Informe de la PROG07_Tarea

Interfaz Imprimible

Solo posee un método devolverInfoString que sera el que se tiene que sobrescribir en todas las clases que se implemente esta interfaz.

```java
public interface Imprimible {

    /**
     * Método que devolvera la información en forma de cadena de texto
     * @return Cadena de texto
     */
    public String devolverInfoString();
}
```

Clase Persona

La clase Persona que implementa la interface Imprimible.

El Constructor que se le tienen que pasar los tres parámetros que inician los atributos.

Todos los atributos son PRIVADOS y se usan los Getter y los Setter para manipular estos.

```java
public class Persona implements Imprimible {
    /**
     * Atributos de la clase PRIVADOS
     */
    private String nombre, apellidos, dni;

    /**
     * Constructor de la clase
     * @param nombre Nombre de la persona
     * @param apellidos Apellidos de la persona
     * @param dni DNI de la persona
     */
    public Persona(String nombre, String apellidos, String dni) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.dni = dni;
    }
```

El método que obliga la interface le sobreescribimos y le decimos que retorne los tres atributos de la clase

```java
    */
    @Override
    public String devolverInfoString() {
        return this.getNombre() + " " + this.getApellidos() + " " + this.getDni();
    }
```

Clase Abstracta CuentaBancaria

La clase impremenata la interface Imprimible.

Contiene en uno de los atributos a la clase Persona para el titular de la cuenta.

Al constructor se los parámetros son para iniciar todos los atributos de la clase.

```java
public abstract class CuentaBancaria implements Imprimible {
    /**
     * Atributos Privados
     */
    private Persona titular;
    private String iban;
    private double saldo;

    /**
     * Contructor que inicializa los atributos
     *
     * @param titular Titular de la cuenta Clase Persona
     * @param iban Numero de Cuenta IBAN
     * @param saldo Saldo en la Cuenta
     */
    public CuentaBancaria(Persona titular, String iban, double saldo) {
        this.titular = titular;
        this.iban = iban;
        this.saldo = saldo;
    }
```

Todos los Atributos son PRIVADOS y para que desde fuera se pueda acceder a estos datos tenemos GETTER y SETTER de todos los Atributos.

En el método getTitular no se retorna directamente titular para que no exista la posibilidad de perder el encapsulamiento.

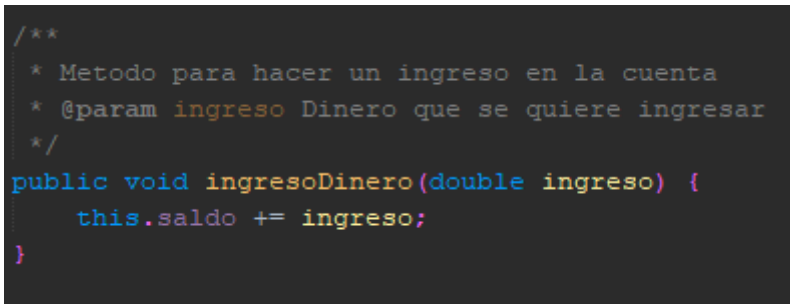
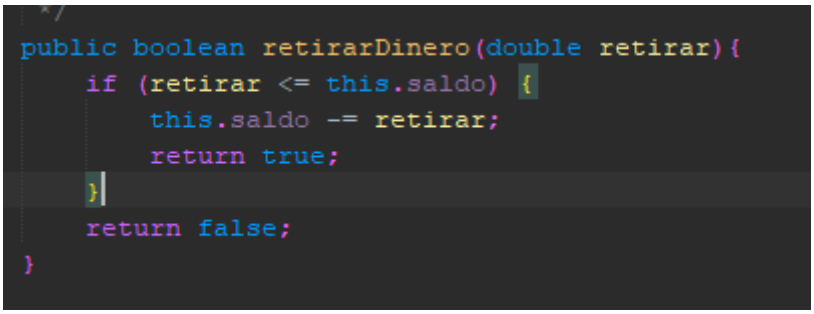
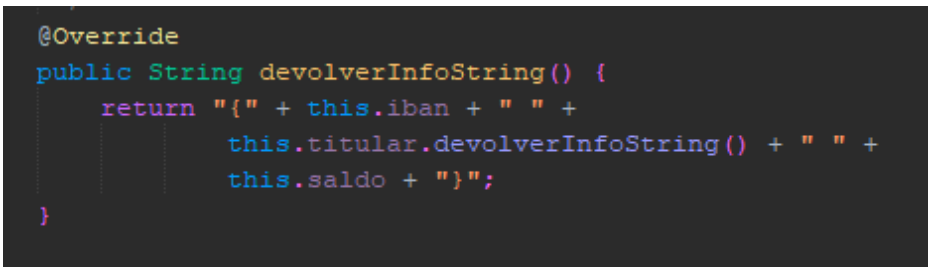
```java
    */
public Persona getTitular() {
    Persona aux = this.titular;
    return aux;
}

/**
 * Metodo setter para modificar el nombre de un titula
 * NOTA: Aunque no es habitual si es posible cambiar c
 * @param titular
 */
public void setTitular(Persona titular) {
    this.titular = titular;
}
/**
 * Método getter del atributo IBAN, número de cuenta
 * @return Devuelve el número de IBAN
 */
public String getIban() {
    return iban;
}

/**
 * Metodo setter del atributo iban
 * NOTA: Aunque no es habitual que el IBAN se tenga qu
 * @param iban IBAN o numero de cuenta
 */
public void setIban(String iban) {
    this.iban = iban;
}

/**
 * Método getter del atributo IBAN, número de cuenta
 * @return Devuelve el número de IBAN
 */
public double getSaldo() {
    return saldo;
}

/**
 * Metodo setter del atributo saldo, cambia todo el sa
 * Nota: No respeta el saldo que se tenga, para eso es
 * @param saldo El saldo que se pone
 */
public void setSaldo(double saldo) {
    this.saldo = saldo;
}
```

