**PRÁCTICA**

**DE MONITORES:**

**SUPERMERCADO FILA ÚNICA**

**Índice**

1. Introducción
   1. Antecedentes
   2. Entorno de Desarrollo
2. Enunciado Práctico
   1. Explicación del Problema a resolver
3. Solución
   1. Código fuente de la Aplicación e Interfaces
   2. Descripción y justificación de la solución
4. Explicación
   1. Clases
   2. Estructura de datos
5. Listado de pruebas
   1. Entrada
   2. Ejecución
   3. Salida
6. Conclusiones

**1. Introducción**

**a. Antecedentes**

Hay que realizar una práctica para clase para poder obtener los conocimientos necesarios para sacar la asignatura adelante. La práctica consiste en aprender a programar mediante la utilización de monitores.

**b. Entorno de desarrollo**

Windows 10 Pro de 64 bits

Con el programa Eclipse Java Oxygen

**2. Introducción**

1. **Explicación del programa a resolver**

Hay que implementar una aplicación que simule el funcionamiento de N cajas de supermercado.

Los clientes acceden al supermercado y pasan por las cajas situándose en una única cola.

Cuando cualquier caja esté disponible, el primero de la cola será atendido en la caja correspondiente.

Es decir, los clientes compran y se colocan en una única fila, cuando una caja esté libre, dará la orden para que pase el primero de la cola.

Existe un tiempo de compra antes de llegar a la cola y un tiempo de estancia en la caja antes de salir del supermercado.

**3. Solución**

Se mostrará el Código Fuente y a continuación una explicación de dicho código

|  |
| --- |
| **Inicialización** |
| **package** fila;  **import** java.util.Random;  **class** Inicializacion {  **public** **final** **static** **int** ***nPersonas*** = 10;  **public** **final** **static** **int** ***nCajas*** = 4;  **public** **static** **int** *personasQuedan* = ***nPersonas***;  **public** **static** **int** *clienteEsperando* = 0;  **public** **static** **boolean**[] *cajaVacia* = **new** **boolean**[***nCajas***];  **public** **static** **int**[] *personasEnCaja* = **new** **int**[***nCajas***];  **public** **static** **boolean**[] *salidaCaja* = **new** **boolean**[***nPersonas***];  **public** **static** Random *aleatorio* = **new** Random();  } |

En esta primera clase inicializamos todas las variables que vamos a tener.

Declaramos en una constante el número de personas y el número de cajas que vamos a querer.

Inicializamos un contador con el valor del número de personas que vamos a tener.

Inicializamos el contador de clientes esperando a cero.

Declaramos el array para comprobar si una caja está vacía o no mediante booleanos.

Declaramos otro array para saber el número de personas que hay en cada caja, para comprobar si ha entrado más de un cliente a alguna caja.

Declaramos un nuevo array para indicar si una persona ha salido de alguna caja o no.

Declaramos un aleatorio que los utilizaremos en los sleeps.

|  |
| --- |
| **Main** |
| **package** fila;  **public** **class** Main {  **private** **static** Personas *personas*[];  **private** **static** Cajas *caja*[];  **private** **static** Controlador *control* = **new** Controlador();  **private** **static** Metodos *m* = **new** Metodos();  **public** **static** **void** main(String[] args) {  // Inicializo todos los arrays  m.inicializarArrayCajasVacias();  m.inicializarArrayNumClientesEnCaja();  m.inicializarArraySalidaCajas();  // Llamo a todas las cajas.  *caja* = **new** Cajas[Inicializacion.***nCajas***];  **for**(**int** i = 0; i < Inicializacion.***nCajas***; i++) {  *caja*[i] = **new** Cajas(i,*control*);  *caja*[i].start();  }  // Llamo a todas las personas  *personas* = **new** Personas[Inicializacion.***nPersonas***];  **for**(**int** i = 0; i < Inicializacion.***nPersonas***; i++) {  *personas*[i] = **new** Personas(i,*control*);  *personas*[i].start();  **try** { Thread.*sleep*(Inicializacion.*aleatorio*.nextInt(500)+100);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  } |

En la clase **Main**, lo primero que hacemos es declarar las otras clases que vamos a utilizar. Una vez declaradas inicializamos los arrays que hemos declarado antes en la clase **Inicialización**. Para inicializarlas, creamos una clase **Métodos** que más adelante hablaremos de ella.

