PROGRAMACIÓN AVANZADA

Práctica de Laboratorio – PECL3

02317427Q – Víctor Gamonal Sánchez
45579166C – Laura Mambrilla Moreno

UNIVERSIDAD DE ALCALÁ
GRADO EN INGENIERÍA INFORMÁTICA
Curso 2019/20 – Convocatoria Ordinaria

1. ANÁLISIS DE ALTO NIVEL

Este programa consiste en conseguir una simulación de un parque acuático con capacidad de 100 usuarios (que pueden ser niños acompañados de un adulto, niños sin acompañar y adultos) que, a su vez, pueden realizar distintas actividades como la piscina de olas, la piscina grande, el tobogán, las tumbonas o la piscina de niños.

Como mínimo el sistema creará hasta 5.000 usuarios, los cuales irán entrando en orden de llegada teniendo en cuenta que el parque acuático tiene un aforo, como hemos mencionado anteriormente, de hasta 100 usuarios de forma simultánea. En caso de alcanzar este aforo, los usuarios esperarán en una cola de entrada hasta que se permita la entrada de alguno/s de ellos.

Cabe destacar la importancia de los monitores, que son personas que se encargarán individualmente de llevar el control de cada una de las actividades del parque acuático, de tal forma que comprueben que los usuarios reúnen los requisitos para realizar dicha actividad (véase que tengan la edad mínima de uso, que vayan acompañados si así lo requiere, etc.).

Al entrar al parque acuático, los usuarios deberán pasar primero por un vestuario con una capacidad de 30 personas: 20 adultos y 10 niños. los niños de entre 1 y 10 años deberán hacerlo siempre acompañados de un adulto (es decir, de una persona mayor de edad) ocupando ambos dos huecos del espacio para niños, mientras que el resto de usuarios podrán hacerlo por su propia cuenta, ocupando un único espacio en su respectiva categoría. La función del monitor aquí será comprobar las edades de los usuarios.

Una vez salen del vestuario, los usuarios tienen libre albedrío para elegir en qué actividad desean participar, con un mínimo de 5 actividades y hasta un máximo de 15 (siendo escogido este número de forma aleatoria para cada usuario y su acompañante, si lo tuviere).

Si las actividades a las que quieren acceder permiten su entrada (por control de aforo, edad, etc.), entonces el usuario pasa y no espera. Por el contrario, si por ejemplo estuviesen completas por otros usuarios, éste tendrá que esperar al final de una cola de entrada, cuyo funcionamiento varía entre las distintas actividades.

Una vez que el usuario ha cubierto el máximo de actividades que puede realizar, éste abandona el parque acuático, dejando así hueco libre para que el siguiente usuario que esté esperando en la cola de entrada del parque acuático pueda realizar el mismo procedimiento y disfrutar de un lindo día de vacaciones.

Quien desee hacer uso y disfrute de esta API, podrá interactuar con el sistema de varias formas distintas y podrá:

Consultar la ubicación y número de actividades a las que ha accedido un usuario
Realizándose mediante la introducción del ID del usuario del que se desee recabar la información.
cu_boton cu_usuario cu_ubicacion cu_actividades

Consultar el número de usuarios en cada zona
Mediante el uso de un botón "Buscar".
ca_boton ca_v ca_pn ca_po ca_pg ca_tu ca_to

Consultar el número total de usuarios menores de edad
Mediante el uso de un botón "Buscar".
cm_boton cm_numMenores

Consultar el número de usuarios que ha utilizado cada uno de los toboganes
Mediante el uso de un botón "Buscar".
ct_boton ct_tA ct_tB ct_tC

Además, se ha instalado un sistema distribuido a través del cual cuando se inicia el parque acuático se inicia consigo un servidor multi-cliente. A través de la interfaz Cliente, se puede abrir un módulo de control que se conecta al servidor y da las opciones mostradas anteriormente.

2. DISEÑO GENERAL DEL SISTEMA Y DE LAS HERRAMIENTAS DE SINCRONIZACIÓN UTILIZADAS

El sistema cuenta con un diseño concurrente a través del cual múltiples tareas son realizadas a la vez.

La clase nombrada como Aquapark es el recurso compartido del sistema, es decir, todos los hilos acceden a dicha clase para realizar su función principal.

El hilo que representa a las personas tiene el nombre de Usuario. Los usuarios son creados e inicializados en la función crearUsuarios del programa. Una vez creados, los usuarios tienen que acceder al parque acuático, y es aquí donde entran en juego las herramientas de comunicación y sincronización necesarias para que la simulación sea correcta y eficiente.

La principal herramienta de comunicación utilizada durante la creación del sistema han sido los cerrojos (locks).

Para explicar el funcionamiento general y diseño del sistema, explicaremos las funciones que realizan los hilos Usuario y Monitor, que son los hilos principales del parque acuático.

1. USUARIO

Cuando el usuario entra en el parque acuático y por lo tanto al vestuario, se encuentra con un cerrojo con el que se consigue exclusión mutua en la zona del código donde solo se desea que un hilo pueda acceder a la vez.

El cerrojo, además del uso de conditions, se utiliza con el objetivo de que, a la hora de actualizar la interfaz del sistema, no haya problemas de sincronización y los usuarios sean incluidos en la interfaz en el mismo orden en el que llegaron al parque acuático.

Tras esto, los usuarios entrarán digitalmente hablando en un bloque de código en el que se comprueba que el contador actual de usuarios dentro del vestuario no iguale el aforo permitido. En caso de que esto ocurriese, los usuarios entrarán en una cola de entrada esperando y comprobando en todo momento que puedan entrar finalmente al vestuario.

Una vez salgan del vestuario y puedan de una vez iniciar su maravilloso día en el parque acuático, elegirán una actividad escogiendo un número aleatorio entre 0 y 7, donde cada uno de ellos corresponde a una posición del arrayList en el que se encuentra cada una de las actividades (dividiéndose Tobogán en 3 tipos), y así siempre que salgan de una actividad para ir a otra, realizando entre 5 y 15 actividades como máximo durante su estancia en el parque acuático.

Piscina de Olas, Piscina de Niños, Piscina Grande, Tobogán

Toda la comunicación y sincronización se ha llevado a cabo mediante el uso de un cerrojo heredado de una clase padre, Actividad, donde ha sido previamente definido; funciona de tal manera que al intentar entrar en la actividad, se cierra el cerrojo y se comprueba que pueda entrar, protegiendo así por exclusión mutua este bloque de código. Una vez realizada esta acción, el cerrojo de desbloquea permitiendo así que el resto de hilos puedan realizar esta misma ejecución.

Tumbona

En el caso de la tumbona, la comunicación y sincronización se realiza mediante el uso de semáforos, ya que el acceso a esta actividad, véase, desde la cola, no tiene ningún tipo de restricción FIFO o similares. Más concretamente, hemos utilizado semáforos no justos para simularlo de la forma más eficiente posible.

Cuando el usuario entra en la tumbona, el semáforo tumbonaLlena, el cual está inicializado a 0, actúa como condición. Se realiza una llamada a tumbonaLlena.acquire(), que resta en 1 de hasta un total 20 los permisos con los que cuenta este semáforo, coincidiendo con el

aforo máximo de la actividad y controlando así que éste nunca se sobrepasa. También cabe destacar la llamada a colaEntrada.release(), que también hace que se atribuya un permiso más al semáforo que controla la cola de entrada a la actividad, por si este usuario hubiese entrado después de haber estado esperando en ella.

Cabe destacar que, en el caso de que el aforo estuviese completo, esta afluencia de personas se controla mediante una llamada a colaEntrada.acquire() para indicar así que se añade en uno los usuarios que actualmente están esperando en la cola.

Anexo de las actividades

Es importante mencionar que además de todo este proceso de sincronización/comunicación del usuario en el parque acuático, también este hace llamadas a funciones relacionadas con la interfaz para actualizarlos y completar la simulación con información visual.

2. MONITOR

En cuanto a los monitores, se puede decir que su finalidad (que es, repetimos, controlar que un usuario reúna condiciones necesarias para poder entrar en una actividad), se controla y sincroniza en cada actividad de igual manera que se hace con los usuarios; es decir:

Piscina de Olas, Piscina de Niños, Piscina Grande, Tobogán, Vestuario

El proceso de control se realiza mediante el uso de cerrojos.

Tumbona

Además de los semáforos mencionados anteriormente para el Usuario, también hemos procedido a la creación de un semáforo de control, “control”, con un único permiso y de tipo justo para que este procedimiento se lleve a cabo de forma individual para cada uno de los usuarios; una vez se haya completado esta acción y se decida si éste es apto o no para entrar en la actividad, llamamos a la función control.release() devolviéndole así el permiso para que otro hilo pueda entrar.

¿Cómo se termina la ejecución de los hilos?

Los hilos que son de la clase Usuario finalizan cuando salen del parque acuático, ya que su método run() consiste en entrar al parque acuático, entrar y salir del vestuario, disfrutar de entre 5 y 15 actividades, volver al vestuario a cambiarse (de nuevo, entrar y salir) y, finalmente, salir del parque acuático.

Los hilos de la clase Monitor terminan cuando el parque acuático cierra, es decir, cuando el último de los 5000 usuarios sale del parque, haciendo que el atributo (boolean) abierto de la clase Aquapark pase a false.

3. CLASES PRINCIPALES, ATRIBUTOS Y MÉTODOS

El sistema consta de 14 clases, dos interfaces y un programa principal. En este apartado se explicará detalladamente la función que tiene cada una de ellas en el sistema.

1. Clase Aquapark

La clase Aquapark va a contener todas las actividades del parque acuático (vestuario, piscina de niños, piscina de olas, piscina grande, tumbona y los tres tipos de toboganes), por lo que es la clase intermediaria entre actividades, usuarios y monitores.

Los hilos de la clase Monitor dependen del atributo boolean abierto, ya que funcionan mientras este sea true. Se pondrá a false cuando hayan salido los 5000 usuarios del parque acuático.

El aforo del parque acuático es de 100 usuarios, por lo que habrá un control que permitirá acceder o no al usuario al parque acuático dependiendo de si el aforo está completo.

Atributos más importantes:

- int contador_total: Para llevar la cuenta de cuántas personas hay en el parque acuático
- boolean abierto: Para llevar el control de que puedan entrar más usuarios o no.
- ArrayList<Usuario> lugar: Que sirve para almacenar los usuarios en las actividades.
- ArrayList<Usuario> colaEspera: Que sirve para almacenar los usuarios en las colas de espera de ciertas actividades.
- ArrayList<Actividad> actividades: Que sirve para almacenar los objetos Actividad.
- Relación de asociación con cada una de las actividades
- Lock cerrojo: Para la sincronización de la entrada y salida de los usuarios
- Condition aquaparkVacio/aquaparkLleno

Métodos más importantes:

- entrar (Usuario usuario): Para que el usuario entre al aquapark.
- meterColaEspera(Usuario usuario): Para meter en la cola de espera del aquapark al usuario.
- sacarColaEspera(): Para sacar al usuario de la cola de espera del aquapark, siempre y cuando se reunan condiciones necesarias como que el aforo no esté completo.
- meterAquapark(Usuario usuario): Para introducir al usuario en el arrayList de lugar, por así decirlo, de forma “física”, sumando en uno el contador total de personas en el aquapark, y en dos en caso de que el usuario vaya acompañado.
- salir(Usuario usuario): Siempre y cuando quede alguien en el aquapark, se podrá sacar a un usuario.
- sacarAquapark(Usuario usuario): Para sacar al usuario del arrayList de lugar.
- imprimir(): Para imprimir en todo momento los usuarios dentro del aquapark.
- imprimirColaEspera(): Para imprimir en todo momento los usuarios que están en la cola de espera del aquapark.

    - Getter y Setter correspondientes a cada uno de los ATOs.
- Las funciones siguientes se utilizan exclusivamente para la segunda parte de la práctica, con las que accederemos de manera remota a lo que retornan cada uno de los siguientes métodos:
    - devolverActividad(String nombre): devuelve el nombre de la actividad en la que se encuentra en ese momento el usuario que se ha insertado por la interfaz Aquapark2.
    - devolverNumActividad(String nombre): devuelve el número de actividades que el usuario introducido por la interfaz Aquapak2 ha hecho hasta el momento.
    - controlToboganes(String nombre): devuelve el número total de personas que han pasado por cada tobogán hasta el momento.
    - controlAforo(String nombre): devuleve el número de personas que están en ese momento dentro de cada actividad.

2. Clase Persona

La clase Persona extiende de Thread y es la clase padre de Monitor y Usuario, clases las cuales heredarán atributos en común como son el ID y la edad (de tipo integer), y un string ident, el cual es una combinación de las anteriores y que nos permite identificar a cada uno de los distintos hilos del programa; todos los atributos tienen sus correspondientes getter y setter para permitir acceder a ellos de forma externa.

Atributos más importantes:

- int id: Como número identificativo del usuario.
- int edad
- String ident: Concatenación de id + edad del usuario

Métodos más importantes:

- Getter y Setter correspondientes a cada uno de los ATOs.

3. Clase Usuario

La clase Usuario es un hilo y clase hija de Persona, que realizará la función de entrar al parque acuático con la intención de pasárselo bien realizando actividades.

El usuario tiene atributos necesarios para el correcto funcionamiento del sistema, como son booleanos adulto y acom, que indica si es adulto y está acompañado por alguien; un int numActividades que establece el total de actividades que realizará ese usuario; y también relaciones de asociación con Actividad, Aquapark y el propio Usuario, por si éste pudiese estar acompañado.

El método entrarActividad permite seleccionar en qué actividad entrará el usuario cada vez que quiera hacerlo, en el que se controla el tiempo que tardará en entrar a cada una de ellas y que se realiza de forma aleatoria haciendo uso de un método externo elegirActividad() que devuelve una posición de arrayList donde se encuentran cada una de ellas.

Atributos más importantes:

- boolean apto: Para comprobar si un usuario es apto para entrar en una actividad.
- int contAptos: Contador de cuántas actividades de las totales ha realizado la el usuario
- boolean adulto: Si es adulto o niño
- boolean acom: Si está acompañado o no
- int numActividades: Número de actividades, aleatorio entre 5 y 15, que realizará el usuario dentro del aquapark.

Métodos más importantes:

- Actividad elegirActividad(): Mediante un Math.random, permite escoger una actividad al azar del arrayList "actividades" previamente definido en Aquapark, que será a la cual acceda el usuario en ese momento.
- void entrarActividad(Actividad actividad): Se comprueba la actividad escogida en el método anterior para meter al usuario en ella.
- void tiempoAleatorio(int t_min, int t_max): Para establecer un tiempo aleatorio entre t_min y t_max que coincidirá con el tiempo que durará el usuario en una actividad determinada.
- void run(): Ejecución del hilo, para acceder a los métodos "entrar" y "salir" del Aquapark y el Vestuario, además de controlar que entre en cada una de las actividades para las que sea apto siempre y cuando contAptos != numActividades.
- Getter y Setter correspondientes a cada uno de los ATOs.

4. Clase Monitor

La clase Monitor es un hilo y clase hija de Persona, que realizará la función de controlar la entrada de los usuarios a las distintas actividades ofrecidas.

Guarda atributos heredados de su clase padre como el ID y la edad, y una relación de asociación con Actividad para establecer de cuál de ellas será responsable.

Los distintos métodos implementados en ella (controlarVestuario, controlarPisciOlas, etc.) controlan, como su propio nombre indica, que el usuario que desea entrar a la actividad correspondiente cumpla los requisitos necesarios para cada una de ellas y el tiempo que tarda en realizar esta comprobación.

Atributos más importantes:

- int id, int edad: Heredados de su clase padre (Persona)
- Relación de asociación con Aquapark y Actividad

Métodos más importantes :

- boolean controlarVestuario(Usuario usuario): Para controlar que un usuario sea apto para entrar en el vestuario.

- boolean controlarPisciOlasEdad(Usuario usuario): Para controlar que un usuario sea apto para entrar en la piscina de olas
- int echarPisciGrande(PiscinaGrande pisciGrande): Para sacar de a piscina grande a un usuario aleatorio cada cierto tiempo.
- void run(): Ejecución del hilo

5. Clase Actividad

La clase Actividad establece los TAD necesarios para el correcto funcionamiento y control de las actividades:

Atributos más importantes:

int aforo → Para definir la capacidad máxima de la actividad

ArrayList<Usuario> colaEspera → Para implementar una cola de entrada en las actividades

ArrayList<Usuario> colaEsperaTablero → En este caso, una cola bidimensional usada para Piscina de Olas.

ArrayList<Usuario> lugar → Para saber qué hueco ocupa un usuario en una actividad

ArrayList<Usuario> lugarTablero → En este caso, el lugar de "forma física", tratándolo como una matriz bidimensional

ArrayList<ArrayList> colaEsperaVestuario/colaEsperaPisciOlas: Para insertar a los usuarios en la cola de espera de la actividad correspondiente, de tipo ArrayList por si los usuarios estuvieran acompañados.

ArrayList<Usuario> colaEsperaTumbona/colaEsperaX: Para insertar a los usuarios en la cola de espera de la actividad correspondiente, solo que en este caso, de forma individual

Lock cerrojo → Para permitir la sincronización y comunicación de las distintas actividades a la hora de permitir la entrada al usuario.

Esta clase hereda sus atributos y métodos a sus clases hijas, que son: PiscinaOlas, PiscinaNinios, PiscinaGrande, Tobogan, Tumbona y Vestuario.

Métodos más importantes:

- Getter y Setter correspondientes a cada uno de los ATOs

6. Clase PiscinaOlas

La clase PiscinaOlas es una de las clases hijas de Actividad.

A esta clase no pueden acceder los usuarios menores de 6 años y los menores de 11 años deben ir con acompañados por un adulto.

A la cola de espera de piscina olas se accede de manera individual (a no ser que el usuario tenga menos de 11 años y se encuentre en la situación explicada anteriormente: que debe ir con el adulto), pero a la piscina se entra por parejas. A través del método public boolean

comprobarEnPisciGrande(Usuario usuario), al cual se le llama desde la clase Usuario, se le da el poder al usuario que ha completado la pareja (es decir, la persona “número 2” de la pareja) de decidir el tiempo que van a estar dentro y de cuándo saldrán.

Atributos más importantes:

- **ArrayList<ArrayList> colaEsperaParejasFormadas:** Para insertar las parejas que ya estén formadas y listas para entrar en la actividad en la cola de espera.
- **ArrayList<Usuario> colaEsperaParejaSinFormar:** Para insertar al usuario que no tenga pareja con la que entrar a la actividad, y formar posteriormente una pareja con otro usuario en su misma situación.

Métodos más importantes:

- void entrar(Usuario usuario)
- void salir (Usuario usuario)
- void elegirParejas(Usuario usuario): Para formar las parejas que pasarán a la cola de espera de la actividad y, posteriormente, a la actividad. En caso de que el usuario venga acompañado, directamente entrará a esa cola de espera de parejas formadas; en caso contrario, entrará a la cola de espera de parejas sin formar, esperando a otro usuario en su misma situación y poder formar pareja, o para formarla con alguien que ya esté esperando.
- void meterPareja(ArrayList<Usuario> pareja): Para insertar la primera pareja que esté esperando en la cola de espera, si el aforo lo permitiese, en la actividad.
- boolean comprobarEnPisciGrande(Usuario usuario): Da el poder de elegir durante cuánto tiempo y hasta cuándo permanecerá la pareja en la piscina de olas al usuario que ha acabado de formar la pareja. Este método es llamado en la clase Usuario.

7. Clase PiscinaNinios

Es una clase hija de Actividad.

Es una clase restringida para los usuarios mayores de 10 años (sin contar acompañantes que). El acompañante, si el niño tiene menos de 6 años, debe entrar a la piscina, mientras que si el niño tiene 6 o más años, el acompañante debe esperar en una cola reservada a adultos con esta situación, la cual abandonará cuando el niño abandone la piscina. El acceso del niño (y, si es el caso, del acompañante) a la piscina y el del acompañantes a la cola de espera de su condición, se maneja en el método: private void meterPiscina(Usuario usuario).

Atributos más importantes:

- Condition piscinaLlena/piscinaVacia
- ArrayList<Usuario> esperaPadres: Para almacenar a los acompañantes de los usuarios de entre 6 y 10 años que entren a la actividad.

Métodos:

- meterColaEspera(Usuario usuario): Para meter en la cola de espera al usuario

- sacarColaEspera(): Para sacar de la cola de espera al primer usuario de la cola de espera.
- meterPiscina(Usuario usuario): Para meter al usuario y a su posible acompañante, si lo requiere, dentro de la actividad.
- sacarPiscina(Usuario usuario): Para sacar al usuario y a su posible acompañante, si lo tuviere, de la actividad buscándole de forma iterativa en la lista.

8. Clase PiscinaGrande

Se trata de una clase hija de Actividad, de la cual cogerá los atributos para almacenar a los usuarios que entran en ella y para controlar el número de ellos.

En PiscinaGrande se han establecido los TAD (todos private) necesarios para la ejecución del programa y la formación de la propia clase, entre ellos los jTextField que mostrarán en la interfaz la evolución de almacenamiento en la propia piscina de olas y en su cola de espera, los Conditions que ayudan al control y un int que indica el número de usuarios que están dentro de la piscina.

