PROGRAMACION CONCURRENTE

Ejemplos III: Sopa de Letras

Laura Barros

Notas:

Objetivo

- Mostrar las diferentes estrategias que puede seguir un gestor que tiene que ejecutar una tarea compleja que puede ser paralelizada en diferentes subtareas, para organizar su ejecución concurrente distribuyendo las subtareas entre un conjunto de objetos activos que denominamos obreros y que las ejecutan concurrentemente.

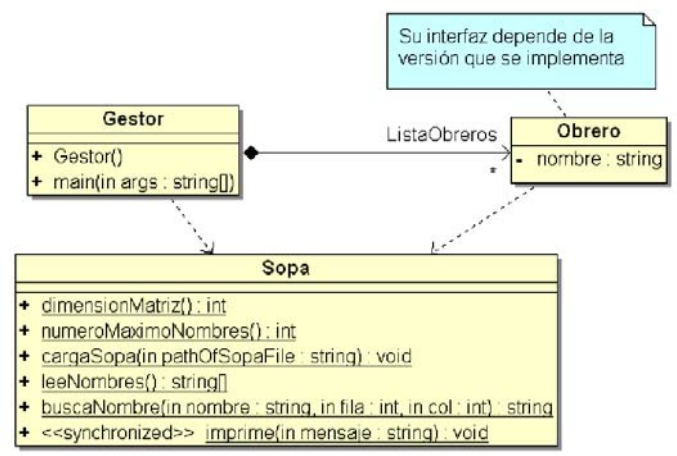
- El objetivo del ejemplo es la búsqueda concurrente en una sopa de letras de un conjunto de nombres (JUAN, INES, CARMEN, JOSE, MARTA, CARLOS, FELIPE, ANA, PEPE, ROSA) que se encuentran en ella en diferentes posiciones y direcciones. El programa debe buscar la ocurrencia de esos nombres en cualquier dirección y los imprime en la consola indicando la posición de la primera letra y la dirección (LR, RL, UD, DU, DR, DL, UR, UL).

```
NCAHPFELIPESOMCPAPEOUJILANEVIO
VABASAEKJINESASANBPXACNAUJVANE
AAJOHAPFZAZAREKUFIATEAONIAJERA
TROSAOZAFENLCARMENPILRKAENAUIS
RPEUNAUJAPOFAERIASAADLLNPEEOCO
AUJNAAFOJSUEJAGALDAFPOAUINISOU
METUALDAOJILPDPALPMIUSULLAEOLA
EKACARLOSAUIACAOAJUANIRAENPAEG
OSJKANIIEUPLAURAIREBANOCFOTEBA
KALSSENIDSAREÑOARUALPASAEJUANS
NABAFKAGPEMMPIJAHJINESAUOSENFO
UJIAMEAAIEPDTAIFOLLADIAPLVAJHR
DAUNFOLANEHAGUEPILEFLSRNIRENZI
ELFAEPOIJUANEPSENINEIPLONEMRAC
OAYENSENPMAAPEERADFORAOMPSSCSO
RSOOAAUJAEEPHZALNAISUDSANAREEG
ANAUJAZAPAPANAFAUIUJUAINEUENIL
EHÑEJUANSAARENCARLOSDEPTAMPIAR
ASDEJUANPAATRAMEAAZAPARLNATAER
BONEINESGAAECARMENÑAPUAIPNDUAH
EFESIMANSENIADJUANFAJPAPAULATA
```


Notas:

Especificación

- **En el diagrama de clases de la figura se muestra la arquitectura de la aplicación basada en tres clases.**



Notas:

Clase **Gestor**: Existe un único objeto manager en la aplicación que es activo a través del thread del procedimiento main() de la aplicación. El gestor carga del fichero cuyo path se recibe en su lanzamiento haciendo uso del método estático Sopa.CargaSopa(). De forma repetida, el gestor invoca el método estático Sopa.leeNombres() y en él se queda suspendido a la espera de que el operador introduzca la lista de nombres que quiere buscar en la sopa. El operador la introduce como una lista de nombres separados por espacios. Por ejemplo:

Juan, Ines, Felipe, Carmen, Pepe, Juan, Luis **return**

El manager utilizando las diferentes estrategias que se irán proponiendo, delega en un conjunto de obreras la búsqueda de cada una de los nombres propuestas. Espera a que todas las obreras hayan finalizado, y luego vuelve a quedar a la espera de que el operador proponga desde el teclado una nueva lista de nombres. El manager cierra la aplicación cuando recibe del operador una lista sin ningún nombre.

