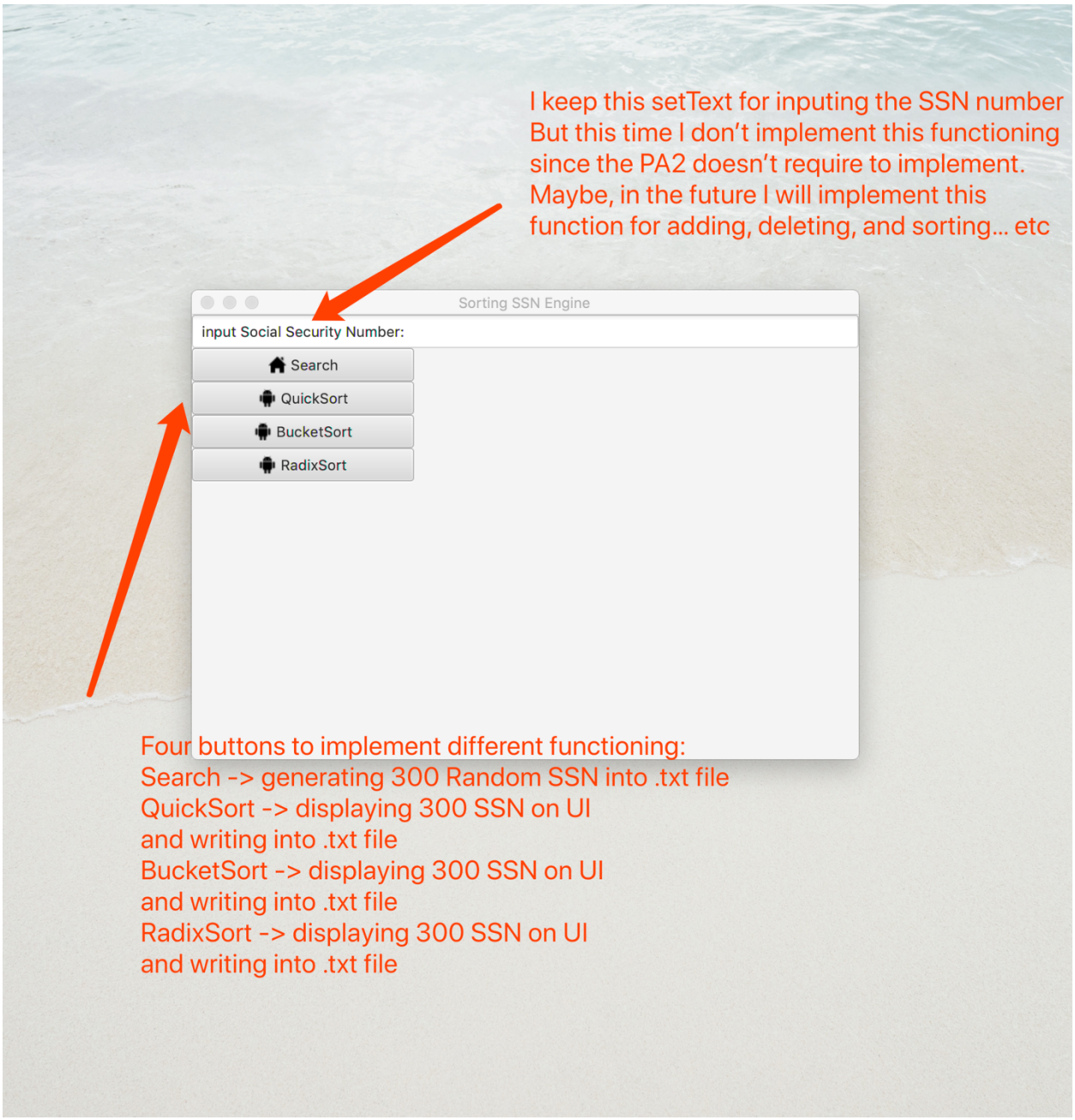


- Then I implement **quick** class for the **quickSort()** method
  The main functioning, I will show in the **Part 4**: A list of **classes/subroutines**

- I also implement **bucket** class for the **bucketSort()** method
  The main functioning, I will show in the **Part 4**: A list of **classes/subroutines**

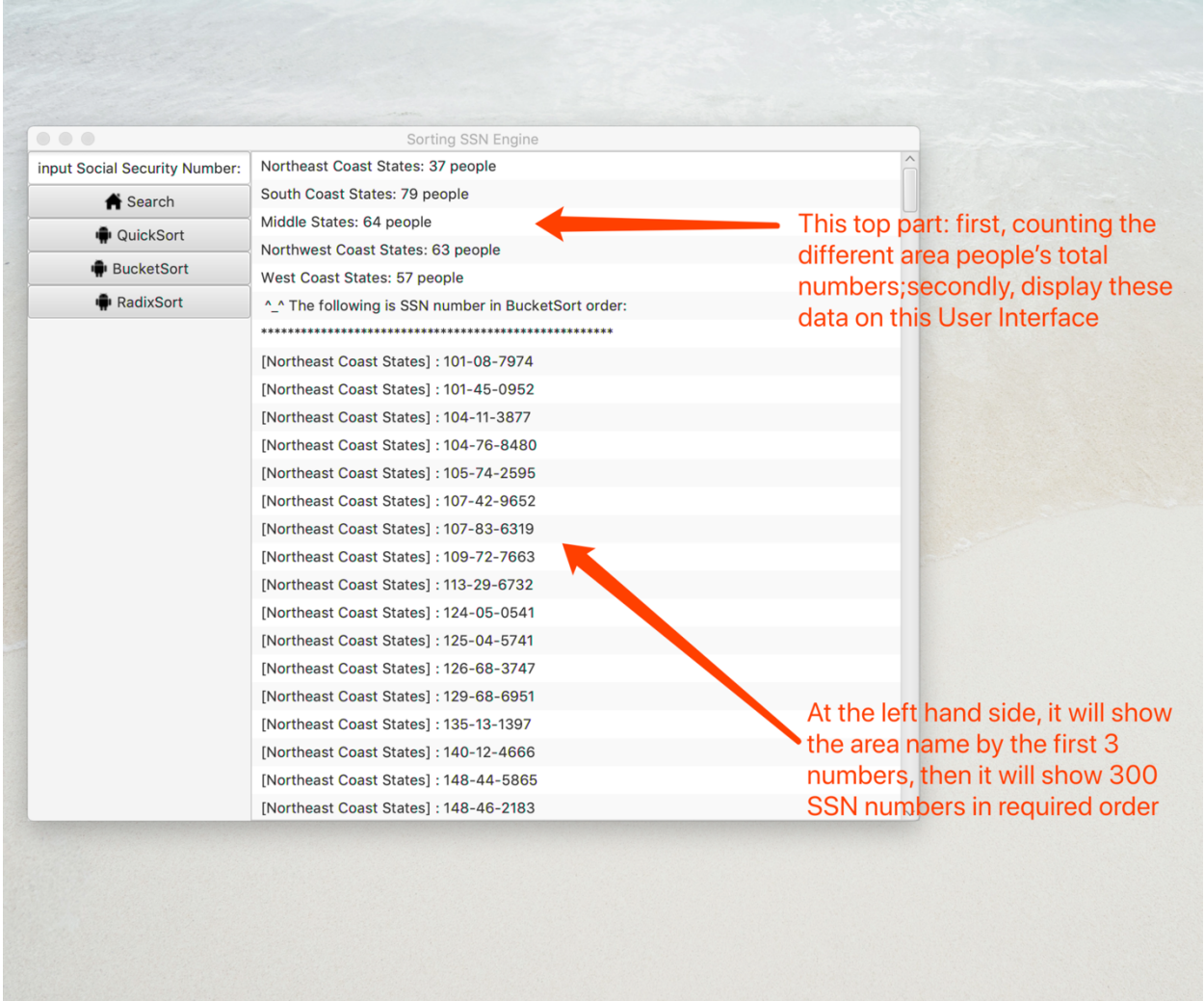
- I implement **radix** class for the **radixSort()** method
  The main functioning, I will show in the **Part 4**: A list of **classes/subroutines**

- After finished the above working, the **Main** class can implement four different required functioning:



- The main User Interface will show two parts:

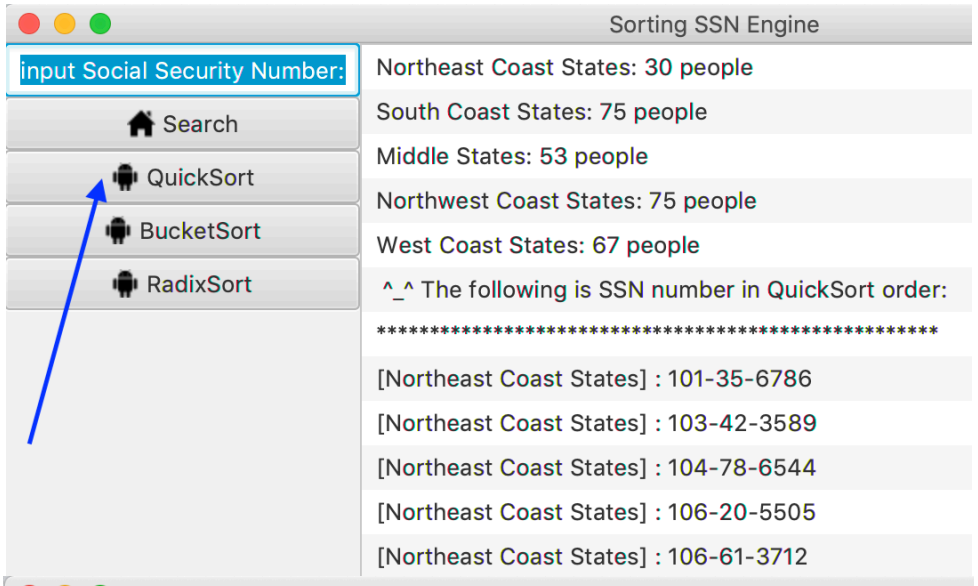
    1. Button component with image icon
    2. Console will display SSN numbers, Area name, and the number of people

