UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SISTEMAS DISTRIBUIDOS

PROFESORA ING. GUADALUPE LIZETH PARRALES ROMAY

TAREA 3: IMPLEMENTACIÓN DE UN COORDINADOR DE TRANSACCIONES

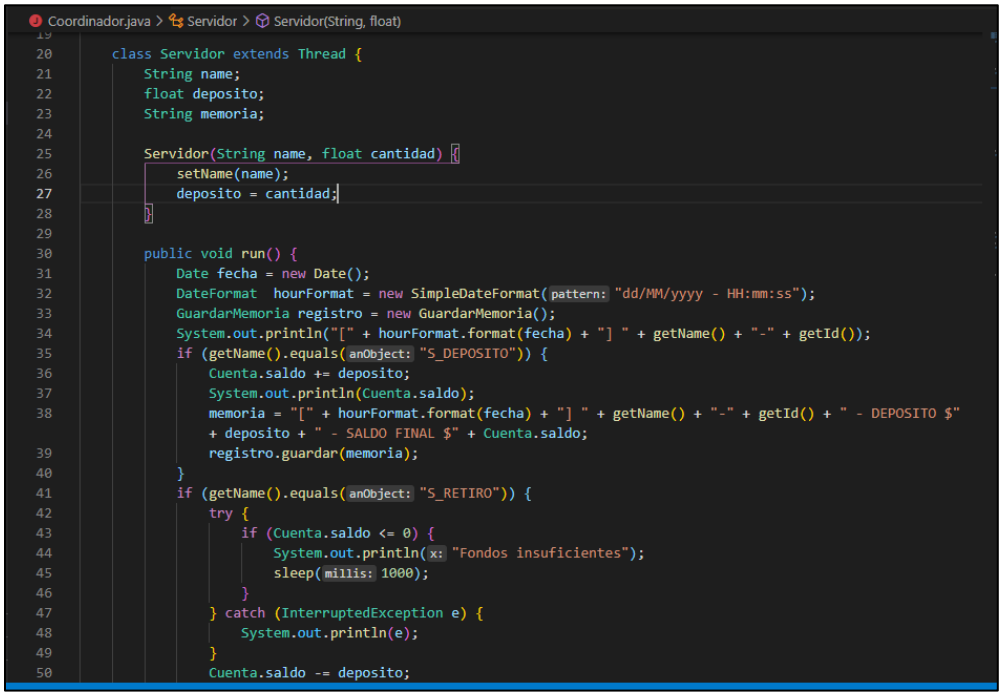
ROJAS MÉNDEZ GABRIEL

FECHA DE ENTREGA: 1 DE MAYO DE 2022

SEMESTRE 2022-2

Planteamiento de la solución.

Para poder realizar con más facilidad la implementación de los diversos métodos solicitados, se hizo uso de hilos los cuales ya cuentan con métodos que permiten modificar el nombre del hilo lo cual sirvió para determinar el origen de las transacciones, ya fuera de cliente o de servidor. Además, también existe la función que retorna el ID del hilo lo que permitió combinarse con el nombre y tener un ID único para cada hilo instanciado, por lo tanto, para esto se crearon dos clases, una denominada Cliente y la otra, Servidor.



```java
class Cliente extends Thread {
    String name;
    float deposito;
    String memoria;

    Cliente(String name, float cantidad) {
        setName(name);
        deposito = cantidad;
    }

    public void run() {
        Date fecha = new Date();
        DateFormat  hourFormat = new SimpleDateFormat(pattern: "dd/MM/yyyy - HH:mm:ss");
        GuardarMemoria registro = new GuardarMemoria();
        System.out.println("[" + hourFormat.format(fecha) + "] " + getName() + "-" + getId());
        if (getName().equals(anObject: "C_DEPOSITO")) {
            Cuenta.saldo += deposito;
            System.out.println(Cuenta.saldo);
            memoria = "[" + hourFormat.format(fecha) + "] " + getName() + "-" + getId() + " - DEPOSITO $"
            + deposito + " - SALDO FINAL $" + Cuenta.saldo;
            registro.guardar(memoria);
        }
        if (getName().equals(anObject: "C_RETIRO")) {
            try {
                if (Cuenta.saldo <= 0) {
                    System.out.println(x: "Fondos insuficientes");
                    sleep(millis: 1000);
                }
            } catch (InterruptedException e) {
                System.out.println(e);
            }
            Cuenta.saldo -= deposito;
```