Para los ingresos en a cuenta Bancaria tenemos un método que lo que hace es aumentar el atributo saldo con el ingreso que se le pase. Al crear este método en la clase Abstracta que van a heredar todas las cuentas pues ya tienen este método.	 ```java /** * Metodo para hacer un ingreso en la cuenta * @param ingreso Dinero que se quiere ingresar */ public void ingresoDinero(double ingreso) { this.saldo += ingreso; } ```
Para retirar dinero al igual que para ingresar la hemos implementado en la clase padre y así todas las clases hijos lo tienen. Este método lo que hace es retirar saldo si hay suficiente, en caso de que no haya retorna falso y no lo retira.	 ```java */ public boolean retirarDinero(double retirar){ if (retirar <= this.saldo) { this.saldo -= retirar; return true; } return false; } ```
Este el método de la interface Imprimible el cual retorna los valores de los atributos,	 ```java @Override public String devolverInfoString() { return "{" + this.iban + " " + this.titular.devolverInfoString() + " " + this.saldo + "}"; } ```

Clase CuentaAhorro

Esta clase hereda de CuentaBancaria

por lo tanto en el constructor ademas de pesarle para iniciar los atributos de esta clase, también le pasamos los de la CuentaBancaria.

Y desde dentro del Constructor llamamos al constructor de la clase padre.

```java
public class CuentaAhorro extends CuentaBancaria{
    /**
     * Atributo
     */
    private float tipoInteresAnual;

    /**
     * Constructor de la Cuenta de Ahorro
     * @param titular Titular de la cuenta que es un Objeto de la clase Persona
     * @param iban Numero de IBAN nuevo
     * @param saldo Dinero de la cuenta.
     */
    public CuentaAhorro(Persona titular, String iban, double saldo, float interesAnual) {
        super(titular, iban, saldo);
        this.tipoInteresAnual = interesAnual;
    }
```

El método de la interface imprimible le sobre escribimos y desde dentro llamamos al mismo método de la clase padre y así nos da los datos de los atributos y añadimos los de esta clase.

```java
@Override
public String devolverInfoString() {
    String contenido = super.devolverInfoString();              //Llamamo a la método d

    contenido = contenido.substring(0, contenido.length()-1);   //Se quita el corchete

    return "C. de Ahorro     " + contenido + " " + this.getTipoInteresAnual() + "}";

}
```

Clase Abstracta CuentaCorriente

Esta clase hereda de CuentaBancaria y tiene un atributo de de tipo cadena de texto.

El Constructor tiene de parámetros los atributos de la clase abstracta, en el cual llama al constructor padre para iniciar esos atributos heredados.

```java
public abstract class CuentaCorriente extends CuentaBancaria{
    /**
     * Atributo lista de Entidades
     */
    private String listaEntidades;

    /**
     * Constructor de la Clase
     * @param titular Objeto que es el titular de la cuenta (Nombre, apellidos y dni)
     * @param iban numero de cuenta IBAN
     * @param saldo Saldo con el que iniciar la cuenta corriente
     * @param lista Lista de Entidades autorizadas
     */
    public CuentaCorriente(Persona titular, String iban, double saldo, String lista) {
        super(titular, iban, saldo);
        this.listaEntidades = lista;
    }
```