Gestor() => Constructor del objeto Gestos. De acuerdo con la estrategia de implementación, instancia todos los elementos que se requieren para ejecutar la aplicación;

main(String[] args) => Método que lanza la aplicación. En args debe pasarse el nombre del fichero en el que se almacena la sopa de letras.

Clase **Obrero:** Durante la ejecución de cada búsqueda, deberán instanciarse o estar instanciadas tantos objetos de la clase Obrero como nombres se vayan a buscar. Cada obrero participante recibe al comienzo de la búsqueda un nombre que debe buscar en la sopa de letras. Recorre la matriz y en cada posición haciendo uso del método estático *Sopa.busca()* comprueba si en ella se inicia el nombre buscado, y por cada uno que encuentra lo escribe sobre la impresora haciendo uso del procedimiento estático *Sopa.imprime()*, indicando su posición y la dirección en que se encuentra. Termina notificando al gestor de alguna manera que ha terminado su tarea. Los obreros ofrecen diferentes interfaces de acuerdo con la estrategia que se utilice.

Clase **Sopa**: Constituye la infraestructura pasiva y estática en la que se encuentra almacenada la sopa de letras y a la que acuden los obreros para buscar los nombres en la sopa, y para imprimir los resultados encontrados. Contiene el algoritmo de búsqueda que es utilizado por los obreros para encontrar su nombre de consigna. Los procedimientos que ofrecen su interfaz son:

static void cargaSopa(String pathOfSopaFile) => Es invocado por el manager para que lea la sopa de letras del fichero cuyo path se pasa como parámetro.

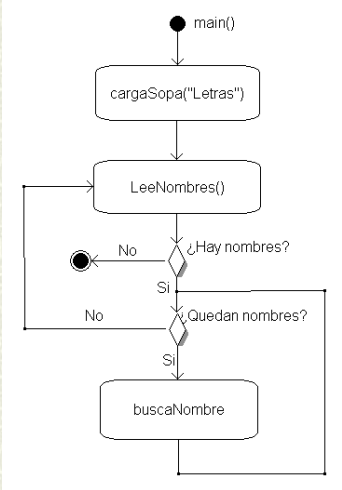
static String[] leeNombre() => Espera que el operador introduzca una lista de nombres desde el teclado. Retorna un array de strings con un nombre en cada elemento. Si no se introduce ningún nobre retorna un string nulo.

static void String buscaNombre(String nombre, int fila, int col) => Procedimiento reentrante que es invocado concurrentemente por cada obrero para buscar su nombre. Busca todas las ocurrencias del nombre que se inicie en la fila y columna que se pasan como parámetro. Retorna un mensaje que contiene el nombre, la fila, la columna, y la dirección o direcciones en que se ha encontrado.

synchronized static void imprime(String mensaje) => Imprime en la consola el texto que se pasa como parámetro mensaje. Es un método synchronaized, por lo que si se invoca concurrentemente, se secuencializa su ejecución.

Diseño I: Implementación secuencial.

- El gestor realiza el proceso completo buscando en el propio thread del método principal main() los nombres que introduce el operador.



Notas:

Diseño I: Código

```
public class Obrero {
     String nombre;
     Obrero(String nombre) {
          this.nombre = nombre;
          String s;
          for (int fila = 0; fila < Sopa.dimensionMatriz(); fila++) {
               for (int col = 0; col < Sopa.dimensionMatriz(); col++) {
                    s = Sopa.busca(nombre, fila, col);
                    if (s != "")
                    Sopa.imprime(s);
               }
          }
     }
}
```


Notas:

Diseño I: Código

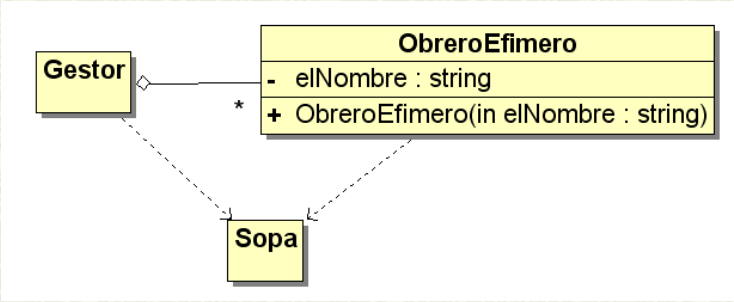
```
import java.io.*;
import java.util.*;
public class Gestor {
     public static void main(String[] args) {
               long tiempoInicial;
          try {
               Sopa.cargaSopa(args[0]);
          } catch (IOException e) {    }    ;

          int numNombres = 1;
          while (numNombres != 0) {
               Vector<Obrero> listaObreros = new Vector<Obrero>(numNombres);
               Vector<String> listaNombres = new Vector<String>();
               listaNombres = Sopa.leeNombres();
               tiempoInicial = System.currentTimeMillis();
               numNombres = listaNombres.size();
               for (int i = 0; i < numNombres; i++) {
                    String nombre = listaNombres.elementAt(i);
                    listaObreros.add(i, new Obrero(nombre));
               }
               System.out.println("Tiempo de ejecución: "
                         + (System.currentTimeMillis() - tiempoInicial) + " ms");
          }
     }
}
```


Notas:

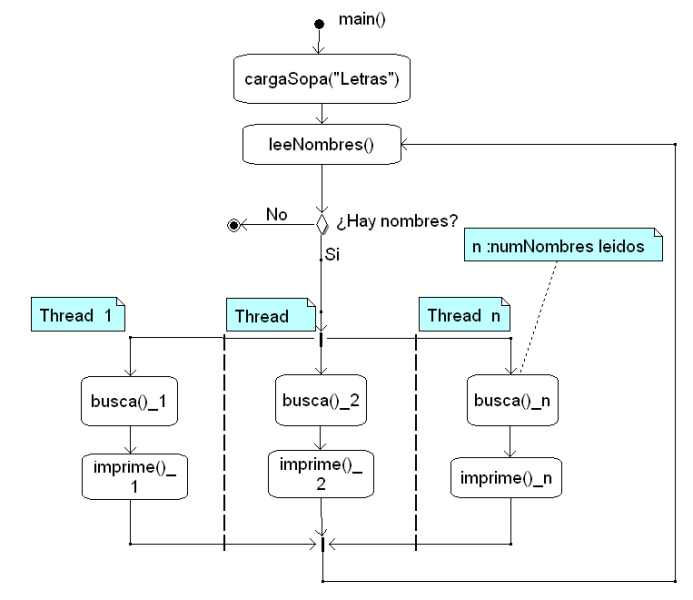
Diseño 2: Creación dinámica de los objetos obreros

- El gestor una vez que ha recibido la lista de nombres que deben ser buscados, crea dinámicamente un nuevo obrero por cada nombre, y le pasa el nombre que debe buscar a través del constructor. Los obreros son efímeros, y se destruyen tras haber realizado la búsqueda y su impresión. El gestor queda suspendido a la espera de que cada uno de los obreros hayan terminado y luego vuelve a requerir al operador una nueva lista de nombres o la orden de finalizar.



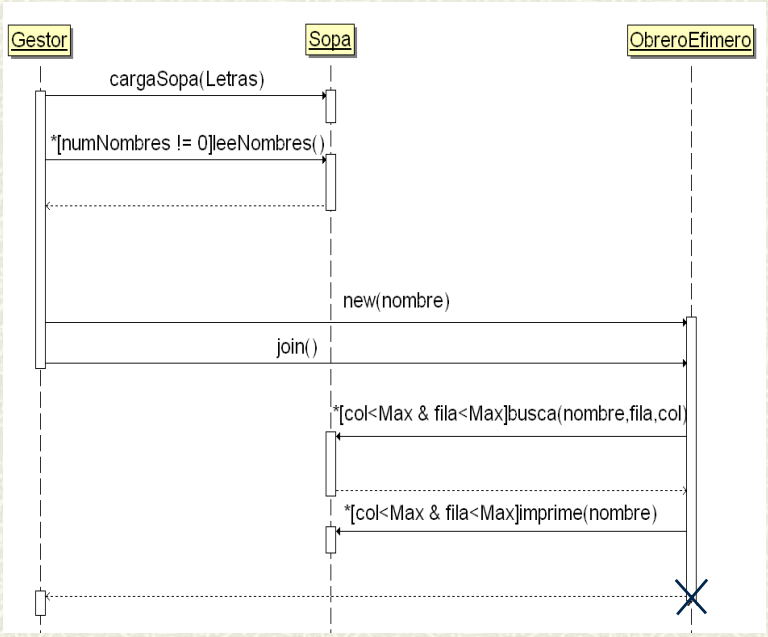
Notas:

Diseño 2:Diagrama de actividad



Notas:

Diseño 2:Diagrama de secuencias



Notas:

Diseño 1: ¿Modificar?