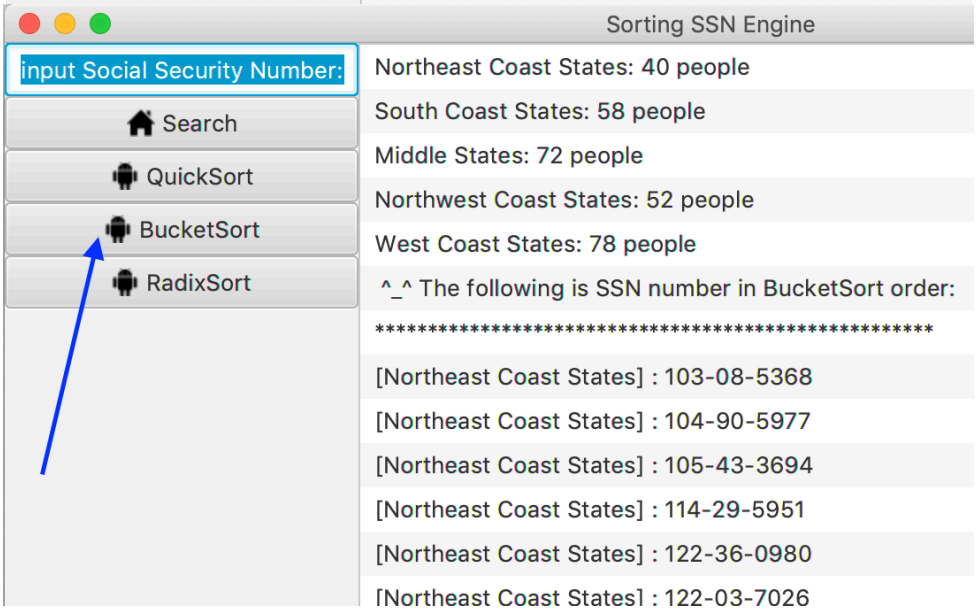


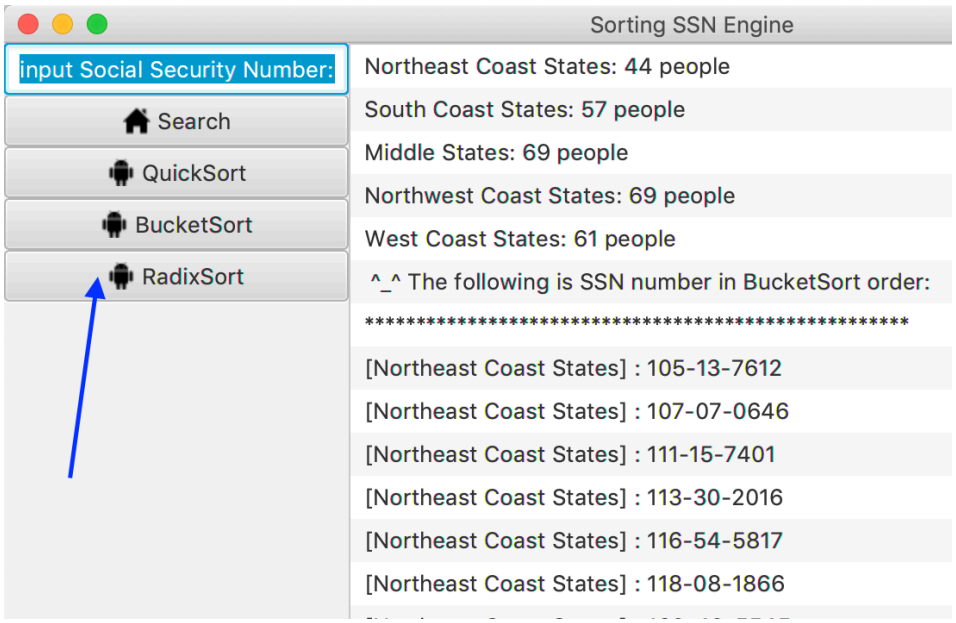
To summary: so far so good, I have **implemented** all class's functioning.
In chapter 4, I will further explain my every class's method, class's subroutines, and every function calls.

4. **classes/subroutines/function calls:**

- In the **extractSSN** class, there is a method `ArrayList<String> generateToRandom_SSN()`
    1. Creating a **object** of **ssnCrawler**; calling the **generate()** method to generate 300 **Random SSN** numbers.
    2. The **generate()** will write 300 SSN into the **transfer.txt** ; this step's purpose is to make **transfer.txt** be a buffered .txt file. Because without **transfer.txt**, we only read data from **Random_SSN.txt** so that we can't see the change after we call **quickSort(), bucketSort(), and radixSort()**. Therefore, we can see the different and new SSN numbers.



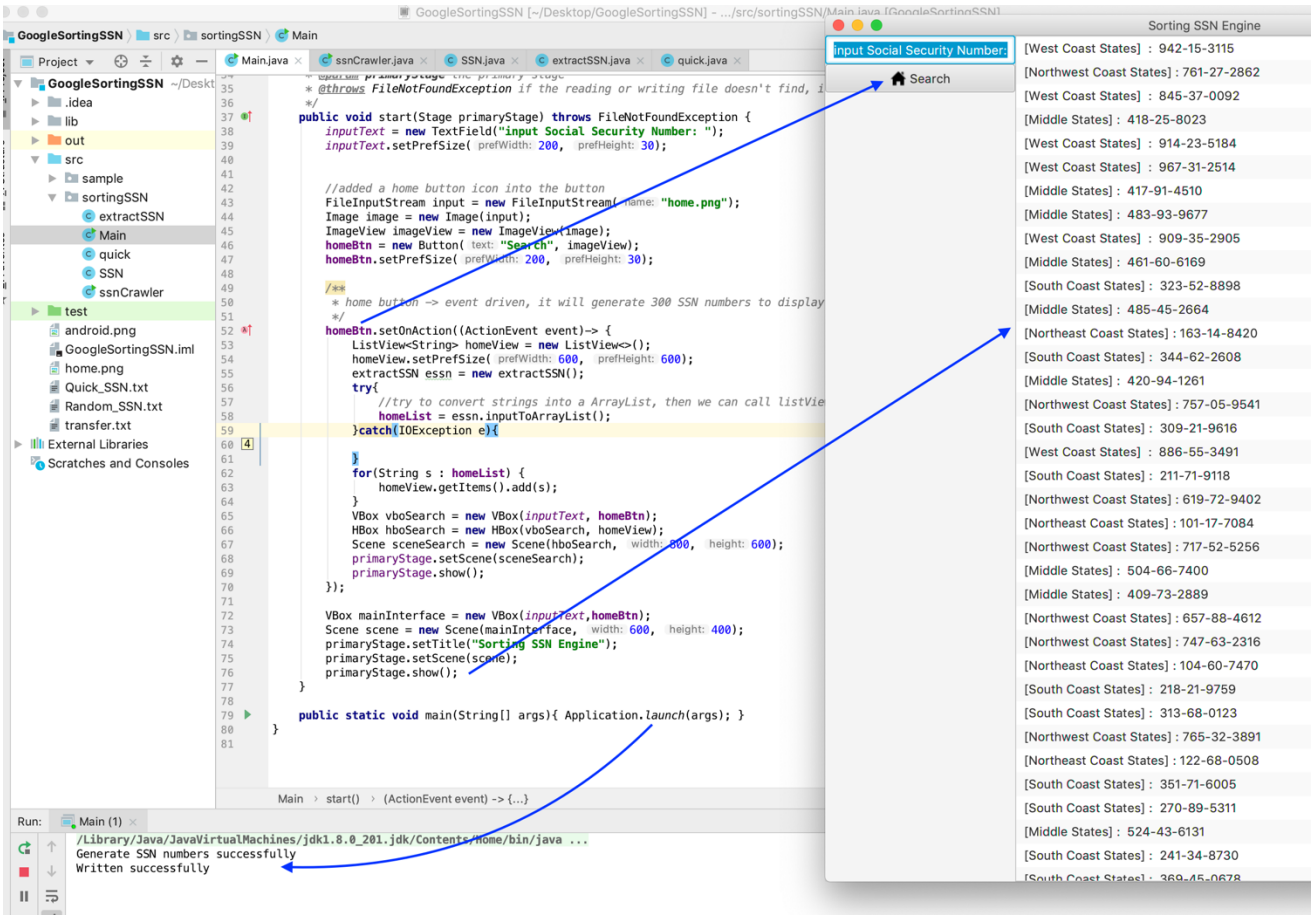




- Inside every method, there is **while()** clause, its purpose is to compare first 3 numbers:
-

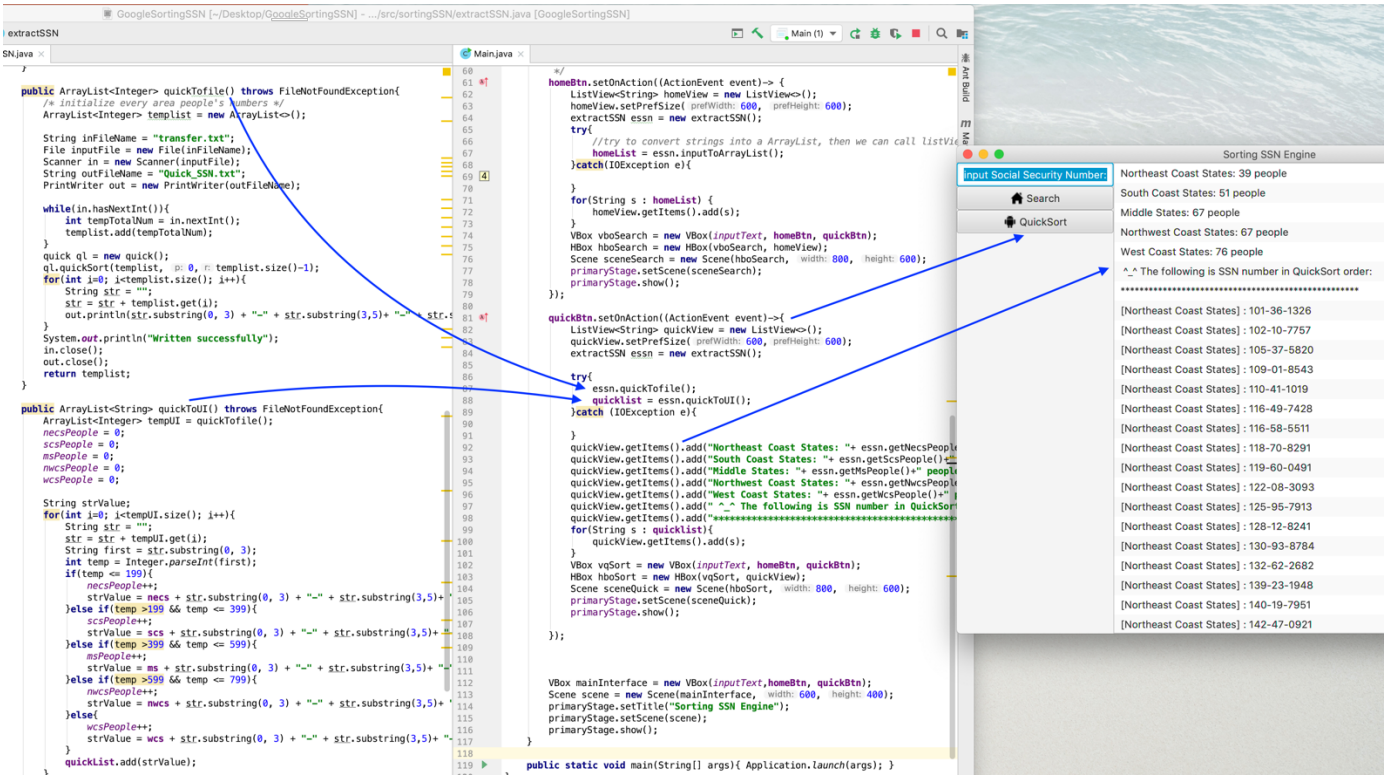
```
if(temp <= 199){ //store every ssn number with the area name
    strValue = necs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
str.substring(5);
}else if(temp >199 && temp <= 399){
    strValue = scs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
str.substring(5);
}else if(temp >399 && temp <= 599){
    strValue = ms + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
str.substring(5);
}else if(temp >599 && temp <= 799){
    strValue = nwcs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
str.substring(5);
}else{
    strValue = wcs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
str.substring(5);
}
//write every ssn number into the Random_SSN.txt file
out.println(str.substring(0, 3) + "-" + str.substring(3,5)+ "-" +
str.substring(5));
```

- We can distinguish the different area name:
  [Northeast Coast States]
  [South Coast States]
  [Middle States]
  [Northwest Coast States]
  [West Coast States]
- **out.println()** will write all data into the .txt file