La clase PiscinaGrande difiere de las demás en que no tiene un límite de edad para acceder a ella, es decir, puede acceder a ella cualquier usuario.

Su inserción puede hacerse de dos formas:

1. a través de la cola de espera de PiscinaGrande(todos los usuarios). Previamente, al usuario le ha debido de conceder el paso el correspondiente monitor.
2. a través de los toboganes (solo los usuarios que puedan tirarse por ellos), ya que desembocan en la PiscinaGrande (método: public void entrarPorTobogan (Usuario usuario)). Una vez dentro, siendo ya usuarios de PiscinaGrande, disfrutarán del servicio en ella.

Debido al doble acceso, las clases Tobogan y PiscinaGrande deben estar vinculadas, ya que si se completa el aforo de PiscinaGrande, los toboganes pararán hasta que vuelva a haber un menor número de usuarios dentro de esta actividad.

Atributos más importantes:

- Condition piscinaVacia/piscinaLlena
- Condition toboganParado/toboganActivado: Que controlan si se puede acceder a la actividad a través de los toboganes.

Métodos:

- meterColaEspera(Usuario usuario)
- sacarColaEspera()
- meterPiscinaGrande(Usuario usuario)

- entrarPorTobogan(Usuario usuario): Para introducir al usuario que se tire por el tobogán a la piscina.

9. Clase Tobogan
En el programa habrá 3 objetos de la clase Tobogan. Cada uno admitirá usuarios dependiendo de su edad.

Esta clase también es hija de Actividad, por lo que utilizará sus atributos controlar y almacenar.

Como hemos explicado en PiscinaGrande, los usuarios que entran a los toboganes, salen directamente a la piscina grande (a través del método: private void entrarEnPiscinaGrande(Usuario usuario)), donde pasarán a formar parte de sus usuarios.

Atributos:
Métodos:

10. Clase Tumbona
Clase hija de Actividad.

A esta clase no pueden acceder los usuarios menores de 15 años.

Esta clase dispone de un espacio pensado para 20 usuarios. Si está completo, los que deseen acceder deben esperar en la cola de espera, pero una vez haya un espacio libre no entrarán por el método FIFO (First In First Out), sino que será "el que ande más rápido", utilizando para ello semáforos no justos:

→ private final Semaphore tumbonaLlena(20): Con un total de hasta 20 permisos, que coinciden así con la capacidad de la actividad.

→ private final Semaphore control(1, true): Para controlar que el monitor comprueba de uno en uno si un usuario es apto para entrar en la actividad (exclusión mutua), y lo harán en orden por ser justo.

→ private final Semaphore tumbonaVacia(0)

→ private final Semaphore colaEntradaTumbona

Métodos más importantes:

- meterTumbona(Usuario usuario)
- meterColaEspera(Usuario usuario)
- sacarColaEspera(Usuario usuario)

11. Clase Vestuario
La clase Vestuario es la última clase de las que hablaremos que es hija de Actividad.

El usuario pasará dos veces por el vestuario: una vez nada más entrar al parque acuático y otras justo antes de salir de él. Ya que hay distintos aforos para niños (con sus acompañantes, si es que tuvieren) y para adultos, tiene dos colas distintas en función de la edad, aunque se las podrían saltar en caso de que hubiera vestuarios libres.

Vestuario no es una actividad como tal, por lo que el paso del usuario por él no le restará una actividad de entre las 5 y 15 que puede realizar.

Atributos más importantes:

- ArrayList<Usuario> colaNinios/colaAdultos: Para meter a los usuarios en su cola de espera correspondiente.
- int aforoNinios/aforoAdultos: Para controlar el aforo máximo de cada uno de ellos.
- int contNinios, contAdultos: Para controlar cuántos niños y adultos hay dentro de la actividad.
- Condition ninioVacio/ninioLleno
- Condition adultoVacio/adultoLleno

Métodos más importantes:

- meterVestuario(Usuario usuario)
- meterColaEspera(Usuario usuario)
- comprobarNoEnCola(Usuario usuario): Para comprobar que un usuario no esté ya en la cola para evitar volver a meterle.
- sacarColaEspera(Usuario usuario)
- sacarVestuario(Usuario usuario)

12. Clase Métodos

Esta clase hace de puente entre los métodos requeridos de la clase Aquapark y la interfaz de Aquapark2.

13. Aquapark1

En esta clase encontramos la interfaz 1.

Aquí se encuentran inicializados todos los objetos.

Al pulsar sobre el botón AQUAPARK los hilos de los monitores empiezan a ejecutarse y se crean y ejecutan los usuarios.

En los jTextFields se imprimen los usuarios que están en cada actividad y en sus respectivas colas de espera.

El boton AQUAPARK y los jTextFields se han moldeado a nuestro gusto en el método transparencia().

Nota: hemos tenido un constante error con la imagen del fondo, el cual no hemos podido solventar pese a haber intentado todo lo posible. Esto ha hecho que no hayamos podido comprobar el funcionamiento del programa.

14. Aquapark2

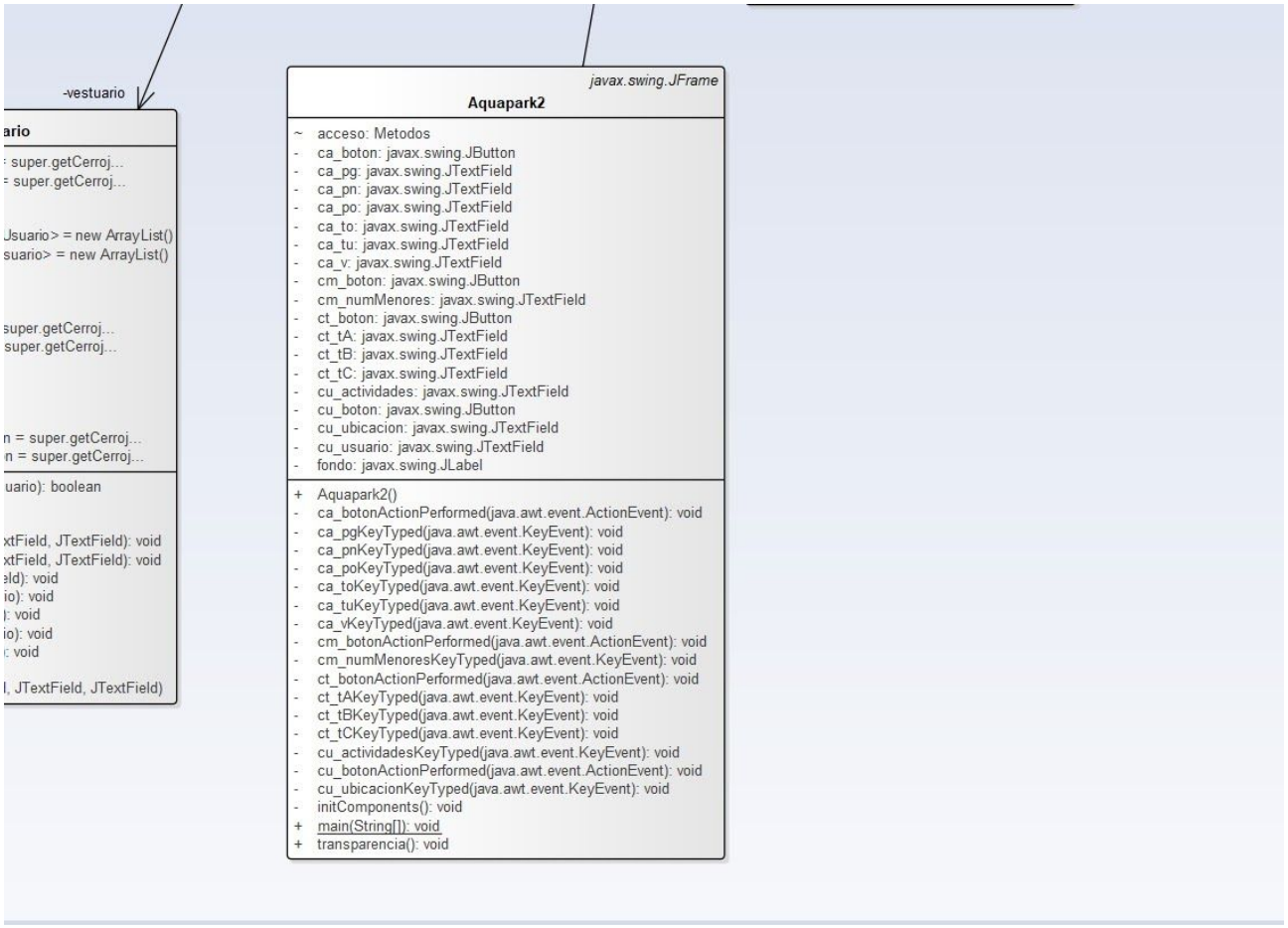
En esta clase encontramos la segunda interfaz, la que accede de forma remota a ciertos métodos de la clase Aquapark a través de Métodos.

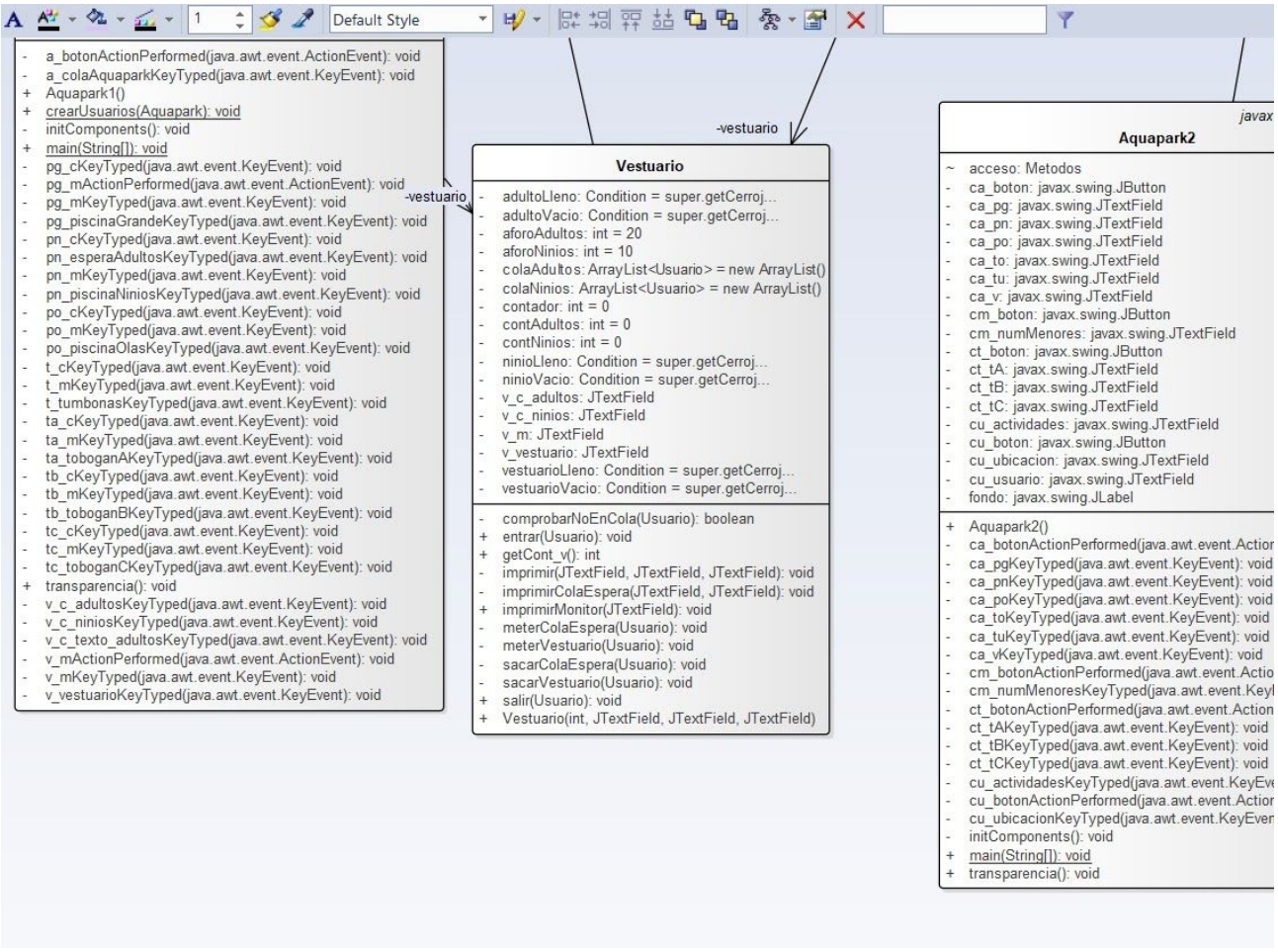
Los jTextFields se han moldeado a nuestro gusto en el método transparencia().

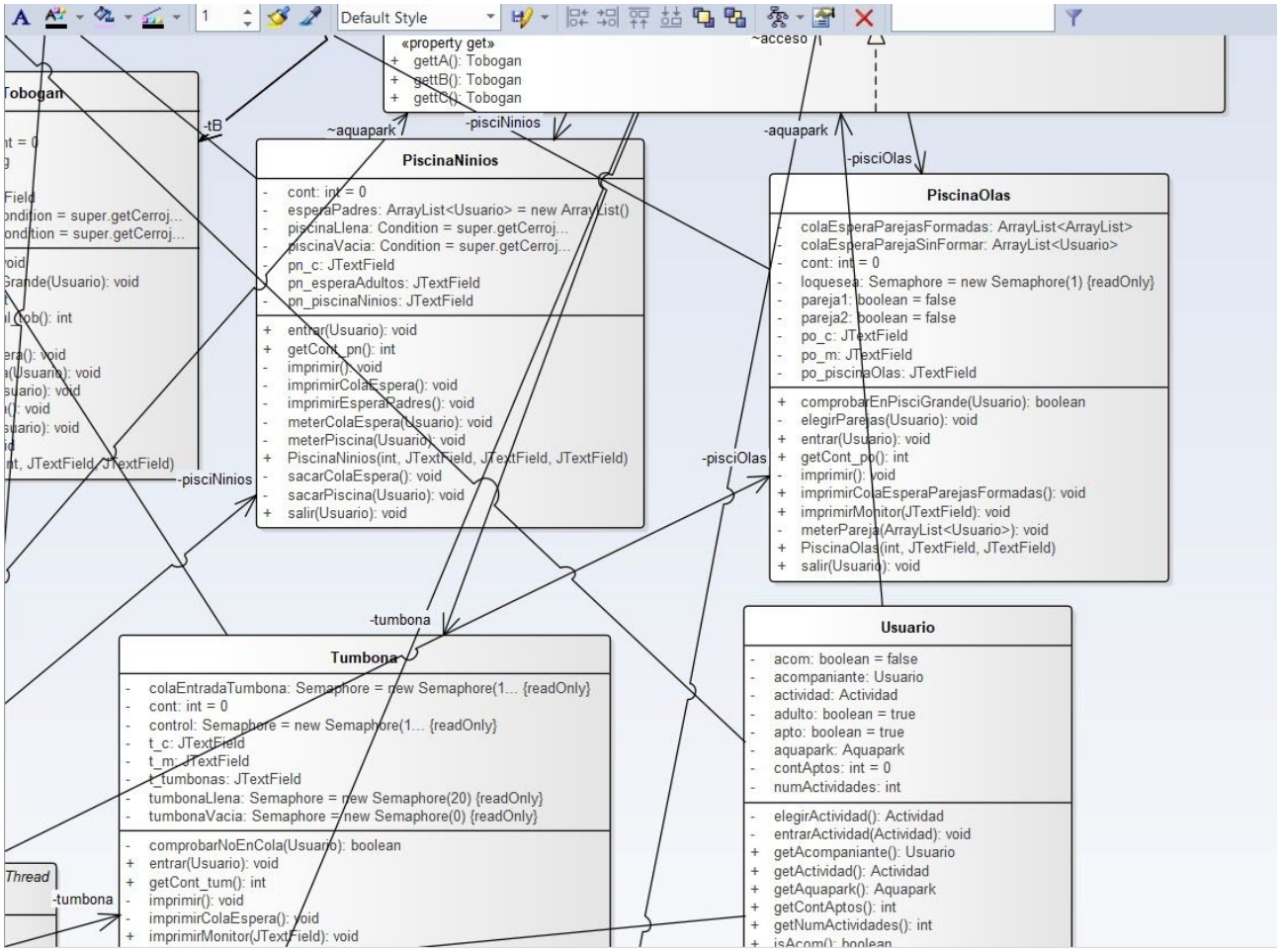
Nota: hemos tenido un constante error con la imagen del fondo, el cual no hemos podido solventar pese a haber intentado todo lo posible. Esto ha hecho que no hayamos podido comprobar el funcionamiento del programa.

