Detalles de la tarea de esta unidad.

Enunciado.
A lo largo de esta unidad has terminado de familiarizarte con el resto de conceptos relacionados con la **Programación Orientada a Objetos** que faltaban por ver de una manera más formal y con ejemplos explícitos: **composición**; **herencia**; **clases y métodos abstractos**; **sobrescritura de métodos**; **interfaces**; **polimorfismo**; **ligadura dinámica**, etc.

Has experimentando con todos estos conceptos y los has utilizado en pequeñas aplicaciones para comprobar su funcionamiento y su utilidad.

Una vez finalizada la unidad se puede decir que tienes ya un dominio adecuado del lenguaje Java como un lenguaje que permite aplicar todas las posibilidades de la **Programación Orientada a Objetos**. Dado ese supuesto, esta tarea tendrá como objetivo escribir una pequeña aplicación en Java empleando algunas de las construcciones que has aprendido a utilizar.

Se trata de desarrollar una aplicación Java, denominada **PROG07_Tarea** que permita gestionar varios tipos de **cuentas bancarias**. Mediante un menú se podrán elegir determinas operaciones:

1. **Abrir una nueva cuenta**.
2. **Ver un listado de las cuentas disponibles** (código de cuenta, titular y saldo actual).
3. **Obtener los datos de una cuenta concreta**.**Realizar un ingreso en una cuenta**.
4. **Retirar efectivo de una cuenta**.
5. **Consultar el saldo actual de una cuenta**.
6. **Salir de la aplicación.**

Las cuentas se irán almacenando en alguna estructura en memoria según vayan siendo creadas. Esta estructura podrá contener un máximo de 100 cuentas bancarias. Cada cuenta será un objeto de una clase que contendrá la siguiente información:

- **Titular** de la cuenta (un objeto de la clase **Persona**, la cual contendrá información sobre el titular: **nombre**, **apellidos** y **DNI**).
- **Saldo** actual de la cuenta (número real).
- **Número de cuenta** (**IBAN**).
- **Tipo de interés** anual (si se trata de una **cuenta de ahorro**).
- **Lista de entidades** autorizadas para cobrar recibos de la cuenta (si se trata de una **cuenta corriente**).
- **Comisión de mantenimiento** (para el caso de una **cuenta corriente personal**).
- **Tipo de interés por descubierto** (si es una **cuenta corriente de empresa**).
- **Máximo descubierto permitido** (si se trata de una **cuenta corriente de empresa**)

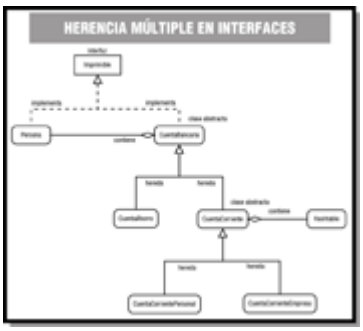
Las **cuentas bancarias** pueden ser de dos tipos: **cuentas de ahorro** o bien **cuentas corrientes**.

- Las **cuentas de ahorro** son **remuneradas** y tienen un determinado **tipo de interés**.
- Las **cuentas corrientes** no son remuneradas, pero tienen asociada una **lista de entidades autorizadas** para cobrar **recibos domiciliados** en la cuenta.

Dentro de las **cuentas corrientes** podemos encontrar a su vez otros dos tipos:

- Las **cuentas corrientes personales**, que tienen una **comisión de mantenimiento** (una cantidad fija anual).
- Las **cuentas corrientes de empresa**, que no la tienen. Además, las **cuentas de empresa** permiten tener una cierta cantidad de **descubierto** (**máximo descubierto permitido**) y por tanto un **tipo de interés por descubierto** y una **comisión fija** por **cada descubierto** que se tenga. Es el único tipo de cuenta que permite tener **descubiertos**.

Aquí tienes un ejemplo de una posible estructura de clases para llevar a cabo la aplicación:



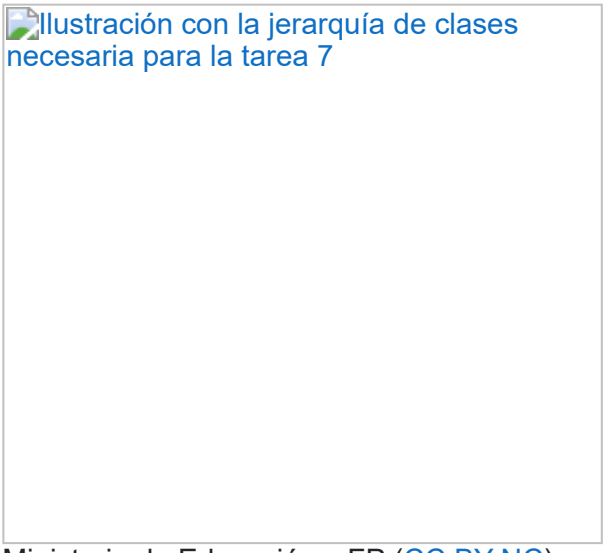


Ilustración con la jerarquía de clases necesaria para la tarea 7

Cuando se vaya a abrir una nueva **cuenta bancaria**, el **usuario de la aplicación** (**empleado del banco**) tendrá que solicitar al **cliente**:

- **Datos personales**: **nombre**, **apellidos y DNI**.
- **Tipo de cuenta** que desea abrir: **cuenta de ahorro**, **cuenta corriente personal** o **cuenta corriente de empresa**.
- **Saldo inicial**.