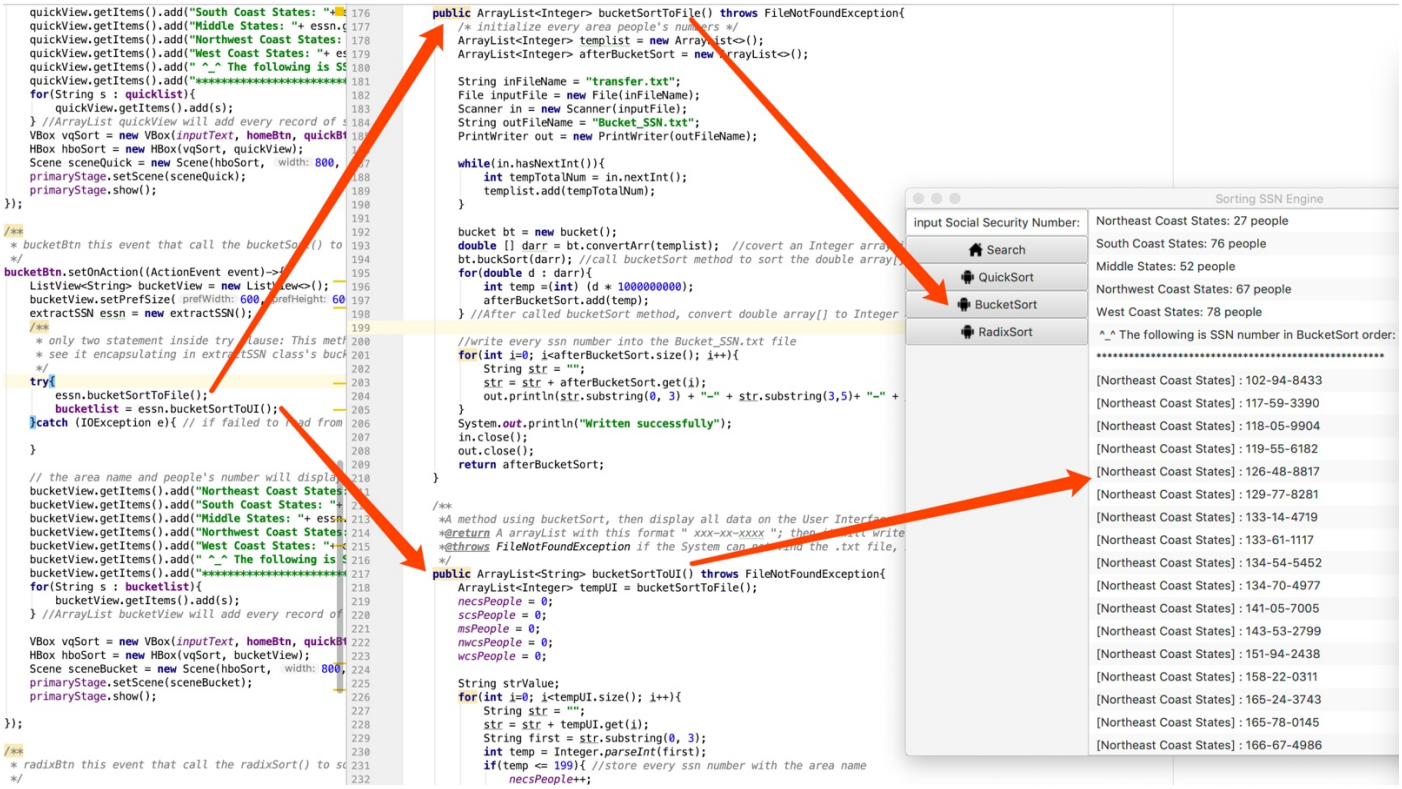
```
public ArrayList<Integer> quickSortToFile()
public ArrayList<String> quickSortToUI()
```

this two method's purpose are to write all data into .txt file and on the User Interface:



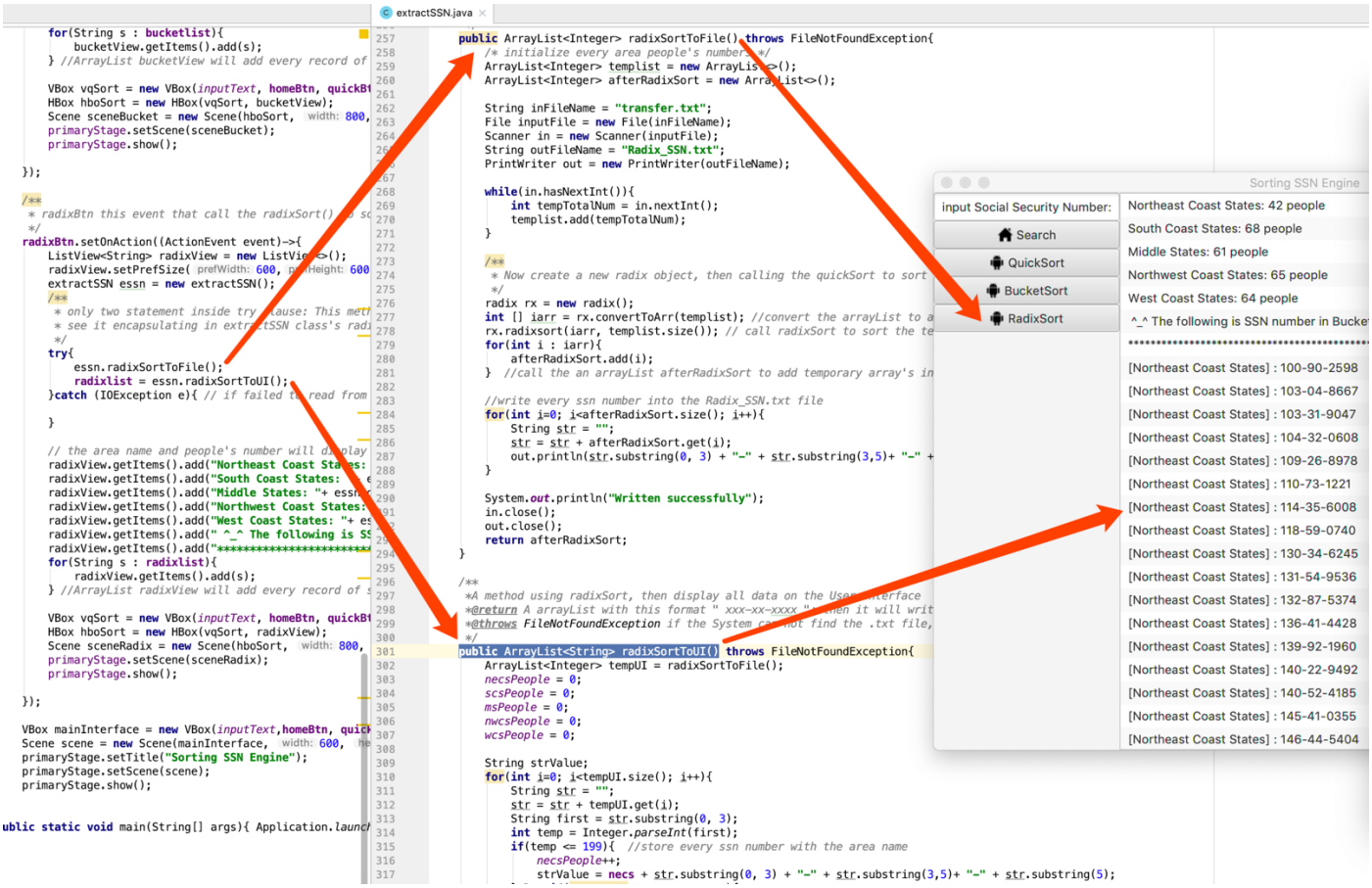
```
public ArrayList<Integer> bucketSortToFile()
public ArrayList<String> bucketSortToUI()
```

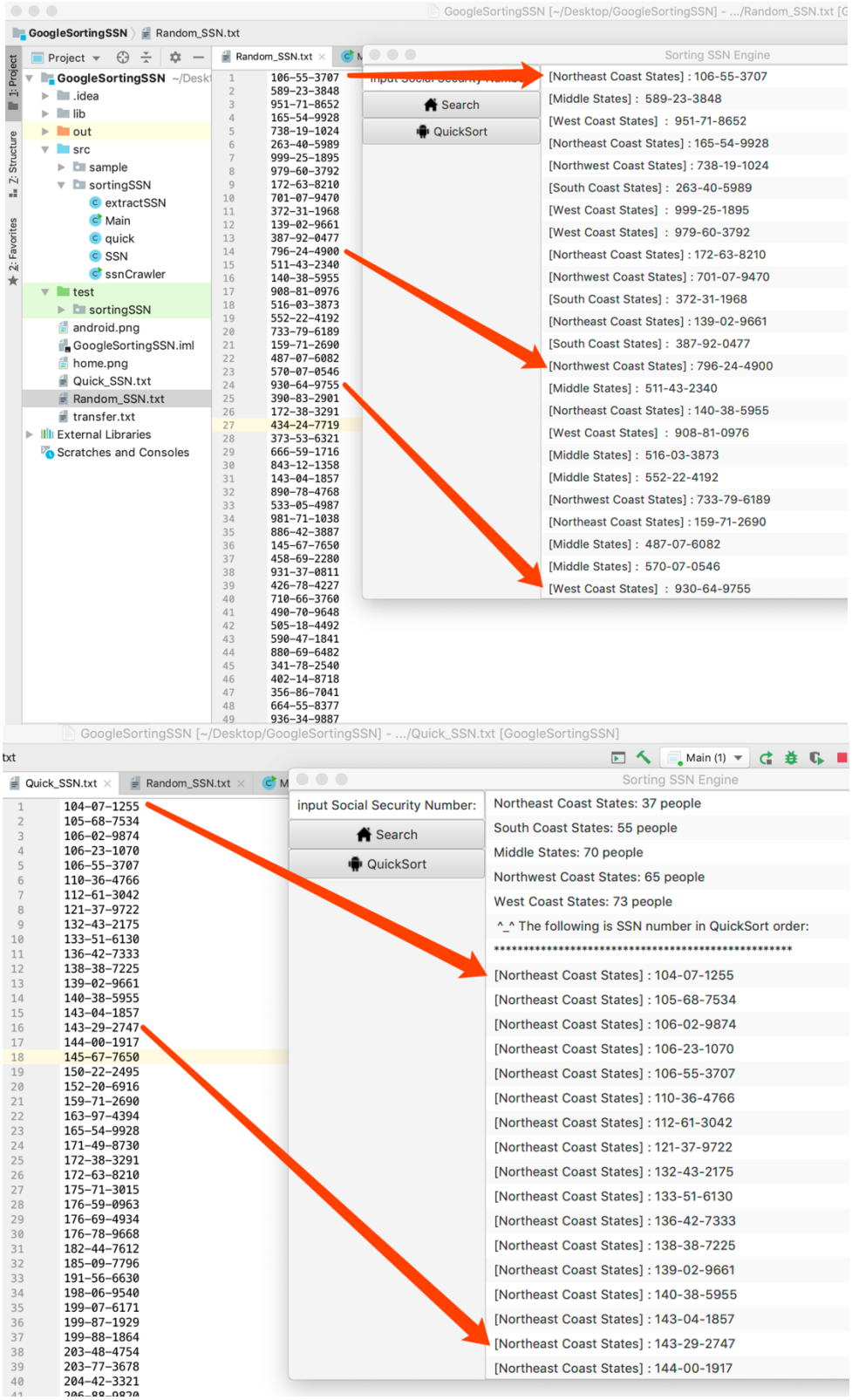
this two method's purpose are to write all data into .txt file and on the User Interface:



```
public ArrayList<Integer> radixSortToFile()
public ArrayList<String> radixSortToUI()
```

this two method's purpose are to write all data into .txt file and on the User Interface:




GoogleSortingSSN
Random_SSN.txt
Project
GoogleSortingSSN
.idea
lib
out
src
sample
sortingSSN
extractSSN
Main
quick
SSN
ssnCrawler
test
sortingSSN
android.png
GoogleSortingSSN.iml
home.png
Quick_SSN.txt
Random_SSN.txt
transfer.txt
External Libraries
Scratches and Consoles
106-55-3707
589-23-3848
951-71-8652
165-54-9928
738-19-1024
263-40-5989
999-25-1895
979-60-3792
172-63-8210
701-07-9470
372-31-1968
139-02-9661
387-92-0477
796-24-4900
511-43-2340
140-38-5955
908-81-0976
516-03-3873
552-22-4192
733-79-6189
159-71-2690
487-07-6082
570-07-0546
930-64-9755
390-83-2901
172-38-3291
434-24-7719
373-53-6321
666-59-1716
843-12-1358
143-04-1857
890-78-4768
533-05-4987
981-71-1038
886-42-3887
145-67-7650
458-69-2280
931-37-0811
426-78-4227
710-66-3760
490-70-9648
505-18-4492
590-47-1841
880-69-6482
341-78-2540
402-14-8718
356-86-7041
664-55-8377
936-34-9887
Sorting SSN Engine
Search
QuickSort
[Northeast Coast States] : 106-55-3707
[Middle States] : 589-23-3848
[West Coast States] : 951-71-8652
[Northeast Coast States] : 165-54-9928
[Northwest Coast States] : 738-19-1024
[South Coast States] : 263-40-5989
[West Coast States] : 999-25-1895
[West Coast States] : 979-60-3792
[Northeast Coast States] : 172-63-8210
[Northwest Coast States] : 701-07-9470
[South Coast States] : 372-31-1968
[Northeast Coast States] : 139-02-9661
[South Coast States] : 387-92-0477
[Northwest Coast States] : 796-24-4900
[Middle States] : 511-43-2340
[Northeast Coast States] : 140-38-5955
[West Coast States] : 908-81-0976
[Middle States] : 516-03-3873
[Middle States] : 552-22-4192
[Northwest Coast States] : 733-79-6189
[Northeast Coast States] : 159-71-2690
[Middle States] : 487-07-6082
[Middle States] : 570-07-0546
[West Coast States] : 930-64-9755
GoogleSortingSSN [~/Desktop/GoogleSortingSSN] - .../Quick_SSN.txt [GoogleSortingSSN]
Main (1)
Quick_SSN.txt
Random_SSN.txt
104-07-1255
105-68-7534
106-02-9874
106-23-1070
106-55-3707
110-36-4766
112-61-3042
121-37-9722
132-43-2175
133-51-6130
136-42-7333
138-38-7225
139-02-9661
140-38-5955
143-04-1857
143-29-2747
144-00-1917
145-67-7650
150-22-2495
152-20-6916
159-71-2690
163-97-4394
165-54-9928
171-49-8730
172-38-3291
172-63-8210
175-71-3015
176-59-0963
176-69-4934
176-78-9668
182-44-7612
185-09-7796
191-56-6630
198-06-9540
199-07-6171
199-87-1929
199-88-1864
203-48-4754
203-77-3678
204-42-3321
Sorting SSN Engine
input Social Security Number:
Search
QuickSort
Northeast Coast States: 37 people
South Coast States: 55 people
Middle States: 70 people
Northwest Coast States: 65 people
West Coast States: 73 people
^_^ The following is SSN number in QuickSort order:
****
[Northeast Coast States] : 104-07-1255
[Northeast Coast States] : 105-68-7534
[Northeast Coast States] : 106-02-9874
[Northeast Coast States] : 106-23-1070
[Northeast Coast States] : 106-55-3707
[Northeast Coast States] : 110-36-4766
[Northeast Coast States] : 112-61-3042
[Northeast Coast States] : 121-37-9722
[Northeast Coast States] : 132-43-2175
[Northeast Coast States] : 133-51-6130
[Northeast Coast States] : 136-42-7333
[Northeast Coast States] : 138-38-7225
[Northeast Coast States] : 139-02-9661
[Northeast Coast States] : 140-38-5955
[Northeast Coast States] : 143-04-1857
[Northeast Coast States] : 143-29-2747
[Northeast Coast States] : 144-00-1917

- Now I will explain how to implemnt **bucketSort()** method step by step:

```
/**
 * Convert a Integer arrayList to a double array[]
 * @param intarr the given arrayList with Integer elements
 * @return a double array[]
 */
public double[] convertArr(ArrayList<Integer> intarr){
    double [] darr = new double[intarr.size()];
    for(int i=0; i<intarr.size(); i++){
        darr[i] =  (intarr.get(i))/ 1000000000.0;
    }
    return darr;
}
```

1. Firstly, trying to **convert** the ArrayLsit<Integer> to the **double [] array**, this step's purpose is to satisfies 0 <= A[i] < 1
2. Create a **double[] array** to receive the converted data
3. Using original arrayList's size()
   - ⇨ For here, the original arrayList's size() is **300**
   - ⇨ So we set **n = size() = 300**
4. **Every int element** / `1000000000.0;` this step's purpose is to make **every new element** <= **1** and new element is a **double**
5. Inside **buckSort(double [] arr)** method, we call a

```
ArrayList<Double>[] bucketList = new ArrayList[n];
```

bucketList is an **array**, which every **array[i]** contains a **arrayList**

- ⇨ **Array[i].arrayList<Double>**

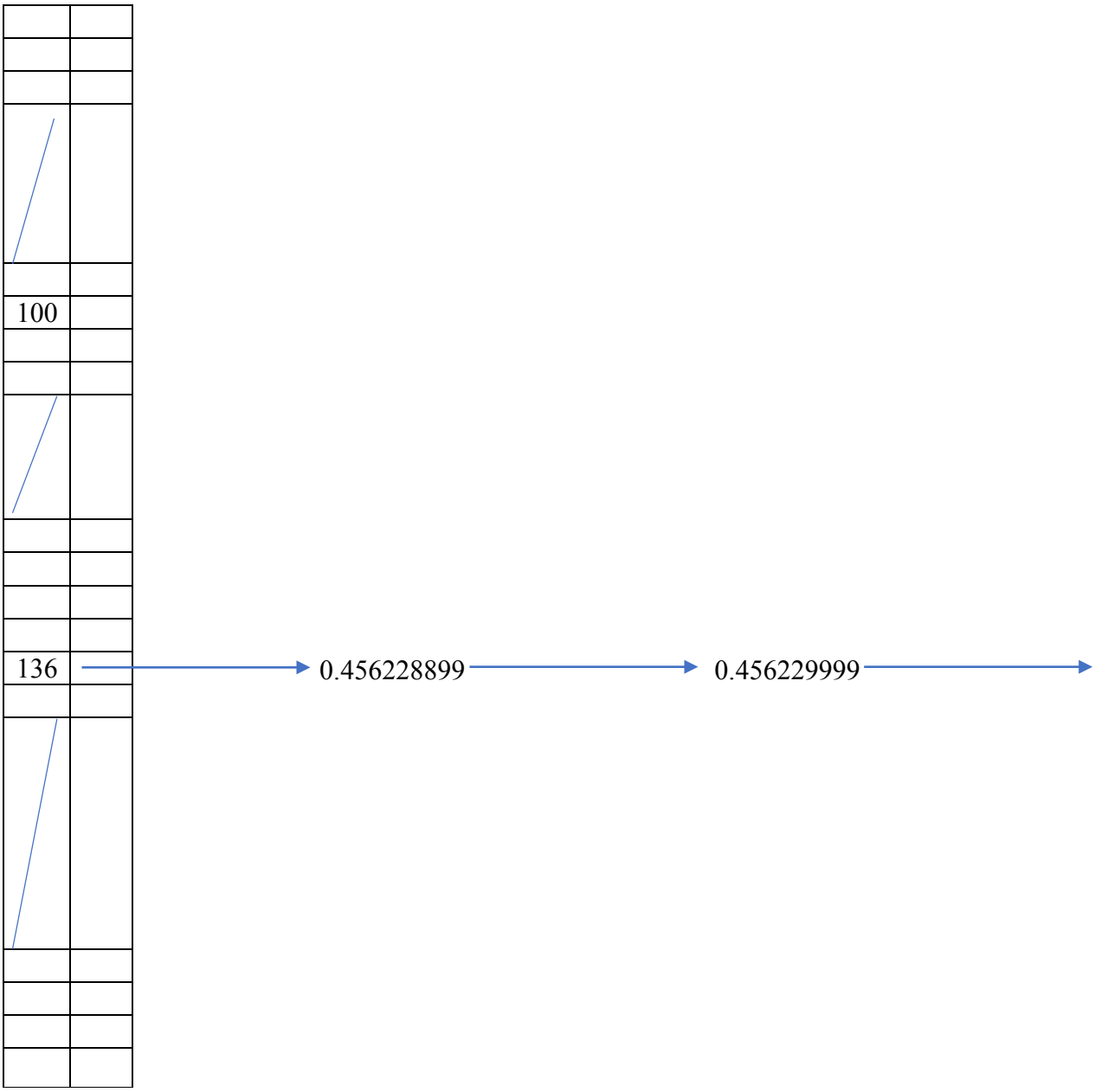
```
/**
 * insert every double element into the appropriate index of bucketList
 */
for(int i=0; i<n; i++){
    int bucketIndex = (int)(arr[i] * n);
    bucketList[bucketIndex].add(arr[i]);
}
```

- ⇨ Set every **bucketIndex = every double elment * 300**
- ⇨ For example: **0.456228899 * 300** = 136.8686697
- ⇨ Convert: **(int)** 136.8686697 => **136**
- ⇨ Finally, **bucketIndex = 136**
- ⇨ Calling the array[] **bucketList[136].add(double elment);**

```
/**
 * calling the insertSort to sort every arrayList
 */
for(int i=0; i<n; i++){
    insertSort(bucketList[i]);
}
```

6. loop a size = 300, everytime calling the **insertSort()** to help sort every bucket's double element:
   - ⇨ We have known this step is stable.
   - ⇨ The **insertSort()** will compare the every double element in this **bucketList[bucketIndex]**

7. the following is the graphically explaination:




```
/* concatenate all buckets */
int index = 0;
for(int i=0; i<n; i++){
    for(int j=0; j<bucketList[i].size(); j++){
        arr[index++] = bucketList[i].get(j);
    }
}
```

Finally, use **two for** loop to **concatenate** all buckets, this step is like 2D array operation.

- Done!

- Now I will explain how to implemnt **radixSort()** method step by step:

```
/**
 * @param arr the required arrayList to convert to a integer array
 * @return a new integer array
 */
public int[] convertToArr(ArrayList<Integer> arr){
    int [] iarr = new int[arr.size()];
    for(int i=0; i<arr.size(); i++){
        iarr[i] =  arr.get(i);
    }
    return iarr;
}
```

  ⇨ Convert **ArrayList<Integer>** to **int[ ]** , the reason why I convert the arrayList because arrayList is difficult to **manipulate.**

```
/**
 * Get the maximum from this array
 * @param arr the required array will be sorted
 * @param n the length of this array
 * @return a maximum
 */
public int getMax(int arr[], int n){
    int max = arr[0];
    for(int i=1; i<n; i++){
        if(arr[i] > max){
            max = arr[i];
        }
    }
    return max;
}
```

  ⇨ Find the max length, this step is to find the max places
  ⇨ For example 10's => 100's => 1000's => 10000's => … …

```
/**
 * The helper method to help radixSort to sort the SSN number
 * @param arr the required array
 * @param n the array's lenght
 * @param places the every digit place
 */
public void countSort(int arr[], int n, int places){
    int B[]  = new int[n];
    int count[] = new int[10];
    /* initialize every element to be 0 */
    Arrays.fill(count,0);

    /* count every digit from 0-9 that the number of times of appearance */
    for(int i=0; i<n; i++){
        count[ (arr[i]/places)%10]++;
    }

    //updated the array C[]
    for(int j=1; j<10; j++){
        count[j] =  count[j] + count[j-1];
    }
```

```
        //input the sorted number into the appropriate index
        for(int j=n-1; j>=0; j--){
            B[count[(arr[j]/places)%10]-1] = arr[j];
            count[ (arr[j]/places)%10 ]--;
        }

        //copy the new array to the original array
        for(int i=0; i<n; i++){
            arr[i] = B[i];
        }
    }
```

In the **countSort()** method, count every digit from 0-9 that the times of this number's apperance:

- Arr[i] / places => find the **nth** place
- %10 => find one of the 0-9 digit
- finally, update the times of this number appearing

- Put the sorted number into the appropriate index, then update the **count array**