- **"Los obreros son efímeros, y se destruyen tras haber realizado la búsqueda"**

```
public class Obrero {
     String nombre;
     Obrero(String nombre) {
          this.nombre = nombre;
          String s;
          for (int fila = 0; fila < Sopa.dimensionMatriz(); fila++) {
               for (int col = 0; col < Sopa.dimensionMatriz(); col++) {
                    s = Sopa.busca(nombre, fila, col);
                    if (s != "")
                    Sopa.imprime(s);
               }
          }
     }
}
```


Notas:

Diseño 2:Código

```
public class ObreroEfimero extends Thread {
     String nombre;
     ObreroEfimero(String nombre){
          this.nombre=nombre;
          this.start();
     }
     public void run() {
          for (int fila=0;fila<Sopa.dimensionMatriz();fila++){
                for (int col=0;col<Sopa.dimensionMatriz();col++){
                     String s = Sopa.busca(nombre,fila,col);
                     if (s!="")
                          Sopa.imprime(s);
                }
          }
     }
}
```


Notas:

Diseño 1: ¿Modificar?

- **"crea dinámicamente un nuevo obrero por cada nombre"**
- **"El gestor queda suspendido a la espera de que cada uno de los obreros hayan terminado"**

```
import java.io.*;
import java.util.*;
public class Gestor {
     public static void main(String[] args) {
          long tiempoInicial;
          try {
               Sopa.cargaSopa(args[0]);
          } catch (IOException e) {    }    ;

          int numNombres = 1;
          while (numNombres != 0) {
               Vector<Obrero> listaObreros = new Vector<Obrero>(numNombres);
               Vector<String> listaNombres = new Vector<String>();
               listaNombres = Sopa.leeNombres();
               tiempoInicial = System.currentTimeMillis();
               numNombres = listaNombres.size();
               for (int i = 0; i < numNombres; i++) {
                    String nombre = listaNombres.elementAt(i);
                    listaObreros.add(i, new Obrero(nombre));
               }
               System.out.println("Tiempo de ejecución: "
                         + (System.currentTimeMillis() - tiempoInicial) + " ms");
          }
     }
}
```


Notas:

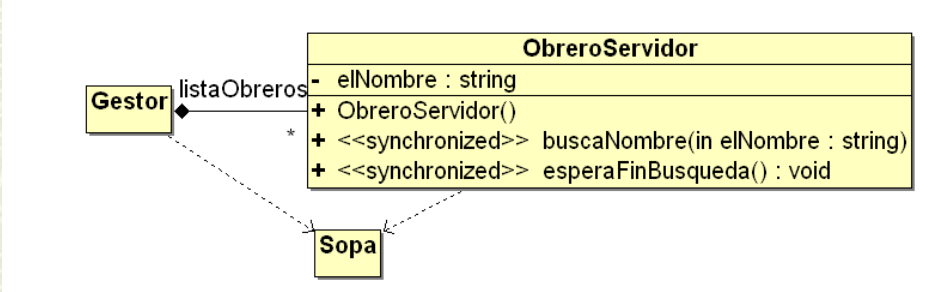
Diseño 2:Código

```
import java.io.*;
import java.util.*;
public class Gestor {
     public static void main(String[] args) {
          Collection<ObreroEfimero> obrerosCreados = new ArrayList<ObreroEfimero>();
          Vector<String> listaNombres = new Vector<String>();
          int numPalabras = 1;
          long tiempoInicial = 0;
          try {
               Sopa.cargaSopa(args[0]);
          } catch (IOException e) {
               System.out.println("el fichero " + "args[0]" + " no existe");
          }
          ;
          while (numPalabras != 0) {
               listaNombres = Sopa.leeNombres();
               tiempoInicial = System.currentTimeMillis();
               numPalabras = listaNombres.size();
               for (int i = 0; i < numPalabras; i++) {
                    obrerosCreados.add(new ObreroEfimero(listaNombres.get(i)));
               }
               Iterator<ObreroEfimero> iter = obrerosCreados.iterator();
               iter = obrerosCreados.iterator();
               while (iter.hasNext()) {
                    try {
                         iter.next().join();
                    } catch (InterruptedException e) {
                    }
               }
               System.out.println("Tiempo de ejecución: "
                         + (System.currentTimeMillis() - tiempoInicial) + " ms");
          }
     }
}
```