Tenemos un getter y un setter para el atributo de la lista de entidades.

```java
/**
 * Método getter del atributo listaEntidades
 * @return Cadena de texto con la lista de Entidades
 */
public String getListaEntidades() {
    return listaEntidades;
}
/**
 * Método setter del atributo listaEntidades
 * @param listaEntidades Establece la lista de Entidades
 */
public void setListaEntidades(String listaEntidades) {
    this.listaEntidades = listaEntidades;
}
```

Este el método de la interface Imprimible el cual retorna los valores de los atributos,

```java
@Override
public String devolverInfoString() {
    String contenido = super.devolverInfoString();

    contenido = contenido.substring(0, contenido.length()-1);

    return contenido + " " + this.getListaEntidades() + "}";
}
```

Clase CuentaCorrientePersonal

Esta clase que es para las cuenta corrientes personales hereda de Cuenta Corriente y tiene un atributo privado.

El Constructor tiene como parámetros los datos de la cuenta corriente los cuales inicializa, tanto los propios como los heredados llamando al constructor del padre.

```java
public class CuentaCorrientePersonal extends CuentaCorriente{
    /**
     * Atributos de la clase
     */
    private float comisionMantenimiento;

    /**
     * Constructor de la clase
     * @param titular Obejeto de tipo Persona (nombre, apellidos y dni)
     * @param iban numero de cuenta IBAN
     * @param saldo saldo de inicio de la cuenta
     * @param lista Lista de entidades
     * @param comisionMantenimiento Comisión de Mantenimiento
     */
    public CuentaCorrientePersonal( Persona titular, String iban, double saldo,String lista,
                                    float comisionMantenimiento) {
        super(titular, iban, saldo, lista);
        this.comisionMantenimiento = comisionMantenimiento;
    }
```

Métodos getter y setter del atributo de la clase, estos métodos son necesarios ya que el atributo es privado.

```java
/**
 * Método Getter del atributo comisionMantenimiento
 * @return Retorna el valor de la comisión de mantenimiento.
 */
public float getComisionMantenimiento() {
    return comisionMantenimiento;
}
/**
 * Método setter del atributo comisionMantenimiento
 * @param comisionMantenimiento Establece la comisión de mantenimiento.
 */
public void setComisionMantenimiento(float comisionMantenimiento) {
    this.comisionMantenimiento = comisionMantenimiento;
}
```

Sobrescribimos el método de la interface Imprimible y le añadimos los atributos de la clase a los atributos heredados y que lo devuelva en una cadena de texto.

```java
@Override
public String devolverInfoString() {
    String contenido = super.devolverInfoString();              //Llamamo a la método de l

    contenido = contenido.substring(0, contenido.length()-1);   //Se quita el corchete fin

    return "C. C. Personal    " + contenido + " " + this.getComisionMantenimiento() + "}";

}

/**
```

Clase CuentaCorrienteEmpresa

Clase para la gestión de las cuentas corrientes de empresa que hereda de la clase CuentaCorriente

Tiene una serie de atributos privados propios para la gestión de los descubiertos.

El constructor tiene por parámetros todos los datos del titular, que hereda estos atributos y los inicializa llamando al constructor padre.

```java
public class CuentaCorrienteEmpresa extends CuentaCorriente {
    /**
     * Atributos de la clase
     */
    private float tipoInteresDescubierto;
    private double maximoDescubiertoPermitido, comisionDescubierto;

    /**
     * Constructor de la clase
     *
     * @param titular Clase persona con los datos del titulas
     * @param iban  Numero de cuenta
     * @param saldo Dinero de inicio de suenta
     * @param lista Lista de Entidades
     * @param interesXDescubierto   Intereses por Descubierto
     * @param maxDescubierto        Máximo por descubierto
     * @param comisionDescubierto   Comisión de descubierto
     */
    public CuentaCorrienteEmpresa(Persona titular, String iban, double saldo, String lista,
                                  float interesXDescubierto,
                                  double maxDescubierto,
                                  double comisionDescubierto) {
        super(titular, iban, saldo, lista);
        this.tipoInteresDescubierto = interesXDescubierto;
        this.maximoDescubiertoPermitido = maxDescubierto;
        this.comisionDescubierto = comisionDescubierto;
    }
```