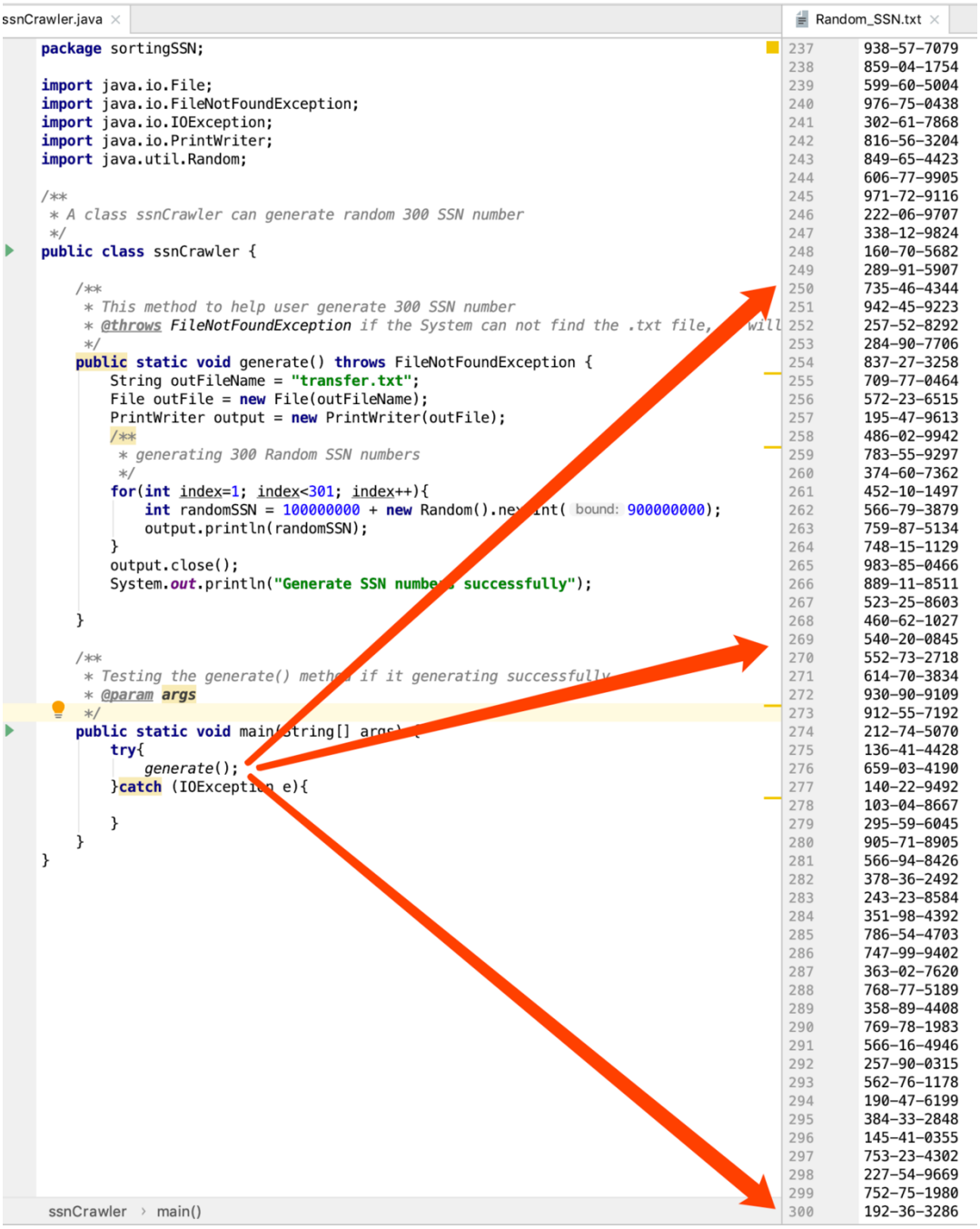
- **Note:** copy the new array to the original array.

- The following table will show the theory to implement **radixSort() :**

Index	Original	One’s place	1000’s place	100xxxxxx’s place
1	751345555	592679900	…	…
2	543456789	239671111		
3	689234545	961567711		
4	264245566	799990011		
5	837349988	393778811		
6	160783434	937882222		
7	239671111	177789732		
8	937882222	236567823		
9	592679900	160783434		
10	961567711	885993344		
11	799990011	412356784		
12	885993344	751345555		
13	393778811	689234545		
14	177789732	264245566		
15	236567823	747234567		
16	364117799	837349988		
17	412356784	543456789		
18	747234567	364117799		

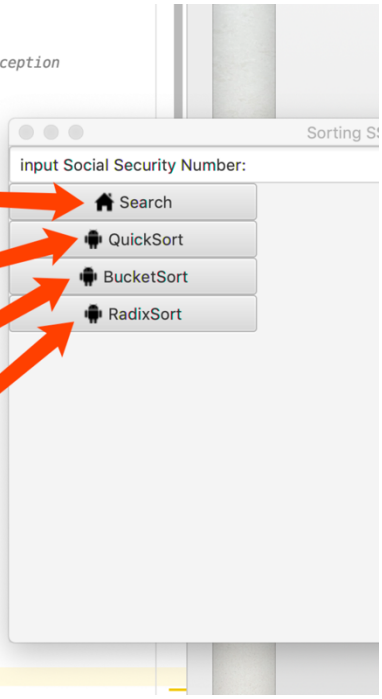
5. Self-testing

- Testing **ssnCrawler.generate()** :
  Generating 300 SSN numbers without sorting



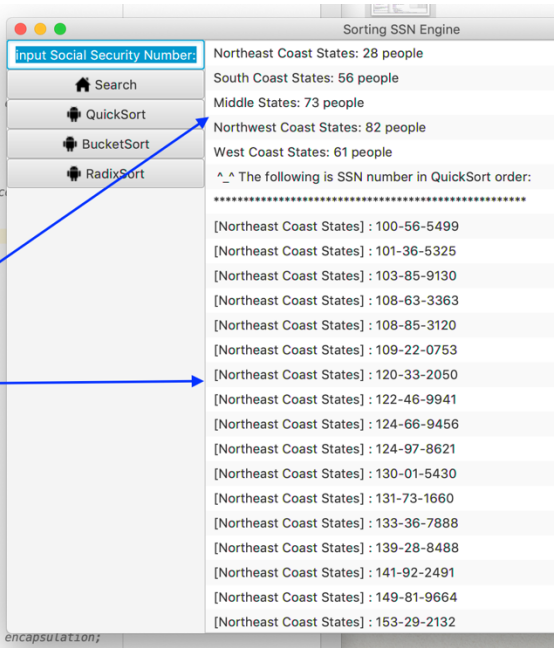
- Testing Main class
  1. testing four **button icon**:

```
/**
 * @param primaryStage the primary stage
 * @throws FileNotFoundException if the reading or writing file doesn't find, it will throw the exception
 */
public void start(Stage primaryStage) throws FileNotFoundException {
    inputText = new TextField("input Social Security Number: ");
    inputText.setPrefSize( prefWidth: 200,  prefHeight: 30);

    //added a home button icon into the button
    FileInputStream input = new FileInputStream( name: "home.png");
    Image image = new Image(input);
    ImageView imageView = new ImageView(image);
    homeBtn = new Button( text: "Search", imageView);
    homeBtn.setPrefSize( prefWidth: 200,  prefHeight: 30);

    //added a quickSort button icon into the button
    FileInputStream input1 = new FileInputStream( name: "android.png");
    Image image1 = new Image(input1);
    ImageView imageView1 = new ImageView(image1);
    quickBtn = new Button( text: "QuickSort", imageView1);
    quickBtn.setPrefSize( prefWidth: 200,  prefHeight: 30);

    //added a bucketSort button icon into the button
    FileInputStream input2 = new FileInputStream( name: "android.png");
    Image image2 = new Image(input2);
    ImageView imageView2 = new ImageView(image2);
    bucketBtn = new Button( text: "BucketSort", imageView2);
    bucketBtn.setPrefSize( prefWidth: 200,  prefHeight: 30);

    //added a radixSort button icon into the button
    FileInputStream input3 = new FileInputStream( name: "android.png");
    Image image3 = new Image(input3);
    ImageView imageView3 = new ImageView(image3);
    radixBtn = new Button( text: "RadixSort", imageView3);
    radixBtn.setPrefSize( prefWidth: 200,  prefHeight: 30);
```

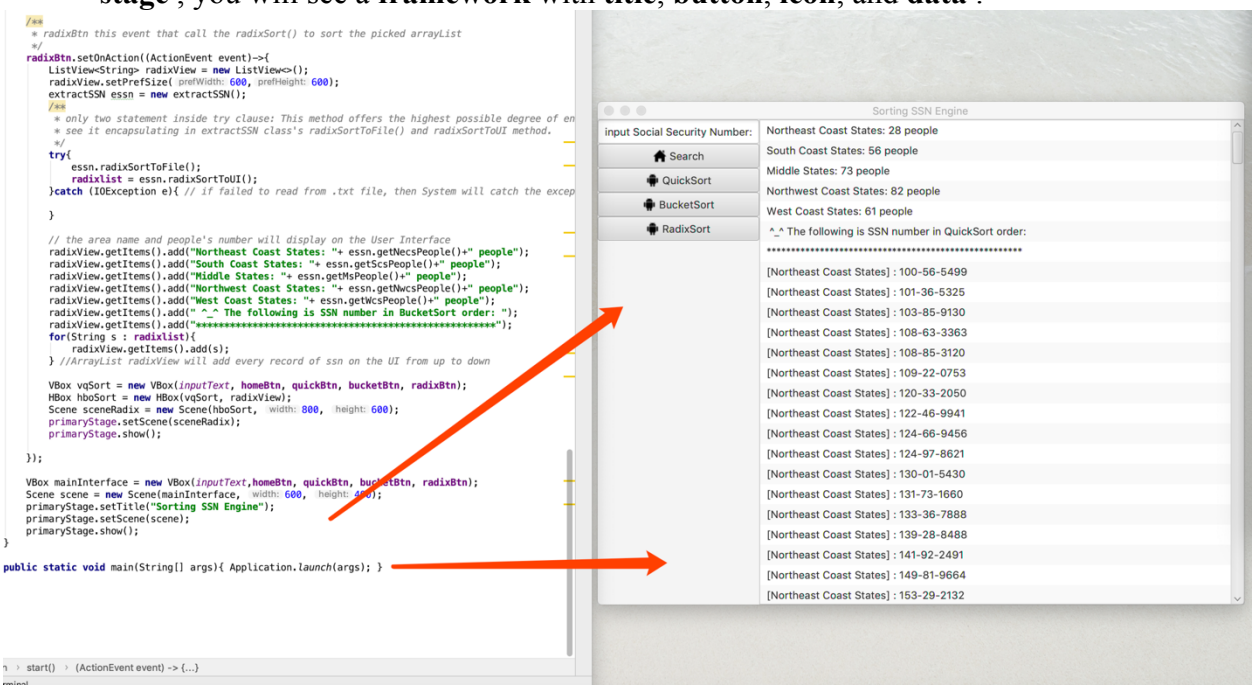


  2. testing **ListView()** , which will drive the button event to display on the **User Interface**:

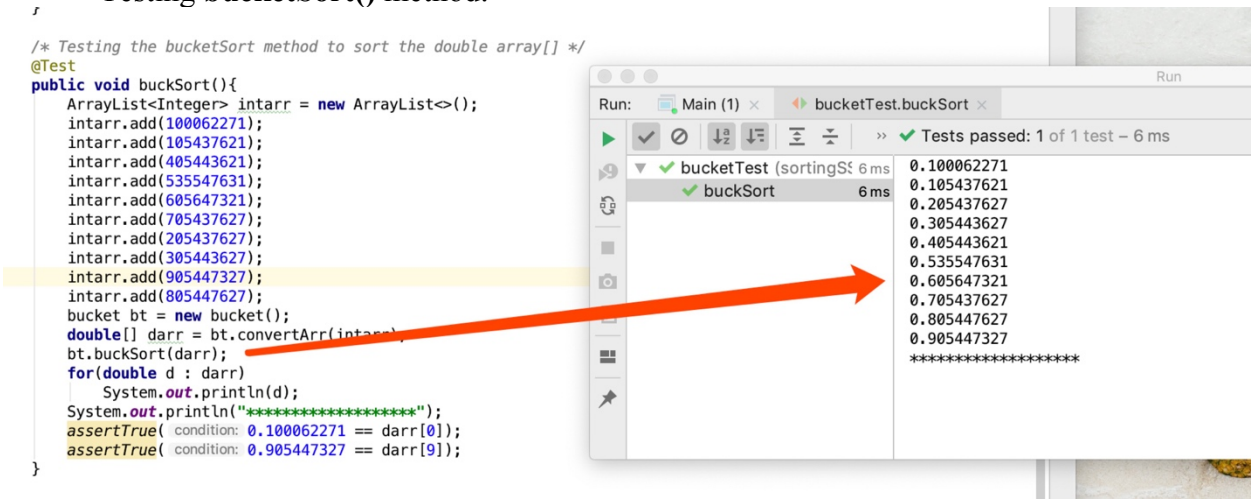
```
 */
quickBtn.setOnAction((ActionEvent event)->{
    ListView<String> quickView = new ListView<>();
    quickView.setPrefSize( prefWidth: 600, prefHeight: 600);
    extractSSN essn = new extractSSN();
    /**
     * only two statement inside try clause: This method offers the highest possible degree of
     * see it encapsulating in extractSSN class's quickSortToFile() and quickSortToUI method.
     */
    try{
        essn.quickSortToFile();
        quicklist = essn.quickSortToUI();
    }catch (IOException e){ // if failed to read from .txt file, then System will catch the exc

    }

    // the area name and people's number will display on the User Interface
    quickView.getItems().add("Northeast Coast States: "+ essn.getNecsPeople()+" people");
    quickView.getItems().add("South Coast States: "+ essn.getScsPeople()+" people");
    quickView.getItems().add("Middle States: "+ essn.getMsPeople()+" people");
    quickView.getItems().add("Northwest Coast States: "+ essn.getNwcsPeople()+" people");
    quickView.getItems().add("West Coast States: "+ essn.getWcsPeople()+" people");
    quickView.getItems().add(" ^_^ The following is SSN number in QuickSort order: ");
    quickView.getItems().add("**************************************************");
    for(String s : quicklist){
        quickView.getItems().add(s);
    } //ArrayList quickView will add every record of ssn on the UI from up to down
    VBox vqSort = new VBox(inputText, homeBtn, quickBtn, bucketBtn, radixBtn);
    HBox hboSort = new HBox(vqSort, quickView);
    Scene sceneQuick = new Scene(hboSort,  width: 800,  height: 600);
    primaryStage.setScene(sceneQuick);
    primaryStage.show();
});

/**
 * bucketBtn this event that call the bucketSort() to sort the picked arrayList
 */
bucketBtn.setOnAction((ActionEvent event)->{
    ListView<String> bucketView = new ListView<>();
    bucketView.setPrefSize( prefWidth: 600, prefHeight: 600);
    extractSSN essn = new extractSSN();
    /**
     * only two statement inside try clause: This method offers the highest possible degree of encapsulation;
     * see it encapsulating in extractSSN class's bucketSortToFile() and bucketToUI method.
     */
    try{
```



