Alumno: Kevin Zamora Amela

Ejercicio 2:

```
public class CuentaBancaria {

    // DEFINICIÓN DE PARÁMETROS
    Persona titular;
    Double saldo;
    String numCuenta;
    String tipoCuenta;

    // CONSTRUCTORES
    public CuentaBancaria() {
    }

    public CuentaBancaria(Persona titular, Double saldo, String numCuenta, String tipoCuenta) {
        this.titular = titular;
        this.saldo = saldo;
        this.numCuenta = numCuenta;
        this.tipoCuenta = tipoCuenta;
    }

    // MÉTODOS GETTER PARA PODER ACCEDER A LOS PARÁMETROS
    public Persona getTitular() {
        return titular;
    }
    public Double getSaldo() {
        return saldo;
    }
    public String getNumCuenta() {
        return numCuenta;
    }
    public String getTipoCuenta() {
        return tipoCuenta;
    }

    // MÉTODOS SETTER PARA PODER ESCRIBIR EN LOS PARÁMETROS
    public void setTitular(String nombre, String Apellidos, String DNI) {
        Persona nuevoTitular = new Persona(nombre, Apellidos, DNI);
        titular = nuevoTitular;
    }
    public void setSaldo(double nuevoSaldo) {
        saldo = nuevoSaldo;
    }
    public void setNumCuenta(String nuevoNumCuenta) {
        numCuenta = nuevoNumCuenta;
    }
    public void setTipoCuenta(String tipoCuentaSelec) {
        tipoCuenta = tipoCuenta;
    }

    // MÉTODOS AUXILIARES
    public String toString() { // Método para poder mostrar los datos por pantalla mediante el
                               terminal
        return "Nº Cuenta: " + numCuenta + ", Titular: " + titular.nombre +
                " " + titular.apellidos + ", Saldo actual: " + saldo + "€ \n ";
    }
}
```