Hacemos un bucle para crear los hilos de las cajas, cada hilo va a llevar el número de caja que es y la clase **Controlador**.

Para terminar, hacemos lo mismo para crear los hilos de las personas, le vamos a pasar por parámetros el número de personas que es y la clase **Controlador**.

|  |
| --- |
| **Métodos** |
| **package** fila;  **public** **class** Metodos {  **public** **void** inicializarArrayCajasVacias() {  **for**(**int** i = 0; i < Inicializacion.***nCajas***; i++) {  Inicializacion.*cajaVacia*[i] = **true**;  }  }  **public** **void** inicializarArrayNumClientesEnCaja() {  **for**(**int** i = 0; i < Inicializacion.***nCajas***; i++) {  Inicializacion.*personasEnCaja*[i] = 0;  }  }  **public** **void** inicializarArraySalidaCajas() {  **for**(**int** i = 0; i < Inicializacion.***nCajas***; i++) {  Inicializacion.*salidaCaja*[i] = **false**;  }  }  } |

En la clase **Métodos** que la llamamos desde la clase **Main** tenemos un método para inicializar el array de cajas vacías. Este array lo inicializamos en *true* y cuando lo estemos utilizando significará que si está en *false* la caja está ocupada sino es que está libre.

Nos encontramos el método para inicializar el array con el número de clientes en las cajas. Se inicializa a cero y con un contador que más adelante explicaremos, podremos saber si las cajas tienen más de un cliente o no.

Por último tenemos el método que inicializa el array para valores si un cliente ha salido por la caja o no.

|  |
| --- |
| **Cajas** |
| **package** fila;  **class** Cajas **extends** Thread{  **int** cajas;  **int** cliente;  Personas personas[];  Controlador control;  Cajas(**int** i, Controlador controla){  cajas = i;  control = controla;  **this**.setName("Caja " + (i+1));  }  **public** **void** run() {  System.***out***.println("Abriendo la " + **this**.getName());  **while**(Inicializacion.*personasQuedan* > 0) { // Mientras que hay personas dentro del centro comercial  **try** { Thread.*sleep*(Inicializacion.*aleatorio*.nextInt(2300)+1800);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println("\t\tSoy " + **this**.getName() + " y está ocupada o no hay clientes esperando.");  **while**(Inicializacion.*clienteEsperando*>0 && Inicializacion.*cajaVacia*[cajas]) { // Mientras que hay clientes esperando y además esté la caja vacía  **try** { Thread.*sleep*(Inicializacion.*aleatorio*.nextInt(2300)+1800);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println("\t\tSoy " + **this**.getName() + " y hay clientes esperando, voy a llamar a un cliente.");  // Si todas las condiciones se cumplen, desbloqueamos un cliente que está en la cola  control.permitirEntrada();  } // fin while clientes esperando y si esta vacía la caja  } // fin while personasQuedan  **try** {  Thread.*sleep*(2000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println("Cerrando la " + **this**.getName());  }  } |

En esta clase vamos a manejar los hilos de las cajas. Lo primero que hacemos es declarar las variables a utilizar y crear un constructor para los datos pasados por parámetros.

En el método run() indicamos que abrimos las cajas, para saber que hemos llegado correctamente a este método. A continuación, creamos un bucle que se repita mientras que queden clientes dentro del supermercado, hacemos que el hilo se duerma un poco porque sino se nos llena el programa con el siguiente mensaje: “Soy la caja x y no hay clientes esperando o las cajas están ocupadas.” para evitar una aglomeración de dicho mensaje, hacemos que se duerma un tiempo limitado.

Después tenemos un nuevo bucle que se entra en el cuando se cumplen dos condiciones, la primera es que haya clientes esperando y la segunda es que la caja esté vacía. Si alguna de estas dos condiciones no se cumple, no entrará en el bucle.

Al entrar en el bucle, realizamos la misma acción de antes, dormimos el hilo para evitar que en el transcurso de la acción que hay a continuación, entre otro hilo y realice la misma acción a la vez. Esto es para evitar tener muchos mensajes repetidos, al estar bien hecho el programa no necesita dormirlo pero los mensajes se repiten continuamente si no los controlamos de alguna manera.