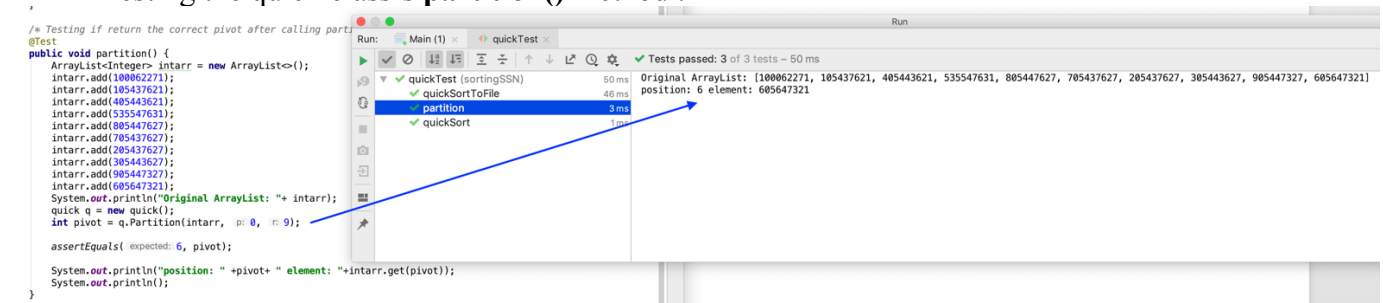
- Testing **primaryStage.setTitle() primaryStage.setScene()primaryStage.show()** Testing **Application.launch(args)** , the previous 4 method will construct a **primary stage** , you will see a **framework** with **title**, **button**, **icon**, and **data** :



- Testing **bucketSort()** method:

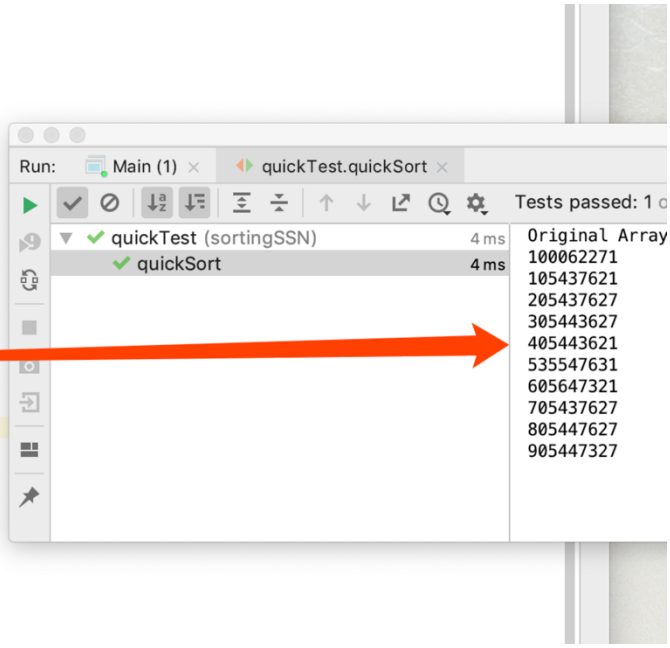


- Testing the quick class's **partition()** method :



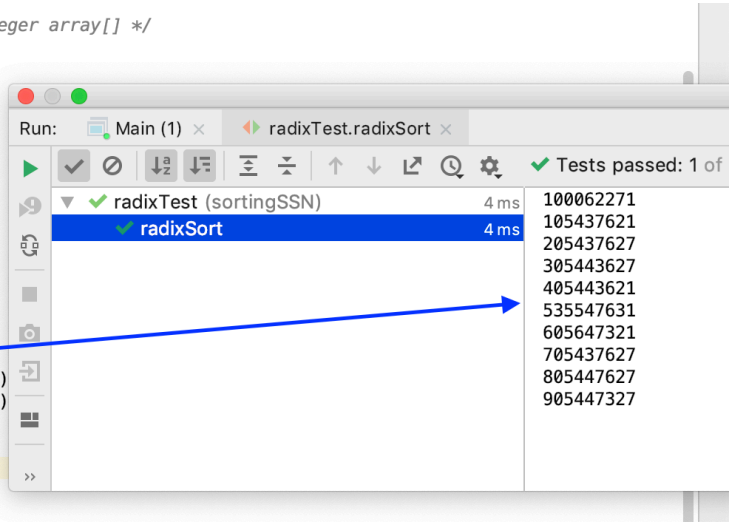
- Testing **quickSort()** method:

```
public void quickSort() {
    ArrayList<Integer> intarr = new ArrayList<>();
    intarr.add(100062271);
    intarr.add(105437621);
    intarr.add(405443621);
    intarr.add(535547631);
    intarr.add(605647321);
    intarr.add(705437627);
    intarr.add(205437627);
    intarr.add(305443627);
    intarr.add(905447327);
    intarr.add(805447627);
    System.out.println("Original ArrayList: "+ intarr);
    ArrayList<Integer> temp = new ArrayList<>(intarr);
    Collections.sort(temp);
    quick q = new quick();
    q.quickSort(intarr,  p: 0,  r: 9);
    for(int i=0; i<intarr.size(); i++){
        System.out.println(intarr.get(i));
    }
    int min = intarr.get(0);
    int max = intarr.get(intarr.size()-1);

    assertEquals( expected: 10, intarr.size());
    assertEquals( expected: 100062271, min);
    assertEquals( expected: 905447327, max);
    assertEquals(temp, intarr);
    System.out.println();
}
```



- Testing **radixSort()** method:

```
/* Testing radixSort method if sorting the integer array[] */
@Test
public void radixSort() {
    int [] intarr = new int [10];
    intarr[0] = 100062271;
    intarr[1] = 105437621;
    intarr[2] = 405443621;
    intarr[3] = 535547631;
    intarr[4] = 605647321;
    intarr[5] = 705437627;
    intarr[6] = 205437627;
    intarr[7] = 305443627;
    intarr[8] = 905447327;
    intarr[9] = 805447627;
    radix r = new radix();
    r.radixsort(intarr,  n: 10);
    assertTrue( condition: 100062271 == intarr[0]);
    assertTrue( condition: 905447327 == intarr[9]);
    for(int i=0; i<intarr.length; i++){
        System.out.println(intarr[i]);
    }
}
```



To summary: the **remaining Running time** and some **detail** , I will illustrate in **Part 8**:
Running Time Big-**O** (Junit -Test)

6. **Problems encountered during implementation**

- Since I design my program can do two things, I found that it is difficult to display all sorted data on the **UI** and **.txt file** at the **same time**.
- Originally, for example, I **combined AfterQuickSortToFile()** and **AfterQuickSortToUserInterface()** this two methods into one method:
  1. problem_1: if I did like this, it will increase its **running time** because I should use some **for loop**, and inside **for()** that I need to compare **300 SSN number**, whatever what functioning I want to implement, **it always occur at the same time**.
  2. problem_2: it is very hard to mainpulate these data structures, since some **bucketSort radixSort** which I **convert** arrayList to **Array;** then after did this step, I convert the **Array** to **arrayList**.

  This is a most troublesome case: you should care the data structure that you using and you should **focus on** which functioning that you want to implement: **on UI** and into **.txt file**.
- Look this following piece of codes:

```
 */
public int[] convertToArr(ArrayList<Integer> arr){
    int [] iarr = new int[arr.size()];
    for(int i=0; i<arr.size(); i++){
        iarr[i] =  arr.get(i);
    }
    return iarr;
}
/**
 * Get the maximum from this array
 * @param arr the required array will be sorted
 * @param n the length of this array
 * @return a maximum
 */
public int getMax(int arr[], int n){
    int max = arr[0];
    for(int i=1; i<n; i++){
        if(arr[i] > max){
            max = arr[i];
        }
    }
    return max;
}

/**
 * The helper method to help radixSort to sort the SSN number
 * @param arr the required array
 * @param n the array's lenght
 * @param places the every digit place
 */
public void countSort(int arr[], int n, int places){
    int B[]  = new int[n];
    int count[] = new int[10];
    /* initialize every element to be 0 */
    Arrays.fill(count, val: 0);

    /* count every digit from 0-9 that the number of times of appearance */
    for(int i=0; i<n; i++){
        count[ (arr[i]/places)%10]++;
    }

    //updated the array C[]
    for(int j=1; j<10; j++){
        count[j] =  count[j] + count[j-1];
    }

    //input the sorted number into the appropriate index
    for(int j=n-1; j>=0; j--){
        B[count[(arr[j]/places)%10]-1] = arr[j];
        count[ (arr[j]/places)%10 ]--;
    }

    //copy the new array to the original array
    for(int i=0; i<n; i++){
        arr[i] = B[i];
    }
}
```

- And this following codes also illustrate the problem that I will encounter:

```
quickBtn.setOnAction((ActionEvent event)->{
       ListView<String> quickView = new ListView<>();
       quickView.setPrefSize(600,600);
       extractSSN essn = new extractSSN();

       try{
           essn.quickSortToFile();
           quicklist = essn.quickSortToUI();
       }catch (IOException e){ // if failed to read from .txt file, then System will
       catch the exception

       }
```

```
quickBtn.setOnAction((ActionEvent event)->{
    ListView<String> quickView = new ListView<>();
    quickView.setPrefSize(600,600);
    extractSSN essn = new extractSSN();

       try{
           essn.bucketSortToFile();
           bucketlist = essn.bucketSortToUI();
       }catch (IOException e){ // if failed to read from .txt file, then System will
       catch the exception

       }
```

```
bucketBtn.setOnAction((ActionEvent event)->{
    ListView<String> bucketView = new ListView<>();
    bucketView.setPrefSize(600,600);
    extractSSN essn = new extractSSN();

       try{
           essn.radixSortToFile();
           radixlist = essn.radixSortToUI();
       }catch (IOException e){ // if failed to read from .txt file, then System will
       catch the exception

       }
```

- Final **strategy**:
  Encapsulation——separating one method into **two** methods, then encapsulating all these methods into the **extractSSN** class.