// **MÉTODOS PARA PODER OPERAR CON LAS DIFERENTES CUENTAS**
// (Se utilizaría una clase auxiliar para leer los diferentes valores por teclado y poder "gestionar/leer/procesar" los diferentes datos relacionados con la cuenta bancaria en cuestión)

```
public class Banco {
    CuentaBancaria cuentaObj = new CuentaBancaria() {};
    ArrayList<CuentaBancaria> listaCuentas = new ArrayList<CuentaBancaria>();
    CuentaBancaria nuevaCuenta;
    CuentaBancaria cuentaSeleccionada;
    String numCuentaIntroducido;

    // Declaración de variables de lectura o de entrada
    Scanner scanner = new Scanner(System.in);

   // Método para ingresar efectivo
    /**
     * Fúnción/Método referente a y contenedor de la funcionalidad necesaria
     * para poder ingresar efectivo en la cuenta en cuestión
     * @param cantidad Inserta la cantidad introducida/deseada
     * @exception Exception Se muestra el error "No se puede ingresar una cantidad negativa"
     * cuando se intenta ingresar un valor inferior a 0
     **/
    public void ingresar() {
        System.out.println("Introduce la cantidad a ingresar: ");
            double cantidad = Double.parseDouble(scanner.nextLine());

            if (cantidad < 0.0)
                System.out.println("No se puede ingresar una cantidad negativa");

        if(cuentaSeleccionada != null) {
            for (CuentaBancaria cuenta : listaCuentas) {
                if ((cuenta.numCuenta).equals(cuentaSeleccionada.numCuenta)) {
                    double nuevoSaldo = cuenta.getSaldo() + cantidad;
                    cuenta.setSaldo(nuevoSaldo);
                    System.out.println("El saldo de su cuenta ha sido actualizado con éxito y este es: "
                            + cuenta.getSaldo() + "\n ");
                }
            }
        } else {
            System.out.println("No se ha detectado ningún número de cuenta guardado en la presente
sesión. \n "
                    + "Por favor, introduzca el número de cuenta en donde quiere realizar el ingreso de
efectivo: ");
            numCuentaIntroducido = scanner.nextLine();
            for (CuentaBancaria cuenta : listaCuentas) {
                if ((cuenta.numCuenta).equalsIgnoreCase(numCuentaIntroducido)) {
                    cuentaSeleccionada = cuenta;
                    double nuevoSaldo = cuenta.getSaldo() + cantidad;
                    cuenta.setSaldo(nuevoSaldo);
                    System.out.println("El saldo de su cuenta ha sido actualizado con éxito y este es: "
                            + cuenta.getSaldo() + "€ \n");
                } else { System.out.println("No se ha encontrado la cuenta donde se quiere ingresar \n"); }
            }
        }
    }
```

```
    // Método para retirar efectivo
    /** @param cantidad Inserta la cantidad introducida/deseada
     * @exception Exception Muestra un error distinto en base a dos supuestos: valor
     * negativo y saldo insuficiente
     **/
    public void retirar() {
        System.out.println("Introduce la cantidad a retirar: ");
            double cantidad = Double.parseDouble(scanner.nextLine());

        if(cuentaSeleccionada != null) {
            for (CuentaBancaria cuenta : listaCuentas) {
                if ((cuenta.numCuenta).equals(cuentaSeleccionada.numCuenta)) {
                    if (cantidad > cuenta.saldo) {
                        System.out.println("No se puede retirar el importe introducido, este excede al "
                                + "saldo disponible en la cuenta"); }

                    double nuevoSaldo = cuenta.getSaldo() - cantidad;
                    cuenta.setSaldo(nuevoSaldo);
                    System.out.println("El saldo de su cuenta ha sido actualizado con éxito y este es: "
                            + cuenta.getSaldo() + "€ \n ");
                }
            }
        } else {
            System.out.println("No se ha detectado ningún número de cuenta guardado en la presente
            sesión. \n "
                    + "Por favor, introduzca el número de cuenta en donde quiere realizar el ingreso de
                      efectivo: ");
            numCuentaIntroducido = scanner.nextLine();
            for (CuentaBancaria cuenta : listaCuentas) {
                if ((cuenta.numCuenta).equalsIgnoreCase(numCuentaIntroducido)) {
                    if (cantidad > cuenta.saldo) {
                        System.out.println("No se puede retirar el importe introducido, este excede al "
                                + "saldo disponible en la cuenta"); }
                    cuentaSeleccionada = cuenta;
                    double nuevoSaldo = cuenta.getSaldo() - cantidad;
                    cuenta.setSaldo(nuevoSaldo);
                    System.out.println("El saldo de su cuenta ha sido actualizado con éxito y este es: "
                            + cuenta.getSaldo() + "€ \n");
                } else { System.out.println("No se ha encontrado la cuenta donde se quiere ingresar \n"); }
            }
        }
    }

public void getSaldo() {
        if (cuentaSeleccionada != null) {
            System.out.println(cuentaSeleccionada.getSaldo() + "€ de saldo disponible \n");
        } else {
            System.out.println("No se ha detectado ningún número de cuenta guardado en la presente
            sesión. \n Por favor, introduzca el número de cuenta en donde quiere realizar el ingreso: ");
            numCuentaIntroducido = scanner.nextLine();
            for (CuentaBancaria cuenta : listaCuentas) {
                if ((cuenta.numCuenta).equalsIgnoreCase(numCuentaIntroducido)) {
                    cuentaSeleccionada = cuenta;
                    System.out.println(cuentaSeleccionada.getSaldo() + "€ de saldo disponible \n");
                } else { System.out.println("No se ha encontrado la cuenta a consultar \n"); }
            }
        }
    }
}
```

Ejercicio 3:

Creamos la clase servidor para construir y habilitar nuestro servidor mediante el uso de un socket UDP.

```
public class Servidor {
```

Creamos un método ‘main’ para que se ejecute el código contenido en este al arrancar nuestra aplicación.

```
public static void main(String[] args) {
```

Definimos un método 'try catch' para el control de excepciones

```
    try {
```

Creamos una variable 'servidor' de tipo 'ServerSocket' y le pasamos como parámetro el puerto 1500, por poner un ejemplo.

```
        ServerSocket servidor = new ServerSocket(1500);
        System.out.println("Se ha iniciado el servidor");
```

Bucle 'mientras' para ejecutar el programa que habilita el servidor mientras sea verdadero (siéndolo siempre) o hasta que se finalice el proceso.

```
        while(true) {
```

Definición de la variable sc, a la cual se le asigna el método 'aceptar' del servidor.

```
            Socket sc = servidor.accept();
```

Creamos una instancia de la clase HiloServidor.

```
            HiloServidor hs = new HiloServidor(sc);
```

Implementamos el método 'start' del objeto 'hs'

```
            hs.start();

        }

    } catch (IOException ex) {
```

Capturamos las excepciones y las guardamos en un 'Logger'

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

  }

}
```

Creamos una clase auxiliar para contener las diferentes operaciones a realizar entre nuestro cliente y nuestro servidor, y así poder mantener la entidad 'Servidor' separada y disponible para poder conectarse otro cliente.

