FAMILIA PROFESIONAL:	**Informática y Comunicaciones**
CICLOS FORMATIVOS:	**Desarrollo de Aplicaciones Multiplataforma**
MÓDULO:	**Programación de Servicios y Procesos**

UNIDAD 2: PROGRAMACIÓN MULTIHILO

ACTIVIDADES

ACTIVIDAD 1x01

Escribe una clase **Fibonacci** que se ejecute como en un hilo y que realice el cálculo de los N primeros números de la sucesión de Fibonacci. Esta sucesión comienza con los números 1 y 1; el siguiente elemento es la suma de los dos elementos anteriores (1, 1, 2, 3, 5, 8, 13, 21,...).

El parámetro N será indicado cuando se instancie un objeto de esta clase.

Codifica una clase **Actividad1x01** que pida al usuario un valor para N e instancie dos objetos de la clase anterior. Haz que se ejecuten en orden. Prueba con valores altos para la N.

ACTIVIDAD 1x02

Escribe una clase **Tic** y otra clase **Tac** que sean derivadas de Thread. La primera clase mostrará en pantalla 100 veces la palabra TIC, y la segunda clase hará lo mismo con la palabra TAC. Dentro del bucle, utiliza el método sleep para que dé tiempo a ver las palabras que se van mostrando y captura la excepción InterrupterException.

Codifica una clase **Actividad1x02** que cree dos objetos de las clases anteriores e intenta que se ejecuten de forma simultánea. Observa el resultado de la ejecución.

ACTIVIDAD 1x03

Tomando como partida el ejemplo del Productor-Consumidor, crea un programa para que el productor envíe las cadenas PING y PONG de forma alternativa y el consumidor tome las cadenas en ese orden y las muestre por pantalla. Se mostrarán 25 veces cada una de las cadenas. El resultado a obtener será: PING PONG PING PONG PING PONG PING PONG ...

Para ello, crea las siguientes clases: **Dato**, que contendrá la cadena a mostrar, **Productor**, **Consumidor** y **Actividad1x03**.

ACTIVIDAD 1x04

Basándote en el ejemplo del Productor-Consumidor, crea un programa con un productor para cada día de la semana y un consumidor que muestre todos los días de la semana varias veces de forma ordenada.

Para ello, crea las siguientes clases: **Dato**, **Productor**, **Consumidor** y **Actividad1x04**.

ACTIVIDAD 1x05

Codifica una clase **Saludo** que se ejecute como un hilo. Tendrá un atributo entero con su identificador de objeto y al ejecutarse mostrará un mensaje identificándose con dicho valor.

Codifica otra clase **Actividad1x05** que instancie dos objetos de la clase anterior con un semáforo y haz que la escritura del segundo sea siempre anterior a la escritura por pantalla del primero.

ACTIVIDAD 1x06

Escribe una clase llamada **Actividad1x06** que cree dos hilos que accedan simultáneamente a un array de 100 enteros. Uno de los hilos leerá del array y el otro escribirá en el mismo de forma indefinida.

El hilo **Escritor** deberá escribir el mismo valor en todos los elementos del array incrementando en uno el valor en cada llamada, es decir, la primera vez pondrá todos elementos a 1, la segunda pondrá todos los elementos del vector con el valor 2 y así sucesivamente.

El hilo **Lector** deberá comprobar que todos los números del array son iguales, mostrando un mensaje de error en caso contrario o un mensaje de "correcto" si la condición se cumple.

El código a realizar utilizará un monitor en el acceso al array.

ACTIVIDAD 1x07

Escribe una clase llamada **Actividad1x07** que simule el funcionamiento de un número determinado de cajas de un supermercado. Los clientes estarán un tiempo aleatorio comprando y con posterioridad seleccionarán de forma aleatoria en qué caja posicionarse para situarse en su cola correspondiente. Cuando les toque el turno serán atendidos procediendo al pago correspondiente e ingresando en los resultados del supermercado. Se deben crear tantos hilos como clientes hay. Los valores para el número de clientes y el número de cajas se definirán como constantes. Para simplificar la implementación, el valor de pago de cada cliente puede ser aleatorio.