  For example:
  Only two statemente(sentences) inside Main class's **try()** clause: this method offers the highest possible degree of encapsulation; see it encapsulating in **extractSSN** class's **bucketSortToFile()** and **bucketSortToUI** methods.
  Look the following piece of codes:

```
     */
    public ArrayList<Integer> bucketSortToFile() throws FileNotFoundException{
        /* initialize every area people's numbers */
        ArrayList<Integer> templist = new ArrayList<>();
        ArrayList<Integer> afterBucketSort = new ArrayList<>();

        String inFileName = "transfer.txt";
        File inputFile = new File(inFileName);
        Scanner in = new Scanner(inputFile);
        String outFileName = "Bucket_SSN.txt";
        PrintWriter out = new PrintWriter(outFileName);

        while(in.hasNextInt()){
            int tempTotalNum = in.nextInt();
            templist.add(tempTotalNum);
        }

        bucket bt = new bucket();
        double [] darr = bt.convertArr(templist);  //covert an Integer arrayList to a double array[]
        bt.buckSort(darr); //call bucketSort method to sort the double array[]
        for(double d : darr){
            int temp =(int) (d * 1000000000);
            afterBucketSort.add(temp);
        } //After called bucketSort method, convert double array[] to Integer arrayList

        //write every ssn number into the Bucket_SSN.txt file
        for(int i=0; i<afterBucketSort.size(); i++){
            String str = "";
            str = str + afterBucketSort.get(i);
            out.println(str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5));
        }
        System.out.println("Written successfully");
        in.close();
        out.close();
        return afterBucketSort;
    }
*/
public ArrayList<String> bucketSortToUI() throws FileNotFoundException{
    ArrayList<Integer> tempUI = bucketSortToFile();
    necsPeople = 0;
    scsPeople = 0;
    msPeople = 0;
    nwcsPeople = 0;
    wcsPeople = 0;

    String strValue;
    for(int i=0; i<tempUI.size(); i++){
        String str = "";
        str = str + tempUI.get(i);
        String first = str.substring(0, 3);
        int temp = Integer.parseInt(first);
        if(temp <= 199){ //store every ssn number with the area name
            necsPeople++;
            strValue = necs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5);
        }else if(temp >199 && temp <= 399){
            scsPeople++;
            strValue = scs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5);
        }else if(temp >399 && temp <= 599){
            msPeople++;
            strValue = ms + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5);
        }else if(temp >599 && temp <= 799){
            nwcsPeople++;
            strValue = nwcs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5);
        }else{
            wcsPeople++;
            strValue = wcs + str.substring(0, 3) + "-" + str.substring(3,5)+ "-" + str.substring(5);
        }
        bucketList.add(strValue);   //call the arrayList bucketList to add every ssn number with the area name
    }
    return bucketList;
}
```

- Change Data Type:

  In **radix** class, I must convert the data type:

```java
public class radix {

    /**
     * @param arr the required arrayList to convert to a integer array
     * @return a new integer array
     */
    public int[] convertToArr(ArrayList<Integer> arr){
        int [] iarr = new int[arr.size()];
        for(int i=0; i<arr.size(); i++){
            iarr[i] =  arr.get(i);
        }
        return iarr;
    }
```

  In **bucket** class, I must covert **ArrayList<Integer>** to **double array[]** :

```java
public class bucket{
    /**
     * Convert a Integer arrayList to a double array[]
     * @param intarr the given arrayList with Integer elements
     * @return a double array[]
     */
    public double[] convertArr(ArrayList<Integer> intarr){
        double [] darr = new double[intarr.size()];
        for(int i=0; i<intarr.size(); i++){
            darr[i] =  (intarr.get(i))/ 1000000000.0;
        }
        return darr;
    }
```

To summary: there is still a way that I learn from textbook and the previous CS course, can address there problems.

7. Leasson Learned

- Import **Javafx** library
- Improt **Java jdk 8**
- Import **Junit Test**

- Learned to debug some problems that I encountered
- Learned to encapsulate all methods into a class **extractSSN**

- Learned to convert data structure : convert **ArrayList to Array**
  and convert **Array to ArrayList**

- For **buketSort()**
  1. convert **ArrayList<Integer>** to **double[ ]**
  2. calling the **insertSort()** to help implement **bucketSort()**
  3. learned to explain and illustrate to show my design a bucket for **bucketSort()**

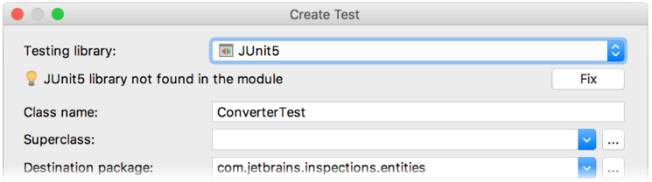
- For **radixSort()**
  1. convert **ArrayList<Integer>** to **int[ ]**
  2. calling the **countSort()** to help implement **raidxSort()**
  3. learned to explain and illustrate to define a digit for my **radixSort()**

8. Running Time Big-**O** (Junit -Test)

- **Install** :

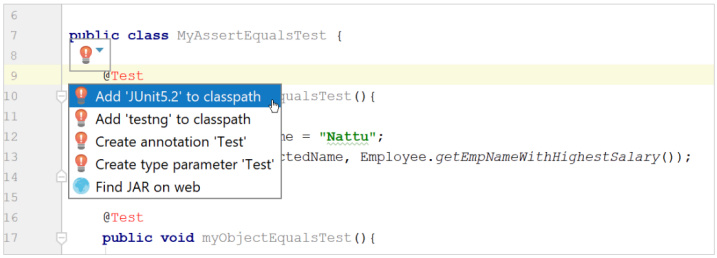
Add a test library to the classpath when creating a test for a class

1. In the editor, place the cursor within the line containing the class declaration.
2. Press ⌥↵ to view the available intention actions.
3. Select **Create Test**.
4. In the **Create Test** dialog, to the right of the text informing you that the corresponding library is not found, click **Fix**.



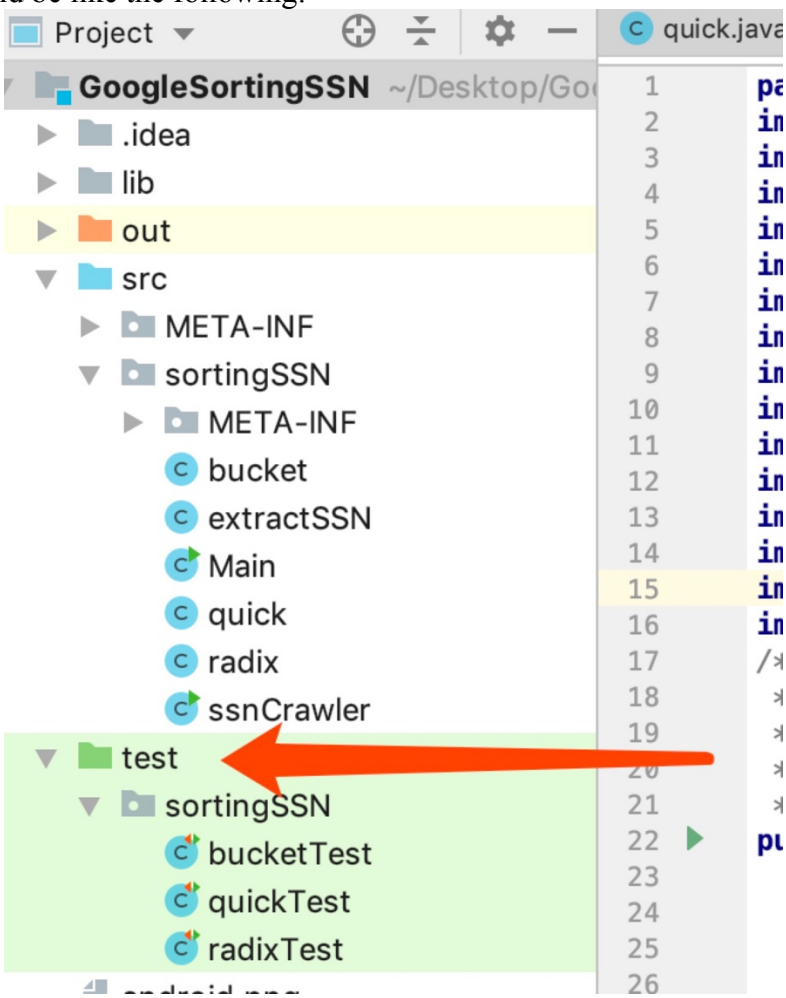
Add a test library to the classpath when writing the code for a test

1. In the source code of a test class, place the cursor within an unresolved reference to **`TestCase`** or annotation.
2. Press ⌥↵ to view the available intention actions.
3. Select **Add <library> to classpath**.



- All **Junit Test** .java files should be inside the **test** folder
  Note: the **folder name** depends on your choice

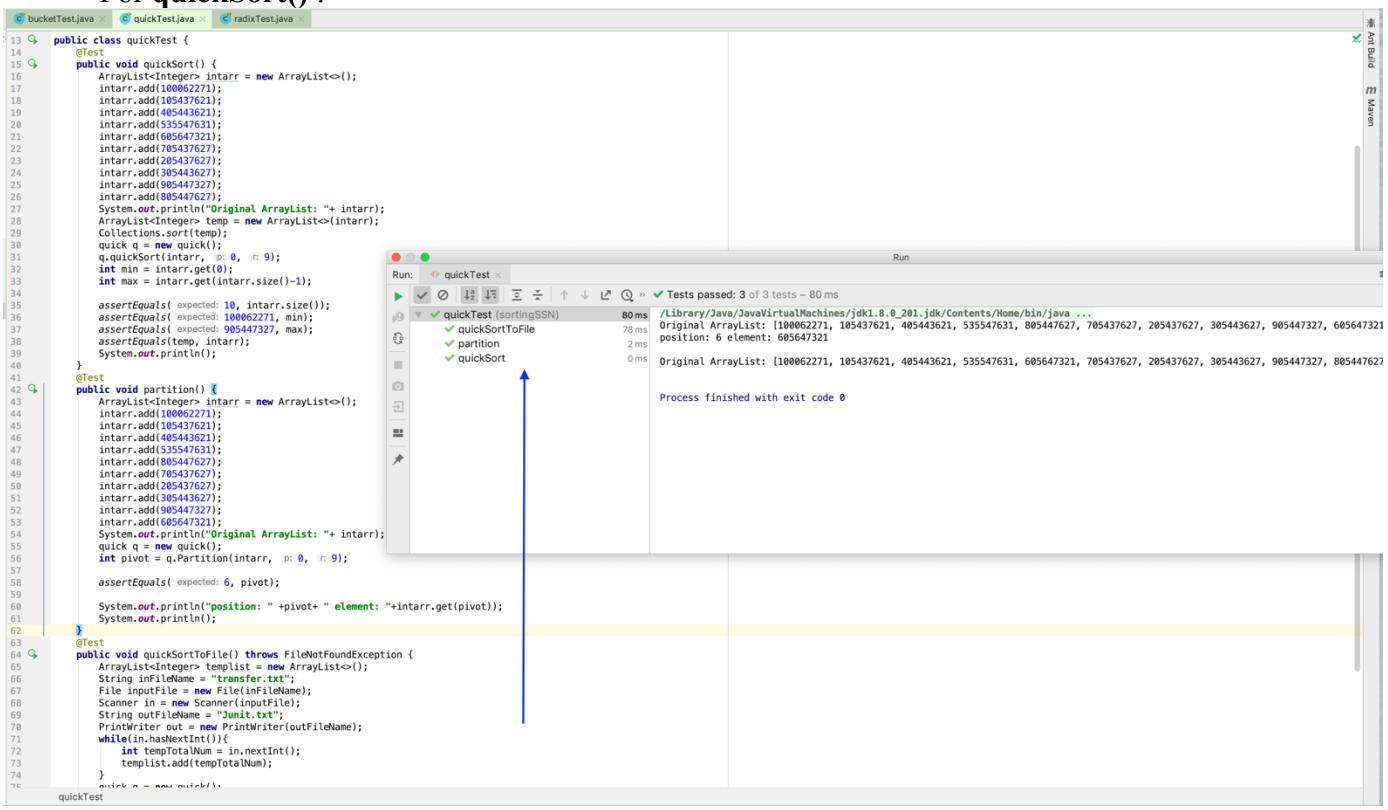
  The path should be like the following:



- If you are installing like the above steps, that means you have installed successfully.

- The following I take it from 10 times's running, I extract the middle data from 10 times
- I picked the same 10 **SSN numbers**, then I will guarantee fairness.
- I picked the **300** SSN numbers that the **ssnCrawler** class generated, to write into the .txt file
- Including **quickSort bucketSort radixSort**

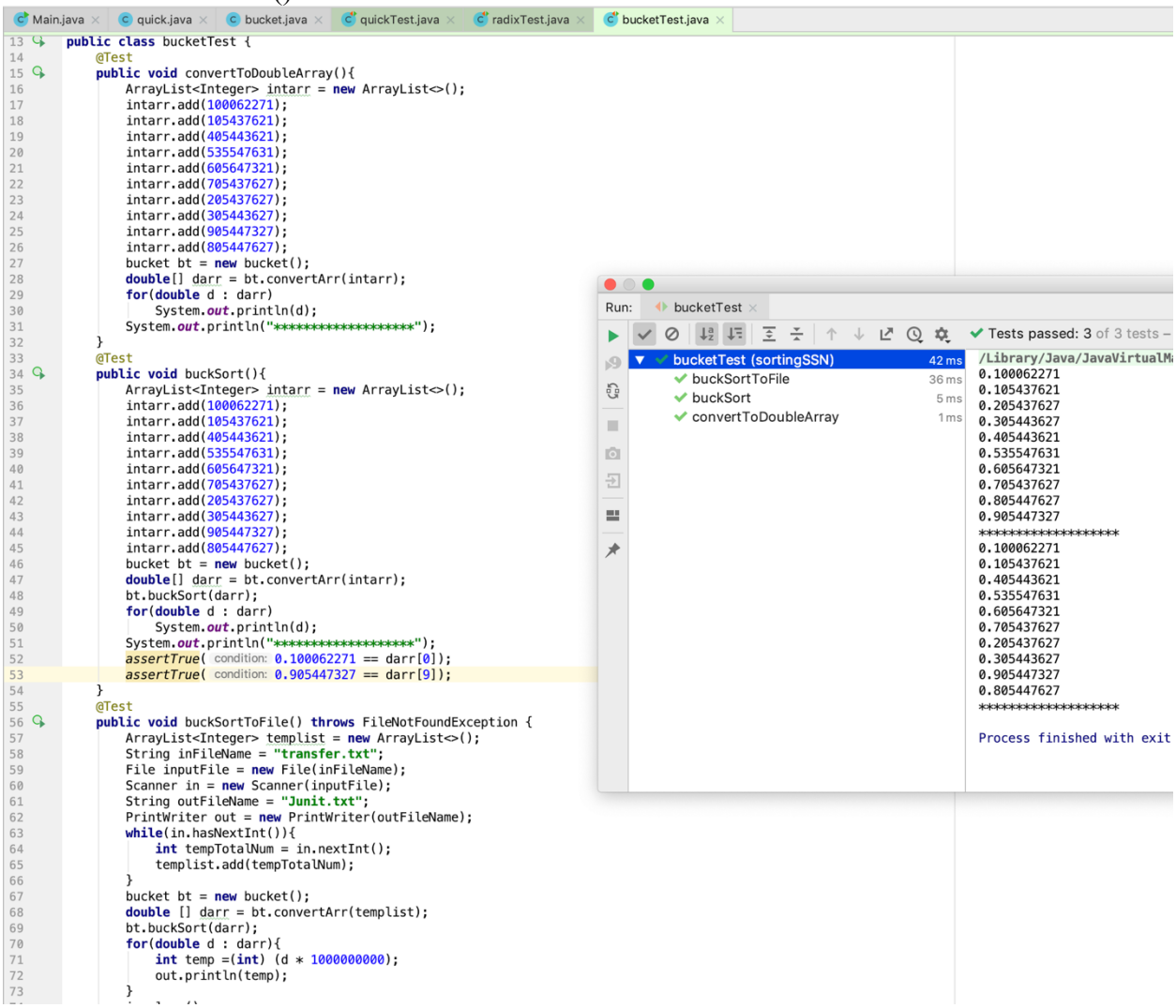
- For **quickSort()** :



Hence for here, QuickSort's running time:

- All quickTest : **80 ms**
- Partition: **2 ms**
- Since within Partition() , it will compare the data
  - ⇨ QuickSort's average running time is O(nlgn)
  - ⇨ The wrost case running time is $O(n^2)$

- For bucketSort()



```java
public class bucketTest {
    @Test
    public void convertToDoubleArray(){
        ArrayList<Integer> intarr = new ArrayList<>();
        intarr.add(100062271);
        intarr.add(105437621);
        intarr.add(405443621);
        intarr.add(535547631);
        intarr.add(605647321);
        intarr.add(705437627);
        intarr.add(205437627);
        intarr.add(305443627);
        intarr.add(905447327);
        intarr.add(805447627);
        bucket bt = new bucket();
        double[] darr = bt.convertArr(intarr);
        for(double d : darr)
            System.out.println(d);
        System.out.println("********************");
    }
    @Test
    public void buckSort(){
        ArrayList<Integer> intarr = new ArrayList<>();
        intarr.add(100062271);
        intarr.add(105437621);
        intarr.add(405443621);
        intarr.add(535547631);
        intarr.add(605647321);
        intarr.add(705437627);
        intarr.add(205437627);
        intarr.add(305443627);
        intarr.add(905447327);
        intarr.add(805447627);
        bucket bt = new bucket();
        double[] darr = bt.convertArr(intarr);
        bt.buckSort(darr);
        for(double d : darr)
            System.out.println(d);
        System.out.println("********************");
        assertTrue( condition: 0.100062271 == darr[0]);
        assertTrue( condition: 0.905447327 == darr[9]);
    }
    @Test
    public void buckSortToFile() throws FileNotFoundException {
        ArrayList<Integer> templist = new ArrayList<>();
        String inFileName = "transfer.txt";
        File inputFile = new File(inFileName);
        Scanner in = new Scanner(inputFile);
        String outFileName = "Junit.txt";
        PrintWriter out = new PrintWriter(outFileName);
        while(in.hasNextInt()){
            int tempTotalNum = in.nextInt();
            templist.add(tempTotalNum);
        }
        bucket bt = new bucket();
        double [] darr = bt.convertArr(templist);
        bt.buckSort(darr);
        for(double d : darr){
            int temp =(int) (d * 1000000000);
            out.println(temp);
        }
```

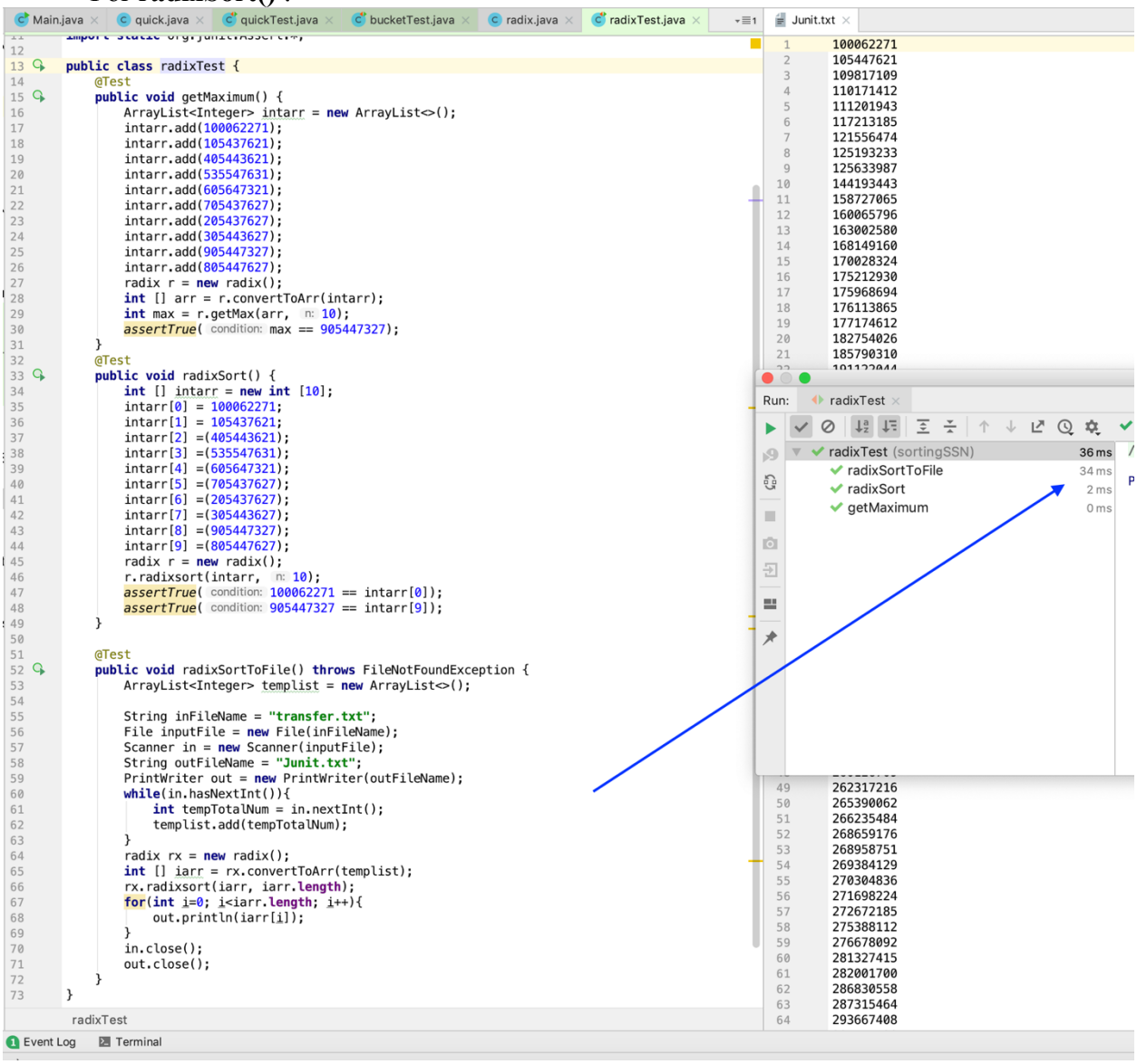
```
Run: bucketTest
Tests passed: 3 of 3 tests –
bucketTest (sortingSSN) 42 ms
  buckSortToFile 36 ms
  buckSort 5 ms
  convertToDoubleArray 1 ms
/Library/Java/JavaVirtualM
0.100062271
0.105437621
0.205437627
0.305443627
0.405443621
0.535547631
0.605647321
0.705437627
0.805447627
0.905447327
********************
0.100062271
0.105437621
0.405443621
0.535547631
0.605647321
0.705437627
0.205437627
0.305443627
0.905447327
0.805447627
********************

Process finished with exit
```

Hence for here, BucketSort's running time:

- All bucketTest : **42 ms**
- buckSort: **5 ms**

⇨ bucketSort's worst case running time is O(n)

- For **radixSort()** :



Hence , RadixSort's running time:

- All quickTest : **36 ms**
- RadixSort: **2 ms**
  - ⇨ RadixSort's total running time is Θ(d(n+k))
  - ⇨ When pass over n d-digit numbers that take time Θ(n+k)
  - ⇨ Total time = O(dn + dk)
  - ⇨ Since d is constant and k= O(n), takes O(n) time
  - ⇨ Summary: Fast ! Stable ! Simple ! (Doesn't sort in place)

- Running time summary:
  - ⇨ Even though, we only pick random 300 SSN numbers, which are not representative.
  - ⇨ However, to be academic research, I still analyze their comparison.
  - ⇨ This is the course's goal——I learn from CS 146 Data Structure Algorithms

- Now I can summary the comparison between **QuickSort BucketSort RadixSort**:
  - ⇨ Since the **quickSort**'s worst case running time is $O(n^2)$
  - ⇨ **bucketSort**'s worst case running time is O(n)
  - ⇨ **radixSort**'s worst case running time is O(n)
  - ⇨ Thus, for my program, quickSort test is **80 ms**
  - ⇨ **bucketSort** and **radixSort** is faster than **quickSort**

- **Programming Assignment 2: Social Security Number**
  is Done!

Thank you for reading !