```java
package Ejercicio01;

public class Palabras implements Comparable<Palabras>  {

    //variables miembro
    private String palabra;
    private int contador;

    //constructor
    public Palabras (String p, int c) {
        this.palabra = p;
        this.contador = c;
    }

    //getters
    public String getPalabra() { return this.palabra; }
    public int getContador() { return this.contador; }

    //setter
    public void setPalabra(String p) { this.palabra = p; }
    public void setContador(int c) { this.contador = contador+c;}

    //este metodo sirve para almacenar el total de apariciones de palabra en el aux
    public void ponContador (int c) { this.contador = c;}

    //metodo para comparar 2 objetos
    //En este caso recibimos 2 objetos
    //vemos si el objeto que llama tiene una palabra con mas de 5 caracteres
    //y los comparamos con el objeto referencia
    //si tiene mas repeteciones, devolvemos 1 para que sea el nuevo objeto referencia
    //sino, pues se devuelve cero
    public int compara(Palabras p){
        //if(this == p) return 0;
        //if(p == null) return 0;
        if(this.getPalabra().length() >= 6){
            if (this.getContador() > p.getContador())return 1;
        }
        return 0;

    }

    @Override
    public int compareTo(Palabras p){
        int valor = this.getPalabra().compareTo(p.getPalabra());
        //de esta manera añadimos uno cuando usamos settree
        if (this.getPalabra().equals(p.getPalabra())) p.setContador(1);
        return valor;
    }

    @Override
    public String toString(){
```

```
52            String cadena = this.palabra + " : " + this.contador;
53            return cadena;
54        }
55
56    }
57
```

```java
package Ejercicio01;

//librerias importadas
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.ListIterator;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;

public class U08E01A {              //inicio clase
    private static Scanner stdin;           //para la lectura del fichero
    private static String cadena = "";      //para coger cada token y manipularlo
    private static ArrayList <Palabras> palabras = new ArrayList<>();   //arrayList donde guardamos Palabras
    private static Palabras temp;           //temporal donde iremos guardando Palabras para comparar
    private static boolean flag;            //bandera que nos permite saber si hay o no que almacenar nueva palabra
    private static int valor;               //para la busqueda de la palabra con mas repeticiones

    public static void main(String[] args) throws FileNotFoundException {   //inicio main

        //ponemos en marcha el crono
        long a = System.currentTimeMillis();    //empezamos a contar

        //fichero desde el cual vamos a trabajar
        File fichero = new File("C:\\temp\\quijote.txt");

        try {
            //indicamos desde vamos a coger los datos, en este caso el fichero
            stdin = new Scanner(fichero);

            while(stdin.hasNext()){     //mientras haya tokens
                //leemos token
                cadena = stdin.next();
                //limpiamos token
                cadena = cadena.replaceAll("[\\.\\,\\:\\/\\-\\*\\?\\¿\\¡\\!\\)\\(\\'\\@\\;\\n]","");
                //quitamos espacios
                cadena = cadena.trim();
                //pasamos token a minusculas
                cadena = cadena.toLowerCase();

                //primero vemos si la lista esta vacia y aï¿½adimos el primer elemento
                //al ser primer objeto le ponemos cero, porque en la primera comprobacion
                //se encontrará consigo mismo y por lo tanto se sumara 1 a su contador de apariciones.
                //posteriormente iremos poniendo el contador a 1 en cada objeto que creemos
                if (palabras.size() == 0) {
                    palabras.add(new Palabras(cadena,0));
                }

                //nos creamos un objeto temporal con la cadena que hemos creado
                temp  = new Palabras(cadena,0);

```

```
52                  //corremos el arrayList para ver si la palabra existe
53                 ListIterator<Palabras> li = palabras.listIterator();
54                 while ( li.hasNext()){
55                     flag = true;
56                     Palabras aux = (Palabras)li.next();
57                     //en este caso usamos el metodo equals de la clase String
58                     //comparamos el atributo palabra de cada objeto del arrayList con el temporal
59                     if(aux.getPalabra().equals(temp.getPalabra())) {
60                         aux.setContador(1);
61                         flag = false;
62                         break;
63                     }
64                 }
65 
66                 //en el caso de no encontrar la palabra, creamos el objeto correspondiente
67                 if(flag == true) {
68                     palabras.add(new Palabras(cadena,1));
69                     flag = true;
70                 }
71             }
72 
73         } catch (FileNotFoundException e) {
74             System.out.println("Error: Fichero no encontrado");
75             System.out.println(e.getMessage());
76         } catch (Exception e) {
77             System.out.println("Error de lectura del fichero");
78             System.out.println(e.getMessage());
79         }
80 
81 
82          //creamos un para empezar la busqueda de la palabra mas larga
83         temp = new Palabras(",",0);
84 
85         //volvemos a correr el array para encontrar la palabra de mas de 5 letras con mas repeticiones
86         //llamamos al metodo comparaTo de Palabras para realizar la comparacion
87         //segun la variable valor, almacenamos los datos en la variable temp
88         //con la que seguimos corriendo el arrayList y comparando
89         ListIterator<Palabras> li = palabras.listIterator();
90         while ( li.hasNext()){
91             Palabras aux = (Palabras)li.next();
92             valor = aux.compara(temp);
93             if(valor > 0) {
94                 temp.setPalabra(aux.getPalabra());
95                 temp.ponContador(aux.getContador());
96             }
97         }
98 
99         //paramos el crono y calculamos el tiempo transcurrido
100        long b = System.currentTimeMillis();
101        long total = b - a;
102        total = TimeUnit.MILLISECONDS.toSeconds(total);
```

```
103
104         //mostramos datos por pantalla
105         System.out.println("palabras localizadas: "+ palabras.size());
106         System.out.println("Palabra: " + temp.getPalabra());
107         System.out.println("Numero repeticiones: " + temp.getContador());
108         System.out.println("Tiempo transcurrido: " + total + " segundos");
109 
110         /*for (Palabras i : palabras) {
111             System.out.println(i.getPalabra() + " " + i.getContador() + " " + i.getPalabra().length());
112         }*/
113 
114     }
115 
116 
117 
118 }
119 
```

```java
package Ejercicio01;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Iterator;
import java.util.Scanner;
import java.util.Set;
import java.util.TreeSet;
import java.util.concurrent.TimeUnit;

public class U08E01B {

    private static Scanner stdin;           //para la lectura del fichero
    private static String cadena = "";      //para coger cada token y manipularlo