Notas:

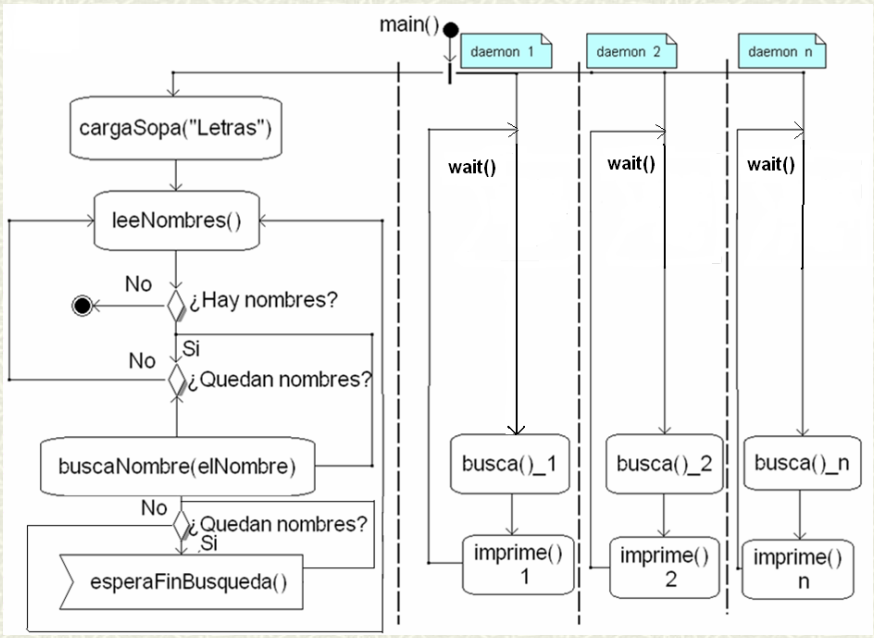
Diseño 3: Creación estática de los objetos obreros.

- El gestor crea en su constructor un número obreros igual al número máximo de nombres que pueden ser requeridos por el operador (*Sopa.numeroMaximoNombres()*). Los obreros permanecen creados durante todo el tiempo de la aplicación.
- Cuando el gestor ha recibido la lista de nombres se comunica con un número de obreros igual al número de nombres recibidos, y a cada uno de ellos le transfiere el nombre que deben buscar. Luego vuelve a comunicar con cada uno de los obreros para esperar a que hayan finalizado su búsqueda. Cuando todos los obreros han finalizado la búsqueda, el gestor requiere del operador una nueva lista de nombres o la orden de finalizar.
- Los obreros son servidores activos. Cuando se crean se suspenden en su propio lock hasta que el gestor le proporciona un nuevo nombre y pasan a buscarlo. Luego se vuelven a suspender hasta que se le proporcione el siguiente. Finalizan cuando detectan que el gestor ha finalizado (como daemons).



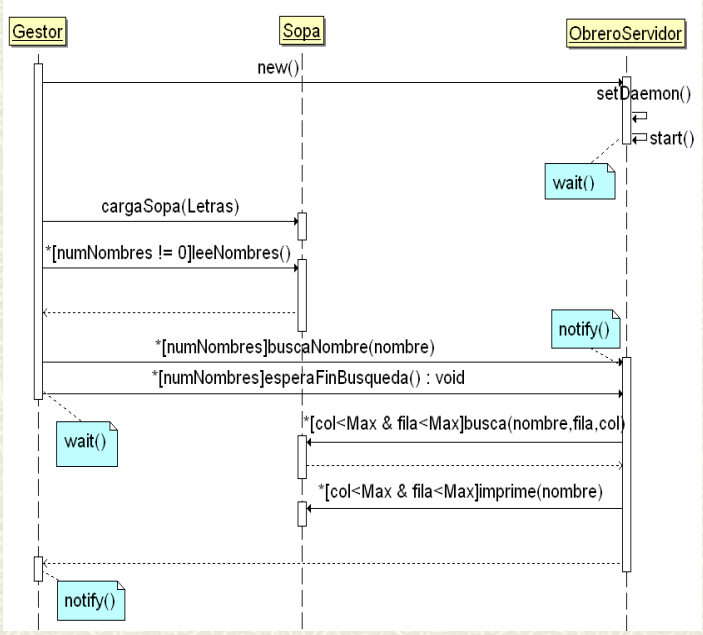
Notas:

Diseño 3:Diagrama de actividad



Notas:

Diseño 3:Diagrama de secuencias



Notas:

BuscaNombre() lo hace sobre cada obrero, pero hemos dibujado uno sólo para evitar la complejidad.

Diseño Base: ¿Modificar?

```
public class ObreroPersistente extends Thread {
   String nombre;
   ObreroPersistente(){
      …
   }

   synchronized public void buscaNombre(String nombre){
      nombre=nombre;
   }

   synchronized public void esperaFinBusqueda(){
      …
   }
```

```
   public void run() {
            …
        for (int fila=0;fila<Sopa.dimensionMatriz();fila++){
            for (int col=0;col<Sopa.dimensionMatriz();col++){
               String s = Sopa.busca(nombre,fila,col);
               if (s!="")
                  Sopa.imprime(s);
            }
         }
        …

   }
}
```

- **Los obreros permanecen creados durante todo el tiempo de la aplicación.**
- **Los obreros son servidores activos. Cuando se crean se suspenden en su propio lock hasta que el gestor le proporciona un nuevo nombre y pasan a buscarlo.**
- **Cuando el gestor ha recibido la lista de nombres se comunica con los obreros y le transfieres el nombre que deben buscar.**
- **Luego vuelve a comunicar con cada uno de los obreros para esperar a que hayan finalizado su búsqueda.**
- **Luego se vuelven a suspender hasta que se le proporcione el siguiente.**
- **Los obreros** son **daemons.**

Notas:

Diseño 3:Código

```
public class ObreroPersistente extends Thread {
   String nombre;
   boolean encontrado;
   ObreroPersistente(){
      encontrado=false;
      nombre=null;
      this.setDaemon(true);
      this.start();
   }

   synchronized public void buscaNombre(String nombre){
      this.nombre=nombre;
      this.notify();
   }

   synchronized public void esperaFinBusqueda(){
      if (!encontrado){
         try {
            this.wait(); // Se suspende el GESTOR
         } catch (InterruptedException e) {}
         // Nunca se produce
      }
   }

    synchronized private void await(){
      try {
         this.wait(); // Se suspende el OBRERO
      } catch (InterruptedException e) {}
    }

    synchronized private void activaGestor(){
      this.notify();
    }

    ...
```

```
   public void run() {
      while(true){
         await(); // Permanece hasta gestor buscaNombre
         for (int fila=0;fila<Sopa.dimensionMatriz();fila++){
              for (int col=0;col<Sopa.dimensionMatriz();col++){
                 String s = Sopa.busca(nombre,fila,col);
                 if (s!="")
                    Sopa.imprime(s);
              }
           }
         encontrado=true;
         activaGestor();
      }
   }
}
```


Notas:

Diseño 3:Código

```
import java.io.*;
import java.util.*;
public class Gestor {
   public static void main(String[] args) {
      ObreroPersistente[] listaObreros=
         new ObreroPersistente[Sopa.numeroMaximoDeNombres()];
      for (int i=0;i<Sopa.numeroMaximoDeNombres();i++){
         listaObreros[i]=new ObreroPersistente();
      }

      Vector<String> listaNombres = new Vector<String>();
      int numNombres = 1;
      long startTime;

      try {
         Sopa.cargaSopa(args[0]);
      } catch (IOException e) {
         System.out.println("el fichero "+"args[0]"+" no existe");
      };

      while (numNombres != 0) {
         listaNombres = Sopa.leeNombres();
         startTime = System.currentTimeMillis();
         numNombres = listaNombres.size();
         for (int i = 0; i < numNombres; i++) {
            listaObreros[i].buscaNombre(listaNombres.elementAt(i));
         }
         for (int i = 0; i < numNombres; i++) {
            listaObreros[i].esperaFinBusqueda();
         }
         System.out.println("Tiempo de ejecución: "
              + (System.currentTimeMillis() - startTime) + " ms");
      }
   }
}
```