Los métodos getter y setter necesarios para los atributos de la clase ya que estos atributos son privados.

```java
    public float getTipoInteresDescubierto() {
        return tipoInteresDescubierto;
    }

    /**
     * Método getter del atributo maximoDescubiertoPermitido
     * @return retorna el máximo de descubierto permitido en la cuenta.
     */
    public double getMaximoDescubiertoPermitido() {
        return maximoDescubiertoPermitido;
    }

    /**
     * Método getter del atributo comisionDescubierto
     * @return Devuelve la comisión que se aplica por descubierto de la cuenta.
     */
    public double getComisionDescubierto() {
        return comisionDescubierto;
    }

    /**
     * Método getter para el atributo tipoInteresDescubierto
     * @param tipoInteresDescubierto Nuevo tipo de intereres por descubierto
     */
    public void setTipoInteresDescubierto(float tipoInteresDescubierto) {
        this.tipoInteresDescubierto = tipoInteresDescubierto;
    }
    /**
     * Método getter para el atributo maximoDescubiertoPermitido
     * @param maximoDescubiertoPermitido nuevo maximo por descubierto permitidol
     */
    public void setMaximoDescubiertoPermitido(double maximoDescubiertoPermitido) {
        this.maximoDescubiertoPermitido = maximoDescubiertoPermitido;
    }
    /**
     * Método getter para el atributo comisionDescubierto
     * @param comisionDescubierto nueva comisión por descubierto
     */
    public void setComisionDescubierto(double comisionDescubierto) {
        this.comisionDescubierto = comisionDescubierto;
```

Sobrescribimos el método de la interface Imprimible y le añadimos los atributos de la clase a los atributos heredados y que lo devuelva en una cadena de texto.

```java
    @Override
    public String devolverInfoString() {

        String contenido = super.devolverInfoString();              //Lla

        contenido = contenido.substring(0, contenido.length()-1);   //Se

        return "C. C. Empresas    " + contenido + " " +
                        this.getTipoInteresDescubierto() + " " +
                        this.getMaximoDescubiertoPermitido() + " " +
                        this.getComisionDescubierto() + "}";
    }
```

Dado que este tipo de cuenta se le permite trabajar con descubiertos (tener la cuenta con números rojos o negativos), es necesario sobrescribir el método de la clase abstracta CuentaBancaria.

En este método lo que realiza es comprobar si el dinero a retirar es menos que el Saldo si es así lo retira y devuelve true.

En caso que vaya a generar números negativos o que la cuenta ya lo este, lo comparar con el descubierto permitido, y si es menos o igual realiza la operación de retirar dinero y devuelve true.

Y como el atributo Saldo de la Clase Abstracta **CuentaBancaria** es privado accedemos a esos datos con sus getter y setter.

NOTA: Para la operación de descubierto al número de descubierto lo multiplico por -1 para que sea más fácil la comparativa de números rojo o negativos.

```java
@Override
public boolean retirarDinero(double retirar) {
    if (retirar <= this.getSaldo()) {
        this.setSaldo(this.getSaldo()-retirar);
        return true;
    } else {
        double NuevoSaldoNegativo = this.getSaldo() - retirar;
        if (NuevoSaldoNegativo >= (this.maximoDescubiertoPermitido*-1)){
            this.setSaldo(NuevoSaldoNegativo);
            return true;
        }
    }
    return false;
}
```

Clase Valida

La clase Valida (Válida) tiene un método estático pata la validación de los numero IBAN con un Expresión Regular, que obliga a que sea en un formato determinado.

```java
public class Valida {

    /**
     * Método que chequea si el IBAN tiene un formato de
     * @param iban Cadena de texto que se quiere validar
     * @return Devuelve verdadero si el formato es el co
     */
    public static boolean checkIban(String iban) {
        return iban.matches("^ES[0-9]{20}$");
    }


}
```

Clase Banco

Clase Banco para el control de todas las cuentas.

Tiene un atributo que es una constante con el número máximo de cuentas que se permite.

El array es donde almacena la clase las diferentes cuenta del banco y para ello es de tipo CuentaBancaria que es de clase abstracta de la que todas las clases de cuentas heredan como clase padre.

El constructor inicia el array con el numero máximo de cuentas y el contador de cuentas creadas a cero.

```
public class Banco {
    private final byte MAX_NUM_CUENTAS = 100;       //Número maximo de Cue

    private CuentaBancaria[] cuentasBancarias;      //Array para guardar l
    private byte num_cuentas;                       //Las cuentas que está

    /**
     * Contructor de la clase
     *
     * Inicializa el array con el Maximo de cuentas que se establece en la
     * MAX_NUM_CUENTAS y pone el num_cuentas a 0
     */
    public Banco() {
        this.cuentasBancarias = new CuentaBancaria[MAX_NUM_CUENTAS];
        this.num_cuentas = 0;
    }
```