    private static Set <Palabras> palabras = new TreeSet<>();   //Treeset donde guardamos Palabras
    private static Palabras temp;           //temporal donde iremos guardando Palabras para comparar
    private static int valor;               //para la busqueda de la palabra con mas repeticiones

    public static void main(String[] args) throws FileNotFoundException {   //inicio main

        //ponemos en marcha el crono
        long a = System.currentTimeMillis();    //empezamos a contar

        //fichero desde el cual vamos a trabajar
        File fichero = new File("C:\\temp\\quijote.txt");

        try {
            //indicamos desde vamos a coger los datos, en este caso el fichero
            stdin = new Scanner(fichero);

            while(stdin.hasNext()){     //mientras haya tokens
                //leemos token
                cadena = stdin.next();
                //limpiamos token
                cadena = cadena.replaceAll("[\\.\\,\\:\\/\\-\\*\\?\\¿\\¡\\!\\)\\(\\'\\@\\;\\n\\<<\\>>]","");
                //quitamos espacios
                cadena = cadena.trim();
                //pasamos token a minusculas
                cadena = cadena.toLowerCase();

                //con la cadena limpia, creamos el objeto Palabras
                temp = new Palabras(cadena, 0);

                //añadimos el objeto a la coleccion
                palabras.add (new Palabras(cadena,1));

            }

        } catch (FileNotFoundException e) {
            System.out.println("Error: Fichero no encontrado");
            System.out.println(e.getMessage());
```

```
52          } catch (Exception e) {
53              System.out.println("Error de lectura del fichero");
54              System.out.println(e.getMessage());
55          }
56
57         //creamos un para empezar la busqueda de la palabra mas larga
58          temp = new Palabras(",",0);
59
60
61          //volvemos a correr el array para encontrar la palabra de mas de 5 letras con mas repeticiones
62          //llamamos al metodo comparaTo de Palabras para realizar la comparacion
63          //segun la variable valor, almacenamos los datos en la variable temp
64          //con la que seguimos corriendo el arrayList y comparando
65          Iterator<Palabras> it = palabras.iterator();
66          while (it.hasNext()){
67              Palabras aux = (Palabras)it.next();
68              valor = aux.compara(temp);
69              if(valor > 0) {
70                  temp.setPalabra(aux.getPalabra());
71                  temp.ponContador(aux.getContador());
72              }
73          }
74
75
76          //paramos el crono y calculamos el tiempo transcurrido
77          long b = System.currentTimeMillis();
78          long total = b - a;
79          total = TimeUnit.MILLISECONDS.toSeconds(total);
80
81          //mostramos datos por pantalla
82          System.out.println("palabras localizadas: "+ palabras.size());
83          System.out.println("Palabra: " + temp.getPalabra());
84          System.out.println("Numero repeticiones: " + temp.getContador());
85          System.out.println("Tiempo transcurrido: " + total + " segundos");
86
87      }
88
89
90
91  }
92
93
94
```

```java
package Ejercicio02;

public class ColaArrayInt implements ColasInt {
    //variables
    private static boolean flag = false; //para controlar la impresion de la linea

    //variables miembro
    private static final int LONGITUD_POR_DEFECTO = 10;
    private int maxLongitud;        //Tamaño por defecto
    private int cabeza;             //indice del primero de la cola
    private int fin;                //indice del ultimo de la cola
    private int[] datos;            //array que almacena elementos

    //constructores
    public ColaArrayInt() { this(LONGITUD_POR_DEFECTO);}

    public ColaArrayInt(int max) {
        this.maxLongitud = max + 1;  //un espacio extra
        this.fin = 0;
        this.cabeza = 1;
        datos = new int[maxLongitud];
    }

    //getters
    public int getFin() {return this.fin;}
    public int getCabeza() {return this.cabeza;}
    public int getMaxLongitud() {return maxLongitud;}

    //metodos
    public void vaciar() {
        this.fin = 0;
        this.cabeza = 1;
        flag = false;
    }

    /** Añadir a la cola e */
    public boolean encolar(int e) {
        //si no se cumple esta condicion, la cola esta llena
        if ((this.fin + 2) % this.maxLongitud != this.cabeza){
            this.fin = (this.fin + 1) % this.maxLongitud; // Incremento circular
            this.datos[this.fin] = e;
            flag = true;
            return true;
        }
        return false;
    }

    /** Eliminar y devolver el primer elemento (cabeza) */
    public int desencolar() {
        //si no se cumple, la cola esta vacia
        if (this.longitud() != 0){
```

```
52              int e = this.datos[this.cabeza];
53              this.cabeza = (this.cabeza + 1) % this.maxLongitud; // Incremento Circular
54              return e;
55          }
56          return 0;
57      }
58
59      /** @return primer valor */
60       public int primero() {
61          //sino se cumple: "La cola está vacía";
62          if(this.longitud() != 0) return this.datos[this.cabeza];
63          return 0;
64      }
65
66      /** @return Cantidad de elementos en la cola */
67       public int longitud(){
68          return ((this.fin + this.maxLongitud) - this.cabeza + 1) % this.maxLongitud;
69      }
70
71     @Override
72     public String toString(){
73
74       String cadena="";
75
76       if (flag == true){
77
78          //mientras la posicion de cabeza no sea mayor que la posicion del fin de la cola
79          if (cabeza <= fin){
80              for(int i = cabeza; i <= fin; i++){
81                  cadena = cadena + datos[i] + " ";
82              }
83          }
84
85          //como es una cola circular se puede dar el caso de que el fin de la cola
86          //sea una posicion menor que el inicio de la cola
87          //debemos leer desde el inicio de la cola hasta el final del array
88          //y posteriormente desde el incicio del array hasta el fin de la colas
89          else {
90              for(int i = cabeza; i < maxLongitud; i++){
91                  cadena = cadena + datos[i] + " ";
92              }
93              for (int i = 0; i <= fin; i++){
94                  cadena = cadena + datos[i] + " ";
95              }
96          }
97       }
98       return cadena;
99
100
101     }
102
```

```
103   }
104
```

```java
package Ejercicio02;

public interface ColasInt {

    public void vaciar();

    public boolean encolar(int e);

    public int desencolar();

    public int primero();

    public int longitud();

}

```

```java
package Ejercicio02;

public class U08E2 {

    public static void main(String[] args) {

        //creamos la cola, con la longitud por defecto
        ColaArrayInt cola = new ColaArrayInt();

        //llenamos la cola con numeros
        for(int i = 0; i < 4; i++){
            cola.encolar(i+1);
        }

        //imprimimos la cola
        System.out.println(cola.toString());

        //sacamos 1 numero
        cola.desencolar();

        //introuducimos otro numero
        cola.encolar(23);

        //imprimimos la cola
        System.out.println(cola.toString());

        //vaciamos la cola
        cola.vaciar();

        //imprimimos la cola. Debe salir una cadena vacia.
        System.out.println(cola.toString());

        //llenamos toda la cola
        for(int i = 0; i < cola.getMaxLongitud(); i++){
            cola.encolar(i+1);
        }

        //imprimimos la cola
        System.out.println(cola.toString());

        //intentamos insertar.
        cola.encolar(23);


        //imprimimos la cola. Como la cola esta llena, el 23 no debe poder ser insertado
        System.out.println(cola.toString());

        //vaciamos un puesto y volvemos a intentar insertar el 23
        cola.desencolar();
        System.out.println(cola.toString());

```

```
52          //
53          cola.encolar(23);
54
55
56          //imprimimos la cola. Ya debe aparecer el 23
57          System.out.println(cola.toString());
58
59          //desencolamos un par de numeros
60          cola.desencolar();
61          cola.desencolar();
62
63          //mostramos la cola
64          System.out.println(cola.toString());
65
66
67      }
68
69  }
70
```

```java
package Ejercicio03;

public class ColaArrayObj implements ColaObj {

    //variables
    private static boolean flag = false; //para controlar la impresion de la linea

    //variables miembro
    private static final int LONGITUD_POR_DEFECTO = 10;
    private int maxLongitud;        //Tamaño por defecto
    private int cabeza;             //indice del primero de la cola
    private int fin;                //indice del ultimo de la cola
    private Object[] datos;            //array que almacena elementos

    //constructores
    public ColaArrayObj() { this(LONGITUD_POR_DEFECTO);}

    public ColaArrayObj(int max) {
        this.maxLongitud = max + 1;  //un espacio extra
        this.fin = 0;
        this.cabeza = 1;
        datos = new Object[maxLongitud];
    }

    //getters
    public int getFin() {return this.fin;}
    public int getCabeza() {return this.cabeza;}
    public int getMaxLongitud() {return maxLongitud;}

    //metodos
    public void vaciar() {
        this.fin = 0;
        this.cabeza = 1;
        flag = false;
    }

    /** Añadir a la cola e */
    public boolean encolar(Object o) {
        //si no se cumple esta condicion, la cola esta llena
        if ((this.fin + 2) % this.maxLongitud != this.cabeza){
            this.fin = (this.fin + 1) % this.maxLongitud; // Incremento circular
            this.datos[this.fin] = o;
            flag = true;
            return true;
        }
        return false;
    }

    /** Eliminar y devolver el primer elemento (cabeza) */
     public Object desencolar() {
        //si no se cumple, la cola esta vacia
```

```
52          if (this.longitud() != 0){
53              Object o = this.datos[this.cabeza];
54              this.cabeza = (this.cabeza + 1) % this.maxLongitud; // Incremento Circular
55              return o;
56          }
57          return null;
58      }
59
60      /** @return primer valor */
61       public Object primero() {
62          //sino se cumple: "La cola está vacía";
63          if(this.longitud() != 0) return this.datos[this.cabeza];
64          return null;
65      }
66
67      /** @return Cantidad de elementos en la cola */
68       public int longitud(){
69          return ((this.fin + this.maxLongitud) - this.cabeza + 1) % this.maxLongitud;
70      }
71
72     @Override
73     public String toString(){
74       String cadena="";
75
76       if (flag == true){
77
78          //mientras la posicion de cabeza no sea mayor que la posicion del fin de la cola
79          if (cabeza <= fin){
80              for(int i = cabeza; i <= fin; i++){
81                  if (datos[i].getClass().getName().equals("Ejercicio03.Persona")){
82                      Persona temp = (Persona)datos[i];
83                      cadena = cadena + temp.getNombre() + " " + temp.getEdad() + " ";
84                  }
85                  else cadena = cadena + datos[i] + " ";
86              }
87          }
88
89          //como es una cola circular se puede dar el caso de que el fin de la cola
90          //sea una posicion menor que el inicio de la cola
91          //debemos leer desde el inicio de la cola hasta el final del array
92          //y posteriormente desde el incicio del array hasta el fin de la colas
93          else {
94              for(int i = cabeza; i < maxLongitud; i++){
95                  if (datos[i].getClass().getName().equals("Ejercicio03.Persona")){
96                      Persona temp = (Persona)datos[i];
97                      cadena = cadena + temp.getNombre() + " " + temp.getEdad() + " ";
98                  }
99                  else cadena = cadena + datos[i] + " ";
100             }
101             for (int i = 0; i <= fin; i++){
102                 if (datos[i].getClass().getName().equals("Ejercicio03.Persona")){
```

```
103                     Persona temp = (Persona)datos[i];
104                     cadena = cadena + temp.getNombre() + " " + temp.getEdad() + " ";
105                 }
106                 else cadena = cadena + datos[i] + " ";
107             }
108         }
109     }
110     return cadena;
111
112
113   }
114
115
116
117 }
118
119
120
```

```java
package Ejercicio03;

public interface ColaObj {

    public void vaciar();

    public boolean encolar(Object e);

    public Object desencolar();

    public Object primero();

    public int longitud();

}

```

```java
package Ejercicio03;

public class U08E03A {

    public static void main(String[] args) {

        //creamos las colas
        ColaArrayObj colaInt = new ColaArrayObj();
        ColaArrayObj colaString = new ColaArrayObj();
        ColaArrayObj colaPersonas = new ColaArrayObj();

        //cola de enteros
        colaInt.encolar(1);
        colaInt.encolar(2);
        colaInt.encolar(3);
        System.out.println(colaInt.toString());
        colaInt.desencolar();
        System.out.println(colaInt.toString());

        //cola de Strings
        colaString.encolar("Jorge");
        colaString.encolar("Victoria");
        colaString.encolar("Andreu");
        System.out.println(colaString.toString());
        colaString.desencolar();
        System.out.println(colaString.toString());

        //cola de Personas
        colaPersonas.encolar(new Persona("Domingo",  45 ));
        colaPersonas.encolar(new Persona("Fiesta", 46));
        colaPersonas.encolar(new Persona("Segura", 44));
        System.out.println(colaPersonas.toString());
        colaPersonas.desencolar();
        System.out.println(colaPersonas.toString());

    }

}

class Persona{

    //variables miembro
    String nombre;
    int edad;

    //constructor
    Persona(String n, int i){
        this.nombre = n;
        this.edad = i;
    }

```

```
52      //getters
53      public String getNombre(){return this.nombre;}
54      public int getEdad(){return this.edad;}
55
56      //setters