Posible solución con monitores:

Clase **Resultados** con atributos de clase para almacenar las ganancias y los clientes atendidos.

Clase **Caja** que implemente una estructura dinámica tipo cola, donde el nodo esté compuesto por el identificador del cliente y una referencia al siguiente elemento. Esta clase será el monitor y tendrá métodos para añadir un cliente a la caja, para atender a un cliente y para mostrar por pantalla la cola de clientes que contiene. Todos estos métodos estarán sincronizados.

Clase **Cliente** con su identificador y un objeto de la caja en la que está ubicado. Esta clase será un hilo y cuando se ejecute hará las siguientes acciones:

- Mostrará la información sobre su identificador e indicará que está eligiendo los productos a comprar.
- Esperará un valor aleatorio de milisegundos simulando el tiempo que tarda en comprar.
- Se pondrá en la cola de la caja.
- Mostrará un mensaje indicando que está en la cola.
- Mostrará el contenido de la Caja en la que está.
- Pasará a ser atendido durante un valor aleatorio de milisegundos.
- Mostrará un mensaje indicando que ya ha sido atendido y que su compra ha finalizado.

Clase **Actividad1x07** con atributos de clase constantes para el número de clientes y el número de cajas. Esta clase creará tantos objetos Caja y tantos objetos Cliente como indiquen las constantes anteriores. Para cada uno de los clientes, se asignará de forma aleatoria el número de caja y se arrancará su ejecución. Cuando finalice la ejecución de todos los hilos cliente, se mostrará un resumen de las ganancias totales y de la cantidad de clientes atendidos.

A continuación, se muestra un ejemplo de ejecución para 10 clientes y 3 cajas:

```
Cliente 0 realizando compra
Cliente 1 realizando compra
Cliente 2 realizando compra
Cliente 3 realizando compra
Cliente 4 realizando compra
Cliente 5 realizando compra
Cliente 6 realizando compra
Cliente 8 realizando compra
Cliente 9 realizando compra
Cliente 7 realizando compra
Cliente 3 en cola con 3-
Cliente 8 en cola con 8-
Cliente 1 en cola con 1-
Cliente 3 atendido
Cliente 1 atendido
Cliente 4 en cola con 4-
Cliente 8 atendido
Cliente 6 en cola con 6-2-9-
Cliente 4 atendido
Cliente 7 en cola con 7-5-0-
Cliente 7 atendido
Cliente 5 en cola con 5-0-
Cliente 6 atendido
Cliente 2 en cola con 2-9-
Cliente 2 atendido
Cliente 9 en cola con 9-
Cliente 9 atendido
Cliente 5 atendido
Cliente 0 en cola con 0-
Cliente 0 atendido
Supermercado cerrado.
Ganancias: 83
```

ACTIVIDAD 1x08

Escribe una clase llamada **`Actividad1x08`** que simule el funcionamiento de un aparcamiento y que contenga el número de plazas del parking y el número de coches existentes en el sistema como constantes. Los coches se representarán como hilos que se ejecuten de forma concurrente.

El parking dispondrá de una única entrada y una única salida. En la entrada de vehículos habrá un dispositivo de control (Barrera) que permita o impida el acceso al parking dependiendo de las plazas de aparcamiento disponibles. Los tiempos de espera de los vehículos dentro del parking serán aleatorios. En el momento en el que un vehículo sale del parking, notifica al dispositivo de control el número de la plaza que tenía asignada y se libera la plaza que estuviera ocupando, quedando así estas nuevamente disponibles.

Posible solución con monitores:

Clase **`Barrera`** con para almacenar la ocupación de las plazas y la cantidad de plazas libres. Será el monitor y tendrá métodos para permitir la entrada, la salida y para mostrar su estado. Todos estos métodos estarán sincronizados.