```
public class HiloServidor extends Thread {
```

Definición de la variable 'sc' de tipo 'socket'/(o zócalo), para habilitar la conexión

```
    private Socket sc;

    public HiloServidor(Socket sc) {
        this.sc = sc;
    }
```

Método sobreescrito 'run()', encargado de ejecutar el funcionamiento principal de nuestra aplicación y también de establecer la conexión entre cliente y servidor.

```
    @Override
    public void run() {
```

Mostramos el mensaje 'cliente conectado' e inicializamos los flujos de entrada y de salida de datos mediante las variables 'in' y 'out'.

```
        System.out.println("Cliente conectado");
        DataInputStream in = null;
        DataOutputStream out = null;
```

Controlamos las diferentes excepciones mediante un método 'try catch'

```
        try {
```

Inicializamos y asignamos los flujos de entrada/salida.

```
            */
            in = new DataInputStream(sc.getInputStream());
            out = new DataOutputStream(sc.getOutputStream());

                boolean salir = false;

                while(!salir) {

                    out.writeUTF("Introduce un mensaje a mostrar: ");
                    String mensaje = in.readUTF().trim();

                    if (mensaje == “*”) {
```

Cerramos la conexión e imprimimos el mensaje correspondiente.

```
                        sc.close();
                        System.out.println("Cliente desconectado");
                    }

                    out.writeUTF(mensaje);

                }
```

```
        } catch (IOException ex) { /** Controlamos las excepciones y guardamos
         la información de estos en un 'Logger'. */
            Logger.getLogger(HiloServidor.class.getName()).log(Level.SEVERE, null, ex);
        } finally { Definimos un método ‘finally’ para que se ejecute antes de finalizar.
            Y definimos también otro método de control de excepciones 'try catch'.
            try {
               Cerramos los flujos de datos de entrada y de salida.
                in.close();
                out.close();
            } catch (IOException ex) {
                Y a su vez, guardamos la información de los posibles errores en un
                'Logger'.
                Logger.getLogger(HiloServidor.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
```

Por último, creamos también la clase cliente para poder realizar la conexión correspondiente con el servidor.

```
public class Cliente {
    public static void main(String[] args) {
        Método 'try catch' dedicado al control de excepciones
          try {
            Creamos una instancia/llamada hacia la clase 'Socket'(o zócalo) y
            definimos el puerto y la dirección de conexión mediante uno de sus
            constructores, introduciendo en este dichos dos parámetros
            Socket sc = new Socket("localhost", 1500); Este ‘socket’ es el que define la ruta
            de acceso al servidor.
            Creamos e inicializamos los flujos de datos de entrada y salida
            DataInputStream in = new DataInputStream(sc.getInputStream());
            DataOutputStream out = new DataOutputStream(sc.getOutputStream());
            Creamos y definimos la variable de la variable 'salir' en 'falso'
            boolean salir = false;
            Creamos y inicializamos un objeto del tipo Scanner
            Scanner scanner = new Scanner(System.in);
            while(!salir) {
                  switch(mensaje) {
                     case "*":
                        salir = true;
                        break;
                     default:
                         String mensaje = in.readUTF().trim();
                         System.out.println(mensaje);
                        break;
                  }
            }
            sc.close(); Cerramos el zócalo de conexión
            Cerramos el método try catch con la gestión de la excepción IOException
        } catch (IOException ex) {
            Capturamos los posibles errores y los registramos en un 'Logger'
            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Ejercicio 4:

```
public class Hilo {

	int veces;
	String mensaje;

	Hilo (int veces, String mensaje) {
		this.veces = veces;
		this.mensaje = mensaje;
	}

	public void imprimirHilo() {
		for (i = 0; i < veces; i++) {
			System.out.println(mensaje);
		}
	}

}

public class Principal {

	Hilo hilo1 = new Hilo(3, “Kevin Zamora: Hola, buenos días”);
	Hilo hilo2 = new Hilo(4, “Kevin Zamora: ¿Se ha mostrado correctamente el
	mensaje?”);
	Hilo hilo3 = new Hilo(5, “Kevin Zamora: Un saludo”);

	hilo1.imprimirHilo();
	hilo2.imprimirHilo();
	hilo3.imprimirHilo();

}
```