57      public void setNombre(String s){this.nombre = s;}
58      public void setEdad(int i){this.edad = i;}
59  }
60
```

```java
package Ejercicio03;

import java.util.ArrayList;

public class ColaArrayT<T> {

    //variables
    private static boolean flag = false; //para controlar la impresion de la linea


    //variables miembro
    private static final int LONGITUD_POR_DEFECTO = 10;
    T obj;          //declara un objeto de tipo T
    private int maxLongitud;        //Tamaño por defecto
    private int cabeza;             //indice del primero de la cola
    private int fin;                //indice del ultimo de la cola
    private Object[] datos;            //array que almacena elementos


    //constructor
    public ColaArrayT(T o){
        this(LONGITUD_POR_DEFECTO, o);
    }

    public ColaArrayT(int max, T o) {
        obj = o;
        this.maxLongitud = max + 1;  //un espacio extra
        this.fin = 0;
        this.cabeza = 1;
        datos = new Object[maxLongitud];
    }

    //getters
    public int getFin() {return this.fin;}
    public int getCabeza() {return this.cabeza;}
    public int getMaxLongitud() {return maxLongitud;}

    //metodos
    public void vaciar() {
        this.fin = 0;
        this.cabeza = 1;
        flag = false;
    }

    /** Añadir a la cola e */
    public boolean encolar(Object o) {
        //si no se cumple esta condicion, la cola esta llena
        if ((this.fin + 2) % this.maxLongitud != this.cabeza){
            this.fin = (this.fin + 1) % this.maxLongitud; // Incremento circular
            this.datos[this.fin] = o;
            flag = true;
```

```
52              return true;
53          }
54          return false;
55      }
56
57      /** Eliminar y devolver el primer elemento (cabeza) */
58       public Object desencolar() {
59          //si no se cumple, la cola esta vacia
60          if (this.longitud() != 0){
61              Object o = this.datos[this.cabeza];
62              this.cabeza = (this.cabeza + 1) % this.maxLongitud; // Incremento Circular
63              return o;
64          }
65          return null;
66      }
67
68      /** @return primer valor */
69       public Object primero() {
70          //sino se cumple: "La cola está vacía";
71          if(this.longitud() != 0) return this.datos[this.cabeza];
72          return null;
73      }
74
75      /** @return Cantidad de elementos en la cola */
76       public int longitud(){
77          return ((this.fin + this.maxLongitud) - this.cabeza + 1) % this.maxLongitud;
78      }
79
80      @Override
81      public String toString(){
82        String cadena="";
83
84        if (flag == true){
85
86            //mientras la posicion de cabeza no sea mayor que la posicion del fin de la cola
87            if (cabeza <= fin){
88                for(int i = cabeza; i <= fin; i++){
89                    if (datos[i].getClass().getName().equals("Ejercicio03.Persona")){
90                        Persona temp = (Persona)datos[i];
91                        cadena = cadena + temp.getNombre() + " " + temp.getEdad() + " ";
92                    }
93                    else cadena = cadena + datos[i] + " ";
94                }
95            }
96
97            //como es una cola circular se puede dar el caso de que el fin de la cola
98            //sea una posicion menor que el inicio de la cola
99            //debemos leer desde el inicio de la cola hasta el final del array
100           //y posteriormente desde el incicio del array hasta el fin de la colas
101           else {
102               for(int i = cabeza; i < maxLongitud; i++){
```

```
                    if (datos[i].getClass().getName().equals("Ejercicio03.Persona")){
                        Persona temp = (Persona)datos[i];
                        cadena = cadena + temp.getNombre() + " " + temp.getEdad() + " ";
                    }
                    else cadena = cadena + datos[i] + " ";
                }
                for (int i = 0; i <= fin; i++){
                    if (datos[i].getClass().getName().equals("Ejercicio03.Persona")){
                        Persona temp = (Persona)datos[i];
                        cadena = cadena + temp.getNombre() + " " + temp.getEdad() + " ";
                    }
                    else cadena = cadena + datos[i] + " ";
                }
            }
        }
        return cadena;


    }



}

```

```java
package Ejercicio03;

public interface ColaGen {

    public void vaciar();

    public boolean encolar(Object e);

    public Object desencolar();

    public Object primero();

    public int longitud();

}

```

```java
package Ejercicio03;

public class U08E03B {

    public static void main(String[] args) {

        ColaArrayT<Integer> cola1 = new ColaArrayT<Integer>(1);
        ColaArrayT<String> cola2 = new ColaArrayT<String>("h");
        ColaArrayT<Persona> cola3 = new ColaArrayT<Persona>(new Persona("Jorge", 10));

         //cola de enteros
         cola1.encolar((Integer)1);
         cola1.encolar((Integer)2);
         cola1.encolar((Integer)3);
         System.out.println(cola1.toString());
         cola1.desencolar();
         System.out.println(cola1.toString());

         //cola de Strings
         cola2.encolar((String)"Jorge");
         cola2.encolar((String)"Victoria");
         cola2.encolar((String)"Andreu");
         System.out.println(cola2.toString());
         cola2.desencolar();
         System.out.println(cola2.toString());

         //cola de Personas
         cola3.encolar(new Persona("Jorge",  45 ));
         cola3.encolar(new Persona("Victoria", 46));
         cola3.encolar(new Persona("Andreu", 44));
         System.out.println(cola3.toString());
         cola3.desencolar();
         System.out.println(cola3.toString());
    }


}

```

```java
package Ejercicio04;

import java.util.Comparator;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.PriorityQueue;

public class U08E04 {

    public static void main(String[] args) {

        //variables locales
        int totalClientes = 0; //total clientes que entran en el dia
        int cuantosEntran = 0;    //nº de clientes que entran por minuto
        double lambda;        //media lambda para el calculo de clientes que pueden entrar cada minuto
        Cliente c;            //para crear clientes
        int tiempoBusqueda;   //para calcular el valor que hay que pasar a tBuscar en el cliente
        int compras;       //para calcular el tiempo de cola de cada cliente
        double tiempoPagar;   //para guardar el tiempo de cola para pagar
        int productosComprados; //para calcular el valor de compra en el cliente
        double media;
        double mediaBusqueda;
        double mediEnCola;
        int totalProductosComprados = 0;
        double totalTiempoBusqueda = 0;
        double totalEnCola=0;