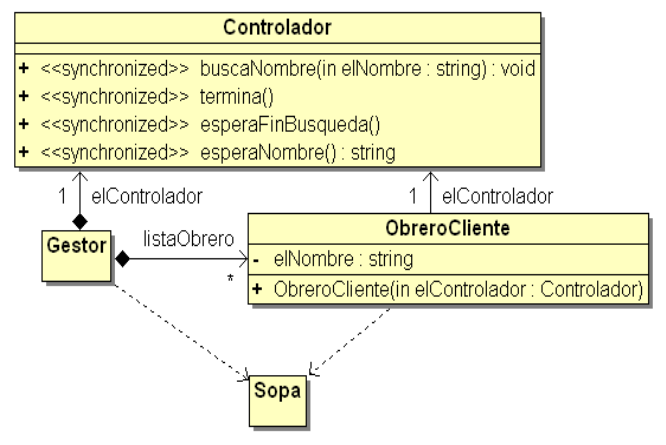

Notas:

Diseño 4: Interacción a través de un controlador pasivo.

- El gestor crea en su constructor un número de obreros igual al número máximo de nombres que pueden ser requeridas por el operador (*Sopa.numeroMaximoNombres()*), así como el controlador. Los obreros y el controlador permanecen creados durante todo el tiempo de la aplicación.
- Cuando el gestor ha recibido la lista de nombres comunica al controlador (*elControlador.buscaNombre()*) los diferentes nombres recibidos. Luego se suspende en el controlador (*elControlador.esperaFinBusqueda()*) hasta que todos los nombres han sido buscados. Luego requiere del operador una nueva lista de nombres o la orden de finalizar, la cual también la comunica al controlador (*elControlador.termina()*).
- Los obreros son también objetos activos y clientes del controlador. Cuando se crean se suspenden en el controlador (*el Controlador.esperaNombre()*)a la espera de que se le proporciones el nombre que deben buscar. Luego busca el nombre y vuelve a repetir la espera hasta una nueva búsqueda. Cada obrero finaliza cuando el controlador le proporciona como nombre a buscar un string nulo.
- El Controlador es un servidor pasivo que sirve de enlace entre el gestor y los obreros que son todos clientes suyos.
  - Cuando recibe el requerimiento de un nuevo nombre (*esperaNombre()*) por parte de un obrero, lo suspende hasta que el nombre haya sido encargada por el gestor (*buscaNombre()*).
  - Cuando el gestor le requiere si ha terminado la búsqueda (*termina()*), lo suspende, hasta que todas los obreros activos hayan finalizado sus búsquedas.
  - Cuando recibe el requerimiento de finalizar, activa todas las obreras suspendidas pasándole un string nulo para que terminen .

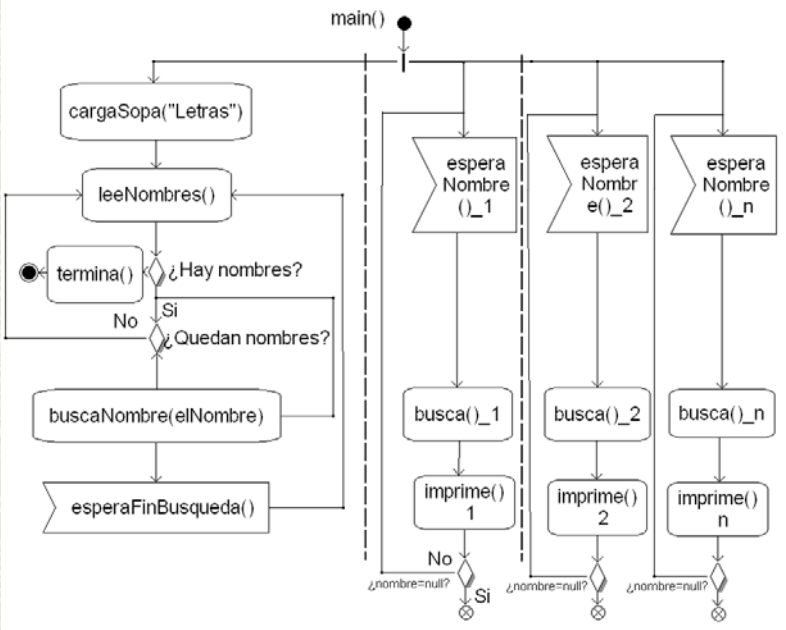
Notas:

Diseño 4: Especificación



Notas:

Diseño 4:Diagrama de actividad



Notas:

Diseño 4:Código

```
public class ObreroCliente extends Thread {
   Controlador elControlador;
   String elNombre="Inicio";
   ObreroCliente(Controlador elControlador) {
      this.elControlador = elControlador;
      this.start();
   }

   public void run() {
      while(!elNombre.equals("")){
         elNombre=elControlador.esperaNombre();
         if (!elNombre.equals("")){ // si es "" no busca sino termina.
            for (int fila=0;fila<Sopa.dimensionMatriz();fila++){
               for (int col=0;col<Sopa.dimensionMatriz();col++){
                  String s = Sopa.busca(elNombre,fila,col);
                  if (s!="") Sopa.imprime(s);
               }
            }
         }
      }
   }
}
```


Notas:

Diseño 4: Controlador

```
import java.util.*;
public class Controlador2 {
int numObrerasEsperando;


Stack<String> colaNombres=new Stack<String>();

Controlador2() {
        numObrerasEsperando=0;
}

Synchronized public void esperaFinBusqueda() {
while (numObrerasEsperando<Sopa.numeroMaximoDeNombres()){
        try {
                this.wait();
        }catch (InterruptedException e) {}
}
}


synchronized public void buscaNombre(String elNombre) {
        this.colaNombres.push(elNombre);
        //despierta las obreras que estan en el wait del controlador
        //despierta al gestor que está en el wait del controlador
        this.notifyAll();
}
```

```
synchronized public void termina() {
for (int i = 0; i < numObrerasEsperando; i++) {
        buscaNombre("");
}
};

synchronized public String esperaNombre() {
numObrerasEsperando++;
while(colaNombres.size()==0){
        try {
                this.wait();
        }catch (InterruptedException e) {}
}
//saldrá del wait cuando tenga un nombre que
        buscar(enbuscaNombre la han hecho notify)
numObrerasEsperando--;
return colaNombres.pop();
}
}
```


Notas:

Diseño 4:Código (variante usando dos lock)

```
import java.util.*;
public class Controlador {
   int numObrerasEsperando;
   Stack<String> colaNombres=new Stack<String>();
   Object esperandoFin=new Object();

   Controlador() {
      numObrerasEsperando=0;
   }
   public void esperaFinBusqueda() {
      while (numObrerasEsperando<Sopa.numeroMaximoDeNombres()){
        synchronized(esperandoFin){
          try {
            esperandoFin.wait();// suspende GESTOR
          }catch (InterruptedException e) {}
        }
      }
   }
```

```
   synchronized public void buscaNombre(String elNombre) {
      this.colaNombres.push(elNombre);
      this.notify();
   }
   synchronized public void termina() {
      System.out.println("Terminado");
      for (int i = 0; i < numObrerasEsperando; i++) {
         buscaNombre("");
      }
   };
   synchronized public String esperaNombre() {
      numObrerasEsperando++;
      synchronized(esperandoFin){
         esperandoFin.notify();
      }
      try {
         this.wait();//lock del controlador suspende OBREROS
      }catch (InterruptedException e) {}
      numObrerasEsperando--;
      return colaNombres.pop();
   }
}
```


Notas:

Diseño 4:Código

```
import java.io.*;
import java.util.*;
public class Gestor {
     public static void main(String[] args) {
          Controlador elControlador=new Controlador();
          int numNombres = 1;
          Vector<String> listaNombres=new Vector<String>();
          long startTime;

        for (int i=0;i<Sopa.numeroMaximoDeNombres();i++){
          new ObreroCliente(elControlador);
        }
          try {
               Sopa.cargaSopa(args[0]);
          } catch (IOException e) {
               System.out.println("el fichero "+"args[0]"+" no existe");
          };

          while (numNombres != 0) {
               listaNombres = Sopa.leeNombres();
               startTime = System.currentTimeMillis();
               numNombres = listaNombres.size();
               for (int i = 0; i < numNombres; i++) {
                    elControlador.buscaNombre(listaNombres.elementAt(i));
                    Thread.yield();
               }
          elControlador.esperaFinBusqueda();
               System.out.println("Tiempo de ejecución: "
                         + (System.currentTimeMillis() - startTime) + " ms");
          }
          elControlador.termina();
     }
}
```


Notas: