

```
OPEN Radix.java
2 import java.io.File;
3 import java.io.FileNotFoundException;
4 import java.io.FileWriter;
5 import java.io.IOException;
6 import java.util.Scanner;
7
8 /**
9  * Clase de Radix. Dado el nombre de un archivo, se verifica si existe, se calcula el número de claves en él
10  * y se ordena mediante 6 iteraciones donde se analiza un dígito y se pasa cada clave a un archivo auxiliar
11  * para después unir las claves de todos los archivos auxiliares por cada iteración.
12  * @author Barreiro Valdez Alejandro, Piña Félix Emilio, Zepeda Baeza Jessica
13  */
14 public class Radix {
15     File archivo;
16     FileWriter escritor;
17     File[] fradix;
18     FileWriter[] fw;
19     int size;
20
21     /**
22      * Constructor que crea una instancia del archivo original, los archivos auxiliares y materializa los arreglos de archivos.
23      * @param nombreArchivo Cadena que almacena el nombre del archivo original
24      */
25     Radix(String nombreArchivo){
26         archivo = new File(nombreArchivo);
27         fradix = new File[10];
28         fw = new FileWriter[10];
29         fradix[0] = new File("f0_" + nombreArchivo);
30         fradix[1] = new File("f1_" + nombreArchivo);
31         fradix[2] = new File("f2_" + nombreArchivo);
32         fradix[3] = new File("f3_" + nombreArchivo);
33         fradix[4] = new File("f4_" + nombreArchivo);
34         fradix[5] = new File("f5_" + nombreArchivo);
35         fradix[6] = new File("f6_" + nombreArchivo);
36         fradix[7] = new File("f7_" + nombreArchivo);
37         fradix[8] = new File("f8_" + nombreArchivo);
38         fradix[9] = new File("f9_" + nombreArchivo);
39     }
40
41     /**
42      * Se calcula el número de claves en el archivo y regresa una variable booleana si este existe o no.
43      * Se crea un Scanner del archivo proporcionado y se aumenta el atributo size por cada línea leída.
44      * Si no se encuentra el archivo se hace una impresión y la variable booleana toma el valor de false.
45      * Se regresa la variable booleana.
46      * @throws IOException Si hay un error se lanza una excepción si hay un error en la lectura de archivos
47      */
48     private boolean tamaño() throws IOException{
49         /*Variable booleana que se regresa, es true si existe el archivo, en caso contrario es false.*/
50         boolean encontrado = true;
51         /*Se crea el Scanner del archivo se sabe si existe o no, se encuentra en un bloque try-catch
```

Aquí existen llamadas al sistema ya que se están creando archivos.
Se está reservando memoria

Se crea un archivo
Se reserva memoria y se crean archivos

Si hay un error se lanza una excepción si hay un error en la lectura de archivos

```
50 boolean encontrado = true;
51 /*Al crear el Scanner del archivo se sabe si existe o no, se encuentra en un bloque try-catch
52 por si no existe.*/
53 try {
54     Scanner sca = new Scanner(archivo);
55
56     while(sca.hasNextLine()){
57         sca.nextLine();
58         size++;
59     }
60     sca.close();
61 } catch (FileNotFoundException ex) {
62     System.out.println("No se encontró el archivo");
63     encontrado = false;
64 }
65 return encontrado;
66 }
67
68 /**
69 * Se realiza el ordenamiento al tener seis iteraciones y en cada una analizar el valor de un cierto dígito,
70 * después se dividen en archivos auxiliares dependiendo su valor y al final se regresan al archivo principal
71 * siguiendo un orden ascendente.
72 * @param archivo File del archivo original.
73 * @throws IOException
74 */
75 private void ordenamiento(File archivo) throws IOException{
76     try{
77         int divisor = 1;
78
79         /*Se le asigna un FileWriter a cada archivo auxiliar.*/
80         for(int i=0; i<10 ;i++){
81             fw[i] = new FileWriter(fradix[i]);
82
83             for(int i=0; i<6 ;i++){
84                 System.out.println("\nIteración " + (i+1) + ", el archivo es: ");
85                 Scanner scf = new Scanner(archivo);
86
87                 for(int j=0; j<10; j++){
88                     fw[j].write("Iteración " + (i+1) + "\n");
89
90                     /*Ciclo que lee todas las claves, obtiene el dígito a analizar de cada una y lo ingresa a su respectivo archivo auxiliar.*/
91                     for(int j=0; j<size; j++){
92                         Integer clave;
93                         String completo = new String();
94
95                         /*Se obtiene encuentra el número de cuenta y la clave completa por separado.*/
96                         while(scf.hasNextInt() == false)
97                             completo += (scf.next() + " ");
98                         clave = scf.nextInt();
99                         completo += (clave.toString() + "\n");
```

Se está leyendo el archivo

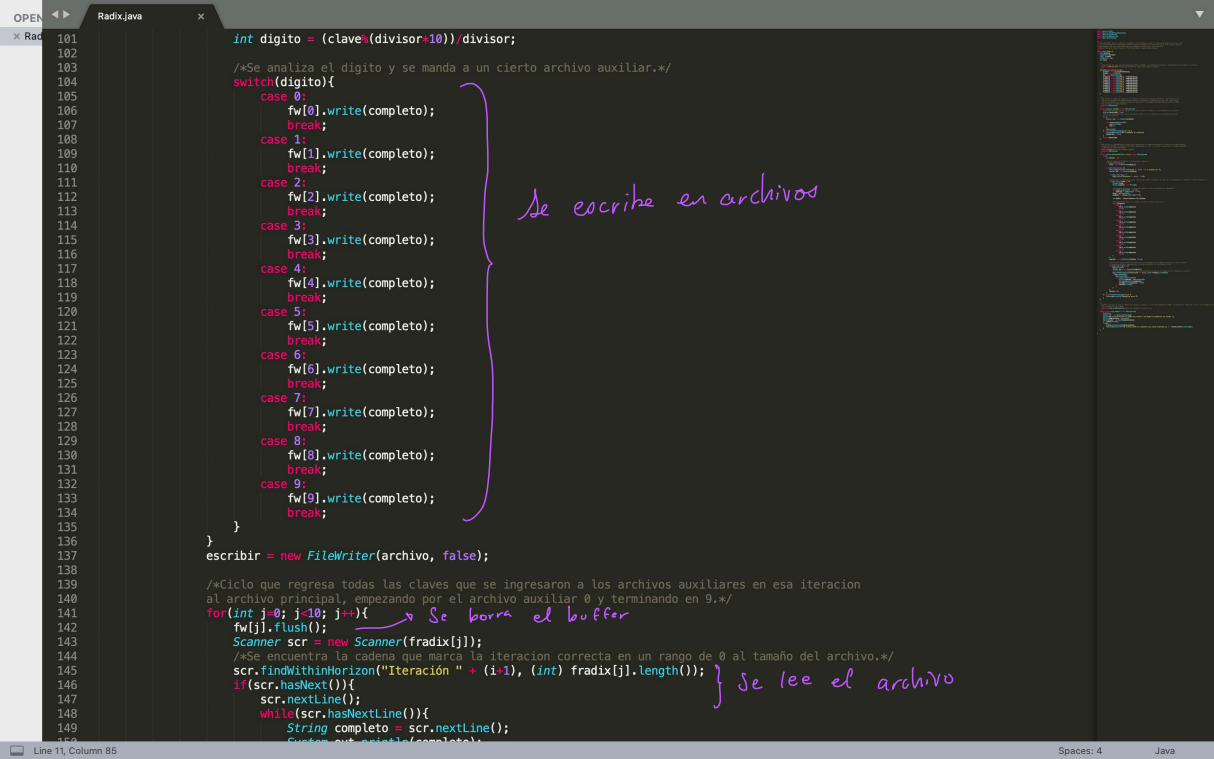
→ Error en la lectura (llamada al sistema)

→ Una impresión es una llamada

→ Misma excepción

→ Impresión de pantalla

→ Se escribe en un archivo



```
OPEN
x Radix.java
133         fw[j].write(completo);
134         break;
135     }
136 }
137 escribir = new FileWriter(archivo, false);
138
139 /*Ciclo que regresa todas las claves que se ingresaron a los archivos auxiliares en esa iteracion
140 al archivo principal, empezando por el archivo auxiliar 0 y terminando en 9.*/
141 for(int j=0; j<10; j++){
142     fw[j].flush();
143     Scanner scr = new Scanner(fradix[j]);
144     /*Se encuentra la cadena que marca la iteracion correcta en un rango de 0 al tamaño del archivo.*/
145     scr.findWithinHorizon("Iteración " + (i+1), (int) fradix[j].length());
146     if(scr.hasNext()){
147         scr.nextLine();
148         while(scr.hasNextLine()){
149             String completo = scr.nextLine();
150             System.out.println(completo);
151             escribir.write(completo + "\n");
152             escribir.flush();
153         }
154     }
155 }
156 divisor*=10;
157 }
158 } catch (FileNotFoundException ex) {
159     System.out.println("Sucedio un error.");
160 }
161 }
162
163 /**
164  * Metodo principal. Se pide el nombre del archivo a ordenar, se crea una instancia de Radix. Se calcula el numero de claves y si se indica que n
165  * Si se encuentra, se ordena.
166  * @throws java.io.IOException Lanza una excepcion si hay un error
167  */
168 public static void menu() throws IOException { → Error
169     boolean x;
170     Scanner sc = new Scanner(System.in);
171     System.out.println("Ingrese el nombre del archivo .txt donde se encuentran las claves: ");
172     String nombreArchivo = sc.next();
173     Radix ordred = new Radix(nombreArchivo);
174     x = ordred.tamaño();
175     if(x){
176         ordred.ordenamiento(ordred.archivo);
177         System.out.println("\nEl archivo donde se encuentran las claves ordenadas es: " + ordred.archivo.toString()); → Impresión
178     }
179 }
180
181 }
182
```

} Se lee y escribe en un archivo

} Error en la lectura e impresion de pantalla

→ Impresión