Clase **`Coche`** con su identificador y un objeto de la barrera del aparcamiento. Esta clase será un hilo y cuando se ejecute hará las siguientes acciones:

- Esperará un valor aleatorio de milisegundos simulando el acceso al recinto.
- Mostrará la información sobre su identificador e indicará que va a intentar entrar al aparcamiento.
- Intentará entrar y cuando lo haga obtendrá su número de plaza.
- Mostrará un mensaje indicando dónde ha aparcado.
- Mostrará el estado del aparcamiento.
- Esperará un número aleatorio de milisegundos simulando el tiempo que permanece aparcado.
- Intentará salir del aparcamiento.
- Mostrará un mensaje indicando que ya ha salido.
- Mostrará de nuevo el estado del aparcamiento.

Clase **Actividad1x08** con atributos de clase constantes para el número de sitios disponibles y el número de coches. Esta clase creará un objeto de Barrera con la cantidad de sitios disponibles que indique la constante anterior y creará tantos objetos Coche como indiquen la otra constante.

Para instanciar cada uno de los coches, se utilizarán números correlativos y el objeto barrera creado anteriormente. Tras crear cada coche, éste arrancará su ejecución

A continuación, se muestra un ejemplo de ejecución para 5 sitios y 5 coches:

```
Coche 6 intenta entrar en parking
Parking: [0] [0] [0] [0] [0]
Coche 6 aparcado en 0
Parking: [6] [0] [0] [0] [0]
Coche 5 intenta entrar en parking
Parking: [6] [0] [0] [0] [0]
Coche 5 aparcado en 1
Parking: [6] [5] [0] [0] [0]
Coche 8 intenta entrar en parking
Parking: [6] [5] [0] [0] [0]
Coche 8 aparcado en 2
Parking: [6] [5] [8] [0] [0]
Coche 4 intenta entrar en parking
Parking: [6] [5] [8] [0] [0]
Coche 4 aparcado en 3
Parking: [6] [5] [8] [4] [0]
Coche 2 intenta entrar en parking
Parking: [6] [5] [8] [4] [0]
Coche 2 aparcado en 4
Parking: [6] [5] [8] [4] [2]
Coche 5 saliendo
Parking: [6] [0] [8] [4] [2]
Coche 8 saliendo
Parking: [6] [0] [0] [4] [2]
Coche 9 intenta entrar en parking
Parking: [6] [0] [0] [4] [2]
Coche 9 aparcado en 1
Parking: [6] [9] [0] [4] [2]
Coche 1 intenta entrar en parking
Parking: [6] [9] [0] [4] [2]
Coche 1 aparcado en 2
Parking: [6] [9] [1] [4] [2]
Coche 10 intenta entrar en parking
Parking: [6] [9] [1] [4] [2]
Coche 10 esperando
Coche 3 intenta entrar en parking
Parking: [6] [9] [1] [4] [2]
Coche 3 esperando
Coche 7 intenta entrar en parking
Parking: [6] [9] [1] [4] [2]
Coche 7 esperando
Coche 6 saliendo
Coche 10 aparcado en 0
Parking: [10] [9] [1] [4] [2]
Parking: [10] [9] [1] [4] [2]
Coche 2 saliendo
Coche 3 aparcado en 4
Parking: [10] [9] [1] [4] [3]
Parking: [10] [9] [1] [4] [3]
Coche 3 saliendo
Parking: [10] [9] [1] [4] [7]
Coche 7 aparcado en 4
Parking: [10] [9] [1] [4] [7]
Coche 7 saliendo
Parking: [10] [9] [1] [4] [0]
Coche 1 saliendo
Parking: [10] [9] [0] [4] [0]
Coche 4 saliendo
Parking: [10] [9] [0] [0] [0]
Coche 10 saliendo
Parking: [0] [9] [0] [0] [0]
Coche 9 saliendo
Parking: [0] [0] [0] [0] [0]
```

ACTIVIDAD 1x09

Escribe una clase llamada **Actividad1x09** que simule una carrera de relevos de la siguiente forma:

- Habrá 4 corredores, que esperarán de recibir el testigo para comenzar a correr. Una vez creados los hilos, se indicará que comience la carrera, con lo que uno de ellos deberá empezar a correr.
- Cuando un corredor termine de correr, mostrará un mensaje en pantalla y espera un segundo, pasando el testigo al siguiente corredor.

- Cuando el último corredor termine, el padre mostrará un mensaje indicando que todos los hijos han terminado.

