Informe PROG07

Estructura inicial

Inicialmente se generan las clases del paquete `prog07`: `Principal`, `Persona`, `Banco`, `CuentaBancaria`, `CuentaAhorro`, `CuentaCorriente`, `CuentaCorrientePersonal` y `CuentaCorrienteEmpresa`. También se crea la interfaz `Imprimible`. Adicionalmente, se crea el paquete `prog07.util` con la clase `Validar`.

En `CuentaBancaria` se establece la estructura general de las cuentas, que será heredada por los distintos tipos de cuenta. Es por ello que en todas las clases hijas se incluye `extends CuentaBancaria` en la declaración de la clase. Se trata de una clase abstracta ya que no se instancian objetos de ella.

En `Banco` se incluyen las variables `listaCuentas`, que es un array de cuentas bancarias, y `numeroCuentas`, que llevará la cuenta de cuántas cuentas hay creadas.

En `Validar` se reutilizan métodos desarrollados en las unidades anteriores adaptándolo a las instrucciones de la entrega, por lo que se crea el método `esIbanCorrecto()` para comprobar que el número de cuenta tenga la estructura correcta. También se crea el método `esMayorOIgualQueCero()` para comprobar si la cantidad introducida es igual o superior a cero, pues considero que una cuenta se puede abrir con un saldo igual a 0, por lo que el método `esMayorQueCero()`, empleado al retirar o ingresar dinero, por ejemplo, no cubría todos los casos.

En `Principal` se escriben las instrucciones para instanciar el banco. Para mostrar el menú se emplea un método propio, así como para las distintas opciones del mismo.

Implementación de todos los requisitos

En esta parte se verifican todos los requisitos del enunciado.

Primeramente, se crean todos los métodos necesarios para el banco: `abrirCuenta()`, `listadoCuentas()`, `informacionCuenta()`, `ingresoCuenta()`, `retiradaCuenta()` y `obtenerSaldo()`,así como un único constructor que contendrá la creación del array de cuentas limitado a 100 y el contador de los mismos iniciado en 0.

En `Principal` desarrollo todos los apartados del menú, optando en la mayor parte de los casos por métodos propios para cada uno de los apartados. En algunos casos opto por subdividirlo, de forma que, por ejemplo, `abrirCuentaDNI()` se encargue de de la solicitud de DNI mientras que `abrirCuentaIban()` haga lo mismo con el IBAN. De esta forma, se pueden realizar comprobaciones de cada uno de los datos, sin permitir que el usuario introduzca datos que después no permitirán la creación de la cuenta. En esta clase hago

uso del polimorfismo, pues inicialmente instancio un objeto `CuentaBancaria` y, dependiendo de la elección del usuario, se crea un tipo concreto de la misma.

En `Persona` se crean los getters para obtener la información del titular, así como la obtención de dichos datos mediante la interfaz `devolverInfoString()`. Inicialmente no había implementado la interfaz, de ahí la creación de los getters. Una vez visto en el gráfico del enunciado que Persona implementa Imprimible, fue cuando incorporé el método. No quité los getters y así hago uso de ellos al solicitar los datos.

En `CuentaBancaria`, la principal tarea novedosa fue el trabajo con clases heredadas. Esto implica crear los métodos generales en ella: `getSaldo()`, `getIban()`, `hacerIngreso()`, `hacerRetirada()` y el método proveniente de la interfaz, `devolverInfoString()`.

El enunciado únicamente permite un método en la interfaz, por lo que en CuentaBancaria creo el método `devolverInfoResumida()` que devolverá el IBAN, titular y saldo de la cuenta bancaria.

En clases de cuenta se modifican los constructores para incluir los datos de la clase padre mediante super(titular, saldo, iban) y con los atributos específicos de cada una de ellas. Al mismo tiempo, también se modifica el método `devolverInfoString()` para incluir la información propia. Para ello se escribe la anotación `@Override`, que nos permite obviar el método heredado y reemplazarlo por el declarado en la clase.

En `CuentaCorrienteEmpresa`, al ser el único tipo de cuenta que permite tener un descubierto, también se reemplaza el método `hacerRetirada()` para que el saldo pueda quedarse en negativo. En este método hago uso de la ligadura dinámica, que me permite acceder a un comportamiento u otro dependiendo del tipo concreto de objeto sobre el que se está operando.

Comprobación y corrección

Una vez creado el programa, se han ejecutado diversas comprobaciones introduciendo datos erróneos para verificar que el comportamiento era el deseado. Se modifica la lógica de `abrirCuenta()`, suponiendo por defecto que la cuenta se va a poder crear ya que todos los datos vienen validados. Únicamente se rechaza la inserción cuando el banco ya está lleno o cuando ya existe el IBAN.

Inicialmente se solicitaba por teclado una sola cadena de entidades autorizadas. Como finalmente este atributo se inicia a `null`, se ha comentado el código tanto en la solicitud de datos como en la clase `CuentaCorriente`

Por último, en `CuentaCorrienteEmpresa` se modifica nuevamente el método `hacerRetirada()` para tener en cuenta la comisión que se cobra cuando se hacen operaciones con descubierto.

Una vez comprobado todo, se añadió la documentación requerida, se completaron los comentarios del código y se escribió este informe.