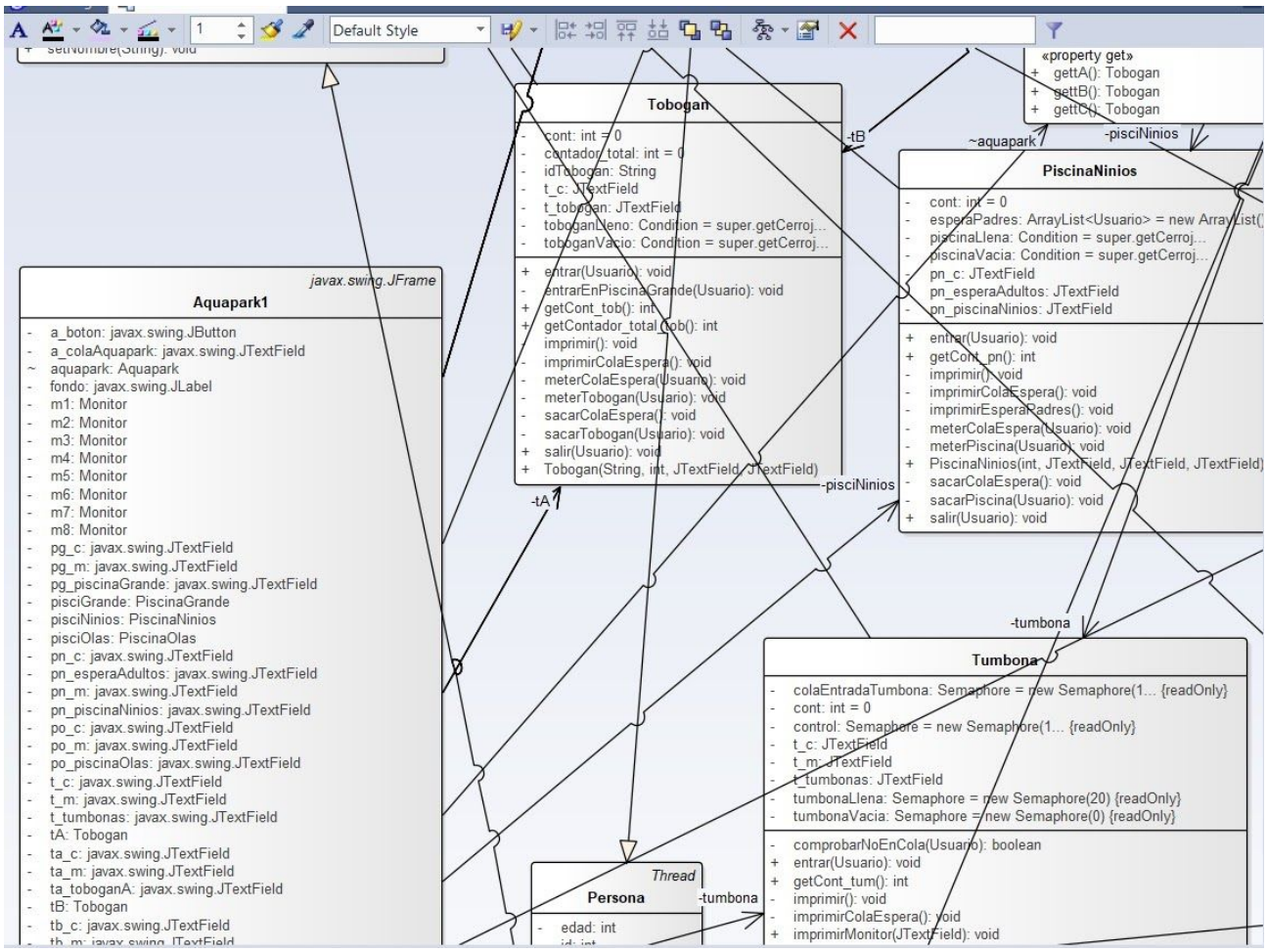
4. DIAGRAMA DE CLASES

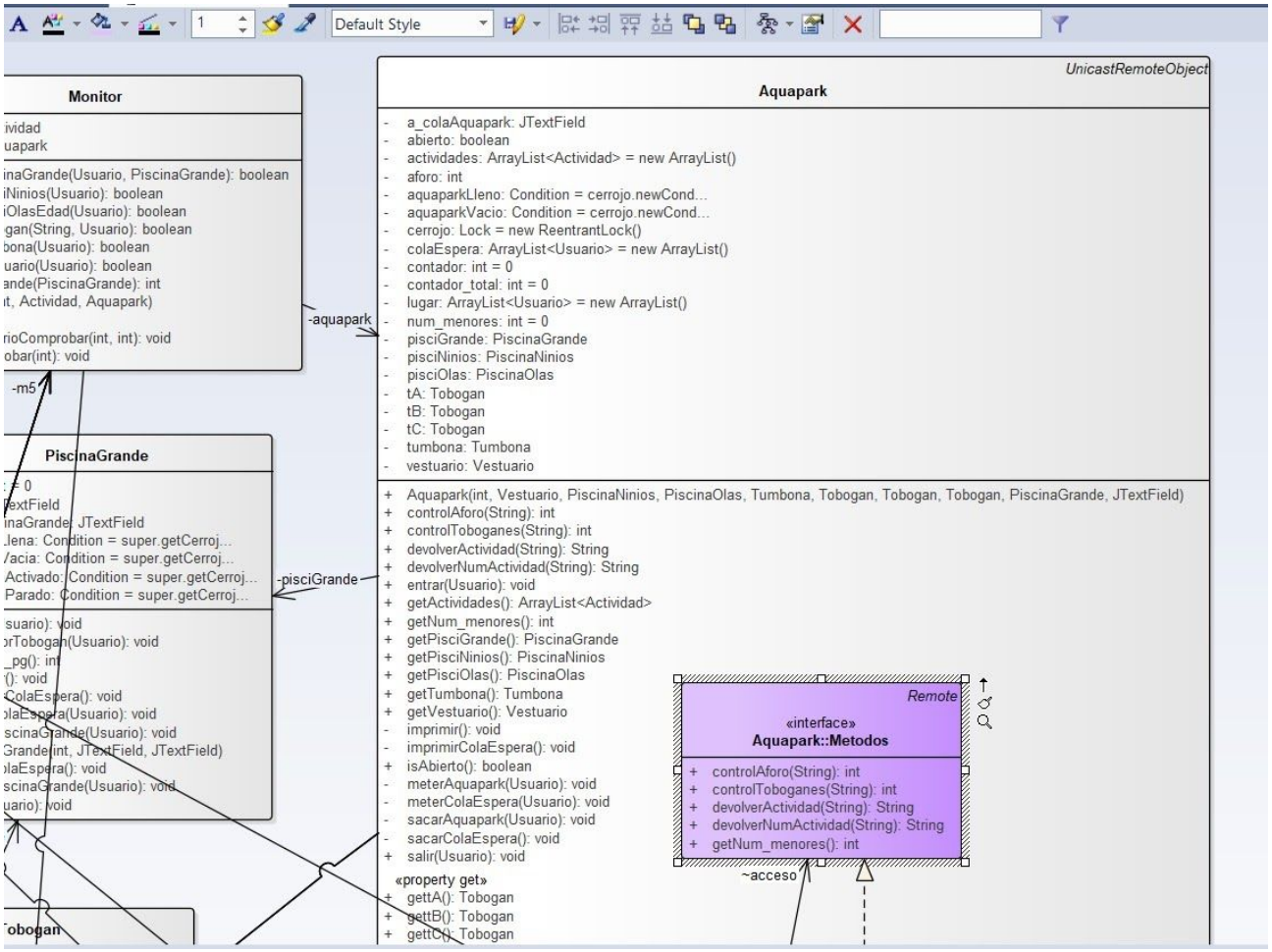
Se adjunta archivo .EAP para comprobarlo con mejor detalle.

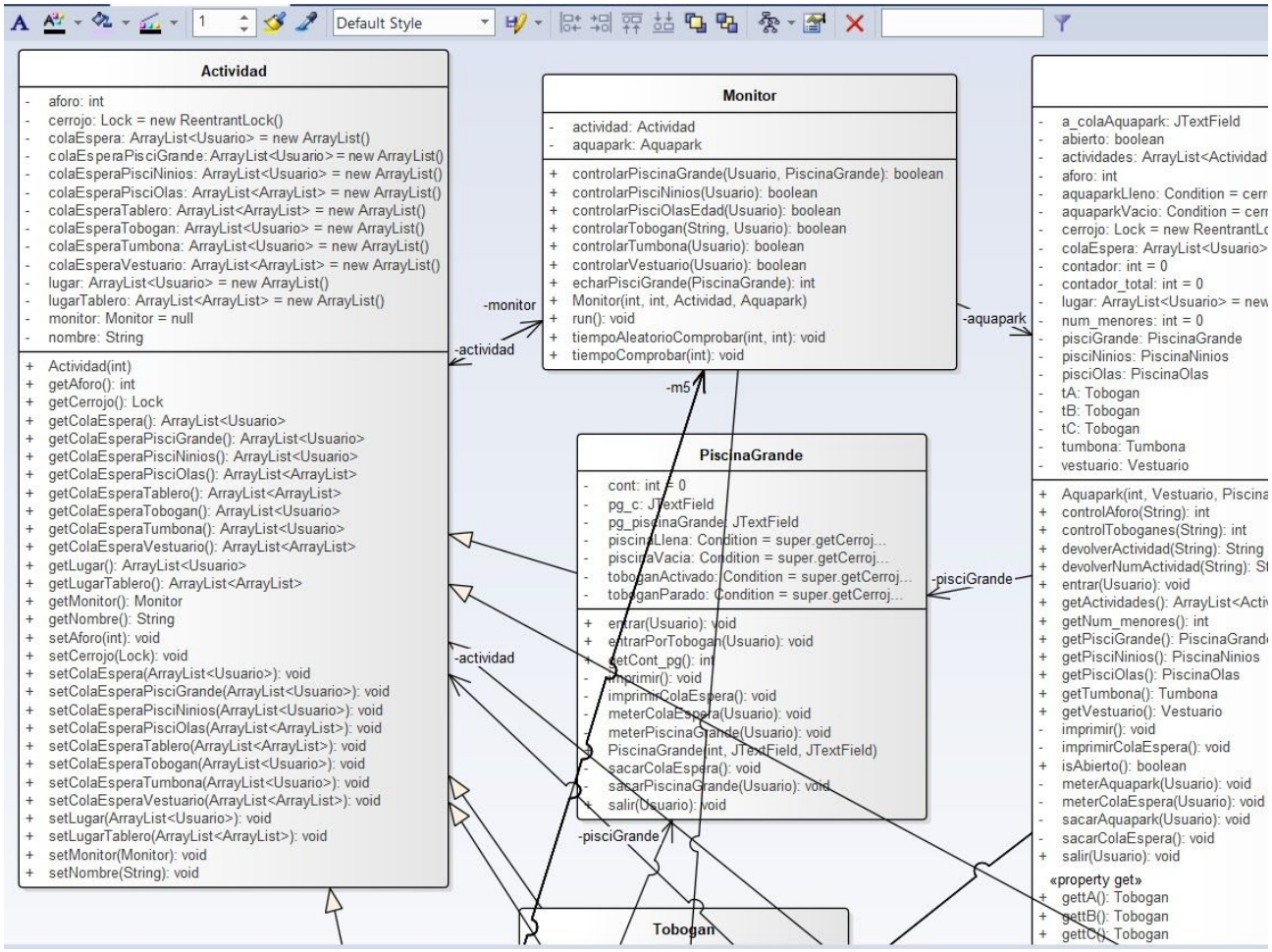

-vestuario
javax.swing.JFrame
Aquapark2
~ acceso: Metodos
- ca_boton: javax.swing.JButton
- ca_pg: javax.swing.JTextField
- ca_pn: javax.swing.JTextField
- ca_po: javax.swing.JTextField
- ca_to: javax.swing.JTextField
- ca_tu: javax.swing.JTextField
- ca_v: javax.swing.JTextField
- cm_boton: javax.swing.JButton
- cm_numMenores: javax.swing.JTextField
- ct_boton: javax.swing.JButton
- ct_tA: javax.swing.JTextField
- ct_tB: javax.swing.JTextField
- ct_tC: javax.swing.JTextField
- cu_actividades: javax.swing.JTextField
- cu_boton: javax.swing.JButton
- cu_ubicacion: javax.swing.JTextField
- cu_usuario: javax.swing.JTextField
- fondo: javax.swing.JLabel
+ Aquapark2()
- ca_botonActionPerformed(java.awt.event.ActionEvent): void
- ca_pgKeyTyped(java.awt.event.KeyEvent): void
- ca_pnKeyTyped(java.awt.event.KeyEvent): void
- ca_poKeyTyped(java.awt.event.KeyEvent): void
- ca_toKeyTyped(java.awt.event.KeyEvent): void
- ca_tuKeyTyped(java.awt.event.KeyEvent): void
- ca_vKeyTyped(java.awt.event.KeyEvent): void
- cm_botonActionPerformed(java.awt.event.ActionEvent): void
- cm_numMenoresKeyTyped(java.awt.event.KeyEvent): void
- ct_botonActionPerformed(java.awt.event.ActionEvent): void
- ct_tAKeyTyped(java.awt.event.KeyEvent): void
- ct_tBKeyTyped(java.awt.event.KeyEvent): void
- ct_tCKeyTyped(java.awt.event.KeyEvent): void
- cu_actividadesKeyTyped(java.awt.event.KeyEvent): void
- cu_botonActionPerformed(java.awt.event.ActionEvent): void
- cu_ubicacionKeyTyped(java.awt.event.KeyEvent): void
- initComponents(): void
+ main(String[]): void
+ transparencia(): void


Default Style
- a_botonActionPerformed(java.awt.event.ActionEvent): void
- a_colaAquaparkKeyTyped(java.awt.event.KeyEvent): void
+ Aquapark1()
+ crearUsuarios(Aquapark): void
- initComponents(): void
+ main(String[]): void
- pg_cKeyTyped(java.awt.event.KeyEvent): void
- pg_mActionPerformed(java.awt.event.ActionEvent): void
- pg_mKeyTyped(java.awt.event.KeyEvent): void
- pg_piscinaGrandeKeyTyped(java.awt.event.KeyEvent): void
- pn_cKeyTyped(java.awt.event.KeyEvent): void
- pn_esperaAdultosKeyTyped(java.awt.event.KeyEvent): void
- pn_mKeyTyped(java.awt.event.KeyEvent): void
- pn_piscinaNiniosKeyTyped(java.awt.event.KeyEvent): void
- po_cKeyTyped(java.awt.event.KeyEvent): void
- po_mKeyTyped(java.awt.event.KeyEvent): void
- po_piscinaOlasKeyTyped(java.awt.event.KeyEvent): void
- t_cKeyTyped(java.awt.event.KeyEvent): void
- t_mKeyTyped(java.awt.event.KeyEvent): void
- t_tumbonasKeyTyped(java.awt.event.KeyEvent): void
- ta_cKeyTyped(java.awt.event.KeyEvent): void
- ta_mKeyTyped(java.awt.event.KeyEvent): void
- ta_toboganAKeyTyped(java.awt.event.KeyEvent): void
- tb_cKeyTyped(java.awt.event.KeyEvent): void
- tb_mKeyTyped(java.awt.event.KeyEvent): void
- tb_toboganBKeyTyped(java.awt.event.KeyEvent): void
- tc_cKeyTyped(java.awt.event.KeyEvent): void
- tc_mKeyTyped(java.awt.event.KeyEvent): void
- tc_toboganCKeyTyped(java.awt.event.KeyEvent): void
+ transparencia(): void
- v_c_adultosKeyTyped(java.awt.event.KeyEvent): void
- v_c_niniosKeyTyped(java.awt.event.KeyEvent): void
- v_c_texto_adultosKeyTyped(java.awt.event.KeyEvent): void
- v_mActionPerformed(java.awt.event.ActionEvent): void
- v_mKeyTyped(java.awt.event.KeyEvent): void
- v_vestuarioKeyTyped(java.awt.event.KeyEvent): void
-vestuario
-vestuario
Vestuario
- adultoLleno: Condition = super.getCerroj...
- adultoVacio: Condition = super.getCerroj...
- aforoAdultos: int = 20
- aforoNinios: int = 10
- colaAdultos: ArrayList<Usuario> = new ArrayList()
- colaNinios: ArrayList<Usuario> = new ArrayList()
- contador: int = 0
- contAdultos: int = 0
- contNinios: int = 0
- ninioLleno: Condition = super.getCerroj...
- ninioVacio: Condition = super.getCerroj...
- v_c_adultos: JTextField
- v_c_ninios: JTextField
- v_m: JTextField
- v_vestuario: JTextField
- vestuarioLleno: Condition = super.getCerroj...
- vestuarioVacio: Condition = super.getCerroj...
- comprobarNoEnCola(Usuario): boolean
+ entrar(Usuario): void
+ getCont_v(): int
- imprimir(JTextField, JTextField, JTextField): void
- imprimirColaEspera(JTextField, JTextField): void
+ imprimirMonitor(JTextField): void
- meterColaEspera(Usuario): void
- meterVestuario(Usuario): void
- sacarColaEspera(Usuario): void
- sacarVestuario(Usuario): void
+ salir(Usuario): void
+ Vestuario(int, JTextField, JTextField, JTextField)
javax
Aquapark2
~ acceso: Metodos
- ca_boton: javax.swing.JButton
- ca_pg: javax.swing.JTextField
- ca_pn: javax.swing.JTextField
- ca_po: javax.swing.JTextField
- ca_to: javax.swing.JTextField
- ca_tu: javax.swing.JTextField
- ca_v: javax.swing.JTextField
- cm_boton: javax.swing.JButton
- cm_numMenores: javax.swing.JTextField
- ct_boton: javax.swing.JButton
- ct_tA: javax.swing.JTextField
- ct_tB: javax.swing.JTextField
- ct_tC: javax.swing.JTextField
- cu_actividades: javax.swing.JTextField
- cu_boton: javax.swing.JButton
- cu_ubicacion: javax.swing.JTextField
- cu_usuario: javax.swing.JTextField
- fondo: javax.swing.JLabel
+ Aquapark2()
- ca_botonActionPerformed(java.awt.event.Actior
- ca_pgKeyTyped(java.awt.event.KeyEvent): void
- ca_pnKeyTyped(java.awt.event.KeyEvent): void
- ca_poKeyTyped(java.awt.event.KeyEvent): void
- ca_toKeyTyped(java.awt.event.KeyEvent): void
- ca_tuKeyTyped(java.awt.event.KeyEvent): void
- ca_vKeyTyped(java.awt.event.KeyEvent): void
- cm_botonActionPerformed(java.awt.event.Actio
- cm_numMenoresKeyTyped(java.awt.event.Keyl
- ct_botonActionPerformed(java.awt.event.Action
- ct_tAKeyTyped(java.awt.event.KeyEvent): void
- ct_tBKeyTyped(java.awt.event.KeyEvent): void
- ct_tCKeyTyped(java.awt.event.KeyEvent): void
- cu_actividadesKeyTyped(java.awt.event.KeyEv
- cu_botonActionPerformed(java.awt.event.Actior
- cu_ubicacionKeyTyped(java.awt.event.KeyEver
- initComponents(): void
+ main(String[]): void
+ transparencia(): void


«property get»
+ gettA(): Tobogan
+ gettB(): Tobogan
+ gettC(): Tobogan
~acceso
-tB
~aquapark
-pisciNinios
-aquapark
-pisciOlas
PiscinaNinios
- cont: int = 0
- esperaPadres: ArrayList<Usuario> = new ArrayList()
- piscinaLlena: Condition = super.getCerroj...
- piscinaVacia: Condition = super.getCerroj...
- pn_c: JTextField
- pn_esperaAdultos: JTextField
- pn_piscinaNinios: JTextField
+ entrar(Usuario): void
+ getCont_pn(): int
- imprimir(): void
- imprimirColaEspera(): void
- imprimirEsperaPadres(): void
- meterColaEspera(Usuario): void
- meterPiscina(Usuario): void
+ PiscinaNinios(int, JTextField, JTextField, JTextField)
- sacarColaEspera(): void
- sacarPiscina(Usuario): void
+ salir(Usuario): void
PiscinaOlas
- colaEsperaParejasFormadas: ArrayList<ArrayList>
- colaEsperaParejaSinFormar: ArrayList<Usuario>
- cont: int = 0
- loquesea: Semaphore = new Semaphore(1) {readOnly}
- pareja1: boolean = false
- pareja2: boolean = false
- po_c: JTextField
- po_m: JTextField
- po_piscinaOlas: JTextField
+ comprobarEnPisciGrande(Usuario): boolean
- elegirParejas(Usuario): void
+ entrar(Usuario): void
+ getCont_po(): int
- imprimir(): void
+ imprimirColaEsperaParejasFormadas(): void
+ imprimirMonitor(JTextField): void
- meterPareja(ArrayList<Usuario>): void
+ PiscinaOlas(int, JTextField, JTextField)
+ salir(Usuario): void
-pisciNinios
-pisciOlas
-tumbona
Tumbona
- colaEntradaTumbona: Semaphore = new Semaphore(1... {readOnly}
- cont: int = 0
- control: Semaphore = new Semaphore(1... {readOnly}
- t_c: JTextField
- t_m: JTextField
- t_tumbonas: JTextField
- tumbonaLlena: Semaphore = new Semaphore(20) {readOnly}
- tumbonaVacia: Semaphore = new Semaphore(0) {readOnly}
- comprobarNoEnCola(Usuario): boolean
+ entrar(Usuario): void
+ getCont_tum(): int
- imprimir(): void
- imprimirColaEspera(): void
+ imprimirMonitor(JTextField): void
Usuario
- acom: boolean = false
- acompaniante: Usuario
- actividad: Actividad
- adulto: boolean = true
- apto: boolean = true
- aquapark: Aquapark
- contAptos: int = 0
- numActividades: int
- elegirActividad(): Actividad
- entrarActividad(Actividad): void
+ getAcompaniante(): Usuario
+ getActividad(): Actividad
+ getAquapark(): Aquapark
+ getContAptos(): int
+ getNumActividades(): int
Thread
-tumbona


Default Style
Tobogan
- cont: int = 0
- contador_total: int = 0
- idTobogan: String
- t_c: JTextField
- t_tobogan: JTextField
- tobaganLleno: Condition = super.getCerroj...
- tobaganVacio: Condition = super.getCerroj...
+ entrar(Usuario): void
- entrarEnPiscinaGrande(Usuario): void
+ getCont_tob(): int
+ getContador_total_tob(): int
- imprimir(): void
- imprimirColaEspera(): void
- meterColaEspera(Usuario): void
- meterTobogan(Usuario): void
- sacarColaEspera(): void
- sacarTobogan(Usuario): void
+ salir(Usuario): void
+ Tobogan(String, int, JTextField, JTextField)
«property get»
+ gettA(): Tobogan
+ gettB(): Tobogan
+ gettC(): Tobogan
-tB
~aquapark
-pisciNinios
PiscinaNinios
- cont: int = 0
- esperaPadres: ArrayList<Usuario> = new ArrayList(
- piscinaLlena: Condition = super.getCerroj...
- piscinaVacia: Condition = super.getCerroj...
- pn_c: JTextField
- pn_esperaAdultos: JTextField
- pn_piscinaNinios: JTextField
+ entrar(Usuario): void
+ getCont_pn(): int
- imprimir(): void
- imprimirColaEspera(): void
- imprimirEsperaPadres(): void
- meterColaEspera(Usuario): void
- meterPiscina(Usuario): void
+ PiscinaNinios(int, JTextField, JTextField, JTextField)
- sacarColaEspera(): void
- sacarPiscina(Usuario): void
+ salir(Usuario): void
javax.swing.JFrame
Aquapark1
- a_boton: javax.swing.JButton
- a_colaAquapark: javax.swing.JTextField
~ aquapark: Aquapark
- fondo: javax.swing.JLabel
- m1: Monitor
- m2: Monitor
- m3: Monitor
- m4: Monitor
- m5: Monitor
- m6: Monitor
- m7: Monitor
- m8: Monitor
- pg_c: javax.swing.JTextField
- pg_m: javax.swing.JTextField
- pg_piscinaGrande: javax.swing.JTextField
- pisciGrande: PiscinaGrande
- pisciNinios: PiscinaNinios
- pisciOlas: PiscinaOlas
- pn_c: javax.swing.JTextField
- pn_esperaAdultos: javax.swing.JTextField
- pn_m: javax.swing.JTextField
- pn_piscinaNinios: javax.swing.JTextField
- po_c: javax.swing.JTextField
- po_m: javax.swing.JTextField
- po_piscinaOlas: javax.swing.JTextField
- t_c: javax.swing.JTextField
- t_m: javax.swing.JTextField
- t_tumbonas: javax.swing.JTextField
- tA: Tobogan
- ta_c: javax.swing.JTextField
- ta_m: javax.swing.JTextField
- ta_toboganA: javax.swing.JTextField
- tB: Tobogan
- tb_c: javax.swing.JTextField
-tA
-pisciNinios
-tumbona
Tumbona
- colaEntradaTumbona: Semaphore = new Semaphore(1... {readOnly}
- cont: int = 0
- control: Semaphore = new Semaphore(1... {readOnly}
- t_c: JTextField
- t_m: JTextField
- t_tumbonas: JTextField
- tumbonaLlena: Semaphore = new Semaphore(20) {readOnly}
- tumbonaVacia: Semaphore = new Semaphore(0) {readOnly}
- comprobarNoEnCola(Usuario): boolean
+ entrar(Usuario): void
+ getCont_tum(): int
- imprimir(): void
- imprimirColaEspera(): void
+ imprimirMonitor(JTextField): void
Thread
Persona
- edad: int
-tumbona


Default Style
Monitor
ividad
uapark
inaGrande(Usuario, PiscinaGrande): boolean
iNinios(Usuario): boolean
iOlasEdad(Usuario): boolean
gan(String, Usuario): boolean
bona(Usuario): boolean
uario(Usuario): boolean
ande(PiscinaGrande): int
it, Actividad, Aquapark)
rioComprobar(int, int): void
obar(int): void
-aquapark
-m5
PiscinaGrande
= 0
TextField
inaGrande: JTextField
.lena: Condition = super.getCerroj...
/acia: Condition = super.getCerroj...
Activado: Condition = super.getCerroj...
Parado: Condition = super.getCerroj...
suario): void
rTobogan(Usuario): void
_pg(): int
(): void
ColaEspera(): void
olaEspera(Usuario): void
scinaGrande(Usuario): void
Grande(int, JTextField, JTextField)
olaEspera(): void
scinaGrande(Usuario): void
uario): void
-pisciGrande
Tobogan
UnicastRemoteObject
Aquapark
- a_colaAquapark: JTextField
- abierto: boolean
- actividades: ArrayList<Actividad> = new ArrayList()
- aforo: int
- aquaparkLleno: Condition = cerrojo.newCond...
- aquaparkVacio: Condition = cerrojo.newCond...
- cerrojo: Lock = new ReentrantLock()
- colaEspera: ArrayList<Usuario> = new ArrayList()
- contador: int = 0
- contador_total: int = 0
- lugar: ArrayList<Usuario> = new ArrayList()
- num_menores: int = 0
- pisciGrande: PiscinaGrande
- pisciNinios: PiscinaNinios
- pisciOlas: PiscinaOlas
- tA: Tobogan
- tB: Tobogan
- tC: Tobogan
- tumbona: Tumbona
- vestuario: Vestuario
+ Aquapark(int, Vestuario, PiscinaNinios, PiscinaOlas, Tumbona, Tobogan, Tobogan, Tobogan, PiscinaGrande, JTextField)
+ controlAforo(String): int
+ controlToboganes(String): int
+ devolverActividad(String): String
+ devolverNumActividad(String): String
+ entrar(Usuario): void
+ getActividades(): ArrayList<Actividad>
+ getNum_menores(): int
+ getPisciGrande(): PiscinaGrande
+ getPisciNinios(): PiscinaNinios
+ getPisciOlas(): PiscinaOlas
+ getTumbona(): Tumbona
+ getVestuario(): Vestuario
- imprimir(): void
- imprimirColaEspera(): void
+ isAbierto(): boolean
- meterAquapark(Usuario): void
- meterColaEspera(Usuario): void
- sacarAquapark(Usuario): void
- sacarColaEspera(): void
+ salir(Usuario): void
«property get»
+ gettA(): Tobogan
+ gettB(): Tobogan
+ gettC(): Tobogan
Remote
«interface»
Aquapark::Metodos
+ controlAforo(String): int
+ controlToboganes(String): int
+ devolverActividad(String): String
+ devolverNumActividad(String): String
+ getNum_menores(): int
~acceso



5. CÓDIGO FUENTE (anexo)

Actividad

```
package clases;

import java.util.ArrayList;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 *
 * @author Laura y Víctor
 */
public class Actividad
{
   private int aforo;
   private Monitor monitor = null;
   private ArrayList<Usuario> lugar = new ArrayList();
   private ArrayList<Usuario> colaEspera = new ArrayList();
   private ArrayList<ArrayList> lugarTablero = new ArrayList();
   private ArrayList<ArrayList> colaEsperaTablero = new ArrayList();
```

```
private ArrayList<ArrayList> colaEsperaVestuario = new ArrayList();
private ArrayList<Usuario> colaEsperaPisciGrande = new ArrayList();
private ArrayList<ArrayList> colaEsperaPisciOlas = new ArrayList();
private ArrayList<Usuario> colaEsperaTumbona = new ArrayList();
private ArrayList<Usuario> colaEsperaTobogan = new ArrayList();
private ArrayList<Usuario> colaEsperaPisciNinios = new ArrayList();

private Lock cerrojo = new ReentrantLock();
private String nombre;

public Actividad(int aforo) {
   this.aforo = aforo;
}

public int getAforo() {
   return aforo;
}

public void setAforo(int aforo) {
   this.aforo = aforo;
}

public ArrayList<Usuario> getLugar() {
   return lugar;
}

public ArrayList<Usuario> getColaEsperaPisciGrande() {
   return colaEsperaPisciGrande;
}

public void setColaEsperaPisciGrande(ArrayList<Usuario> colaEsperaPisciGrande) {
   this.colaEsperaPisciGrande = colaEsperaPisciGrande;
}

public ArrayList<ArrayList> getColaEsperaPisciOlas() {
   return colaEsperaPisciOlas;
}

public void setColaEsperaPisciOlas(ArrayList<ArrayList> colaEsperaPisciOlas) {
   this.colaEsperaPisciOlas = colaEsperaPisciOlas;
}

public ArrayList<Usuario> getColaEsperaTumbona() {
   return colaEsperaTumbona;
}

public void setColaEsperaTumbona(ArrayList<Usuario> colaEsperaTumbona) {
   this.colaEsperaTumbona = colaEsperaTumbona;
}

public ArrayList<Usuario> getColaEsperaTobogan() {
```

```
    return colaEsperaTobogan;
}

public void setColaEsperaTobogan(ArrayList<Usuario> colaEsperaTobogan) {
    this.colaEsperaTobogan = colaEsperaTobogan;
}

public ArrayList<Usuario> getColaEsperaPisciNinios() {
    return colaEsperaPisciNinios;
}

public void setColaEsperaPisciNinios(ArrayList<Usuario> colaEsperaPisciNinios) {
    this.colaEsperaPisciNinios = colaEsperaPisciNinios;
}



    public ArrayList<ArrayList> getColaEsperaVestuario() {
    return colaEsperaVestuario;
}

public void setColaEsperaVestuario(ArrayList<ArrayList> colaEsperaVestuario) {
    this.colaEsperaVestuario = colaEsperaVestuario;
}

public void setLugar(ArrayList<Usuario> lugar) {
    this.lugar = lugar;
}

public ArrayList<Usuario> getColaEspera() {
    return colaEspera;
}

public void setColaEspera(ArrayList<Usuario> colaEspera) {
    this.colaEspera = colaEspera;
}

public ArrayList<ArrayList> getLugarTablero() {
    return lugarTablero;
}

public void setLugarTablero(ArrayList<ArrayList> lugarTablero) {
    this.lugarTablero = lugarTablero;
}

public ArrayList<ArrayList> getColaEsperaTablero() {
    return colaEsperaTablero;
}

public void setColaEsperaTablero(ArrayList<ArrayList> colaEsperaTablero) {
    this.colaEsperaTablero = colaEsperaTablero;
```

```
    }

    public Lock getCerrojo() {
        return cerrojo;
    }

    public void setCerrojo(Lock cerrojo) {
        this.cerrojo = cerrojo;
    }

    public Monitor getMonitor() {
        return monitor;
    }

    public void setMonitor(Monitor monitor) {
        this.monitor = monitor;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

Aquapark

```
package clases;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
import javax.swing.JTextField;

/**
 *
 * @author Laura y Víctor
 */
public class Aquapark extends UnicastRemoteObject implements Metodos {

    private int num_menores = 0;
    private int contador = 0;
    private int contador_total = 0;
    private int aforo;
    private boolean abierto;
    private ArrayList<Usuario> lugar = new ArrayList();
```

```
    private ArrayList<Usuario> colaEspera = new ArrayList();

    private Vestuario vestuario;
    private PiscinaNinios pisciNinios;
    private PiscinaOlas pisciOlas;
    private Tumbona tumbona;
    private Tobogan tA;
    private Tobogan tB;
    private Tobogan tC;
    private PiscinaGrande pisciGrande;
    private ArrayList<Actividad> actividades = new ArrayList();
    private Lock cerrojo = new ReentrantLock();
    private Condition aquaparkVacio = cerrojo.newCondition();
    private Condition aquaparkLleno = cerrojo.newCondition();
    private JTextField a_colaAquapark;

     public Aquapark(int aforo, Vestuario vestuario, PiscinaNinios pisciNinios, PiscinaOlas pisciOlas, Tumbona tumbona, Tobogan tA, Tobogan tB, Tobogan tC, PiscinaGrande pisciGrande, JTextField a_colaAquapark) throws RemoteException {
        this.aforo = aforo;
        this.pisciGrande = pisciGrande;
        this.vestuario = vestuario;
        this.pisciNinios = pisciNinios;
        this.pisciOlas = pisciOlas;
        this.tumbona = tumbona;
        this.tA = tA;
        this.tB = tB;
        this.tC = tC;

        //arrayList actividades
        actividades.add(vestuario);
        actividades.add(pisciGrande);
        actividades.add(pisciNinios);
        actividades.add(pisciOlas);
        actividades.add(tumbona);
        actividades.add(tA);
        actividades.add(tB);
        actividades.add(tC);

        this.abierto = true;
        this.a_colaAquapark = a_colaAquapark;
    }

    /**
     * Entrar en Aquapark
     *
     * @param usuario
     */
    public void entrar(Usuario usuario) {
        meterColaEspera(usuario);
        sacarColaEspera();
```

```
}

/**
 * Meter en cola de espera
 *
 * @param usuario
 */
private void meterColaEspera(Usuario usuario) {
   cerrojo.lock();
   try {
      colaEspera.add(usuario);
      imprimir();
   } finally {
      cerrojo.unlock();
   }
}

/**
 * Sacar a los usuarios de la cola de espera
 *
 * @param usuario
 */
private void sacarColaEspera() {
   cerrojo.lock();
   try {
      while (contador >= aforo) {
         try {
            aquaparkLleno.await();
         } catch (InterruptedException e) {
            System.out.println("Error en entrar()" + e);
         }
      }
      meterAquapark(colaEspera.get(0));
      colaEspera.remove(0);
      imprimir();
      aquaparkVacio.signal();
   } finally {
      cerrojo.unlock();
   }
}

/**
 * Meter en aquapark
 *
 * @param usuario
 */
private void meterAquapark(Usuario usuario) {

   lugar.add(usuario);
   if (usuario.getEdad() < 18) {
      num_menores++;
```

```
    }
    if (usuario.isAcom()) {
        lugar.add(usuario.getAcompañante());
        contador++;
        contador_total++;
    }
    contador++;
    contador_total++;
}

//////////////////////////////////////////////
//SALIR
/**
 * Salir del aquaprk
 *
 * @param usuario
 */
public void salir(Usuario usuario) {
    cerrojo.lock();
    try {
        while (contador < 1) {
            try {
                aquaparkVacio.await();
            } catch (InterruptedException e) {
                System.out.println("Error en salir() " + e);
            }
        }
        sacarAquapark(usuario);
        imprimir();
        usuario.setContAptos(usuario.getContAptos() + 1);
        aquaparkLleno.signal();
    } finally {
        cerrojo.unlock();
    }
}

/**
 * Sacar a los usuarios del aquapark
 *
 * @param usuario
 */
private void sacarAquapark(Usuario usuario) {
    for (int i = 0; i < lugar.size(); i++) {
        if (lugar.get(i) == usuario) {
            lugar.remove(i);
        }
    }
    contador--;
    if (contador_total == aforo) {
        abierto = false;
    }
```

```
}

//////////////////////////////////////////
//IMPRIMIR
private void imprimir() {
   imprimirColaEspera();
}

/**
 * Imprime las colas de espera
 *
 */
private void imprimirColaEspera() {
   if (colaEspera.size() > 0) {
      a_colaAquapark.setText(colaEspera.get(0).toString());
      for (int i = 1; i < (colaEspera.size()); i++) {
         String previo = a_colaAquapark.getText();
         a_colaAquapark.setText(previo + " || " + colaEspera.get(i).toString());

      }
   }
}

//////////////////////////////
//GETTERS Y SETTERS
public ArrayList<Actividad> getActividades() {
   return actividades;
}

public Vestuario getVestuario() {
   return vestuario;
}

public PiscinaNinios getPisciNinios() {
   return pisciNinios;
}

public PiscinaOlas getPisciOlas() {
   return pisciOlas;
}

public Tumbona getTumbona() {
   return tumbona;
}

public Tobogan gettA() {
   return tA;
}

public Tobogan gettB() {
   return tB;
```

```
}

public Tobogan gettC() {
   return tC;
}

public PiscinaGrande getPisciGrande() {
   return pisciGrande;
}

public boolean isAbierto() {
   return abierto;
}

//////////////////////////////////////////
//REMOTO
/**
 * Para PARTE 2
 *
 * @return
 * @throws java.rmi.RemoteException
 */
@Override
public int getNum_menores() throws RemoteException //así comparto el método
{
   return num_menores;
}

/**
 * Devuelve la actividad que lleva el usuario si está en el aquapark
 *
 * @param nombre
 * @return
 * @throws RemoteException
 */
@Override
public String devolverActividad(String nombre) throws RemoteException //así comparto el método
{
   String actividad;
   if (lugar.size() > 0) {
      int pos = -1;
      for (int i = 0; i < lugar.size(); i++) {
         if (nombre.equals(lugar.get(i).toString())) {
            pos = i;
         }
      }
      if (pos == -1) {
         actividad = nombre + " NO ESTÁ EN AQUAPARK!!!";
      } else {
         actividad = lugar.get(pos).getActividad().getNombre();
      }
```

```
    } else {
        actividad = "No hay usuarios en Aquapark";
    }
    return actividad;
}

/**
 * Devuelve el número de actividades que lleva el usuario si está en el
 * aquapark
 *
 * @param nombre
 * @return
 * @throws RemoteException
 */
@Override
public String devolverNumActividad(String nombre) throws RemoteException //así comparto el método
{
    String actividad;
    if (lugar.size() > 0) {
        int pos = -1;
        for (int i = 0; i < lugar.size(); i++) {
            if (nombre.equals(lugar.get(i).toString())) {
                pos = i;
            }
        }
        if (pos == -1) {
            actividad = nombre + " NO ESTÁ EN AQUAPARK!!!";
        } else {
            actividad = String.valueOf(lugar.get(pos).getNumActividades());
        }
    } else {
        actividad = "No hay usuarios en Aquapark";
    }
    return actividad;
}

/**
 * Devuelve el número de usuarios que han pasado por cada tobogán
 *
 * @param nombre
 * @return
 * @throws RemoteException
 */
@Override
public int controlToboganes(String nombre) throws RemoteException //así comparto el método
{
    int cont = 0;
    if (null != nombre) {
        switch (nombre) {
            case "Tobogan A":
                cont = tA.getContador_total_tob();
```

```
            break;
        case "Tobogan B":
            cont = tB.getContador_total_tob();
            break;
        case "Tobogan C":
            cont = tC.getContador_total_tob();
            break;
        default:
            break;
      }
   }
   return cont;
}

/**
 * Dice el número de usuarios que hay en la actividad en ese momento
 *
 * @param nombre
 * @return
 * @throws RemoteException
 */
@Override
public int controlAforo(String nombre) throws RemoteException //así comparto el método
{
   int cont = 0;
   if (null != nombre) {
      switch (nombre) {
         case "Toboganes":
            cont = tA.getCont_tob() + tB.getCont_tob() + tC.getCont_tob();
            break;
         case "Piscina Grande":
            cont = pisciGrande.getCont_pg();
            break;
         case "Piscina Ninios":
            cont = pisciNinios.getCont_pn();
            break;
         case "Piscina Olas":
            cont = pisciOlas.getCont_po();
            break;
         case "Tumbona":
            cont = tumbona.getCont_tum();
            break;
         case "Vestuario":
            cont = vestuario.getCont_v();
            break;
         default:
            break;
      }
   }
   return cont;
}
```

```
}
```

Metodos

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package clases;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 *
 * @author Lau y Vic
 */
public interface Metodos extends Remote
{
   int getNum_menores() throws RemoteException;
   int controlAforo (String nombre) throws RemoteException;
    int controlToboganes (String nombre) throws RemoteException;  //cuántos usuarios han pasado por cada tobogán
    String devolverActividad(String nombre) throws RemoteException; //devuelve el nombre de la actividad en la que se encuentra el usua
   String devolverNumActividad(String nombre) throws RemoteException;
}
```

Monitor

```
package clases;

import static java.lang.Thread.sleep;


/**
 *
 * @author Laura y Víctor
 */
public class Monitor extends Persona
{
   private Actividad actividad;
   private Aquapark aquapark;

   /**
    * constructor
    * @param id
    * @param edad
    * @param actividadd
```

```
*/
public Monitor(int id, int edad, Actividad actividadd, Aquapark aquaparkk) {
   super(id, edad);
   this.actividad = actividadd;
   this.aquapark = aquaparkk;

   System.out.println("\n ++ MONITOR " + super.getIdent() + " --> " + actividad.getNombre());
}

/**
 * Controla la edad en VESTUARIO
 * 1-11 años NO acompañados --> no pueden
 * @param usuario
 * @return
*/
public boolean controlarVestuario(Usuario usuario)
{
   boolean puede = true;
   if (usuario.getEdad() < 11)
   {
      if (usuario.getAcompañante() == null)   //6-10 años no acompañados --> no pueden
      {
         puede = false;
      }
   }
   tiempoComprobar(1000);
   return puede;
}

/**
 * Controla la edad en PISCINA OLAS
 * 1-5 años --> no pueden
 * 6-10 años NO acompañados --> no pueden
 * @param usuario
 * @return
*/
public boolean controlarPisciOlasEdad(Usuario usuario)
{
   boolean puede = true;
   //si edad < 5
   if (usuario.getEdad() < 6)  //1-5 años --> no pueden
   {
      puede = false;
   }
   else
   {
      if (usuario.getEdad() < 11)
      {
         if (usuario.getAcompañante() == null)   //6-10 años no acompañados --> no pueden
         {
            puede = false;
```

```
            }
          }
        }
        tiempoComprobar(1000);
        return puede;
    }

    /**
     * controla la edad en PISCINA NINIOS
     * 1-5 años NO acompañados --> no pueden
     * >11 años  --> no pueden
     * @param usuario
     * @return
    */
    public boolean controlarPisciNinios(Usuario usuario)
    {
        boolean puede = true;
          if (usuario.getEdad() < 6 && usuario.getAcompañante() == null)  //1-5 años NO acompañados--> no
pueden
        {
          puede = false;
        }
        else if (usuario.getEdad() > 11)  // >11 --> no pueden
        {
          puede = false;
        }
        tiempoAleatorioComprobar(1000, 1500);
        return puede;
    }

    /** controla edad en TUMBONA
     * 1-14 --> no pueden
     * @param usuario
     * @return
    */
    public boolean controlarTumbona(Usuario usuario)
    {
        boolean puede = true;
        if (usuario.getEdad() < 15)  //1-14 años --> no pueden
        {
          puede = false;
        }
        tiempoAleatorioComprobar(500, 900);
        return puede;
    }

    /** controla edad en TOBOGAN
     * 11-14 --> A
     * 15-17 --> B
     * >=18 --> C
     * @param idTobogan
```

```
     * @param usuario
     * @return
    */
    public boolean controlarTobogan(String idTobogan, Usuario usuario)
    {
        boolean puede = false;
        if ("A".equals(idTobogan) && (10<usuario.getEdad() && usuario.getEdad()<15))  //11-14 --> A
        {
            puede = true;
        }
        else if ("B".equals(idTobogan) && (14<usuario.getEdad() && usuario.getEdad()<18))  //15-17 --> B
        {
            puede = true;
        }
        else if ("C".equals(idTobogan) && (17<usuario.getEdad()))   //>=18 --> C
        {
            puede = true;
        }

        tiempoAleatorioComprobar(400, 500);
        return puede;
    }

    /**
     * Controlar la piscina grande
     * @param usuario
     * @param pisciGrande
     * @return
     */
    public boolean controlarPiscinaGrande(Usuario usuario, PiscinaGrande pisciGrande)
    {
        boolean puede = true;
        tiempoComprobar(500);
        if (!(pisciGrande.getAforo() > pisciGrande.getCont_pg()))
        {
            Usuario u = pisciGrande.getLugar().get(echarPisciGrande (pisciGrande));
            System.out.println(this.toString() + " ha echado a " + u.toString());
            pisciGrande.salir(u);
        }
        return puede;
    }

    /** Echar a alguien aleatoriamente
   Si echar != 1, devuelve -1 y se debe comprobar en Z_PiscinaGrande que no sea -1
     * @param pisciGrande
     * @return
    */
    public int echarPisciGrande (PiscinaGrande pisciGrande)
    {
        int pos = (int) (pisciGrande.getAforo() * Math.random());
        tiempoAleatorioComprobar(500, 1000);
```

```
        return pos;
    }

    /** tiempo sleep
     * @param tiempo
    */
    public void tiempoComprobar(int tiempo)
    {
        try
        {
            sleep(tiempo);
        }
        catch (InterruptedException e)
        {
            System.out.println("Error en el sleep " + e);
        }
    }

    /** tiempo ALEATORIO sleep
     * @param t_min
     * @param t_max
    */
    public void tiempoAleatorioComprobar(int t_min, int t_max)
    {
        try
        {
            sleep((int) (t_min + (t_max - t_min) * Math.random()) );
        }
        catch (InterruptedException e)
        {
            System.out.println("Error en el sleep " + e);
        }
    }

    @Override
    public void run()
    {
        actividad.setMonitor(this);
        while(aquapark.isAbierto())
        {
            //acordeActividad(actividad);
        }
    }
}
```

Persona

```
package clases;

/**
```

```
 *
 * @author Laura y Víctor
 */
public class Persona extends Thread
{
   private int id;
   private int edad;
   private String ident;  //ID3-41  IDid-edad

   public Persona(int id, int edad)
   {
      this.id = id;
      this.edad = edad;

      //ident
      ident = "ID" + id + "-" + edad;
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public int getEdad() {
      return edad;
   }

   public void setEdad(int edad) {
      this.edad = edad;
   }

   public String getIdent() {
      return ident;
   }

   public void setIdent(String ident) {
      this.ident = ident;
   }
}
```

PiscinaGrande

```
package clases;

import java.util.concurrent.locks.Condition;
import javax.swing.JTextField;
```

```
/**
 *
 * @author Laura y Víctor
 */
public class PiscinaGrande extends Actividad
{

   private int cont = 0;

   private Condition piscinaVacia = super.getCerrojo().newCondition();
   private Condition piscinaLlena = super.getCerrojo().newCondition();
   private Condition toboganParado = super.getCerrojo().newCondition();
   private Condition toboganActivado = super.getCerrojo().newCondition();
   private JTextField pg_c;
   private JTextField pg_piscinaGrande;

   public PiscinaGrande(int aforo, JTextField pg_c, JTextField pg_piscinaGrande) {
      super(aforo);
      this.pg_c = pg_c;
      this.pg_piscinaGrande = pg_piscinaGrande;
      super.setNombre("Piscina Grande");
   }

   /**
    * Entrar en la piscina grande
    * @param usuario
    */
   public void entrar (Usuario usuario)
   {
      usuario.setApto(super.getMonitor().controlarPiscinaGrande(usuario, this));
      if (usuario.isApto())
      {
         meterColaEspera(usuario);
         sacarColaEspera();
      }
   }

   /** Meter en cola de espera
    *
    * @param usuario
    */
   private void meterColaEspera (Usuario usuario)
   {
      super.getCerrojo().lock();
      try
      {
         super.getColaEsperaPisciGrande().add(usuario);
         imprimir();
      }
      finally
```

```
    {
        super.getCerrojo().unlock();
    }
}

 /** Sacar a los usuarios de la cola de espera
 *
 * @param usuario
 */
private void sacarColaEspera()
{
    super.getCerrojo().lock();
    try
    {
        while (cont >= super.getAforo()) // si está lleno
        {
            try
            {
                piscinaLlena.await();
            }
            catch (InterruptedException e)
            {
                System.out.println("Error en entrar()" + e);
            }
        }
        meterPiscinaGrande(super.getColaEsperaPisciGrande().get(0));
        super.getColaEsperaPisciGrande().remove(0);
        imprimir();
        piscinaVacia.signal();
    }
    finally
    {
        super.getCerrojo().unlock();
    }
}

/**Meter en piscina grande
 *
 * @param usuario
 */
private void meterPiscinaGrande(Usuario usuario)
{
        super.getLugar().add(usuario);
        if (usuario.isAcom())
        {
            super.getLugar().add(usuario.getAcompañante());
            cont++;
        }
    cont++;
}
```

```
//////////////////////////////
//SALIR

/**
 * Salir de la piscina grande
 * @param usuario
 */
public void salir (Usuario usuario)
{
   super.getCerrojo().lock();
   try
   {
      while (cont < 1) // piscina vacía
      {
         try
         {
            piscinaVacia.await();
            toboganActivado.await();
         }
         catch (InterruptedException e)
         {
            System.out.println("Error en salir() " + e);
         }
      }
      sacarPiscinaGrande(usuario);
      imprimir();
      usuario.setContAptos(usuario.getContAptos() + 1);
      piscinaLlena.signal();
      toboganParado.signal();
   }
   finally
   {
      super.getCerrojo().unlock();
   }
}

/**
 * Sacar a los usuarios de la piscina grande
 * @param usuario
 */
private void sacarPiscinaGrande(Usuario usuario)
{
   for (int i = 0; i < super.getLugar().size(); i++)
   {
      if (super.getLugar().get(i) == usuario)
      {
         super.getLugar().remove(i);
      }
   }
   cont--;
}
```

```
////////////////////////////////////
//TOBOGAN

/**
 * Los que se han tirado por el tobogan acaban en la pisci grande
 * @param usuario
 */
public void entrarPorTobogan(Usuario usuario)
{
   super.getCerrojo().lock();
   try
   {
      while (cont >= super.getAforo())
      {
         try
         {
            toboganParado.await();
         }
         catch (InterruptedException e)
         {
            System.out.println("Error en entrar()" + e);
         }
      }
      super.getLugar().add(usuario);
      cont++;
      imprimir();
      toboganActivado.signal();
   }
   finally
   {
      super.getCerrojo().unlock();
   }
}

//////////////////////////////////////////////////////////

/**Imprime la piscina grande
 *
 */
private void imprimir() {
   if (super.getLugar().size() > 1) {
      for (int i = 0; i < (super.getLugar().size()); i++)
      {
         String previo;
         if (i == 0) {
            previo = "";
         } else {
            previo = pg_piscinaGrande.getText() + " || ";
         }
         pg_piscinaGrande.setText(previo + super.getLugar().get(i).toString());
```

```
            }
        }
        imprimirColaEspera();
    }

    /**
     * Imprime las colas de espera
     *
     */
    private void imprimirColaEspera() {
        if (super.getColaEsperaTumbona().size() > 0) {
            pg_c.setText(super.getColaEsperaTumbona().get(0).toString());

            if (super.getColaEsperaTumbona().size() > 1) {
                for (int i = 1; i < (super.getColaEsperaTumbona().size()); i++) {
                    String previo = pg_c.getText();
                    pg_c.setText(previo + " || " + super.getColaEsperaTumbona().get(i).toString());
                }
            }
        }
    }

    /**
     * Para la clase Monitor
     * @return
     */
    public int getCont_pg()
    {
        return cont;
    }
}
```

PiscinaNinios

```
package clases;

import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.concurrent.locks.Condition;
import javax.swing.JTextField;

/**
 *
 * @author Laura y Víctor
 */
public class PiscinaNinios  extends Actividad
{
    private int cont = 0;
    private Condition piscinaVacia = super.getCerrojo().newCondition();
```

```
    private Condition piscinaLlena = super.getCerrojo().newCondition();
    private ArrayList<Usuario> esperaPadres = new ArrayList();
    private JTextField pn_c;
    private JTextField pn_esperaAdultos;
    private JTextField pn_piscinaNinios;

    /**
     *
     * @param aforo
     * @param pn_c
     * @param pn_esperaAdultos
     * @param pn_piscinaNinios
     */
     public PiscinaNinios(int aforo, JTextField pn_c, JTextField pn_esperaAdultos, JTextField pn_piscinaNinios)
{
        super(aforo);
        this.pn_c = pn_c;
        this.pn_esperaAdultos = pn_esperaAdultos;
        this.pn_piscinaNinios = pn_piscinaNinios;
        super.setNombre("Piscina Ninios");
    }


    /////////////////
    //ENTRAR

    public void entrar (Usuario usuario)
    {
        usuario.setApto(super.getMonitor().controlarPisciNinios(usuario));
        if (usuario.isApto())
        {
            meterColaEspera(usuario);
            sacarColaEspera();
            imprimir();
        }
    }



    /** Meter en cola de espera
     *
     * @param usuario
     */
    private void meterColaEspera(Usuario usuario)
    {
        super.getCerrojo().lock();
        try
        {
            super.getColaEsperaPisciNinios().add(usuario);
        }
        finally
```

```
    {
      super.getCerrojo().unlock();
    }
  }

  /** Sacar a los usuarios de la cola de espera
   *
   */
  private void sacarColaEspera ()
  {
    super.getCerrojo().lock();
    try
    {
      while (cont > super.getAforo()) // si está lleno
      {
        try
        {
          piscinaLlena.await();
        }
        catch (InterruptedException e)
        {
          System.out.println("Error en entrar()" + e);
        }
      }

      meterPiscina(super.getColaEsperaPisciNinios().get(0));
      super.getColaEsperaPisciNinios().remove(0);
      piscinaVacia.signal();
    }
    finally
    {
      super.getCerrojo().unlock();
    }
  }

  /**
   * Meter en Piscina Ninios
   *
   * @param usuario
   */
  private void meterPiscina(Usuario usuario) {
    super.getLugar().add(usuario);
    cont++;
      if (usuario.getEdad() < 6) //si el niño tiene menos de 6 años, el acompañante también se mete en la
piscina
    {
      super.getLugar().add(usuario.getAcompañante());
      cont++;
    } else {
      esperaPadres.add(usuario.getAcompañante());
    }
```

```
}

//////////////////////////
//SALIR

/** Salir de la piscina
 *
 * @param usuario
 */
public void salir (Usuario usuario)
{
   super.getCerrojo().lock();
   try
   {
      while (cont < 1)
      {
         try
         {
            piscinaVacia.await();
         }
         catch (InterruptedException e)
         {
            System.out.println("Error en salir() " + e);
         }
      }
      sacarPiscina(usuario);
      imprimir();
      usuario.setContAptos(usuario.getContAptos() + 1);
      piscinaLlena.signal();
   }
   finally
   {
      super.getCerrojo().unlock();
   }
}

/**
 * Sacar a los usuarios de la piscina y a los padres de la cola de espera de
 * padres
 *
 * @param usuario
 */
private void sacarPiscina(Usuario usuario) {
   for (int i = 0; i < super.getLugar().size(); i++) {
      if (super.getLugar().get(i) == usuario) {
         if (usuario.getEdad() < 6) {
            super.getLugar().remove(i + 1);
         }
         super.getLugar().remove(i);
      }
   }
```

```
    for (int i = 0; i < esperaPadres.size(); i++) {
        if (esperaPadres.get(i) == usuario.getAcompañante()) {
            esperaPadres.remove(i);
        }
    }
    cont--;
}

////////////////////////////////////
//IMPRIMIR

/**Imprime la piscina de niños
 *
 */
private void imprimir() {
    if (super.getLugar().size() > 1) {
        for (int i = 0; i < (super.getLugar().size()); i++)
        {
            String previo;
            if (i == 0) {
                previo = "";
            } else {
                previo = pn_piscinaNinios.getText();
            }
            pn_piscinaNinios.setText(previo + " || " + super.getLugar().get(i).toString());
        }
    }
    imprimirColaEspera();
    imprimirEsperaPadres();
}

/**
 * Imprime las colas de espera
 *
 */
private void imprimirColaEspera() {
    if (super.getColaEsperaPisciNinios().size() > 0) {
        pn_c.setText(super.getColaEsperaPisciNinios().get(0).toString());

        if (super.getColaEsperaPisciNinios().size() > 1) {
            for (int i = 1; i < (super.getColaEsperaPisciNinios().size()); i++) {
                String previo = pn_c.getText();
                pn_c.setText(previo + " || " + super.getColaEsperaPisciNinios().get(i).toString());
            }
        }
    }
}

/**
 * Imprime a los padres que están esperando a sus hijos
 */
```

```
    private void imprimirEsperaPadres() {
        if (esperaPadres.size() > 0) {
            pn_esperaAdultos.setText(esperaPadres.get(0).getIdent());

            if (esperaPadres.size() > 1) {
                for (int i = 1; i < (esperaPadres.size()); i++) {
                    String previo = pn_esperaAdultos.getText();
                    pn_esperaAdultos.setText(previo + " || " + esperaPadres.get(i).getIdent());

                }
            }
        }
    }

    ////////////////////
    //GETTERS Y SETTERS

    /**
     * PARA PARTE 2
     * @return
     * @throws java.rmi.RemoteException
     */
    public int getCont_pn() throws RemoteException  //así comparto el método
    {
        return cont;
    }
}
```

PiscinaOlas

```
package clases;

import java.rmi.RemoteException;
import java.util.concurrent.Semaphore;
import javax.swing.JTextField;
import java.util.ArrayList;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
/**
*
* @author laura
 */
public class PiscinaOlas extends Actividad {

    private int cont = 0;
    private ArrayList<ArrayList> colaEsperaParejasFormadas;
    private ArrayList<Usuario> colaEsperaParejaSinFormar;
```

```
private JTextField po_c;
private JTextField po_m;
private JTextField po_piscinaOlas;

private boolean pareja1 = false;
private boolean pareja2 = false;
private final Semaphore loquesea = new Semaphore(1);

/**
 * constructor
 *
 * @param aforo
 * @param po_c
 * @param po_piscinaOlas
 */
public PiscinaOlas(int aforo, JTextField po_c, JTextField po_piscinaOlas) {
   super(aforo);
   this.po_c = po_c;
   this.po_piscinaOlas = po_piscinaOlas;
   super.setNombre("Piscina Olas");
   colaEsperaParejasFormadas = new ArrayList<>();
   colaEsperaParejaSinFormar = new ArrayList<>();

}

/**
 * Saca al monitor por el jText Field
 *
 * @param m
 */
public void imprimirMonitor(JTextField m) {
   this.po_m = m;
   po_m.setText(super.getMonitor().getIdent());
}


/**
 * entrar en Piscina de olas
 *
 * @param usuario
 */
public void entrar(Usuario usuario) {
   if (super.getMonitor().controlarPisciOlasEdad(usuario)) {
      try {
         elegirParejas(usuario); //se le asigna al usuario una pareja (completa o incompleta)
      } finally {

      }
   }
}
```

```
   /**
    * salir de la piscina de olas
    *
    * @param usuario
    */
   public void salir(Usuario usuario) {
      super.getCerrojo().lock();
      int i = 0;
      try {
            while ((super.getLugarTablero().get(i).get(0) != usuario || super.getLugarTablero().get(i).get(1) !=
usuario) && (i < super.getLugarTablero().size())) {
            i++;
         }
         super.getLugarTablero().remove(i);
         cont = cont - 2;
      } finally {
         super.getCerrojo().unlock();
      }
   }

   /**
    * Formar parejas y meterlas en la cola de espera
    *
    */
   private void elegirParejas(Usuario usuario) {
      int i = 0;
      ArrayList<Usuario> nueva_pareja = new ArrayList<>();
      if (!usuario.isAdulto() && usuario.isAcom()) //si es un niño acompañado
      {
         nueva_pareja.add(usuario);
         nueva_pareja.add(usuario.getAcompañante());

         colaEsperaParejasFormadas.add(nueva_pareja);
         meterPareja(colaEsperaParejasFormadas.get(i));

      } else //niños sin acompañantes y adultos
      {
         if (colaEsperaParejaSinFormar.size() < 1) // si la cola para formar parejas está vacía
         {
            colaEsperaParejaSinFormar.add(usuario);
           } else // si no está vacía, significa que hay una persona esperando y por tanto se puede formar la
pareja
         {
            colaEsperaParejaSinFormar.add(usuario);
            nueva_pareja.add(colaEsperaParejaSinFormar.get(0));
            nueva_pareja.add(colaEsperaParejaSinFormar.get(1));
            colaEsperaParejasFormadas.add(nueva_pareja);
            meterPareja(colaEsperaParejasFormadas.get(i));
            for (int k = 0; k < 2; k++) {
               colaEsperaParejaSinFormar.remove(0);
            }
```

```
            }
        }
        imprimir();
    }

    /**
     * UBICAR LA PAREJA: Si puede meterse la pareja en la piscina, si tiene que
     * esperar en colaparejaformada o si permanece en la cola de espera sin
     * pareja.
     */
    private void meterPareja(ArrayList<Usuario> pareja)
    {
        if (super.getAforo() > cont) // si piscina no está llena
        {
            super.getLugarTablero().add(pareja);
            cont = cont + 2;
            colaEsperaParejasFormadas.remove(0);
        }
        imprimir();
    }

    public boolean comprobarEnPisciGrande(Usuario usuario) {
        int comprobante = 0;
        for (int i = 0; i < super.getLugarTablero().size(); i++) {
                if ((super.getLugarTablero().get(i).get(0) == usuario || super.getLugarTablero().get(i).get(1) ==
usuario)) {
                comprobante++;
            }
        }
        return comprobante != 0;
    }

    /**
     * Imprime la piscina olas
     *
     */
    private void imprimir() {
        if (super.getLugar().size() > 1) {
            for (int i = 0; i < (super.getLugar().size()); i++)
            {
                String previo;
                if (i == 0) {
                    previo = "";
                } else {
                    previo = po_piscinaOlas.getText();
                }
                po_piscinaOlas.setText(previo + " || " + super.getLugar().get(i).toString());
            }
        }
        imprimirColaEsperaParejasFormadas();
    }
```

```
    /**
     * Imprime las colas de espera de pareja formadas
     *
     */
    public void imprimirColaEsperaParejasFormadas() {
        if (colaEsperaParejasFormadas.size() > 0) {
                                    po_c.setText(colaEsperaParejasFormadas.get(0).get(0).toString()  +  "  +  "  +
colaEsperaParejasFormadas.get(0).get(1).toString());

            for (int i = 1; i < (colaEsperaParejasFormadas.size()); i++) {
                String previo = po_c.getText();
                        po_c.setText(previo + " || " + colaEsperaParejasFormadas.get(i).get(0).toString() + " + " +
colaEsperaParejasFormadas.get(i).get(1).toString());
            }
        }
    }

    /**
     * Para parte 2
     *
     * @return
     * @throws java.rmi.RemoteException
     */
    public int getCont_po() throws RemoteException //así comparto el método
    {
        return cont;
    }
}
```

Tobogan

```
package clases;

import java.rmi.RemoteException;
import java.util.concurrent.locks.Condition;
import javax.swing.JTextField;

/**
 *
 * @author Laura y Víctor
 */
public class Tobogan extends Actividad {

    private int cont = 0;
    private int contador_total = 0;
    private String idTobogan;
    private Condition toboganVacio = super.getCerrojo().newCondition();
    private Condition toboganLleno = super.getCerrojo().newCondition();
```

```
private JTextField t_c;
private JTextField t_tobogan;

public Tobogan(String idTobogan, int aforo, JTextField t_c, JTextField t_tobogan) {
   super(aforo);
   this.idTobogan = idTobogan;
   this.t_c = t_c;
   this.t_tobogan = t_tobogan;
   super.setNombre("Tobogan " + idTobogan);
}

//////////////////////////
//ENTRAR
/**
 * Entrar en tobogan
 *
 * @param usuario
 */
public void entrar(Usuario usuario) {
   usuario.setApto(super.getMonitor().controlarTobogan(idTobogan, usuario));
   if (usuario.isApto()) {
      meterColaEspera(usuario);
      sacarColaEspera();
   }
}

/**
 * Meter en cola de espera
 *
 * @param usuario
 */
private void meterColaEspera(Usuario usuario) {
   super.getCerrojo().lock();
   try {
      super.getColaEsperaTobogan().add(usuario);
      imprimir();
   } finally {
      super.getCerrojo().unlock();
   }
}

/**
 * Sacar a los usuarios de la cola de espera
 *
 * @param usuario
 */
private void sacarColaEspera(){
   super.getCerrojo().lock();
   try {
      while (cont >= super.getAforo()) {
         try {
```

```
                tobogánLleno.await();
            } catch (InterruptedException e) {
                System.out.println("Error en entrar()" + e);
            }
        }
        meterTobogan(super.getColaEsperaTobogan().get(0));
        imprimir();
        tobogánVacio.signal();
            super.getColaEsperaTobogan().get(0).tiempoAleatorio(2000, 3000);  //el tiempo que está en el tobogán
        salir(super.getColaEsperaTobogan().get(0));
    } finally {
        super.getCerrojo().unlock();
    }
  }

  /**
   * Meter en tobogan
   *
   * @param usuario
   */
  private void meterTobogan(Usuario usuario) {
    if (super.getColaEsperaTobogan().size() < 1) {
        super.getLugar().add(usuario);
    } else {
        super.getLugar().add(super.getColaEsperaTobogan().get(0));
    }
    contador_total++;
    cont++;

  }

  ////////////////////////////
  //SALIR
  /**
   *
   * @param usuario
   */
  public void salir(Usuario usuario) {
    super.getCerrojo().lock();
    try {
        while (cont < 1) {
            try {
                tobogánVacio.await();
            } catch (InterruptedException e) {
                System.out.println("Error en salir() " + e);
            }
        }
        sacarTobogan(usuario);
        imprimir();
        usuario.setContAptos(usuario.getContAptos() + 1);
```

```
            tobogan.signal();
        } finally {
            super.getCerrojo().unlock();
        }
    }

    /**
     * Sacar a los usuarios del tobogan
     *
     * @param usuario
     */
    private void sacarTobogan(Usuario usuario) {
        for (int i = 0; i < super.getLugar().size(); i++) {
            if (super.getLugar().get(i) == usuario) {
                super.getLugar().remove(i);
            }
        }
        cont--;
        entrarEnPiscinaGrande(usuario);
    }

    /**
     * Cuando se tiran por el tobogán acaban en la piscina grande
     *
     * @param usuario
     */
    private void entrarEnPiscinaGrande(Usuario usuario) {
        usuario.getAquapark().getPisciGrande().entrarPorTobogan(usuario);
    }

    ////////////////////////////////////////
    /**Imprime el tobogán
     *
     */
    private void imprimir() {
        if (super.getLugar().size() > 1) {
            for (int i = 0; i < (super.getLugar().size()); i++)
            {
                String previo;
                if (i == 0) {
                    previo = "";
                } else {
                    previo = t_tobogan.getText();
                }
                t_tobogan.setText(previo + " || " + super.getLugar().get(i).toString());
            }
        } imprimirColaEspera();
    }

    /**
     * Imprime las colas de espera
```

```
     *
     */
    private void imprimirColaEspera() {
        if (super.getColaEsperaTobogan().size() > 0) {
            t_c.setText(super.getColaEsperaTobogan().get(0).toString());

            if (super.getColaEsperaTobogan().size() > 1) {
                for (int i = 1; i < (super.getColaEsperaTobogan().size()); i++) {
                    String previo = t_c.getText();
                    t_c.setText(previo + " || " + super.getColaEsperaTobogan().get(i).toString());
                    //System.out.print(super.getColaEspera().get(i).toString() + " || ");
                }
            }
        }

    }

    ////////////////////////
    //GETTERS AND SETTERS
    /**
     * Parte 2
     *
     * @return
     * @throws java.rmi.RemoteException
     */
    public int getCont_tob() throws RemoteException //así comparto el método
    {
        return cont;
    }

    /**
     * Parte 2
     *
     * @return
     * @throws java.rmi.RemoteException
     */
    public int getContador_total_tob() throws RemoteException //así comparto el método
    {
        return contador_total;
    }
}
```

Tumbona

```
package clases;

import java.rmi.RemoteException;
import java.util.concurrent.Semaphore;
import javax.swing.JTextField;
```

```
/**
 *
 * @author laura
 */
public class Tumbona extends Actividad {

   private int cont = 0;

   private final Semaphore tumbonaLlena = new Semaphore(20);
   private final Semaphore tumbonaVacia = new Semaphore(0);
   private final Semaphore control = new Semaphore(1, true);
   private final Semaphore colaEntradaTumbona = new Semaphore(10, true);
   private JTextField t_c;
   private JTextField t_m;
   private JTextField t_tumbonas;

   public Tumbona(int aforo, JTextField t_c, JTextField t_tumbonas) {
      super(aforo);
      this.t_c = t_c;
      this.t_tumbonas = t_tumbonas;
      super.setNombre("Tumbona");
   }

   /**
    * Saca al monitor por el jText Field
    *
    * @param m
    */
   public void imprimirMonitor(JTextField m) {
      this.t_m = m;
      t_m.setText(super.getMonitor().getIdent());
   }

   /**
    * Entrar en tumbona
    *
    * @param usuario
    */
   public void entrar(Usuario usuario) {
      try // Establezco un semáforo para que el monitor compruebe de uno en uno
      {
         control.acquire();
         usuario.setApto(super.getMonitor().controlarTumbona(usuario));
         control.release();
      } catch (InterruptedException e) {
      }

      if (usuario.isApto()) {
         try {
            while (cont == super.getAforo())
            {
```

```
            try {
               colaEntradaTumbona.acquire();
               meterColaEspera(usuario);
               imprimirColaEspera();
            } catch (InterruptedException e) {
               System.out.println("Error en entrar() - Meter en Cola de Espera" + e);
            }
         }

         meterColaEspera(usuario);
         meterTumbona(super.getColaEsperaTumbona().get(0));
         imprimir();

         tumbonaLlena.acquire();
         colaEntradaTumbona.release();
         tumbonaVacia.release();
      } catch (InterruptedException e) {
         System.out.println("Error en entrar() - Meter usuario en Tumbona");
      }
   }
}

/**
 *
 * @param usuario
 */
public void salir(Usuario usuario) {
   try {
      while (cont < 1) {
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            System.out.println("Error en salir() - while vacío" + e);
         }
      }
      sacarTumbona(usuario);
      usuario.setContAptos(usuario.getContAptos() + 1);

      imprimir();

      tumbonaLlena.release();
      tumbonaVacia.acquire();

   } catch (InterruptedException e) {
      System.out.println("Error en salir() - Sacar usuario de tumbona " + e);
   }
}

/**
 * Sacar a los usuarios de las tumbonas
 *
```

```
     * @param usuario
     */
    private void sacarTumbona(Usuario usuario) {
        for (int i = 0; i < super.getLugar().size(); i++) {
            if (super.getLugar().get(i) == usuario) {
                super.getLugar().remove(i);
            }
        }
        cont--;
    }

    /**
     * Meter en Tumbona --> semafor NO justo
     *
     * @param usuario
     */
    private void meterTumbona(Usuario usuario) {
        super.getLugar().add(usuario);
        cont++;
        sacarColaEspera(usuario);
    }

    /**
     * Meter en cola de espera
     *
     * @param usuario
     */
    private void meterColaEspera(Usuario usuario) {
        super.getColaEsperaTumbona().add(usuario);

    }

    /**
     * Comprobamos que todavía no está en la cola
     *
     * @param usuario
     * @return
     */
    private boolean comprobarNoEnCola(Usuario usuario) {
        boolean noEsta = true;
        boolean esta;
        for (int i = 0; i < super.getColaEsperaTumbona().size(); i++) {
            esta = super.getColaEsperaTumbona().get(i).equals(usuario);
            noEsta = (noEsta && !esta);
        }
        return noEsta;
    }

    /**
     * Sacar a los usuarios de la cola de espera
     *
```

```
     * @param usuario
     */
    private void sacarColaEspera(Usuario usuario) {
        for (int i = 0; i < super.getColaEsperaTumbona().size(); i++) {
            if (super.getColaEsperaTumbona().get(i) == usuario) {
                super.getColaEsperaTumbona().remove(i);
            }
        }
    }

    /**
     * Imprime la tumbona
     *
     */
    private void imprimir() {
        if (super.getLugar().size() > 1) {
            for (int i = 0; i < (super.getLugar().size()); i++)
            {
                String previo;
                if (i == 0) {
                    previo = "";
                } else {
                    previo = t_tumbonas.getText();
                }
                t_tumbonas.setText(previo + " || " + super.getLugar().get(i).toString());
            }
        }
        imprimirColaEspera();
    }

    /**
     * Imprime las colas de espera
     *
     */
    private void imprimirColaEspera() {
        if (super.getColaEsperaTumbona().size() > 0) {
            t_c.setText(super.getColaEsperaTumbona().get(0).toString());

            if (super.getColaEsperaTumbona().size() > 1) {
                for (int i = 1; i < (super.getColaEsperaTumbona().size()); i++) {
                    String previo = t_c.getText();
                    t_c.setText(previo + " || " + super.getColaEsperaTumbona().get(i).toString());
                }
            }
        }
    }

    /**
     * PARTE 2
     *
     * @return
```

```
     * @throws java.rmi.RemoteException
     */
    public int getCont_tum() throws RemoteException //así comparto el método
    {
        return cont;
    }
}
```

Usuario

```
package clases;

import static java.lang.Thread.sleep;


/**
 *
 * @author Laura y Víctor
 */
public class Usuario extends Persona
{
    private boolean apto = true;  //apto a realizar la actividad
    private int contAptos = 0;
    private boolean adulto = true; //si es niño o adulto
    private boolean acom = false;  // si es niño =acompañado  si es adulto = acompañante
    private Usuario acompañante;
    private int numActividades;  //actividades a realizar
    private Actividad actividad;
    private Aquapark aquapark;

    /**Constructor
     * @param acom
     * @param acompañante
     * @param aquaparkk
     * @param id
     * @param edad
     */
    public Usuario(boolean acom, Usuario acompañante, Aquapark aquaparkk, int id, int edad) {
        super(id, edad);
        this.acom = acom;
        this.acompañante = acompañante;
        this.aquapark = aquaparkk;

        //adulto o niño
        if (edad < 18)
        {
            adulto = false;
        }
        //num actividades para todos menos para los acompañantes
        numActividades = (int)(5 + 10 * Math.random());
```

```
      //ident concatenado
      if (acom && adulto)
      {
         super.setIdent(super.getIdent() + "-" + (id-1));
      }
      else if (acom && !adulto)
      {
         super.setIdent(super.getIdent() + "-" + (id+1));
            acompañante.setNumActividades(numActividades);  //num actividades del acompañante igual que
el del niño al que acompaña
      }
   }

   /** Elige Actividad aleatoriamente
   *
   */
   private Actividad elegirActividad()
   {
      int i = (int) (1 + 7 * Math.random());
      actividad = aquapark.getActividades().get(i);
      return actividad;
   }

   /**
    * Entra a la actividad correspondiente
    * @param actividad
    */
   private void entrarActividad (Actividad actividad)
   {
      if (actividad instanceof PiscinaGrande)
      {
         aquapark.getPisciGrande().entrar(this);
         if (apto)
         {
            tiempoAleatorio(3000, 5000);
            aquapark.getPisciGrande().salir(this);
         }
      }
      else if (actividad instanceof PiscinaNinios)
      {
         aquapark.getPisciNinios().entrar(this);
         if (apto)
         {
            tiempoAleatorio(1000, 3000);
            aquapark.getPisciNinios().salir(this);
         }
      }
      else if (actividad instanceof PiscinaOlas)
      {
         aquapark.getPisciOlas().entrar(this);
```

```
            boolean puede = aquapark.getPisciOlas().comprobarEnPisciGrande(this);

            if (puede)
            {
               tiempoAleatorio(2000, 5000);
               aquapark.getPisciOlas().salir(this);
            }
         }
         else if (actividad instanceof Tumbona)
         {
            aquapark.getTumbona().entrar(this);
            if (apto)
            {
               tiempoAleatorio(2000, 4000);
               aquapark.getTumbona().salir(this);
            }
         }
         else if (actividad instanceof Tobogan)
         {
            if (actividad == aquapark.gettA())
            {
               aquapark.gettA().entrar(this);
               if (apto)
               {
                  tiempoAleatorio(2000, 3000);
                  aquapark.gettA().salir(this);
               }
            }
            else if (actividad == aquapark.gettB())
            {
               aquapark.gettB().entrar(this);
               if (apto)
               {
                  tiempoAleatorio(2000, 3000);
                  aquapark.gettB().salir(this);
               }
            }
            else if (actividad == aquapark.gettC())
            {
               aquapark.gettC().entrar(this);

               if (apto)
               {
                  tiempoAleatorio(2000, 3000);
                  aquapark.gettC().salir(this);
               }
            }
         }
      }

      /** tiempo sleep
```

```
 * @param tiempo
*/
public void tiempo(int tiempo)
{
   try
   {
      sleep(tiempo);
   }
   catch (InterruptedException e)
   {
      System.out.println("Error en el sleep " + e);
   }
}

/** tiempo ALEATORIO sleep
 * @param t_min
 * @param t_max
*/
public void tiempoAleatorio(int t_min, int t_max)
{
   try
   {
      sleep((int) (t_min + (t_max - t_min) * Math.random()) );
   }
   catch (InterruptedException e)
   {
      System.out.println("Error en el sleep " + e);
   }
}

@Override
public void run()
{
   aquapark.entrar(this);
   //entra aquapark VESTUARIO
   aquapark.getVestuario().entrar(this);
   tiempo(3000); //lo que tarda en entrar
   tiempo(3000); //lo que tarda en salir
   aquapark.getVestuario().salir(this);

   //realiza actividades
   while (contAptos != numActividades)
   {
      entrarActividad(elegirActividad());
   }

   //Sale aquapark VESTUARIO
   aquapark.getVestuario().entrar(this);
   tiempo(3000); //lo que tarda en entrar
   tiempo(3000); //lo que tarda en salir
   aquapark.getVestuario().salir(this);
```

```
    aquapark.salir(this);
}

public boolean isApto() {
    return apto;
}

public void setApto(boolean apto) {
    this.apto = apto;
}

public int getContAptos() {
    return contAptos;
}

public void setContAptos(int contAptos) {
    this.contAptos = contAptos;
}

public boolean isAdulto() {
    return adulto;
}

public void setAdulto(boolean adulto) {
    this.adulto = adulto;
}

public boolean isAcom() {
    return acom;
}

public void setAcom(boolean acom) {
    this.acom = acom;
}

public Usuario getAcompañante() {
    return acompañante;
}

public void setAcompañante(Usuario acompañante) {
    this.acompañante = acompañante;
}

public int getNumActividades() {
    return numActividades;
}

public void setNumActividades(int numActividades) {
    this.numActividades = numActividades;
}
```

```
    @Override
    public String toString() {
        return super.getIdent();
    }

    public Actividad getActividad() {
        return actividad;
    }

    public Aquapark getAquapark() {
        return aquapark;
    }
}
```

Vestuario

```
package clases;

import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.concurrent.locks.Condition;
import javax.swing.JTextField;

/**
 *
 * @author Laura y Víctor
 */
public class Vestuario extends Actividad
{
    private ArrayList<Usuario> colaNinios = new ArrayList();
    private ArrayList<Usuario> colaAdultos = new ArrayList();
    private int aforoNinios = 10;
    private int aforoAdultos = 20;
    private int contNinios = 0;
    private int contAdultos = 0;
    private int contador = 0;
    private Condition vestuarioVacio = super.getCerrojo().newCondition();
    private Condition vestuarioLleno = super.getCerrojo().newCondition();
    private Condition ninioVacio = super.getCerrojo().newCondition();
    private Condition ninioLleno = super.getCerrojo().newCondition();
    private Condition adultoVacio = super.getCerrojo().newCondition();
    private Condition adultoLleno = super.getCerrojo().newCondition();
    private JTextField v_c_adultos;
    private JTextField v_c_ninios;
    private JTextField v_vestuario;
    private JTextField v_m;

    /**constructor
     * @param aforo
     * @param v_c_adultos
```

```
 * @param v_c_ninios
 * @param v_vestuario
*/
public Vestuario(int aforo, JTextField v_c_adultos, JTextField v_c_ninios, JTextField v_vestuario)
{
   super(aforo);
   super.setNombre("Vestuario");

   //Meter en cola de espera la cola de niños y la de adultos
   super.getColaEsperaVestuario().add(colaNinios);
   super.getColaEsperaVestuario().add(colaAdultos);
   this.v_c_adultos = v_c_adultos;
   this.v_c_ninios = v_c_ninios;
   this.v_vestuario = v_vestuario;
}

/**
 * Saca al monitor por el jText Field
 * @param m
 */
public void imprimirMonitor(JTextField m)
{
   this.v_m = m;
   v_m.setText(super.getMonitor().toString());
}


/**Entrar en el vestuario
 *
 * @param usuario
 */
public void entrar(Usuario usuario)
{
   usuario.setApto(super.getMonitor().controlarVestuario(usuario));
   if (usuario.isApto())
   {
      super.getCerrojo().lock();
      try
      {
         //vestuario lleno
         if (super.getAforo() == contador)
         {
            while (super.getAforo() == contador)
            {
               //si es niño acompañado
               if (usuario.isAdulto() == false && usuario.isAcom()) //acompañado
               {
                  while (contador > (super.getAforo() - 2))
                  {
                     try
                     {
```

```
                meterColaEspera (usuario);
                vestuarioLleno.await();
            }
            catch (InterruptedException e)
            {
                System.out.println("Error en entrar()" + e);
            }
        }
    }
    else
    {
        while (contador > (super.getAforo() - 1))
        {
            try
            {
                meterColaEspera (usuario);
                vestuarioLleno.await();
            }
            catch (InterruptedException e)
            {
                System.out.println("Error en entrar()" + e);
            }
        }
    }
    meterVestuario (usuario);
    imprimir(v_c_adultos, v_c_ninios, v_vestuario);
    vestuarioVacio.signal();
  }
}

else
{
  if (usuario.isAdulto() == false) //niño
  {
    if (usuario.isAcom()) //acompañado
    {
        while (contNinios >= (aforoNinios - 1))
        {
            try
            {
                meterColaEspera (usuario);
                ninioLleno.await();
            }
            catch(InterruptedException e)
            {
                System.out.println("Error en entrar()" + e);
            }
        }
    }
    else
    {
```

```
                while (contNinios >= aforoNinios)
                {
                   try
                   {
                      meterColaEspera(usuario);
                      ninioLleno.await();
                   }
                   catch(InterruptedException e)
                   {
                      System.out.println("Error en entrar()" + e);
                   }
                }
             }
             meterVestuario (usuario);
             imprimir(v_c_adultos, v_c_ninios, v_vestuario);
             ninioVacio.signal();
          }
          else
          {
             while (contAdultos >= aforoAdultos)
             {
                try
                {
                   meterColaEspera (usuario);
                   adultoLleno.await();
                }
                catch(InterruptedException e)
                {
                   System.out.println("Error en entrar()" + e);
                }
             }
             meterVestuario (usuario);
             imprimir(v_c_adultos, v_c_ninios, v_vestuario);
             adultoVacio.signal();
          }
       }
    }
    finally
    {
       super.getCerrojo().unlock();
    }
  }
}

/** Salir del vesturaio
 *
 * @param usuario
 */
public void salir (Usuario usuario)
{
   super.getCerrojo().lock();
```

```
    try
    {
      while (contador < 1)
      {
        try
        {
          vestuarioVacio.await();
        }
        catch (InterruptedException e)
        {
          System.out.println("Error en salir() " + e);
        }
      }
      sacarVestuario(usuario);
      imprimir(v_c_adultos, v_c_ninios, v_vestuario);
      vestuarioLleno.signal();
    }
    finally
    {
      super.getCerrojo().unlock();
    }
}

/**
 * Imprime el vestuario
 *
 */
private void imprimir(JTextField v_c_adultos, JTextField v_c_ninios, JTextField v_vestuario) {
    if (super.getLugar().size() > 1) {
      for (int i = 0; i < (super.getLugar().size()); i++) {
        String previo;
        if (i == 0) {
          previo = "";
        } else {
          previo = v_vestuario.getText();
        }
        v_vestuario.setText(previo + " || " + super.getLugar().get(i).toString());
      }
    }
    imprimirColaEspera(v_c_adultos, v_c_ninios);
}

/**Imprime las colas de espera
 *
 */
private void imprimirColaEspera(JTextField v_c_adultos, JTextField v_c_ninios)
{
    if (super.getColaEsperaVestuario().size() > 0)
    {
      v_c_ninios.setText(super.getColaEsperaVestuario().get(0).get(0).toString());
      v_c_adultos.setText(super.getColaEsperaVestuario().get(1).get(0).toString());
```

```
        if (super.getColaEsperaVestuario().size() > 1)
        {
            for (int i= 1; i < (super.getColaEsperaVestuario().get(0).size()); i++)
            {
                String previo = v_c_ninios.getText();
                v_c_ninios.setText(previo + " || " + super.getColaEsperaVestuario().get(0).get(i).toString());

            }
            //Text Field ADULTOS

            for (int j= 1; j < (super.getColaEsperaVestuario().get(1).size()); j++)
            {
                String previo = v_c_adultos.getText();
                v_c_adultos.setText(previo + " || " + super.getColaEsperaVestuario().get(1).get(j).toString());
            }
        }
    }
}

/**Meter en Vestuario
 *
 * @param usuario
 */
private void meterVestuario (Usuario usuario)
{
    super.getLugar().add(usuario);
    contador++;
    if (!usuario.isAdulto() && usuario.isAcom())
    {
        super.getLugar().add(usuario.getAcompañante());
        contador++;
        contNinios++;
    }
    if (usuario.isAdulto())
    {
        contAdultos++;
    }
    else
        contNinios++;
    sacarColaEspera(usuario);
}

/** Meter en cola de espera
 *
 * @param usuario
 */
private void meterColaEspera (Usuario usuario)
{
    if (comprobarNoEnCola(usuario))
    {
```

```
        if (usuario.isAdulto())
        {
            super.getColaEsperaVestuario().get(1).add(usuario);
        }
        else
        {
            super.getColaEsperaVestuario().get(0).add(usuario);
            if (usuario.isAcom())
            {
                super.getColaEsperaVestuario().get(0).add(usuario.getAcompañante());
            }
        }
        imprimir(v_c_adultos, v_c_ninios, v_vestuario);
    }
}

/**Comprobamos que todavía no está en la cola
 *
 * @param usuario
 * @return
 */
private boolean comprobarNoEnCola (Usuario usuario)
{
    boolean noEsta = true;
    boolean esta;
    if (usuario.isAdulto())
    {
        for (int i = 0; i < super.getColaEsperaVestuario().get(1).size(); i++)
        {
            esta = super.getColaEsperaVestuario().get(1).get(i).equals(usuario);
            noEsta = (noEsta && !esta);
        }
    }
    else
    {
        for (int i = 0; i < super.getColaEsperaVestuario().get(0).size(); i++)
        {
            esta = super.getColaEsperaVestuario().get(0).get(i).equals(usuario);
            noEsta = (noEsta && !esta);
        }
    }
    return noEsta;
}

/** Sacar a los usuarios de la cola de espera
 *
 * @param usuario
 */
private void sacarColaEspera (Usuario usuario)
{
    if (usuario.isAdulto())
```

```
    {
        for (int i = 0; i < super.getColaEsperaVestuario().get(1).size(); i++)
        {
            if (super.getColaEsperaVestuario().get(1).get(i) == usuario)
            {
                super.getColaEsperaVestuario().get(1).remove(i);
            }
        }
    }
    else
    {
        for (int j = 0; j < super.getColaEsperaVestuario().get(0).size(); j++)
        {
            if (super.getColaEsperaVestuario().get(0).get(j) == usuario)
            {
                super.getColaEsperaVestuario().get(0).remove(j);
                if (usuario.isAcom())
                {
                    super.getColaEsperaVestuario().get(0).remove(j); //no sé si i o i+1
                }
            }
        }
    }
}

/** Sacar a los usuarios del vestuario
 *
 * @param usuario
 */
private void sacarVestuario (Usuario usuario)
{
    for (int i = 0; i < super.getLugar().size(); i++)
    {
        if (super.getLugar().get(i) == usuario)
        {
            super.getLugar().remove(i);
        }
    }
    contador--;
    if (!usuario.isAdulto() && usuario.isAcom())
    {
        sacarVestuario (usuario.getAcompañante());
    }
    if (usuario.isAdulto() && !usuario.isAcom())
    {
        contAdultos--;
        adultoLleno.signal();
    }
    else
    {
        contNinios--;
```

```
            ninioLleno.signal();
        }
    }

    /**
     * PARA PARTE 2
     * @return
     * @throws java.rmi.RemoteException
     */
    public int getCont_v() throws RemoteException
    {
        return contador;
    }
}
```

Aquapark1

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package interfaces;

import clases.Aquapark;
import clases.Monitor;
import clases.PiscinaGrande;
import clases.PiscinaNinios;
import clases.PiscinaOlas;
import clases.Tobogan;
import clases.Tumbona;
import clases.Usuario;
import clases.Vestuario;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author laura y Víctor
 */
public class Aquapark1 extends javax.swing.JFrame
{
    Aquapark aquapark;
    private Vestuario vestuario;
    private PiscinaOlas pisciOlas;
```

```
    private PiscinaNinios pisciNinios;
    private PiscinaGrande pisciGrande;
    private Tobogan tA;
    private Tobogan tB;
    private Tobogan tC;
    private Tumbona tumbona;

    private Monitor m1;
    private Monitor m2;
    private Monitor m3;
    private Monitor m4;
    private Monitor m5;
    private Monitor m6;
    private Monitor m7;
    private Monitor m8;

    /**
     * Creates new form aquapark
     * @throws java.rmi.RemoteException
     * @throws java.net.MalformedURLException
     */
    public Aquapark1() throws RemoteException, MalformedURLException{
        initComponents();
        this.pisciOlas = new PiscinaOlas(20, po_c, po_piscinaOlas);
        this.tumbona = new Tumbona(20, t_c, t_tumbonas);
        this.pisciGrande = new PiscinaGrande(50, pg_c, pg_piscinaGrande);
        this.pisciNinios = new PiscinaNinios(15, pn_c, pn_esperaAdultos, pn_piscinaNinios);
        this.tA = new Tobogan("A", 1, ta_c, ta_toboganA);
        this.tB = new Tobogan("B", 1, tb_c, tb_toboganB);
        this.tC = new Tobogan("C", 1, tc_c, tc_toboganC);
        this.vestuario = new Vestuario(30,v_c_adultos, v_c_ninios, v_vestuario);
          this.aquapark = new Aquapark(100, vestuario, pisciNinios, pisciOlas, tumbona, tA, tB, tC, pisciGrande
,a_colaAquapark);
        this.m1 = new Monitor(1, (int)(18 + 20*Math.random()), pisciOlas, aquapark);
        this.m2 = new Monitor(2, (int)(18 + 20*Math.random()), tumbona, aquapark);
        this.m3 = new Monitor(3, (int)(18 + 20*Math.random()), pisciGrande, aquapark);
        this.m4 = new Monitor(4, (int)(18 + 20*Math.random()), pisciNinios, aquapark);
        this.m5 = new Monitor(5, (int)(18 + 20*Math.random()), tA, aquapark);
        this.m6 = new Monitor(6, (int)(18 + 20*Math.random()), tB, aquapark);
        this.m7 = new Monitor(7, (int)(18 + 20*Math.random()), tC, aquapark);
        this.m8 = new Monitor(8, (int)(18 + 20*Math.random()), vestuario, aquapark);

        transparencia();
        this.setLocationRelativeTo(null);  //ajustarlo en el medio de la pantalla

        try
        {
           Registry registry = LocateRegistry.createRegistry(1099); //Arranca rmiregistry local en el puerto 1099
             Naming.rebind("//localhost/Metodos", aquapark);   //rebind sólo funciona sobre una url del equipo
local
        } catch (RemoteException | MalformedURLException ex) {
```

```
            Logger.getLogger(Aquapark1.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        a_boton = new javax.swing.JButton();
        a_colaAquapark = new javax.swing.JTextField();
        v_m = new javax.swing.JTextField();
        v_c_texto_adultos = new javax.swing.JTextField();
        v_c_adultos = new javax.swing.JTextField();
        v_c_texto_ninios = new javax.swing.JTextField();
        v_c_ninios = new javax.swing.JTextField();
        v_vestuario = new javax.swing.JTextField();
        pn_m = new javax.swing.JTextField();
        pn_c = new javax.swing.JTextField();
        pn_piscinaNinios = new javax.swing.JTextField();
        pn_esperaAdultos = new javax.swing.JTextField();
        po_m = new javax.swing.JTextField();
        po_c = new javax.swing.JTextField();
        po_piscinaOlas = new javax.swing.JTextField();
        pg_m = new javax.swing.JTextField();
        pg_c = new javax.swing.JTextField();
        pg_piscinaGrande = new javax.swing.JTextField();
        t_m = new javax.swing.JTextField();
        t_c = new javax.swing.JTextField();
        t_tumbonas = new javax.swing.JTextField();
        ta_m = new javax.swing.JTextField();
        ta_c = new javax.swing.JTextField();
        ta_toboganA = new javax.swing.JTextField();
        tb_m = new javax.swing.JTextField();
        tb_c = new javax.swing.JTextField();
        tb_toboganB = new javax.swing.JTextField();
        tc_m = new javax.swing.JTextField();
        tc_c = new javax.swing.JTextField();
        tc_toboganC = new javax.swing.JTextField();
        fondo = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        a_boton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                a_botonActionPerformed(evt);
```

```
        }
    });
    getContentPane().add(a_boton, new org.netbeans.lib.awtextra.AbsoluteConstraints(320, 20, 450, 50));

    a_colaAquapark.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
    a_colaAquapark.setForeground(new java.awt.Color(255, 255, 255));
    a_colaAquapark.setBorder(null);
    a_colaAquapark.addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyTyped(java.awt.event.KeyEvent evt) {
            a_colaAquaparkKeyTyped(evt);
        }
    });
        getContentPane().add(a_colaAquapark, new org.netbeans.lib.awtextra.AbsoluteConstraints(180, 80,
910, 40));

    v_m.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
    v_m.setForeground(new java.awt.Color(255, 255, 255));
    v_m.setBorder(null);
    v_m.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            v_mActionPerformed(evt);
        }
    });
    v_m.addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyTyped(java.awt.event.KeyEvent evt) {
            v_mKeyTyped(evt);
        }
    });
    getContentPane().add(v_m, new org.netbeans.lib.awtextra.AbsoluteConstraints(170, 150, 230, 30));

    v_c_texto_adultos.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
    v_c_texto_adultos.setForeground(new java.awt.Color(0, 102, 102));
    v_c_texto_adultos.setText(" Adultos:");
    v_c_texto_adultos.setBorder(null);
    v_c_texto_adultos.addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyTyped(java.awt.event.KeyEvent evt) {
            v_c_texto_adultosKeyTyped(evt);
        }
    });
        getContentPane().add(v_c_texto_adultos, new org.netbeans.lib.awtextra.AbsoluteConstraints(170,
190, -1, -1));

    v_c_adultos.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
    v_c_adultos.setForeground(new java.awt.Color(255, 255, 255));
    v_c_adultos.setBorder(null);
    v_c_adultos.addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyTyped(java.awt.event.KeyEvent evt) {
            v_c_adultosKeyTyped(evt);
        }
    });
```

```
        getContentPane().add(v_c_adultos, new org.netbeans.lib.awtextra.AbsoluteConstraints(250, 190, 350,
30));

        v_c_texto_ninios.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        v_c_texto_ninios.setForeground(new java.awt.Color(0, 102, 102));
        v_c_texto_ninios.setText(" Niños:");
        v_c_texto_ninios.setBorder(null);
        getContentPane().add(v_c_texto_ninios, new org.netbeans.lib.awtextra.AbsoluteConstraints(630, 190,
-1, -1));

        v_c_ninios.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        v_c_ninios.setForeground(new java.awt.Color(0, 102, 102));
        v_c_ninios.setBorder(null);
        v_c_ninios.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                v_c_niniosKeyTyped(evt);
            }
        });
        getContentPane().add(v_c_ninios, new org.netbeans.lib.awtextra.AbsoluteConstraints(700, 190, 390,
30));

        v_vestuario.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        v_vestuario.setForeground(new java.awt.Color(255, 255, 255));
        v_vestuario.setBorder(null);
        v_vestuario.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                v_vestuarioKeyTyped(evt);
            }
        });
        getContentPane().add(v_vestuario, new org.netbeans.lib.awtextra.AbsoluteConstraints(170, 220, 920,
30));

        pn_m.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        pn_m.setForeground(new java.awt.Color(255, 255, 255));
        pn_m.setBorder(null);
        pn_m.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                pn_mKeyTyped(evt);
            }
        });
        getContentPane().add(pn_m, new org.netbeans.lib.awtextra.AbsoluteConstraints(140, 300, 220, 30));

        pn_c.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        pn_c.setForeground(new java.awt.Color(255, 255, 255));
        pn_c.setBorder(null);
        pn_c.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                pn_cKeyTyped(evt);
            }
        });
        getContentPane().add(pn_c, new org.netbeans.lib.awtextra.AbsoluteConstraints(140, 330, 400, 30));
```

```
        pn_piscinaNinios.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        pn_piscinaNinios.setForeground(new java.awt.Color(255, 255, 255));
        pn_piscinaNinios.setBorder(null);
        pn_piscinaNinios.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                pn_piscinaNiniosKeyTyped(evt);
            }
        });
         getContentPane().add(pn_piscinaNinios, new org.netbeans.lib.awtextra.AbsoluteConstraints(140, 360,
400, 30));

        pn_esperaAdultos.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        pn_esperaAdultos.setForeground(new java.awt.Color(255, 255, 255));
        pn_esperaAdultos.setBorder(null);
        pn_esperaAdultos.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                pn_esperaAdultosKeyTyped(evt);
            }
        });
            getContentPane().add(pn_esperaAdultos, new org.netbeans.lib.awtextra.AbsoluteConstraints(140,
394, 400, 30));

        po_m.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        po_m.setForeground(new java.awt.Color(255, 255, 255));
        po_m.setBorder(null);
        po_m.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                po_mKeyTyped(evt);
            }
        });
        getContentPane().add(po_m, new org.netbeans.lib.awtextra.AbsoluteConstraints(680, 330, 200, 30));

        po_c.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        po_c.setForeground(new java.awt.Color(255, 255, 255));
        po_c.setBorder(null);
        po_c.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                po_cKeyTyped(evt);
            }
        });
        getContentPane().add(po_c, new org.netbeans.lib.awtextra.AbsoluteConstraints(680, 360, 400, 40));

        po_piscinaOlas.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        po_piscinaOlas.setForeground(new java.awt.Color(255, 255, 255));
        po_piscinaOlas.setBorder(null);
        po_piscinaOlas.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                po_piscinaOlasKeyTyped(evt);
            }
        });
```

```
        getContentPane().add(po_piscinaOlas, new org.netbeans.lib.awtextra.AbsoluteConstraints(680, 400,
400, 30));

        pg_m.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        pg_m.setForeground(new java.awt.Color(255, 255, 255));
        pg_m.setBorder(null);
        pg_m.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                pg_mActionPerformed(evt);
            }
        });
        pg_m.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                pg_mKeyTyped(evt);
            }
        });
        getContentPane().add(pg_m, new org.netbeans.lib.awtextra.AbsoluteConstraints(140, 500, 220, 30));

        pg_c.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        pg_c.setForeground(new java.awt.Color(255, 255, 255));
        pg_c.setBorder(null);
        pg_c.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                pg_cKeyTyped(evt);
            }
        });
        getContentPane().add(pg_c, new org.netbeans.lib.awtextra.AbsoluteConstraints(140, 534, 400, 30));

        pg_piscinaGrande.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        pg_piscinaGrande.setForeground(new java.awt.Color(255, 255, 255));
        pg_piscinaGrande.setBorder(null);
        pg_piscinaGrande.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                pg_piscinaGrandeKeyTyped(evt);
            }
        });
            getContentPane().add(pg_piscinaGrande, new org.netbeans.lib.awtextra.AbsoluteConstraints(140,
564, 400, 30));

        t_m.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        t_m.setForeground(new java.awt.Color(255, 255, 255));
        t_m.setBorder(null);
        t_m.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                t_mKeyTyped(evt);
            }
        });
        getContentPane().add(t_m, new org.netbeans.lib.awtextra.AbsoluteConstraints(690, 500, 190, 30));

        t_c.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        t_c.setForeground(new java.awt.Color(255, 255, 255));
```

```
        t_c.setBorder(null);
        t_c.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                t_cKeyTyped(evt);
            }
        });
        getContentPane().add(t_c, new org.netbeans.lib.awtextra.AbsoluteConstraints(690, 540, 400, -1));

        t_tumbonas.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        t_tumbonas.setForeground(new java.awt.Color(255, 255, 255));
        t_tumbonas.setBorder(null);
        t_tumbonas.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                t_tumbonasKeyTyped(evt);
            }
        });
         getContentPane().add(t_tumbonas, new org.netbeans.lib.awtextra.AbsoluteConstraints(690, 570, 400,
-1));

        ta_m.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        ta_m.setForeground(new java.awt.Color(255, 255, 255));
        ta_m.setBorder(null);
        ta_m.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                ta_mKeyTyped(evt);
            }
        });
        getContentPane().add(ta_m, new org.netbeans.lib.awtextra.AbsoluteConstraints(200, 660, 240, -1));

        ta_c.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        ta_c.setForeground(new java.awt.Color(255, 255, 255));
        ta_c.setBorder(null);
        ta_c.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                ta_cKeyTyped(evt);
            }
        });
        getContentPane().add(ta_c, new org.netbeans.lib.awtextra.AbsoluteConstraints(200, 690, 240, -1));

        ta_tobogan A.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        ta_tobogan A.setForeground(new java.awt.Color(255, 255, 255));
        ta_tobogan A.setBorder(null);
        ta_tobogan A.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                ta_tobogan AKeyTyped(evt);
            }
        });
        getContentPane().add(ta_tobogan A, new org.netbeans.lib.awtextra.AbsoluteConstraints(200, 720, 240,
-1));

        tb_m.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
```

```
        tb_m.setForeground(new java.awt.Color(255, 255, 255));
        tb_m.setBorder(null);
        tb_m.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                tb_mKeyTyped(evt);
            }
        });
        getContentPane().add(tb_m, new org.netbeans.lib.awtextra.AbsoluteConstraints(520, 660, 230, -1));

        tb_c.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        tb_c.setForeground(new java.awt.Color(255, 255, 255));
        tb_c.setBorder(null);
        tb_c.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                tb_cKeyTyped(evt);
            }
        });
        getContentPane().add(tb_c, new org.netbeans.lib.awtextra.AbsoluteConstraints(520, 690, 230, -1));

        tb_toboganB.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        tb_toboganB.setForeground(new java.awt.Color(255, 255, 255));
        tb_toboganB.setBorder(null);
        tb_toboganB.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                tb_toboganBKeyTyped(evt);
            }
        });
            getContentPane().add(tb_toboganB, new org.netbeans.lib.awtextra.AbsoluteConstraints(520, 720,
230, 30));

        tc_m.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        tc_m.setForeground(new java.awt.Color(255, 255, 255));
        tc_m.setBorder(null);
        tc_m.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                tc_mKeyTyped(evt);
            }
        });
        getContentPane().add(tc_m, new org.netbeans.lib.awtextra.AbsoluteConstraints(850, 660, 230, -1));

        tc_c.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        tc_c.setForeground(new java.awt.Color(255, 255, 255));
        tc_c.setBorder(null);
        tc_c.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                tc_cKeyTyped(evt);
            }
        });
        getContentPane().add(tc_c, new org.netbeans.lib.awtextra.AbsoluteConstraints(850, 690, 230, -1));

        tc_toboganC.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
```

```
        tc_tobogan C.setForeground(new java.awt.Color(255, 255, 255));
        tc_tobogan C.setBorder(null);
        tc_tobogan C.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                tc_tobogan CKeyTyped(evt);
            }
        });
        getContentPane().add(tc_tobogan C, new org.netbeans.lib.awtextra.AbsoluteConstraints(850, 720, 230, -1));

                                                                                    fondo.setIcon(new javax.swing.ImageIcon(getClass().getResource("/interfaces/imagenes/fondo1.png"))); // NOI18N
        getContentPane().add(fondo, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, -10, 1110, 790));

        pack();
    }// </editor-fold>

    /**
     * crea a los usuarios
     * @param aquapark
     */
    public static void crearUsuarios(Aquapark aquapark)
    {
                //Usuario(boolean acom, Usuario acompañante, Vestuario vestuario, PiscinaOlas pisciOlas, Z_PiscinaNinios pisciNinios, Tumbona tumbona, Z_PiscinaGrande pisciGrande, Tobogan tobogan, int id, int edad) {

        int contador = 1;
        while (contador < 5000) //5000
        {
            int edad = (int) (1 + 51 * Math.random());

            if (edad < 11)
            {
                //acompañante
                int edadAcompañante = (int) (18 + (51-18) * Math.random());
                    Usuario uAcompañante = new Usuario (true, null, aquapark, (contador + 1), edadAcompañante); //boolean acom, Usuario acompañante, int id, int edad
                //niño
                Usuario u = new Usuario (true, uAcompañante, aquapark, contador, edad);
                contador = contador + 2;
                u.start();
            }
            else
            {
                Usuario u = new Usuario (false, null, aquapark, contador, edad);
                contador ++;
                u.start();
            }

            try
```

```
        {
            Thread.sleep((int) (400 + 300 + Math.random()));
        }
        catch (InterruptedException e)
        {
            System.out.println("Error en esperar" + e);
        }
    }
}

/**
 * empieza todo
 * @param evt
 */
private void a_botonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    po_m.setText(m1.getIdent());
    t_m.setText(m2.getIdent());
    pg_m.setText(m3.getIdent());
    pn_m.setText(m4.getIdent());
    ta_m.setText(m5.getIdent());
    tb_m.setText(m6.getIdent());
    tc_m.setText(m7.getIdent());
    v_m.setText(m8.getIdent());

    m1.start();
    m2.start();
    m3.start();
    m4.start();
    m5.start();
    m6.start();
    m7.start();
    m8.start();

    /**
    pisciOlas.imprimirMonitor(po_m);
    vestuario.imprimirMonitor(v_m);
    pisciNinios.imprimirMonitor(pn_m);
    pisciGrande.imprimirMonitor(pg_m);
    tA.imprimirMonitor(ta_m);
    tB.imprimirMonitor(tb_m);
    tC.imprimirMonitor(tc_m);
    tumbona.imprimirMonitor(t_m);
    */
    crearUsuarios(aquapark);

}

private void v_mActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

```
    private void pg_mActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }

    private void a_colaAquaparkKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void v_mKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void v_c_adultosKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void v_vestuarioKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void pn_mKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void pn_cKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
```

```
    }

    private void pn_piscinaNiniosKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void pn_esperaAdultosKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void po_mKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void po_cKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void po_piscinaOlasKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void pg_mKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void pg_cKeyTyped(java.awt.event.KeyEvent evt) {
```

```
// No admite el ingreso ni de letras ni de caracteres
      char c = evt.getKeyChar();
      if (c < 'a' || c > 'z' || c < '0' || c > '9') {
         evt.consume();
      }       // TODO add your handling code here:
   }

   private void pg_piscinaGrandeKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
      char c = evt.getKeyChar();
      if (c < 'a' || c > 'z' || c < '0' || c > '9') {
         evt.consume();
      }       // TODO add your handling code here:
   }

   private void t_mKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
      char c = evt.getKeyChar();
      if (c < 'a' || c > 'z' || c < '0' || c > '9') {
         evt.consume();
      }       // TODO add your handling code here:
   }

   private void t_cKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
      char c = evt.getKeyChar();
      if (c < 'a' || c > 'z' || c < '0' || c > '9') {
         evt.consume();
      }       // TODO add your handling code here:
   }

   private void t_tumbonasKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
      char c = evt.getKeyChar();
      if (c < 'a' || c > 'z' || c < '0' || c > '9') {
         evt.consume();
      }       // TODO add your handling code here:
   }

   private void ta_mKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
      char c = evt.getKeyChar();
      if (c < 'a' || c > 'z' || c < '0' || c > '9') {
         evt.consume();
      }       // TODO add your handling code here:
   }

   private void ta_cKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
      char c = evt.getKeyChar();
      if (c < 'a' || c > 'z' || c < '0' || c > '9') {
```

```
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void ta_tobogan AKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void tb_mKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void tb_cKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void tb_toboganBKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void tc_mKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void tc_cKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }
```

```
    private void tc_tobogancKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }       // TODO add your handling code here:
    }

    private void v_c_niniosKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }     }

    private void v_c_texto_adultosKeyTyped(java.awt.event.KeyEvent evt) {
// No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }     }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
                                                    for      (javax.swing.UIManager.LookAndFeelInfo      info      :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
                    } catch (ClassNotFoundException | InstantiationException | IllegalAccessException |
javax.swing.UnsupportedLookAndFeelException ex) {
                java.util.logging.Logger.getLogger(Aquapark1.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>

        //</editor-fold>
```

```
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            try {
                try {
                    new Aquapark1().setVisible(true);
                } catch (MalformedURLException ex) {
                    Logger.getLogger(Aquapark1.class.getName()).log(Level.SEVERE, null, ex);
                }
            } catch (RemoteException ex) {
                Logger.getLogger(Aquapark1.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    });
}

/**
 * Hacemos transparentes tanto los Text Field como el botón Aquapark para que empiece
 */
public void transparencia() {
    a_boton.setOpaque(false);
    a_boton.setContentAreaFilled(false);
    a_boton.setBorderPainted(false);

    a_colaAquapark.setBackground(new java.awt.Color(0,0,0,1));
    pg_c.setBackground(new java.awt.Color(0,0,0,1));
    pg_m.setBackground(new java.awt.Color(0,0,0,1));
    pg_piscinaGrande.setBackground(new java.awt.Color(0,0,0,1));
    pn_c.setBackground(new java.awt.Color(0,0,0,1));
    pn_esperaAdultos.setBackground(new java.awt.Color(0,0,0,1));
    pn_m.setBackground(new java.awt.Color(0,0,0,1));
    pn_piscinaNinios.setBackground(new java.awt.Color(0,0,0,1));
    po_c.setBackground(new java.awt.Color(0,0,0,1));
    po_m.setBackground(new java.awt.Color(0,0,0,1));
    po_piscinaOlas.setBackground(new java.awt.Color(0,0,0,1));
    t_c.setBackground(new java.awt.Color(0,0,0,1));
    t_m.setBackground(new java.awt.Color(0,0,0,1));
    t_tumbonas.setBackground(new java.awt.Color(0,0,0,1));
    ta_c.setBackground(new java.awt.Color(0,0,0,1));
    ta_m.setBackground(new java.awt.Color(0,0,0,1));
    ta_toboganA.setBackground(new java.awt.Color(0,0,0,1));
    tb_c.setBackground(new java.awt.Color(0,0,0,1));
    tb_m.setBackground(new java.awt.Color(0,0,0,1));
    tb_toboganB.setBackground(new java.awt.Color(0,0,0,1));
    tc_c.setBackground(new java.awt.Color(0,0,0,1));
    tc_m.setBackground(new java.awt.Color(0,0,0,1));
    tc_toboganC.setBackground(new java.awt.Color(0,0,0,1));
    v_c_adultos.setBackground(new java.awt.Color(0,0,0,1));
```

```
        v_m.setBackground(new java.awt.Color(0,0,0,1));
        v_vestuario.setBackground(new java.awt.Color(0,0,0,1));

    }

    // Variables declaration - do not modify
    private javax.swing.JButton a_boton;
    private javax.swing.JTextField a_colaAquapark;
    private javax.swing.JLabel fondo;
    private javax.swing.JTextField pg_c;
    private javax.swing.JTextField pg_m;
    private javax.swing.JTextField pg_piscinaGrande;
    private javax.swing.JTextField pn_c;
    private javax.swing.JTextField pn_esperaAdultos;
    private javax.swing.JTextField pn_m;
    private javax.swing.JTextField pn_piscinaNinios;
    private javax.swing.JTextField po_c;
    private javax.swing.JTextField po_m;
    private javax.swing.JTextField po_piscinaOlas;
    private javax.swing.JTextField t_c;
    private javax.swing.JTextField t_m;
    private javax.swing.JTextField t_tumbonas;
    private javax.swing.JTextField ta_c;
    private javax.swing.JTextField ta_m;
    private javax.swing.JTextField ta_toboganA;
    private javax.swing.JTextField tb_c;
    private javax.swing.JTextField tb_m;
    private javax.swing.JTextField tb_toboganB;
    private javax.swing.JTextField tc_c;
    private javax.swing.JTextField tc_m;
    private javax.swing.JTextField tc_toboganC;
    private javax.swing.JTextField v_c_adultos;
    private javax.swing.JTextField v_c_ninios;
    private javax.swing.JTextField v_c_texto_adultos;
    private javax.swing.JTextField v_c_texto_ninios;
    private javax.swing.JTextField v_m;
    private javax.swing.JTextField v_vestuario;
    // End of variables declaration
}
```

Aquapark2

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package interfaces;

import clases.Metodos;
```

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author laura y Víctor
 */
public class Aquapark2 extends javax.swing.JFrame {

    Metodos acceso;

    public Aquapark2() throws NotBoundException, MalformedURLException, RemoteException
    {
        initComponents();
        transparencia();
        this.setLocationRelativeTo(null); //ajustarlo en el medio de la pantalla
        acceso = (Metodos) Naming.lookup("//localhost/Metodos");
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        cu_usuario = new javax.swing.JTextField();
        cu_ubicacion = new javax.swing.JTextField();
        cu_actividades = new javax.swing.JTextField();
        cu_boton = new javax.swing.JButton();
        cm_numMenores = new javax.swing.JTextField();
        cm_boton = new javax.swing.JButton();
        ct_tA = new javax.swing.JTextField();
        ct_tB = new javax.swing.JTextField();
        ct_tC = new javax.swing.JTextField();
        ca_v = new javax.swing.JTextField();
        ca_pn = new javax.swing.JTextField();
        ca_po = new javax.swing.JTextField();
        ca_pg = new javax.swing.JTextField();
        ca_tu = new javax.swing.JTextField();
        ca_to = new javax.swing.JTextField();
        ct_boton = new javax.swing.JButton();
        ca_boton = new javax.swing.JButton();
        fondo = new javax.swing.JLabel();
```

```
        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setResizable(false);
        getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        cu_usuario.setFont(new java.awt.Font("Arial Black", 0, 18)); // NOI18N
        cu_usuario.setForeground(new java.awt.Color(29, 93, 88));
        cu_usuario.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        cu_usuario.setBorder(null);
        getContentPane().add(cu_usuario, new org.netbeans.lib.awtextra.AbsoluteConstraints(30, 250, 250,
60));

        cu_ubicacion.setFont(new java.awt.Font("Arial Black", 0, 18)); // NOI18N
        cu_ubicacion.setForeground(new java.awt.Color(255, 255, 255));
        cu_ubicacion.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        cu_ubicacion.setBorder(null);
        cu_ubicacion.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                cu_ubicacionKeyTyped(evt);
            }
        });
        getContentPane().add(cu_ubicacion, new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 470, 260,
60));

        cu_actividades.setFont(new java.awt.Font("Arial Black", 0, 18)); // NOI18N
        cu_actividades.setForeground(new java.awt.Color(255, 255, 255));
        cu_actividades.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        cu_actividades.setToolTipText("");
        cu_actividades.setBorder(null);
        cu_actividades.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                cu_actividadesKeyTyped(evt);
            }
        });
        getContentPane().add(cu_actividades, new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 560,
260, 60));

        cu_boton.setText("Buscar");
        cu_boton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                cu_botonActionPerformed(evt);
            }
        });
        getContentPane().add(cu_boton, new org.netbeans.lib.awtextra.AbsoluteConstraints(90, 350, 130,
30));

        cm_numMenores.setFont(new java.awt.Font("Arial Black", 0, 18)); // NOI18N
        cm_numMenores.setForeground(new java.awt.Color(255, 255, 255));
        cm_numMenores.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        cm_numMenores.setBorder(null);
        cm_numMenores.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
```

```
            cm_numMenoresKeyTyped(evt);
        }
    });
    getContentPane().add(cm_numMenores, new org.netbeans.lib.awtextra.AbsoluteConstraints(310, 250,
260, 60));

    cm_boton.setText("Buscar");
    cm_boton.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            cm_botonActionPerformed(evt);
        }
    });
        getContentPane().add(cm_boton, new org.netbeans.lib.awtextra.AbsoluteConstraints(370, 330, 130,
30));

    ct_tA.setFont(new java.awt.Font("Arial Black", 0, 18)); // NOI18N
    ct_tA.setForeground(new java.awt.Color(255, 255, 255));
    ct_tA.setHorizontalAlignment(javax.swing.JTextField.CENTER);
    ct_tA.setBorder(null);
    ct_tA.addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyTyped(java.awt.event.KeyEvent evt) {
            ct_tAKeyTyped(evt);
        }
    });
    getContentPane().add(ct_tA, new org.netbeans.lib.awtextra.AbsoluteConstraints(590, 250, 260, 60));

    ct_tB.setFont(new java.awt.Font("Arial Black", 0, 18)); // NOI18N
    ct_tB.setForeground(new java.awt.Color(255, 255, 255));
    ct_tB.setHorizontalAlignment(javax.swing.JTextField.CENTER);
    ct_tB.setBorder(null);
    ct_tB.addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyTyped(java.awt.event.KeyEvent evt) {
            ct_tBKeyTyped(evt);
        }
    });
    getContentPane().add(ct_tB, new org.netbeans.lib.awtextra.AbsoluteConstraints(590, 360, 260, 60));

    ct_tC.setFont(new java.awt.Font("Arial Black", 0, 18)); // NOI18N
    ct_tC.setForeground(new java.awt.Color(255, 255, 255));
    ct_tC.setHorizontalAlignment(javax.swing.JTextField.CENTER);
    ct_tC.setBorder(null);
    ct_tC.addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyTyped(java.awt.event.KeyEvent evt) {
            ct_tCKeyTyped(evt);
        }
    });
    getContentPane().add(ct_tC, new org.netbeans.lib.awtextra.AbsoluteConstraints(590, 470, 260, 50));

    ca_v.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
    ca_v.setForeground(new java.awt.Color(255, 255, 255));
    ca_v.setHorizontalAlignment(javax.swing.JTextField.CENTER);
```

```
ca_v.setBorder(null);
ca_v.addKeyListener(new java.awt.event.KeyAdapter() {
   public void keyTyped(java.awt.event.KeyEvent evt) {
      ca_vKeyTyped(evt);
   }
});
getContentPane().add(ca_v, new org.netbeans.lib.awtextra.AbsoluteConstraints(880, 250, 250, 30));

ca_pn.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
ca_pn.setForeground(new java.awt.Color(255, 255, 255));
ca_pn.setHorizontalAlignment(javax.swing.JTextField.CENTER);
ca_pn.setBorder(null);
ca_pn.addKeyListener(new java.awt.event.KeyAdapter() {
   public void keyTyped(java.awt.event.KeyEvent evt) {
      ca_pnKeyTyped(evt);
   }
});
getContentPane().add(ca_pn, new org.netbeans.lib.awtextra.AbsoluteConstraints(880, 310, 250, 40));

ca_po.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
ca_po.setForeground(new java.awt.Color(255, 255, 255));
ca_po.setHorizontalAlignment(javax.swing.JTextField.CENTER);
ca_po.setBorder(null);
ca_po.addKeyListener(new java.awt.event.KeyAdapter() {
   public void keyTyped(java.awt.event.KeyEvent evt) {
      ca_poKeyTyped(evt);
   }
});
getContentPane().add(ca_po, new org.netbeans.lib.awtextra.AbsoluteConstraints(880, 380, 250, 40));

ca_pg.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
ca_pg.setForeground(new java.awt.Color(255, 255, 255));
ca_pg.setHorizontalAlignment(javax.swing.JTextField.CENTER);
ca_pg.setToolTipText("");
ca_pg.setBorder(null);
ca_pg.addKeyListener(new java.awt.event.KeyAdapter() {
   public void keyTyped(java.awt.event.KeyEvent evt) {
      ca_pgKeyTyped(evt);
   }
});
getContentPane().add(ca_pg, new org.netbeans.lib.awtextra.AbsoluteConstraints(880, 440, 250, 40));

ca_tu.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
ca_tu.setForeground(new java.awt.Color(255, 255, 255));
ca_tu.setHorizontalAlignment(javax.swing.JTextField.CENTER);
ca_tu.setBorder(null);
ca_tu.addKeyListener(new java.awt.event.KeyAdapter() {
   public void keyTyped(java.awt.event.KeyEvent evt) {
      ca_tuKeyTyped(evt);
   }
});
```

```
        getContentPane().add(ca_tu, new org.netbeans.lib.awtextra.AbsoluteConstraints(880, 510, 250, 40));

        ca_to.setFont(new java.awt.Font("Arial Black", 0, 14)); // NOI18N
        ca_to.setForeground(new java.awt.Color(255, 255, 255));
        ca_to.setHorizontalAlignment(javax.swing.JTextField.CENTER);
        ca_to.setBorder(null);
        ca_to.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent evt) {
                ca_toKeyTyped(evt);
            }
        });
        getContentPane().add(ca_to, new org.netbeans.lib.awtextra.AbsoluteConstraints(880, 580, 250, 40));

        ct_boton.setText("Buscar");
        ct_boton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                ct_botonActionPerformed(evt);
            }
        });
            getContentPane().add(ct_boton, new org.netbeans.lib.awtextra.AbsoluteConstraints(660, 540, 130,
30));

        ca_boton.setText("Buscar");
        ca_boton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                ca_botonActionPerformed(evt);
            }
        });
            getContentPane().add(ca_boton, new org.netbeans.lib.awtextra.AbsoluteConstraints(940, 630, 130,
30));

                                                                                fondo.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/interfaces/imagenes/fondo2.png"))); // NOI18N
        getContentPane().add(fondo, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, -1, 697));

        pack();
    }// </editor-fold>

    private void cu_ubicacionKeyTyped(java.awt.event.KeyEvent evt) {
        // No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
        }
    }

    private void cu_actividadesKeyTyped(java.awt.event.KeyEvent evt) {
        // No admite el ingreso ni de letras ni de caracteres
        char c = evt.getKeyChar();
        if (c < 'a' || c > 'z' || c < '0' || c > '9') {
            evt.consume();
```

```
    }
}

private void cm_numMenoresKeyTyped(java.awt.event.KeyEvent evt) {
    // No admite el ingreso ni de letras ni de caracteres
    char c = evt.getKeyChar();
    if (c < 'a' || c > 'z' || c < '0' || c > '9') {
        evt.consume();
    }
}

private void ct_tAKeyTyped(java.awt.event.KeyEvent evt) {
    // No admite el ingreso ni de letras ni de caracteres
    char c = evt.getKeyChar();
    if (c < 'a' || c > 'z' || c < '0' || c > '9') {
        evt.consume();
    }
}

private void ct_tBKeyTyped(java.awt.event.KeyEvent evt) {
    // No admite el ingreso ni de letras ni de caracteres
    char c = evt.getKeyChar();
    if (c < 'a' || c > 'z' || c < '0' || c > '9') {
        evt.consume();
    }
}

private void ct_tCKeyTyped(java.awt.event.KeyEvent evt) {
    // No admite el ingreso ni de letras ni de caracteres
    char c = evt.getKeyChar();
    if (c < 'a' || c > 'z' || c < '0' || c > '9') {
        evt.consume();
    }
}

private void ca_vKeyTyped(java.awt.event.KeyEvent evt) {
    // No admite el ingreso ni de letras ni de caracteres
    char c = evt.getKeyChar();
    if (c < 'a' || c > 'z' || c < '0' || c > '9') {
        evt.consume();
    }       // TODO add your handling code here:
}

private void ca_pnKeyTyped(java.awt.event.KeyEvent evt) {
    // No admite el ingreso ni de letras ni de caracteres
    char c = evt.getKeyChar();
    if (c < 'a' || c > 'z' || c < '0' || c > '9') {
        evt.consume();
    }
}
```

```
private void ca_poKeyTyped(java.awt.event.KeyEvent evt) {
    // No admite el ingreso ni de letras ni de caracteres
    char c = evt.getKeyChar();
    if (c < 'a' || c > 'z' || c < '0' || c > '9') {
        evt.consume();
    }
}

private void ca_pgKeyTyped(java.awt.event.KeyEvent evt) {
   // No admite el ingreso ni de letras ni de caracteres
    char c = evt.getKeyChar();
    if (c < 'a' || c > 'z' || c < '0' || c > '9') {
        evt.consume();
    }
}

private void ca_tuKeyTyped(java.awt.event.KeyEvent evt) {
    // No admite el ingreso ni de letras ni de caracteres
    char c = evt.getKeyChar();
    if (c < 'a' || c > 'z' || c < '0' || c > '9') {
        evt.consume();
    }
}

private void ca_toKeyTyped(java.awt.event.KeyEvent evt) {
    // No admite el ingreso ni de letras ni de caracteres
    char c = evt.getKeyChar();
    if (c < 'a' || c > 'z' || c < '0' || c > '9') {
        evt.consume();
    }
}

private void cm_botonActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // TODO add your handling code here:
        cm_numMenores.setText(String.valueOf(acceso.getNum_menores()));
    } catch (RemoteException ex) {
        Logger.getLogger(Aquapark2.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private void ct_botonActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // TODO add your handling code here:
        ct_tA.setText(String.valueOf(acceso.controlToboganes("Tobogan A")));
        ct_tB.setText(String.valueOf(acceso.controlToboganes("Tobogan B")));
        ct_tC.setText(String.valueOf(acceso.controlToboganes("Tobogan C")));
    } catch (RemoteException ex) {
        Logger.getLogger(Aquapark2.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

```
    private void ca_botonActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            // TODO add your handling code here:
            ca_v.setText(String.valueOf(acceso.controlAforo("Vestuario")));
            ca_tu.setText(String.valueOf(acceso.controlAforo("Tumbona")));
            ca_to.setText(String.valueOf(acceso.controlAforo("Toboganes")));
            ca_po.setText(String.valueOf(acceso.controlAforo("Piscina Olas")));
            ca_pn.setText(String.valueOf(acceso.controlAforo("Piscina Ninios")));
            ca_pg.setText(String.valueOf(acceso.controlAforo("Piscina Grande")));

        } catch (RemoteException ex) {
            Logger.getLogger(Aquapark2.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    private void cu_botonActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            // TODO add your handling code here:
            String nombre_usuario = cu_usuario.getText();
            cu_ubicacion.setText(acceso.devolverActividad(nombre_usuario));
            cu_actividades.setText(acceso.devolverNumActividad(nombre_usuario));
        } catch (RemoteException ex) {
            Logger.getLogger(Aquapark2.class.getName()).log(Level.SEVERE, null, ex);
        }
    }



    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
                                                    for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
                java.util.logging.Logger.getLogger(Aquapark2.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (InstantiationException ex) {
```

```
            java.util.logging.Logger.getLogger(Aquapark2.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(Aquapark2.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(Aquapark2.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                try {
                    new Aquapark2().setVisible(true);
                } catch (NotBoundException | MalformedURLException | RemoteException ex) {
                    Logger.getLogger(Aquapark2.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        });
    }

    /**
     * hace el fondo del text field transparente
     */
    public void transparencia()
    {
        ca_pg.setBackground(new java.awt.Color(0,0,0,1));
        ca_pn.setBackground(new java.awt.Color(0,0,0,1));
        ca_po.setBackground(new java.awt.Color(0,0,0,1));
        ca_to.setBackground(new java.awt.Color(0,0,0,1));
        ca_tu.setBackground(new java.awt.Color(0,0,0,1));
        ca_v.setBackground(new java.awt.Color(0,0,0,1));
        cm_numMenores.setBackground(new java.awt.Color(0,0,0,1));
        ct_tA.setBackground(new java.awt.Color(0,0,0,1));
        ct_tB.setBackground(new java.awt.Color(0,0,0,1));
        ct_tC.setBackground(new java.awt.Color(0,0,0,1));
        cu_actividades.setBackground(new java.awt.Color(0,0,0,1));
        cu_ubicacion.setBackground(new java.awt.Color(0,0,0,1));
        cu_usuario.setBackground(new java.awt.Color(0,0,0,1));
    }

    // Variables declaration - do not modify
    private javax.swing.JButton ca_boton;
    private javax.swing.JTextField ca_pg;
    private javax.swing.JTextField ca_pn;
    private javax.swing.JTextField ca_po;
    private javax.swing.JTextField ca_to;
```

```
    private javax.swing.JTextField ca_tu;
    private javax.swing.JTextField ca_v;
    private javax.swing.JButton cm_boton;
    private javax.swing.JTextField cm_numMenores;
    private javax.swing.JButton ct_boton;
    private javax.swing.JTextField ct_tA;
    private javax.swing.JTextField ct_tB;
    private javax.swing.JTextField ct_tC;
    private javax.swing.JTextField cu_actividades;
    private javax.swing.JButton cu_boton;
    private javax.swing.JTextField cu_ubicacion;
    private javax.swing.JTextField cu_usuario;
    private javax.swing.JLabel fondo;
    // End of variables declaration
}
```