A continuación, se muestra un ejemplo de ejecución:

```
Todos los hilos creados.
Comienza la carrera!
Corredor 1 comienza . . .
Paso el testigo al hilo 2
Corredor 2 comienza . . .
Paso el testigo al hilo 3
Corredor 3 comienza . . .
Paso el testigo al hilo 4
Corredor 4 comienza . . .
Terminé!
Todos los hilos terminaron.
```

ACTIVIDAD 1x10

Realiza una simulación de la atención a los clientes de una frutería mediante el uso de semáforos. Los clientes van llegando al establecimiento cada centésima y media de segundo y si el dependiente está ocupado, deberán esperar. Cuando el primer cliente llega, el dependiente ya está disponible para realizar la atención.

El tiempo de atención de cada cliente podrá ser uno de los siguientes valores expresados en segundos: 1, 1.1, 1.5, 2.1 o 2.5. Estos valores se presentarán como un vector constante y el valor concreto para un cliente se obtendrá de forma aleatoria. Dicho tiempo de espera lo decide el dependiente, no es un valor que tenga que saber el cliente.

La frutería estará abierta de forma indefinida y finalizará mediante una interrupción que enviará el programa principal para indicar que ya no quedan clientes.

Cada cliente mostrará un mensaje por pantalla en las siguientes ocasiones: cuando espere para ser atendido, cuando esté siendo atendido y cuando finalice su atención.

El dependiente mostrará un mensaje por pantalla indicando el tiempo de atención del cliente al que va a atender y el total de la cuenta del cliente, que deberá calcular el dependiente de forma aleatoria. Y finalmente, cuando pague, mostrará un mensaje informativo con el tiempo total de atención que ha realizado y el total que ha hecho de caja.

Una posible solución:

Clase **`Cliente`** que contiene su número de cliente y hará uso de dos semáforos: uno que marca la atención del dependiente y otro que marca cuando puede salir de la tienda. Esta clase será un hilo, de manera que los clientes se ejecutarán de forma simultánea. En su ejecución, tendrán que hacer lo siguiente:

- Mostar mensaje indicando su número y diciendo que están esperando su turno.
- Indicar al dependiente que están esperando para ser atendidos.
- Mostrar mensaje indicando su número y diciendo que están siendo atendidos.
- Salir de la tienda.

Clase **`Frutero`** (o Dependiente) que almacenará valores para su tiempo total de atención y la cantidad total de caja que haga. Además, hará uso de dos semáforos, al igual que la clase cliente. Esta clase será un hilo, de manera que el dependiente se ejecutará de forma simultánea a los clientes. En su ejecución, tendrán que hacer de forma indefinida lo siguiente:

- Esperará a tener un cliente a quien atender.

- Generará de forma aleatoria el tiempo de espera basado en las constantes indicadas anteriormente y realizará la espera de dicho tiempo.
- Generará de forma aleatoria el importe a pagar por el cliente.
- Mostrará los valores obtenidos anteriormente de forma aleatoria e incrementará los contadores de total de tiempo y total de caja.
- Indicará que ha finalizado su atención.

Clase **`Actividad1x10`** donde se crearán dos semáforos, uno para indicar si el dependiente está libre u ocupado y otro para indicar si el cliente ya ha sido atendido y puede salir de la tienda. En esta clase se instanciará un frutero y un número aleatorio de clientes. A continuación, se arrancarán todos los clientes hayan finalizado, se interrumpirá la ejecución del frutero finalizando de esta manera su ejecución y obteniendo el total de tiempo invertido junto con el total de caja obtenido.

A continuación, se muestra un ejemplo de ejecución para 3 clientes:

```
Cliente 1 espera su turno
Cliente 1 es atendido
Cliente 1 sale de la tienda
Cliente 2 espera su turno
Cliente 3 espera su turno
Tiempo de atención 1000
Cuenta 4.36
Cliente 2 es atendido
Cliente 2 sale de la tienda
Tiempo de atención 2500
Cuenta 8.25
Cliente 3 es atendido
Cliente 3 sale de la tienda
Frutería cerrada por falta de clientes. Tiempo total: 3500. Total caja: 12.61
```