Después de dormirlos, llamamos a un método de la clase **Controlador** que es el que nos va a permitir que pasen los clientes que están en la cola.

Una vez que ninguna condición de los bucles, significa que ya han pasado todos los clientes y se pueden cerrar las cajas.

|  |
| --- |
| **Personas** |
| **package** fila;  **class** Personas **extends** Thread {  Cajas c[];  Controlador control;  **int** cliente;  **boolean** yaHeEsperadoCola = **false**;  Personas(**int** i, Controlador controla) {  cliente = i;  **this**.control = controla;  **this**.setName("Cliente " + (i+1));  }  **public** **void** entrarCaja(**int** cliente, **int** cajas) {  System.***out***.println("\t\tEntrando cliente "+ (cliente+1) + " en caja " + (cajas+1));  // Indicamos que la caja está ocupada  Inicializacion.*cajaVacia*[cajas] = **false**;  // Contador de personas en caja para comprobar si hay mas de una persona en alguna caja  Inicializacion.*personasEnCaja*[cajas]++;  System.***out***.println("\t\tNúmero de clientes en caja "+ (cajas+1) + ": " + Inicializacion.*personasEnCaja*[cajas] + "");  // Esperamos un tiempo simulando que está pagando y salimos de la caja después  System.***out***.println("\t\tCliente " + (cliente+1) + " pagando.");  **try** { Thread.*sleep*(Inicializacion.*aleatorio*.nextInt(3000)+1500);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println("\t\tSaliendo cliente "+ (cliente+1) + " de caja " + (cajas+1));  // Bajamos el contador de las personas en caja y lo mostramos  Inicializacion.*personasEnCaja*[cajas]--;  System.***out***.println("\t\tNúmero de clientes en caja "+ (cajas+1) + ": " + Inicializacion.*personasEnCaja*[cajas] + "");  // Indicamos que la caja está vacía de nuevo e indicamos que el cliente ha pasado por caja para permitirle salir del supermercado  Inicializacion.*cajaVacia*[cajas] = **true**;  Inicializacion.*salidaCaja*[cliente] = **true**;  System.***out***.println("Clientes esperando: " + Inicializacion.*clienteEsperando*);  }  **public** **void** comprobarSiCajaVacia() {  **int** j = 0;  **boolean** comprobarCajaVacia = **false**;  // Mientras que este alguna caja ocupada se repite el bucle  **while**(!comprobarCajaVacia && j<Inicializacion.***nCajas***) {  // Comprobamos que esté vacía la caja o no.  **if**(Inicializacion.*cajaVacia*[j]) { // Si hay una caja vacía entra en la caja  entrarCaja(cliente,j);  comprobarCajaVacia = **true**;  } **else** { // sino indicamos la caja que estamos chequeando y decimos que está ocupada  System.***out***.println("\t\tSoy el " + (cliente+1) + " y la caja " + (j+1) + " está ocupada.");  }  j++;  } // fin while nCajas  }  **public** **void** run() {  System.***out***.println("Ha llegado el "+ **this**.getName() + " y estoy comprando.\n");  **try** { Thread.*sleep*(Inicializacion.*aleatorio*.nextInt(1500)+500);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  // Hacemos que los clientes después de comprar esperen en la cola  control.esperarCaja(cliente);  // Cuando nos den la orden entrando en el bucle, hasta que no salgamos por alguna caja, no salimos del bucle  **while**(!Inicializacion.*salidaCaja*[cliente]) {  **try** {  Thread.*sleep*(500);  } **catch** (InterruptedException e) {  System.***out***.println("Error en la espera para que no pasen los clientes tan rápido.");  }  // Vamos a comprobar cual de las cajas está vacía  comprobarSiCajaVacia();  }  System.***out***.println("Soy " + **this**.getName() + " Ya he comprado. Me voy del super.");  // Restamos el número de personas que quedan dentro del centro comercial para después poder cerrar las cajas  Inicializacion.*personasQuedan*--;  System.***out***.println("Quedan " + Inicializacion.*personasQuedan* + " personas.");  }  } |

Lo primero que hacemos en esta clase es declarar las variables que vamos a utilizar y crear un constructor para los datos pasados por parámetros.

Al primer método que entramos al crear el hilo es al método run(). En él lo primero que hacemos es indicar que ha llegado el cliente al supermercado. Hacemos que se duerma un tiempo aleatorio el hilo para indicar que el cliente está comprando y así no van en orden a la cola para esperar a las cajas.

Cuando los clientes han comprado, llaman dentro de la clase **Controlador** al método esperarCaja y le paso el cliente que soy.

Una vez que la espera ha finalizado, el proceso vuelve a su hilo principal y entra en un bucle del que no sale hasta que el cliente haya pasado por alguna caja. Dentro de este bucle comprobamos que caja es la que está vacía y cual no.

Nos vamos al método de comprobación y lo primero que hacemos en este método es declarar e inicializar dos variables que solo se utilizarán en este método, uno es el contador de cajas y el otro es un booleano que indica si la caja está vacía o no. A continuación tendremos un bucle en el que se tienen que cumplir dos condiciones, la primera es que la caja debe de estar vacía y la segunda es que el contador no debe de ser mayor al número de cajas, si una de estas dos condiciones no se cumple, se sale del bucle. Dentro de este bucle tenemos una condición, si hay una caja vacía, entra en la caja y al booleano le indicamos que la caja ya no está vacía para después poder salir del bucle sin que chequee las siguientes cajas, para hacer esta comprobación tenemos que utilizar el array de cajas vacías, cuando está a *true* significa que puede entrar, sino es que está ocupada la caja y debe de chequear la siguiente.

Nos vamos a la entrada de la caja al que le hemos pasado por parámetros en cliente y la caja que es. Mostramos un mensaje por pantalla indicando que el cliente ha entrado en la caja e indicamos en el array de cajas vacías que la caja que acaba de entrar el cliente está ocupada. Con el array de contadores de personas en caja, le sumamos uno y mostramos cuantas personas hay por si ha habido un error y han pasado dos personas al mismo tiempo en la caja. Ahora dormimos el hilo un tiempo aleatorio que es el tiempo que el cliente está siendo atendido en la caja.

Una vez pasado este tiempo, mostramos un mensaje de que el cliente está saliendo de la caja y restamos uno al contador y mostramos las personas que hay de nuevo, si todo está bien, en dicha caja debería de haber cero personas en este momento.

Al array de cajas vacías le volvemos a indicar que se ha vaciado la caja y con el array de clientes saliendo de la caja, le indicamos que dicho cliente ha salido por la caja para que pueda abandonar el supermercado.

Como medida de control, hemos sacado por pantalla, nada más salir el cliente por la caja, el número de clientes que están esperando en la cola.

Ya que todas las condiciones se han cumplido para que el proceso salga de los bucles a los que le hemos sometido, mostramos por pantalla que ya ha comprado y que se va del supermercado.

Una vez mostrado dicho mensaje, restamos uno a la variable de personas que quedan en el super y las mostramos por pantalla para un mejor control del programa. Gracias a este contador en negativo, las cajas podrían cerrar sus procesos.

|  |
| --- |
| **Controlador** |
| **package** fila;  **public** **class** Controlador{  // Bloqueamos a los clientes que van a las cajas  **public** **synchronized** **void** esperarCaja(**int** cliente) {  System.***out***.println("\tSoy cliente " + (cliente+1) + " y estoy esperando a que una caja esté libre.");  Inicializacion.*clienteEsperando*++;  **try** {  wait();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  Inicializacion.*clienteEsperando*--;  System.***out***.println("\tSoy cliente " + (cliente+1) + " y acabo de salir de la espera.");  }  // Desbloqueamos a un cliente que está en la cola  **public** **synchronized** **void** permitirEntrada() {  System.***out***.println("\tLlamo a un cliente.");  notify();  }  } |

En esta última clase tenemos el método que es llamado por la clase **Personas**, este método es el de esperar a que las cajas estén disponibles.

Mostramos un mensaje por pantalla indicando que cliente es y que está esperando en la cola, sumamos uno al contador de clientes esperando para poder avisar a las cajas de que hay alguna persona esperando y a continuación llamamos a wait().

Cuando las cajas ven que están esperando, llaman al método permitir entrada a las cajas, indico con un mensaje que llamo a la persona y después llama al notify().

Una vez llamado al notify(), el primer hilo que haya quedado en espera, sale de esa espera, resta uno al contador de personas esperando y muestra por pantalla un mensaje indicando que ha salido.

**4. Explicación**

**a. Clases**

Tenemos una clase principal llamada Main de la cual se hacen las inicializaciones y las llamadas a las otras clases.

Una clase Inicialización que es donde inicializamos todo lo que podemos utilizar en varias clases, hacemos esta clase para tener todas las variables locales juntas.

Una clase método que utilizamos para inicializar los arrays.

Una clase cajas que es donde vamos a manejar los de dicha clase y el control del paso de los clientes a las cajas.

Una clase personas que es donde vamos a manejar los hilos de dicha clase y el control de los clientes.

Una clase controlador para llevar la comunicación entre la espera de las personas y la llamada de las cajas a los clientes.

**b. Estructura**

Como hemos explicado anteriormente se ha estructurado en varios bloques.

El primer bloque hacemos las inicializaciones.

El segundo bloque llamamos a los métodos.

El tercer bloque inicializamos los arrays de los hilos y los lanzamos.

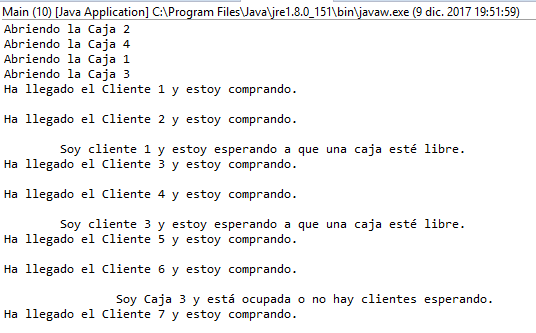
El cuarto bloque manejamos los hilos tanto de cajas como de clientes y dentro de este bloque tendríamos un quinto bloque que controla las clases de cajas y de clientes para poder realizar el ejercicio correctamente.

**5. Listado de pruebas**

**a. Entrada**

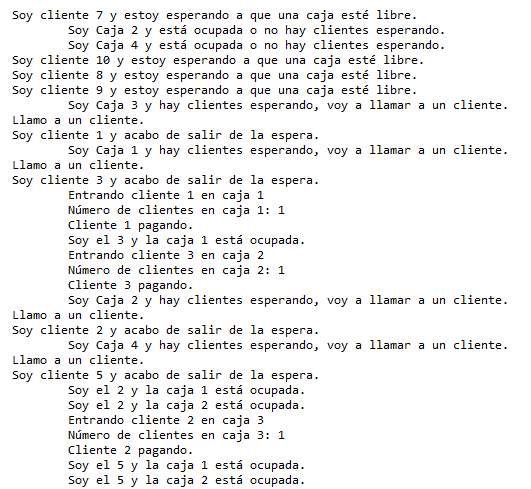
Los datos que tenemos lo escribimos en la clase **Inicialización** fuera de esta clase no pedimos ningún valor.

Nos aparecerá por consola el número de cliente y de caja se está ejecutando.



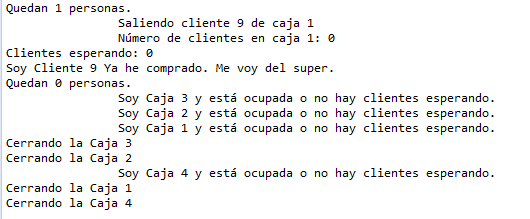
**b. Ejecución**

Después de ver que está todo correctamente funcionando, vemos como se inicializan las cajas y entran los clientes. Los clientes van llegando a la cola y cuando una caja está libre, llama al primer cliente de la cola. Cuando un cliente ha sido llamado, comprueba la caja que está libre y entra en ella.



**c. Salida**

Una vez que los clientes han pasado por las cajas y ya no queda ningún cliente dentro del supermercado, las cajas se cierran.



**6. Conclusiones**

La práctica ha resultado más difícil de lo que parecía. El mayor problema ha sido conseguir implementar correctamente el wait() y el notify() debido a que los hilos se quedaban en espera pero al ir a llamarlos, nunca salían de esa espera, se ha procedido a investigar como solucionar este problema y una vez resuelto, hemos podido acabar correctamente el ejercicio.