        double tiempoPagando=0;
        double mediaPagando=0;
        double totalTiempo=0;
        double totalMinutos=0;
        double totalCaja1=0;
        double totalCaja2=0;

        //creamos el array con la tasa de clientes
        int[] tasaClientes = new int[11];

        //rellenamos el array con la tasa de clientes esperada
        tasaClientes[0] = 25;
        tasaClientes[1] = 40;
        tasaClientes[2] = 50;
        tasaClientes[3] = 65;
        tasaClientes[4] = 80;
        tasaClientes[5] = 42;
        tasaClientes[6] = 18;
        tasaClientes[7] = 21;
        tasaClientes[8] = 32;
        tasaClientes[9] = 40;
        tasaClientes[10] = 60;

        //creamos las colas
```

```
52          LinkedList<Cliente> colaBuscar = new LinkedList<>();
53          LinkedList<Cliente> caja1 = new LinkedList<>();
54          LinkedList<Cliente> caja2 = new LinkedList<>();
55
56          //creamos el reloj y su minutero
57          double reloj = 9.0;
58          double minuto = 1.0 / 60;   //ritmo de avance del reloj de la simulacion
59
60          while ( reloj <= 20 ){ //inicio del while reloj
61
62              //vemos cuantos clientes entran cada minuto
63              lambda = minuto * tasaClientes[(int)reloj - 9];
64              cuantosEntran = getPoisson(lambda);
65
66              //creamos los clientes de ese minuto
67              for(int i = 1; i <= cuantosEntran; i++){ //inicio for
68                  tiempoBusqueda = (int)calculoNormal(10,5.8);
69                  tiempoBusqueda = Math.abs(tiempoBusqueda); //me salen valores negativos
70                  compras = (int)(tiempoBusqueda * (Math.random() * 2));
71                  tiempoPagar = Math.round(2 + compras / 4.0);
72                  tiempoPagar = cuadraSegundos(tiempoPagar);
73                  c = new Cliente(reloj,tiempoBusqueda, tiempoBusqueda, compras, 0, tiempoPagar, 0 );
74                  colaBuscar.add(c);
75                  totalProductosComprados = totalProductosComprados + compras; //poner despues?
76                  totalTiempoBusqueda = totalTiempoBusqueda + tiempoBusqueda; //total minutos buscando
77                  tiempoPagando = tiempoPagando + tiempoPagar;
78                  totalTiempo = totalTiempoBusqueda + tiempoPagando;
79              }
80              totalClientes += cuantosEntran;
81
82              //restar 1 al campo tQueda de los clientes de la colaBuscar
83              Iterator it = colaBuscar.iterator();
84              while(it.hasNext()){
85                  Cliente aux = (Cliente)it.next();
86                  aux.setTQueda(aux.getTQueda()-1);
87              }
88
89              //ordenar los clientes por el tiempo de espera
90              colaBuscar.sort(
91                  new Comparator<Cliente>(){
92                      public int compare(Cliente c1, Cliente c2){
93                          if(c1.getTQueda() < c2.getTQueda()) return 1;
94                          if(c1.getTQueda() < c2.getTQueda()) return -1;
95                          return 0;
96                      }
97
98                  }
99              );
100
101             //ver los clientes que tiempo queda es menor o igual que 0
102             Iterator ite = colaBuscar.iterator();
```

```
103             while(ite.hasNext()){
104                 Cliente aux = (Cliente)ite.next();
105                 if(aux.getTQueda() <= 0){
106                     aux.setTCola(reloj);
107                     aux.setTQueda(aux.getTCobrar());
108                     ite.remove();
109                     if(caja1.size() <= caja2.size()) caja1.add(aux);
110                     else caja2.add(aux);
111                 }
112             }
113 
114             //contamos cuanta gente hay en caja
115             totalCaja1 = totalCaja1 + caja1.size();
116             totalCaja2 = totalCaja2 + caja2.size();
117 
118 
119 
120 
121 
122             //vemos cola de caja 1
123             for(int i = 0; i < caja1.size(); i++){
124                 Cliente q = caja1.getFirst();
125                 q.setTQueda(q.getTQueda()-1);
126                 if(q.getTQueda() <= 0){
127                     caja1.removeFirst();
128                     q.setTSale(reloj);
129                     totalEnCola = totalEnCola + (q.getTSale() - q.getTCola());
130                 }
131             }
132 
133             //vemos cola de caja 2
134             for(int i = 0; i < caja2.size(); i++){
135                 Cliente q = caja2.getFirst();
136                 q.setTQueda(q.getTQueda()-1);
137                 if(q.getTQueda() <= 0){
138                     caja2.removeFirst();
139                     q.setTSale(reloj);
140                     totalEnCola = totalEnCola + (q.getTSale() - q.getTCola());
141                 }
142             }
143 
144         //control de tiempo
145         reloj = reloj + minuto;
146         reloj = cuadraSegundos(reloj);
147         totalMinutos++;
148         if(reloj > 20){
149             System.out.println("Se cierran las cajas. Pendientes en caja1 " + caja1.size() + " y en caja 2 " + caja2.size() +
                " en local " + colaBuscar.size());
150             System.out.println();
151             if(colaBuscar.size() > 0) colaBuscar.removeAll(colaBuscar);
152             if(caja1.size() > 0) caja1.removeAll(caja1);
```

```
153                 if(caja2.size() > 0) caja2.removeAll(caja2);
154             }
155         }
156 
157         //Impresion por pantalla
158             System.out.println("Local abierto de 9:00 a 20:00");
159             System.out.println("Entran " + totalClientes + " clientes");
160             System.out.print("Total productos comprados: ");
161             System.out.println(totalProductosComprados);
162             media = (double)totalProductosComprados/totalClientes;
163             System.out.printf("Compran %.2f productos media.",media );
164             mediaBusqueda = (double)totalTiempoBusqueda/totalClientes;
165             mediaBusqueda = cuadraSegundos(mediaBusqueda);
166             System.out.printf("\nPasa %.2f minutos buscando productos",mediaBusqueda);
167             /*mediEnCola = totalEnCola/totalClientes;
168             mediEnCola = cuadraSegundos(mediEnCola);
169             System.out.printf("\nPasa %.2f minutos en cola", mediEnCola);*/
170             mediaPagando = tiempoPagando/totalClientes;
171             mediaPagando = cuadraSegundos(mediaPagando);
172             System.out.printf("\nPasa %.2f minutos pagando productos",mediaPagando);
173             System.out.printf("\nTiempo por cliente: COMPRANDO: %.2f", (totalTiempoBusqueda*100)/totalTiempo);
174             System.out.print("%");
175             System.out.printf(" PAGANDO: %.2f ", (tiempoPagando*100)/totalTiempo);
176             System.out.println("%");
177             System.out.printf("Tamaño cola caja 1 %.2f clientes/min", totalCaja1/totalMinutos);
178             System.out.printf("\nTamaño cola caja 2 %.2f clientes/min", totalCaja2/totalMinutos);
179         }
180 
181 
182 
183         private static double calculoNormal(double i, double d) {
184             double r1 = Math.random();
185             double r2 = Math.random();
186             double z1 = Math.sqrt((-2)*Math.log(r1))*Math.sin(2*Math.PI*r2);
187             double n1 = z1*i+d;
188             return n1;
189         }
190 
191         private static int getPoisson(double lambda) {
192            double L = Math.exp(-lambda);
193            double p = 1.0;
194            int k = 0;
195            do {
196                k++;
197                p = Math.random();
198            } while (p > L);
199            return k-1;
200         }
201 
202         //para cuadrar la hora, ya que a veces sales mas de 60 segundos
203         private static double cuadraSegundos(double dato){
```

```
204
205             String tiempo = String.valueOf(dato);
206             int pos = tiempo.indexOf(".");
207             pos++;
208             char caracter = tiempo.charAt(pos);
209             if (caracter == 54|| caracter == 55 || caracter == 56 || caracter == 57){
210                 dato = dato + 1;
211                 dato = dato -0.6;
212             }
213
214             return dato;
215         }
216
217
218
219     }
220
221
222
223     class Cliente {
224
225         //variables miembro
226         private double tEntra;
227         private double tBuscar;
228         private double tQueda;
229         private int compra;
230         private double tCola;
231         private double tCobrar;
232         private double tSale;
233
234         //constructor
235         public Cliente(double entra, double busca, double queda, int cuantos, double cola, double cobra, double sale){
236             this.tEntra = entra;
237             this.tBuscar = busca;
238             this.tQueda = queda;
239             this.compra = cuantos;
240             this.tCola = cola;
241             this.tCobrar = cobra;
242             this.tSale = sale;
243         }
244
245         //getters
246         public double getTEntra() { return this.tEntra;}
247         public double getTBuscar() { return this.tBuscar;}
248         public double getTQueda() { return this.tQueda;}
249         public int getCompra() { return this.compra;}
250         public double getTCola() { return this.tCola; }
251         public double getTCobrar() { return this.tCobrar;}
252         public double getTSale() { return this.tSale;}
253
254         //setters
```

```
255       public void setTEntra(double entra) { this.tEntra = entra;}
256       public void setTBuscar(double busca) { this.tBuscar = busca;}
257       public void setTQueda(double queda) { this.tQueda = queda;}
258       public void setCompra(int cantidad) { this.compra = cantidad;}
259       public void setTCola(double cola) { this.tCola = cola; }
260       public void setTCobrar(double cobra) { this.tCobrar = cobra;}
261       public void setTSale(double sale) { this.tSale = sale;}
262   }
263
```

```java
package Ejercicio05;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Arbol {

    private static int contador = 0;

    Nodo raiz;

    public Arbol(){
        raiz=null;
    }

    public void insertar(int i, NodoS palabra){
        Nodo n = new Nodo(i);
        n.contenido = palabra;

        if(raiz == null) {
            raiz=n;
        } else {
            Nodo aux = raiz;
            while ( aux!=null){
                n.padre=aux;
                if(n.llave >= aux.llave){
                    aux=aux.derecha;
                }else{
                    aux=aux.izquierda;
                }
            }

        if(n.llave < n.padre.llave){
            n.padre.izquierda = n;
        } else {
            n.padre.derecha = n;
        }
    }

}

    public void recorrer(Nodo n){
        if(n != null) {
            while(contador < 10){
                recorrer(n.izquierda);
                if(n.contenido.getNCaracteres() > 5){
                    int apariciones = n.contenido.buscarPalabra(n.contenido.getPalabra());
                    System.out.println("Indice " + n.llave + " palabra: " + n.contenido.getPalabra() + " aparece " +
```

```
                    apariciones + " veces.");
52                  contador++;
53              }
54              recorrer(n.derecha);
55          }
56      }
57  }
58
59
60
61      private class Nodo {
62
63          //variables miembro
64          public Nodo padre;
65          public Nodo derecha;
66          public Nodo izquierda;
67          public int llave;
68          public NodoS contenido;
69
70          public Nodo(int indice){
71              llave=indice;
72              derecha=null;
73              izquierda=null;
74              padre=null;
75              contenido=null;
76          }
77      }
78
79  }
80
81  class Principal{
82
83      public static void main(String[] args) {
84
85          Arbol miArbol = new Arbol();
86          boolean puedoInsertar = false;
87          String cadena;
88          NodoS temp;
89          int indice = 1;
90          List<String> paraules = new ArrayList<>();
91
92          //fichero desde el cual vamos a trabajar
93          File fichero = new File("C:\\temp\\quijote.txt");
94
95          try {
96              //indicamos desde vamos a coger los datos, en este caso el fichero
97              Scanner stdin = new Scanner(fichero);
98
99              while(stdin.hasNext()){     //mientras haya tokens
100                 //leemos token
101                 cadena = stdin.next();
```

```
                    //limpiamos token
                    cadena = cadena.replaceAll("[\\.\\,\\:\\/\\-\\*\\?\\¿\\¡\\!\\)\\(\\'\\@\\;\\n\\<<\\>>]","");
                    //quitamos espacios
                    cadena = cadena.trim();
                    //pasamos token a minusculas
                    cadena = cadena.toLowerCase();
                    //añadimos esa cadena al arrayList
                    if (!paraules.contains(cadena)){
                        paraules.add(cadena);
                        puedoInsertar = true;
                        indice++;
                    }
                    //con la cadena limpia, creamos el objeto NodoS
                    temp = new NodoS(cadena);
                    //si puedo insetar, creamos el nodo con un indice y el objeto
                    if (puedoInsertar) miArbol.insertar(indice, temp);
                    //ponemos el flag a false
                    puedoInsertar = false;
                }

            } catch (FileNotFoundException e) {
                System.out.println("Error: Fichero no encontrado");
                System.out.println(e.getMessage());
            } catch (Exception e) {
                System.out.println("Error de lectura del fichero");
                System.out.println(e.getMessage());
            }

            miArbol.recorrer(miArbol.raiz);


        }
    }

    class NodoS implements Comparable<NodoS>{

        //variables miembro
        String palabra;
        int nCaracteres;

        //constructor
        public NodoS(String pal){
            this.palabra = pal;
            this.nCaracteres = pal.length();
        }

        //getters
        public String getPalabra() { return this.palabra; }
        public int getNCaracteres() { return this.nCaracteres;}

        //setters
```

```
153       public void setPalabra(String pal) { this.palabra = pal;}
154       public void setNCaracteres(int cont) { this.nCaracteres = cont;}
155
156       @Override
157       public int compareTo(NodoS sub){
158           int valor = this.getPalabra().compareTo(sub.getPalabra());
159           return valor;
160       }
161
162       public int buscarPalabra(String paraula){
163           String cadena;
164           int contador=0;
165
166           //fichero desde el cual vamos a trabajar
167           File fichero = new File("C:\\temp\\quijote.txt");
168
169           try {
170               //indicamos desde vamos a coger los datos, en este caso el fichero
171               Scanner stdin = new Scanner(fichero);
172
173               while(stdin.hasNext()){     //mientras haya tokens
174                   //leemos token
175                   cadena = stdin.next();
176                   //limpiamos token
177                   cadena = cadena.replaceAll("[\\.\\,\\:\\/\\-\\*\\?\\¿\\¡\\!\\)\\(\\'\\@\\;\\n\\<<\\>>]","");
178                   //quitamos espacios
179                   cadena = cadena.trim();
180                   //pasamos token a minusculas
181                   cadena = cadena.toLowerCase();
182                   //añadimos esa cadena al arrayList
183                   if (paraula.equals(cadena)){
184                       contador++;
185                   }
186               }
187
188           } catch (FileNotFoundException e) {
189               System.out.println("Error: Fichero no encontrado");
190               System.out.println(e.getMessage());
191           } catch (Exception e) {
192               System.out.println("Error de lectura del fichero");
193               System.out.println(e.getMessage());
194           }
195
196           return contador;
197
198       }
199   }
200
```

```java
package Ejercicio06;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class U08E06 {
    //para puntuar los chistes que salen
    private static int[] puntuacion = new int[3]; //para añmacenar la puntuacion del usuario
    private static int[] posChistes = new int[3]; //para almacenar las posiciones de los chistes del random
    private static double[] comparacion;          //para almacenar el manhattam de los usuarios
    private static List<Usuario> misUsuarios = new ArrayList<>();   //para almacenar usuarios
    private static List<Chiste> misChistes = new ArrayList<>();     //para almacenar chistes
    private static List<Puntos> misPuntos = new ArrayList<>();      //para almacenar puntuaciones

    public static void main(String[] args) {

        //creamos los usuarios

        try {
            //nos creamos un filereader envuelto en un buffer de lectura
            BufferedReader br = new BufferedReader(new FileReader("c:\\temp\\usuarios.csv"));

            //leemos la primera linea
            String linea = br.readLine();

            //mientras hayan lineas que leer
            while (linea != null) {

                //construimos un array con los campos de la linea. Separamos los campos con split
                String [] fields = linea.split(";");

                //si hay info de la provincia, la mostramos por pantalla
                if(fields.length == 2) {
                    int id = Integer.parseInt(fields[0]);
                    String usuari = fields[1];
                    Usuario temp = new Usuario(id, usuari);
                    misUsuarios.add(temp);
                }

                 //leemos la siguiente linea
                linea = br.readLine();
            }

            br.close();


```

```
52
53          } catch (FileNotFoundException e) {
54              System.out.println(e.getMessage());
55          } catch (IOException e) {
56              System.out.println(e.getMessage());
57          }
58
59          //creamos los chistes
60
61
62         try {
63              //nos creamos un filereader envuelto en un buffer de lectura
64              BufferedReader br = new BufferedReader(new FileReader("c:\\temp\\chistes.csv"));
65
66              //leemos la primera linea
67              String linea = br.readLine();
68
69              //mientras hayan lineas que leer
70              while (linea != null) {
71
72                  //construimos un array con los campos de la linea. Separamos los campos con split
73                  String [] fields = linea.split(";");
74
75                  //si hay info de la provincia, la mostramos por pantalla
76                  if(fields.length == 2) {
77                      int id = Integer.parseInt(fields[0]);
78                      String text = fields[1];
79                      Chiste temp = new Chiste(id, text);
80                      misChistes.add(temp);
81                  }
82
83                   //leemos la siguiente linea
84                  linea = br.readLine();
85              }
86
87              br.close();
88
89
90
91          } catch (FileNotFoundException e) {
92              System.out.println(e.getMessage());
93          } catch (IOException e) {
94              System.out.println(e.getMessage());
95          }
96
97          //creamos los puntos
98
99
100         try {
101             //nos creamos un filereader envuelto en un buffer de lectura
102             BufferedReader br = new BufferedReader(new FileReader("c:\\temp\\puntos.csv"));
```

```
103
104             //leemos la primera linea
105             String linea = br.readLine();
106
107             //mientras hayan lineas que leer
108             while (linea != null) {
109
110                 //construimos un array con los campos de la linea. Separamos los campos con split
111                 String [] fields = linea.split(";");
112
113                 //si hay info de la provincia, la mostramos por pantalla
114                 if(fields.length == 3) {
115                     int idu = Integer.parseInt(fields[0]);
116                     int idc = Integer.parseInt(fields[1]);
117                     int punts = Integer.parseInt(fields[2]);
118                     Puntos temp = new Puntos(idu, idc, punts);
119                     misPuntos.add(temp);
120                 }
121
122                  //leemos la siguiente linea
