2) a) Se desarrollaron dos servidores que atienden clientes de forma concurrente, a través del uso de hilos. Uno de ellos se encarga de procesar depósitos, y el otro, de extracciones. Ambos acceden a un recurso compartido, un archivo, que contiene el saldo de la “cuenta” a la que desean alcanzar los clientes.

Ambos servidores se valen de servidores hijos que atienden a cada cliente, siendo estos últimos los que ejecutan la acción correspondiente. En el caso del depósito, se obtiene el saldo actual de la cuenta, se suma el valor a depositar, y se sobrescribe el archivo con el nuevo saldo. Por otro lado, a la hora de realizar una extracción, el servidor hijo verifica si el saldo de la cuenta es suficiente para realizar la operación. En caso que sí, descuenta el monto indicado; de lo contrario, la cuenta se mantiene en el saldo previo.

Para poder ejecutar los programas, los pasos son:

1. Iniciamos la ejecución de los servidores: ServerDepositos.java y ServerExtracciones.java
2. Iniciamos la ejecución automática de los clientes: ejecutando Main.java se instanciará un número aleatorio de ClienteDeposito y ClienteExtraccion que realizarán operaciones sobre la cuenta (cada cliente realizará, una vez, la operación que indica su nombre). Si pasamos un número n como argumento, se instanciarán n clientes de cada tipo.

De manera alternativa al paso 2, podemos ejecutar, directamente, tantos clientes de depósito como de extracción queramos, de forma manual.,

Una vez terminada la ejecución de los clientes, se habrán generado tres archivos: cuenta.txt, logDepositos.txt y logExtracciones.txt

En el primero de estos archivos estará el importe de la cuenta. En los otros, encontramos el log de movimientos realizados sobre dicha cuenta.

El formato de las entradas es el siguiente:

abr 10, 2020 4:21:48 PM SDyPP.SDyPP\_tp2\_punto2.copy.ServerDepositos log

INFO: (1586546508260)---> Saldo anterior: 0.0 - A depositar: 920.0 - Nuevo Saldo: 920.0

Tal como puede observarse, informa el saldo anterior de la cuenta, el importe a depositar (o extraer, cuando corresponda) y el saldo posterior a la realización de la operación (que podrá ser el mismo, en caso que no haya suficiente saldo para la extracción). En celeste puede observarse la marca de tiempo en milisegundos, que nos otorga la granularidad suficiente para determinar el orden de ejecución de las operaciones.

Veremos a continuación un fragmento de ambos logs luego de la ejecución de los pasos mencionados, y cómo se producen errores al acceder al recurso compartido. El número al comienzo de cada log, indica el orden en que se ejecutaron las operaciones.

Log depósitos:

2. abr 10, 2020 4:21:48 PM SDyPP.SDyPP\_tp2\_punto2.copy.ServerDepositos log

INFO: (1586546508260)---> Saldo anterior: 0.0 - A depositar: 920.0 - Nuevo Saldo: 920.0

4. abr 10, 2020 4:21:48 PM SDyPP.SDyPP\_tp2\_punto2.copy.ServerDepositos log

INFO: (1586546508327)---> Saldo anterior: 920.0 - A depositar: 590.0 - Nuevo Saldo: 1510.0

5. abr 10, 2020 4:21:48 PM SDyPP.SDyPP\_tp2\_punto2.copy.ServerDepositos log

INFO: (1586546508337)---> Saldo anterior: 920.0 - A depositar: 560.0 - Nuevo Saldo: 1480.0

Log extracciones:

1. abr 10, 2020 4:21:48 PM SDyPP.SDyPP\_tp2\_punto2.copy.ServerExtracciones log

INFO: (1586546508215)---> Saldo anterior: 0.0 - A extraer: 690.0 - Nuevo Saldo: 0.0

3. abr 10, 2020 4:21:48 PM SDyPP.SDyPP\_tp2\_punto2.copy.ServerExtracciones log

INFO: (1586546508322)---> Saldo anterior: 920.0 - A extraer: 960.0 - Nuevo Saldo: 920.0

6. abr 10, 2020 4:21:48 PM SDyPP.SDyPP\_tp2\_punto2.copy.ServerExtracciones log

INFO: (1586546508357)---> Saldo anterior: 920.0 - A extraer: 830.0 - Nuevo Saldo: 90.0

Podemos quedarnos con los últimos 3 números en celeste, para poder comparar estados. Recordar que el número al comienzo de cada log, indica el orden en que se ejecutaron las operaciones. Los logs indican lo siguiente:

1. Se produce un intento de extracción insatisfactorio por falta de fondos. El saldo permanece en 0.

2. Se depositan 920 con éxito. El saldo pasa a 920.

3. Se produce otro intento sin éxito, porque el saldo está en 920 (lo cual es correcto) pero se intenta extraer 960. El saldo continúa en 920.

4. Un cliente deposita 590, por lo que al saldo previo de 920 se adicionan 590, dejando 1510 en la cuenta.

5. Otro proceso cliente deposita ahora 560, por lo que al saldo anterior de 1510, deberían agregarse estos 560 obteniendo 2070. Sin embargo, tal y como puede observarse, el valor resultante es 1480. El motivo radica en que al leer el saldo actual, se obtiene 920 (valor previo a 4) y no 1510 (valor correcto). Aquí se produce la primera falla.

6. Un cliente realiza una extracción satisfactoria de 830 pero en lugar de un saldo inicial correcto, lo hace sobre la base de 920. Como consecuencia, la cuenta queda en 90.

De esta manera, queda claro que el acceso concurrente sin sincronización al recurso, produce malfuncionamiento y obtenemos un resultado equivocado.

b) En una segunda versión, agregamos un proceso ServerLauncher que será el encargado de la instanciación de los servidores del punto anterior. De esta forma, ambos procesos servidores se ejecutarán como hilos del proceso principal, compartiendo el archivo cuenta. Ahora bien, para lograr que efectivamente se lleve a cabo un acceso sincronizado, los servidores deben hacer uso de la directiva “synchronized” a la hora de acceder a la cuenta, leer su valor y modificarlo.

El acceso sincronizado puede verse en los procesos hijos, ServerDepositosHijo y ServerExtraccionesHijo.

Para poder ejecutar esta segunda versión de los programas, los pasos son:

1. Iniciamos la ejecución de los servidores: a través de ServerLauncher.java
2. Iniciamos la ejecución automática de los clientes: ejecutando Main.java se instanciará un número aleatorio de ClienteDeposito y ClienteExtraccion que realizarán operaciones sobre la cuenta (cada cliente realizará, una vez, la operación que indica su nombre). Si pasamos un número n como argumento, se instanciarán n clientes de cada tipo.