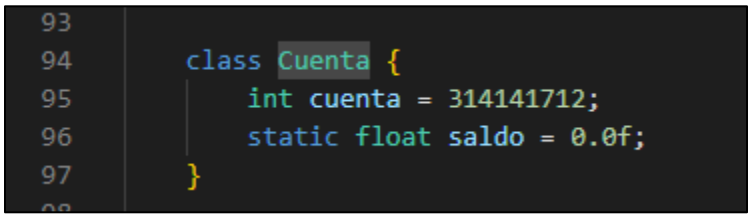
En cada una de estas clases se implementaron los métodos de deposito y retiro de manera implícita, ya que mediante el constructor de cada una de la clase se modifico el atributo de nombre del hilo y de esta manera se emplearon estructuras de control que determinaban que transacción realizar y quién debía hacerla.

Coordinador.java > Servidor > Servidor(String, float)

```java
    }

    class Coordinador {
        Run | Debug
        public static void main(String[] args) {
            Cliente cl1 = new Cliente(name: "C_DEPOSITO", cantidad: 100.00f);
            Servidor sv1 = new Servidor(name: "S_DEPOSITO", cantidad: 200.0f);
            Cliente cl2 = new Cliente(name: "C_RETIRO", cantidad: 100.00f);
            Servidor sv2 = new Servidor(name: "S_RETIRO", cantidad: 200.0f);
            cl1.start();
            sv1.start();
            System.out.println(Cuenta.saldo);
            cl2.start();
            sv2.start();
            System.out.println(Cuenta.saldo);
        }
    }
```

Con esta estructura las instancias de hilos a nivel de procesador se ejecutarían conforme ganaran tiempo de Quantum y así se podría ver las anomalías de lecturas sucias y escrituras prematuras, para que se logrará esto se declaró una clase denominada Cuenta con una variable estática para que así tanto cliente como servidor tuvieran acceso a un solo tipo de variable y no hicieran copia de esta para observar el comportamiento del valor de esta.



Finalmente, para la implementación de ir guardando las transacciones en un archivo de texto se empleó otra clase que hacia uso de la clase FileWriter a la cual se le paso un buffer para que así se capturaran los datos de cada transacción

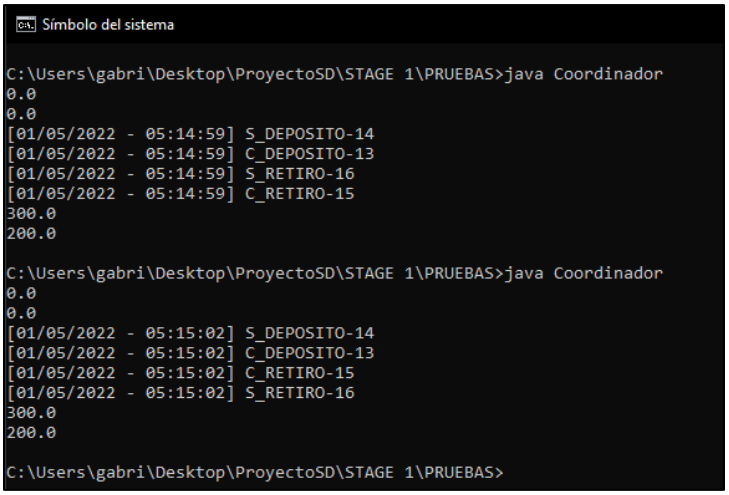
```java
class GuardarMemoria {
    public void guardar(String registro) {
        try {
            FileWriter memoria = new FileWriter( fileName: "memoria.txt",  append: true);
            for (int i = 0; i < registro.length(); i++) {
                memoria.write(registro.charAt(i));
            }
            memoria.write( str: "\n");
            memoria.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Y los resultados de la implementación de esta clase quedaron de la siguiente manera, pudiendo observar la fecha y hora de la transacción, el origen y la acción realizada y el estado de la variable estática al termino de cada transacción.

memoria.txt