123                 linea = br.readLine();
124             }
125
126             br.close();
127
128
129
130         } catch (FileNotFoundException e) {
131             System.out.println(e.getMessage());
132         } catch (IOException e) {
133             System.out.println(e.getMessage());
134         }
135
136
137
138         Scanner stdin = new Scanner(System.in);
139
140         //Escogemos 3 chistes al azar y los puntuamos
141         for(int i = 0; i <= 2; i++){
142             puntuacion[i] = 0;                                          //inicialimos la nota
143             int pos = (int)(Math.random() * misChistes.size());         //contamos un chiste al azar
144             posChistes[i] = pos;                                        //almacenamos la posicion del chiste
145             System.out.println(misChistes.get(pos).getTexto());         //mostramos el chiste
146             while (puntuacion[i] <= 0 || puntuacion[i] >= 6){
147                 System.out.print("Puntacion del chiste[1-5]: ");
148                 puntuacion[i] = stdin.nextInt();                        //almacenamos la puntuacion
149             }
150         }
151
152         //algoritmo de comparacion con el resto de usuarios
153
```

```
154             //creamos el array donde iremos almacenando los distintos calculos
155             comparacion = new double[misUsuarios.size()];
156 
157             //calculamos el manhattam vs los usuarios
158             for(int i = 0; i < comparacion.length; i++){
159                 comparacion[i] = manhattam(i);
160             }
161 
162             /*//para ver los calculos
163             for(int i = 0; i < comparacion.length; i++){
164                 System.out.println(comparacion[i]);
165             }*/
166 
167             //obtenemos la menor distancia
168             double menor=comparacion[0];
169             for(int i = 0; i < comparacion.length; i++){
170                 if (comparacion[i] < menor) menor = comparacion[i];
171             }
172 
173             //System.out.println(menor);
174 
175             //System.out.println();
176 
177             System.out.println("Usuarios parecidos: ");
178 
179             //imprimimos usuarios parecidos
180             for(int i = 0; i < comparacion.length; i++){
181                 if (comparacion[i] == menor){
182                     System.out.println(misUsuarios.get(i).getNombre());
183                     otrosChistes(i);
184                 }
185             }
186 
187 
188 
189         }
190 
191         public static double manhattam(int usuario){
192             //aumentamos 1 el usuario para que coincida la posicion
193             usuario++;
194 
195             double nota1=0;
196             double nota2=0;
197             double nota3=0;
198 
199             //corremos el arrayList de puntos para el chiste 1.
200             for(int i = 0; i < misPuntos.size();i++){
201                 if(misPuntos.get(i).getIdUsuario() == usuario && misPuntos.get(i).getIdChiste() == posChistes[0]) nota1 =
                    misPuntos.get(i).getPuntos();
202             }
203 
```

```
204             //corremos el arrayList de puntos para el chiste 2
205             for(int i = 0; i < misPuntos.size();i++){
206                 if(misPuntos.get(i).getIdUsuario() == usuario && misPuntos.get(i).getIdChiste() == posChistes[1]) nota2 =
                    misPuntos.get(i).getPuntos();
207             }
208
209             //corremos el arrayList de puntos para el chiste 3
210             for(int i = 0; i < misPuntos.size();i++){
211                 if(misPuntos.get(i).getIdUsuario() == usuario && misPuntos.get(i).getIdChiste() == posChistes[2]) nota3 =
                    misPuntos.get(i).getPuntos();
212             }
213
214             double calculo = (Math.abs(nota1-puntuacion[0] + Math.abs(nota2-puntuacion[1]) + Math.abs(nota3-puntuacion[2]))/3);
215
216
217
218
219             return calculo;
220         }
221
222
223         public static void otrosChistes(int usu){
224
225             usu++;
226
227             for(int i = 0; i < misPuntos.size(); i++){
228                 if (misPuntos.get(i).getIdUsuario() == usu){
229                     int chiste = misPuntos.get(i).getIdChiste();
230                     for(int z = 0; z < misChistes.size();z++){
231                         if (misChistes.get(z).getId() == chiste && posChistes[0] != z && posChistes[1] != z && posChistes[2] != z){
232                             System.out.println (misChistes.get(z).getTexto());
233                             System.out.println("Puntos otorgados: " + misPuntos.get(i).getPuntos());
234                         }
235                     }
236                 }
237             }
238         }
239
240
241
242
243     } //fin clase U08E06
244
245     class Usuario{
246
247         //variables miembro
248         int Id;
249         String nombre;
250
251         //constructor
252         public Usuario(int id, String nom){
```

```
253             this.Id = id;
254             this.nombre = nom;
255         }
256 
257         //getters
258         public int getId() { return this.Id;}
259         public String getNombre() { return this.nombre;}
260 
261         //setters
262         public void setId(int id) { this.Id = id;}
263         public void setNombre(String nom) { this.nombre = nom;}
264 
265     } //fin clase Usuario
266 
267     class Chiste{
268 
269         //variables miembro
270         int Id;
271         String texto;
272 
273         public Chiste (int id, String chiste){
274             this.Id = id;
275             this.texto = chiste;
276         }
277 
278         //getters
279         public int getId() { return this.Id;}
280         public String getTexto() { return this.texto;}
281 
282         //setters
283         public void setId(int id) { this.Id = id;}
284         public void setNombre(String chiste) { this.texto = chiste;}
285 
286     }//fin clase chiste
287 
288     class Puntos{
289 
290         //variables miembro
291         int IdUsuario;
292         int IdChiste;
293         int puntos;
294 
295         public Puntos(int idUser, int idText, int punts){
296             this.IdUsuario = idUser;
297             this.IdChiste = idText;
298             this.puntos = punts;
299         }
300 
301         //getters
302         public int getIdUsuario() { return this.IdUsuario;}
303         public int getIdChiste() { return this.IdChiste;}
```

```
304         public int getPuntos() { return this.puntos;}
305 
306         //setters
307         public void setIdUsuario(int IdUser) { this.IdUsuario = IdUser;}
308         public void setIdChiste(int idtext) { this.IdChiste = idtext;}
309         public void setPuntos(int punts) { this.puntos = punts;}
310 
311     }//fin clase Puntos
312 
313 
314 
315 
```