Método informacionCuenta

El método informacionCuenta que se le pasa por parámetro el Iban, recorre el array de cuentas bancarias y si encuentra el iban llama al método devolverinfoString y retorna la cadena de texto.
En caso que no lo encuentre retorna NULL.

```
/**
 * Método para buscar una cuenta bancaria por medio de su IBAN
 *
 * @param iban numero IBAN que se desea buscar
 * @return Devuelve un cadena de texto con los datos de la cuenta o NULL sino
 */
public String informacionCuenta(String iban) {
    for (int i=0;i<this.num_cuentas;i++){
        if (this.cuentasBancarias[i].getIban().equalsIgnoreCase(iban)) {
            return this.cuentasBancarias[i].devolverInfoString();   //Si le h
        }
    }
    return null;
}
```

Método abrirCuenta

Método para Abrir Cuentas nuevas en el banco, se le pasa por parámetro un objeto de tipo CuentaBancaria. Se le puede pasar cualquiera de los tres tipos de cuenta ya que usando el polimorfismo introducirá en el array de cuentas el objeto correspondiente.

El método lo que hace es mirar si n o se ha alcanzo el máximo de cuentas y busca sino existe el IBAN y sino existe añade el nuevo objeto que se pasa por parametro.

```
/**
 * Método pata abrir una cuenta, añadiendo al array la cuenta que se
 * @param cb Cuenta Bancaria, usando el polimorfismo almacenamos en e
 * @return Devuelve true si todo es correcto y false sino ha añadido
 */
public boolean abrirCuenta(CuentaBancaria cb) {
    if (cb != null) {                                   //Verifico que el
        if (this.num_cuentas < MAX_NUM_CUENTAS) {   //Si hay menos cu
            if (this.informacionCuenta(cb.getIban()) == null) {
                this.cuentasBancarias[num_cuentas] = cb;
                num_cuentas++;
                return true;
            }
        }
    }
    return false;
}
```

Método listadoCuentas

Este método devuelve un String con todas las cuentas que tiene el banco y usando el polimorfismo accede a los diferentes **devolverInfoString** de los tres tipos de cuentas que se guardan en el array.

```java
/**
 * Método que lista todas las cuentas del Banco
 * @return Cadena de caracteres con los datos de las diferentes cuentas, si
 */
public String[] listadoCuentas() {
    String[] listado = null;
    if (this.num_cuentas != 0) {
        listado = new String[this.num_cuentas];
        for (int i=0;i<this.num_cuentas;i++) {
            listado[i] = this.cuentasBancarias[i].devolverInfoString();
        }
    }
    return listado;     //Returnamos el listado de las cuentas
}
```

Método retiradaCuenta

Este método busca el iban en el array y si le encuentra llama al método **retirarDinero** y usando el polimorfismo accede al correspondiente método de cada objeto.

```java
/**
 * Método para retirar dinero de un cuenta
 * @param iban Número de IBAN de la cuenta que se quiere retirar dinero
 * @param retira Dinero que se quiere retirar
 * @return True todo correcto false incorrecto
 */
public boolean retiradaCuenta(String iban, double retira){
    for (int i=0;i<this.num_cuentas;i++){
        if (this.cuentasBancarias[i].getIban().equalsIgnoreCase(iban)) {
            return this.cuentasBancarias[i].retirarDinero(retira);
        }
    }
    return false;   //En caso que el bucle se recorra entero es que no ha
}
```

Método obtenerSaldo

En este método buscamos la cuenta que corresponda con el IBAN y llama a **getSaldo** para que nos devuelva, en caso de que no exista devuelve -1.

NOTA: aunque no ha sido controlado no seria recomendado devolver -1 ya que en el caso de la Cuenta Corriente de Empresa si puede ser -1 por que si admite números negativos.

```java
/**
 * Método que devuelve el Saldo de la cuenta bancaria
 * @param iban Número de IBAN del que se quiere obtener el saldo.
 * @return Devuelve el saldo o si no existe -1
 */
public double obtenerSaldo(String iban) {
    for (int i=0;i<this.num_cuentas;i++){
        if (this.cuentasBancarias[i].getIban().equalsIgnoreCase(iban)) {
            return this.cuentasBancarias[i].getSaldo();   //Si le ha encontr
        }
    }
    return -1;  //En caso que no se encuentre devuelve -1
}
```

Clase Principal

Método **menu**() muestra en pantalla el menú principal

```
public static void menu() {
    System.out.println("\n");
    System.out.println("================================");
    System.out.println("           BANCO DAM            ");
    System.out.println("================================");
    System.out.println("1. Abrir una nueva cuenta");
    System.out.println("2. Listado de cuentas");
    System.out.println("3. Ver cuenta. Ingresar Dinero");
    System.out.println("4. Retirar efectivo en cuenta");
    System.out.println("5. Consultar saldo de una cuenta");
    System.out.println("6. Salir");
    System.out.println("================================");
    System.out.print("Elige una opción [1-6] ");
}
```

MenuCuentas() muestra en pantalla el menú para seleccionar el tipo de cuenta

```
public static void menuCuentas() {
    System.out.println("\n");
    System.out.println("-----------------------------");
    System.out.println("       Tipo de cuenta");
    System.out.println("-----------------------------");
    System.out.println("1. Cuenta de ahorro");
    System.out.println("2. Cuenta corriente personal");
    System.out.println("3. Cuenta corriente de empresa");
    System.out.println("4. Volver al menú principal");
    System.out.println("-----------------------------");
    System.out.print("Elige una opción[1-4] ");

}
```

Método nuevaCuenta()

Este método es usado para introducir los datos de una nueva cuenta y devuelve un objeto compatible con **CuentaBancaria**.

Al comienzo del método definimos todas las variables que necesitaremos para introducir los diferentes datos de una cuenta.

```
public static CuentaBancaria nuevaCuenta(){

    /**
     * Variable para introducir los datos de las diferentes tipos de cuentas
     */
    Scanner sc = new Scanner(System.in);
    boolean check=false;                                    //Usada para lo
    String nombre, apellidos, dni, iban;                    //Para los datc
    double dinero, maxDescubierto, comisionDescubierto;     //Para los dife
    byte opcion;                                            //Para el menú
    float tipoInteresRemunerado,comisionMantenimiento,interesDescubierto;   //pa
```

El IBAN lo repetirá hasta que introduzcamos un IBAN correcto que se valida por expresiones regulares.

```
System.out.println("Introduce el nombre:");
nombre = sc.nextLine();
System.out.println("Introduce apellidos:");
apellidos = sc.nextLine();
System.out.println("Introduce DNI:");
dni = sc.nextLine();
do {
    System.out.println("Introduce nuevo IBAN");
    iban = sc.nextLine();
    check = Valida.checkIban(iban.toUpperCase());
} while (!check);                    //Se repite mie
```

Después de introducir el saldo inicial sale el menú que muestra las diferentes cuentas que se pueden crear y el usuario tiene que seleccionar una de ellas.

```
System.out.println("Introduce saldo inicial:");
dinero = sc.nextDouble();
sc.nextLine();
menuCuentas();          //LLama al método que mue
opcion = sc.nextByte();
sc.nextLine();
switch (opcion) {                    //Los case no
```

Para la opción 1 que es de Cuenta de Ahorro se pide el dato que necesita y se crea un objeto de tipo **CuentaAhorro** y se le pasan los parámetros del constructor.

En vez de crear un objeto persona y pasar este como parámetro lo que hago en el propio parámetro llamo al constructor de Persona.

Y llama a return devolviendo el objeto creado.

NOTA: al llamar return en el case no es necesario el break, ya que el return termina con todo el método.

```
case 1: //1. Cuenta de ahorro
    System.out.println("Introduce tipo interes remunerado:");
    tipoInteresRemunerado = sc.nextFloat();
    sc.nextLine();
    /**
     * Crea el objeto de la clase Cuenta Ahorro
     */
    CuentaAhorro ca = new CuentaAhorro(new Persona(nombre,apellidos,dni),iban,dinero,tipoInteresRemunerado);
    return ca;      //Devolvemos el objeto CuentaAhorro
```

La opción 2 es para las cuentas corrientes personales por lo tanto pide el dato necesario para esa clase y crea el objeto de la clase **CuentaCorrientePersonal**.

En vez de crear un objeto persona y pasar este como parámetro lo que hago en el propio parámetro llamo al constructor de Persona.

Y llama a return devolviendo el objeto creado.

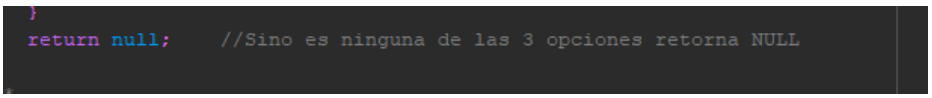
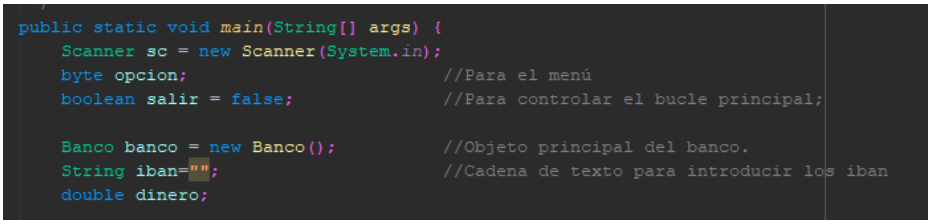
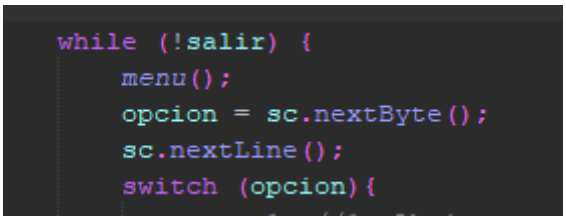
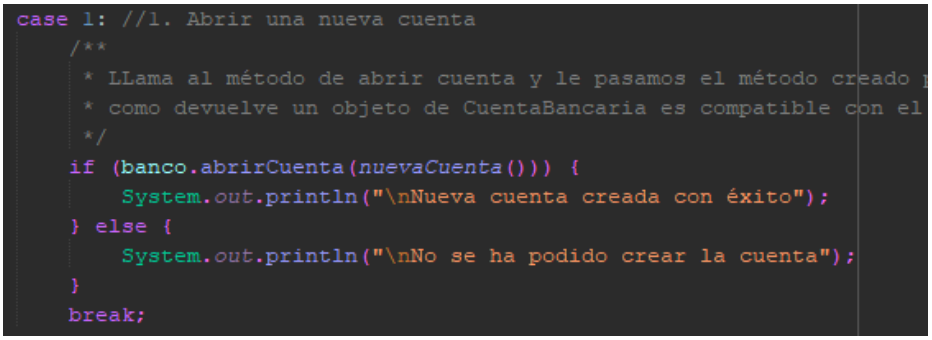
NOTA: al llamar return en el case no es necesario el break, ya que el return termina con todo el método.

```
case 2: //2. Cuenta corriente personal
    System.out.println("Introduce comisión de mantenimiento:");
    comisionMantenimiento = sc.nextFloat();
    sc.nextLine();
    /**
     * Crea el objeto de la clase Cuenta Corriente Persona
     */
    CuentaCorrientePersonal ccp = new CuentaCorrientePersonal(new Persona(nombre, apellidos, dni),iban,
                                                              dinero,null,comisionMantenimiento);
    return ccp;      //Devolvemos el Objeto CuentaCorrientePersonal
```

La opción 3 es para la cuentas corrientes de empresa y solicita los datos que son necesarios y crea el objeto de la clase **CuentaCorrienteEmpresa**.

En vez de crear un objeto persona y pasar este como parámetro lo que

```
case 3: //3. Cuenta corriente de empresa
    System.out.println("Introduce máximo de descubierto:");
    maxDescubierto = sc.nextDouble();
    sc.nextLine();
    System.out.println("Introduce comisión por descubierto:");
    comisionDescubierto = sc.nextDouble();
    sc.nextLine();
    System.out.println("Introduce interes por descubierto:");
    interesDescubierto = sc.nextFloat();
    sc.nextLine();
    /**
     * Crea el objeto de la clase de la cuenta corriente de empresa
     */
    CuentaCorrienteEmpresa cce = new CuentaCorrienteEmpresa(new Persona(nombre, apellidos, dni),iban,
                                                            dinero,null,interesDescubierto,maxDescubierto,comisionDescubierto);
    return cce;     //Devolvemos el objeto CuentaCorrienteEmpresa
```

hago en el propio parámetro llamo al constructor de Persona. Y llama a return devolviendo el objeto creado. NOTA: al llamar return en el case no es necesario el break, ya que el return termina con todo el método.	
Si el usuario no selecciona ningún tipo de cuenta retorna NULL	
Metodo Main	
Es el método principal de la aplicación, desde donde comienza el programa. Al principio del método definimos las variables que vamos a necesitar. Y tambien creamos el Objeto **banco** de la clase Banco que es donde esta toda la lógica del negocio de la aplicación.	
Con un while controlamos el bucle principal y mostramos el menú y pregunta al usuario que opción desea.	
La opción 1 es para abrir una nueva cuenta en el banco en ella llamamos al metodo del banco abrircuenta y le pasamos como parámetro el método **nuevaCuenta** que devuelve el mismo tipo que necesita **abrirCuenta**. Gestionamos los que devuelve abrirCuenta para saber si todo ha ido correctamente.	

La opción 2 es para listar cuentas se llama al método **listadoCuentas** que devuelve un array de String, si hay datos en el array le recorremos y mostramos en pantalla las cuentas.

```
case 2: //2. Listado de cuentas
    String[] listado = banco.listadoCuentas();
    if (listado != null) {
        for (int i=0;i<listado.length;i++) {  //Recorremos
            System.out.println(listado[i]);
        }
    } else {
        System.out.println("No hay cuentas");
    }
```

La opción 3 es para mostrar información de una cuenta y para ingresar en ella.

Lo primero que hace es buscar con el método de **informacionCuenta** si existe información y si es así la muestra.

Después solicita al usuario si quiere hacer un ingreso en caso que sea afirmativo solicita la cantidad y llama al método **ingresoCuenta** y gestiona el retorno para saber si todo está correcto.

```
case 3: //3. Ver cuenta. Ingresar Dinero
    System.out.println("\nIntroduce número de cuenta (IBAN):");
    iban = sc.nextLine();
    if (Valida.checkIban(iban.toUpperCase())) {     //Chequeamos que el IBAN este correctamer
        String infoCuenta = banco.informacionCuenta(iban);      //Obtenemos la información de
        if (infoCuenta != null) {       //Si existe la cuenta con el IBAN
            System.out.println(infoCuenta);
            System.out.println("\nDesea hacer un ingreso? [S/N]");  //Pregunta si quiere el t
            String pregunta = sc.next();
            if (pregunta.equalsIgnoreCase("S")) {
                System.out.println("Dinero que va a ingresar:");
                dinero = sc.nextDouble();
                sc.nextLine();
                if (banco.ingresoCuenta(iban, dinero)) {        //Realiza el igreso
                    System.out.println("Operación de INGRESO realizada con éxito.");
                }else {
                    System.out.println("No se ha podido realizar la operación.");
                }
            }
        }
    }else {
        System.out.println("Formato de IBAN Incorrecto");
    }
    break;
```

La opción 4 es para retirar el dinero de la cuenta con el método de la clase Banco **retiradaCuenta** que se le pasa por parámetro el Iban y el dinero que se deses retirar y que previamente se ha solicitado al usuario.

Controla la salida de la función para saber si es correcto.

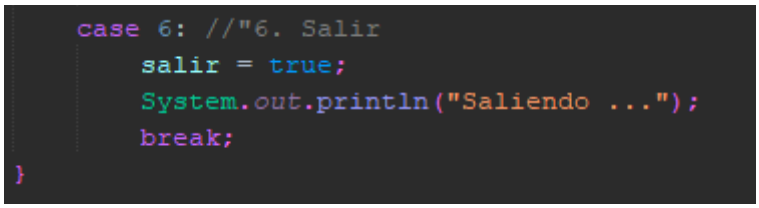
```
case 4: //4. Retirar efectivo en cuenta
    System.out.println("\nIntroduce número de cuenta (IBAN):");
    iban = sc.nextLine();
    System.out.println("Dinero que desea retirar de la cuenta:");
    dinero = sc.nextDouble();
    sc.nextLine();
    if (Valida.checkIban(iban.toUpperCase())) {     //Chequeamos que el IBAN este
        if (banco.retiradaCuenta(iban, dinero)) {   //llama al método que retira
            System.out.println("Operación de RETIRAR realizada con éxito.");
        } else {
            System.out.println("No se ha podido realizar la operación.");
        }
    }else {
        System.out.println("Formato de IBAN Incorrecto");
    }
    break;
```

La opción 5 devuelve el Saldo de una cuenta que se pide su Iban al usuario y después llamamos al método **ObtenerSaldo** y mostramos el resultado.

NOTA: al igual que mencione cuando describí este método en la clase Banco, se pude dar una curiosa situación con un Objeto de la clase **CuentaCorrienteEmpresa** que permite números negativos que genere confusión.

```
case 5: //5. Consultar saldo de una cuenta
    System.out.println("\nIntroduce número de cuenta (IBAN):");
    iban = sc.nextLine();
    if (Valida.checkIban(iban.toUpperCase())) {
        double saldo = banco.obtenerSaldo(iban);
        if (saldo == -1) {
            System.out.println("No se encuentra la cuenta: " + iban.toUpperCase());
        } else {
            System.out.println("El saldo de la cuenta: " + iban.toUpperCase() +
                               " es de " + saldo + " €");
        }
    } else {
        System.out.println("Formato de IBAN Incorrecto");
    }

    break;
```

Si la opción elegida es 6 se sale del bucle y termina el programa.	

```
    case 6: //"6. Salir
        salir = true;
        System.out.println("Saliendo ...");
        break;
}
```