```
[01/05/2022 - 04:34:39] C_RETIRO-12 - RETIRO $100.0 - SALDO FINAL $200.0
[01/05/2022 - 04:34:39] C_DEPOSITO-10 - DEPOSITO $100.0 - SALDO FINAL $300.0
[01/05/2022 - 04:34:39] S_DEPOSITO-11 - DEPOSITO $200.0 - SALDO FINAL $200.0
[01/05/2022 - 04:34:39] S_RETIRO-13 - RETIRO $200.0 - SALDO FINAL $0.0
[01/05/2022 - 04:34:43] C_DEPOSITO-10 - DEPOSITO $100.0 - SALDO FINAL $100.0
[01/05/2022 - 04:34:43] C_RETIRO-12 - RETIRO $100.0 - SALDO FINAL $0.0
[01/05/2022 - 04:34:43] S_DEPOSITO-11 - DEPOSITO $200.0 - SALDO FINAL $200.0
[01/05/2022 - 04:34:43] S_RETIRO-13 - RETIRO $200.0 - SALDO FINAL $0.0
[01/05/2022 - 04:34:52] S_DEPOSITO-11 - DEPOSITO $200.0 - SALDO FINAL $300.0
[01/05/2022 - 04:34:52] C_RETIRO-12 - RETIRO $100.0 - SALDO FINAL $200.0
[01/05/2022 - 04:34:52] S_RETIRO-13 - RETIRO $200.0 - SALDO FINAL $0.0
[01/05/2022 - 04:34:52] C_DEPOSITO-10 - DEPOSITO $100.0 - SALDO FINAL $200.0
[01/05/2022 - 04:34:55] C_RETIRO-12 - RETIRO $100.0 - SALDO FINAL $200.0
[01/05/2022 - 04:34:55] C_DEPOSITO-10 - DEPOSITO $100.0 - SALDO FINAL $0.0
[01/05/2022 - 04:34:55] S_RETIRO-13 - RETIRO $200.0 - SALDO FINAL $0.0
[01/05/2022 - 04:34:55] S_DEPOSITO-11 - DEPOSITO $200.0 - SALDO FINAL $200.0
[01/05/2022 - 04:34:57] C_DEPOSITO-10 - DEPOSITO $100.0 - SALDO FINAL $300.0
[01/05/2022 - 04:34:57] S_RETIRO-13 - RETIRO $200.0 - SALDO FINAL $100.0
[01/05/2022 - 04:34:57] C_RETIRO-12 - RETIRO $100.0 - SALDO FINAL $0.0
[01/05/2022 - 04:34:57] S_DEPOSITO-11 - DEPOSITO $200.0 - SALDO FINAL $0.0
[01/05/2022 - 04:43:47] C_RETIRO-15 - RETIRO $100.0 - SALDO FINAL $0.0
[01/05/2022 - 04:43:47] C_DEPOSITO-13 - DEPOSITO $100.0 - SALDO FINAL $0.0
[01/05/2022 - 04:43:47] S_DEPOSITO-14 - DEPOSITO $200.0 - SALDO FINAL $100.0
[01/05/2022 - 04:43:47] S_RETIRO-16 - RETIRO $200.0 - SALDO FINAL $-100.0
```

Esos son los datos de la memoria, pero también se emplearon unos printf para poder observar esos datos en tiempo de ejecución y ahí era más evidente lo de las anomalías ya mencionadas.



Y aunque los print y la ejecución de los hilos tenían un orden en el código en la consola se veía otro al igual que los valores esto debido a lo ya antes mencionado de la carrera por recursos otorgados por el sistema operativo.

```
99      class Coordinador {
            Run | Debug
100         public static void main(String[] args) {
101             Cliente cl1 = new Cliente(name: "C_DEPOSITO", cantidad: 100.00f);
102             Servidor sv1 = new Servidor(name: "S_DEPOSITO", cantidad: 200.0f);
103             Cliente cl2 = new Cliente(name: "C_RETIRO", cantidad: 100.00f);
104             Servidor sv2 = new Servidor(name: "S_RETIRO", cantidad: 200.0f);
105             cl1.start();
106             sv1.start();
107             System.out.println(Cuenta.saldo);
108             cl2.start();
109             sv2.start();
110             System.out.println(Cuenta.saldo);
111         }
112     }
```

Y así es como se logró generar un coordinador de transacciones sin algún tipo de optimización para evitar las lecturas sucias y las escrituras prematuras, pero el mismo uso de hilos permite implementar esta optimización si es requerida mediante el uso de métodos synchronized, los cuales bloquean sus respectivos atributos hasta que se termina de operar con ellos, pero esto ya será implementación para la segunda parte de este coordinador.