Además de esa información, el **usuario de la aplicación** deberá también incluir:

- **Número de cuenta** (**IBAN**) de la nueva cuenta, el cual se validará con una expresión regular y deberá tener formato ESNNNNNNNNNNNNNNNNNNNN, donde N es un dígito del 0 al 9.
- **Tipo de interés** de remuneración, si se trata de una **cuenta de ahorro**.
- **Comisión de mantenimiento**, si es una **cuenta corriente personal**.
- **Máximo descubierto permitido**, si se trata de una **cuenta corriente de empresa**.
- **Tipo de interés por descubierto**, en el caso de una **cuenta corriente de empresa**.
- **Comisión fija por cada descubierto**, también para el caso de una **cuenta corriente de empresa**.

El programa que escribas debe cumplir al menos los siguientes requisitos:

- Para almacenar los objetos de tipo cuenta deberás utilizar un array.
- Para trabajar con los datos personales, debes utilizar una clase **Persona** que contenga la información sobre los datos personales básicos del cliente (**nombre**, **apellidos**, **DNI**).
- Para guardar las entidades autorizadas en las cuentas corrientes utiliza una cadena de caracteres.

En cuanto a la organización del código, se deberán crear las siguientes clases:

- **Principal:** Contendrá el método main y todo el código de interacción con el usuario (lectura de teclado y salida por pantalla).
- **Banco**: mantendrá como atributo la estructura que almacena las cuentas. Dispondrá de los siguientes métodos:
  - **Constructor o constructores**.
  - **abrirCuenta**: recibe por parámetro un objeto `CuentaBancaria` y lo almacena en la estructura. Devuelve `true` o `false` indicando si la operación se realizó con éxito.

    - **listadoCuentas**: no recibe parámetro y devuelve un array donde cada elemento es una cadena que representa la información de una cuenta.
    - **informacionCuenta**: recibe un iban por parámetro y devuelve una cadena con la información de la cuenta o null si la cuenta no existe.
    - **ingresoCuenta**: recibe un iban por parámetro y una cantidad e ingresa la cantidad en la cuenta. Devuelve `true` o `false` indicando si la operación se realizó con éxito.
    - **retiradaCuenta**: recibe un iban por parámetro y una cantidad y trata de retirar la cantidad de la cuenta. Devuelve `true` o `false` indicando si la operación se realizó con éxito.
    - obtenerSaldo: Recibe un iban por parámetro y devuelve el saldo de la cuenta si existe. En caso contrario devuelve -1.
- Todas clases e interfaces necesarias para representan la jerarquía de cuentas.
    - La interfaz `Imprimible` tan solo declarará el `devolverInfoString`, que devolverá la información de una cuenta como una cadena de caracteres.
- Otras clases que pudieran ser de utilidad.

El **código fuente** Java de **cada clase** debería incluir **comentarios** en cada **atributo** (o en cada conjunto de atributos) y **método** (o en cada conjunto de métodos del mismo tipo) indicando su **utilidad**.
El **programa principal** (**clase principal**) también debería incluir algunos comentarios explicativos sobre su funcionamiento y la utilización de objetos de las distintas clases utilizadas.

Además del programa deberás escribir también un **informe** con todas las consideraciones oportunas que se necesiten para entender cómo has realizado la tarea.

Se debe entregar el proyecto de Netbeans completo. Para empaquetar un proyecto en Netbeans, utiliza la opción **File - Export Project** de Netbeans: generarás un fichero .zip con el contenido completo del proyecto.
Criterios de puntuación. Total 10 puntos.

Para poder empezar a aplicar estos criterios es necesario que la aplicación compile y se ejecute correctamente en un ordenador. En caso contrario la puntuación será directamente de 0,00.

Criterios de puntuación.

Existe una clase `CuentaBancaria` base que proporciona los **atributos** y **métodos** necesarios para cualquier tipo de cuenta bancaria genérica. La clase funciona correctamente.	1,00
Existe una clase `CuentaAhorro` que proporciona los **atributos** y **métodos** necesarios para trabajar con una **cuenta de ahorro**. La clase funciona correctamente.	1,00
Existe una clase `CuentaCorrientePersonal` que proporciona los **atributos** y **métodos** necesarios para trabajar con una **cuenta de corriente personal**. La clase funciona correctamente.	1,00
Existe una clase `CuentaCorrienteEmpresa` que proporciona los **atributos** y **métodos** necesarios para trabajar con una **cuenta de corriente de empresa**. La clase funciona correctamente.	1,00
Exite una clase **Banco** que proporcione la funcionalidad.	1,00
La estructura del código es correcta.	1,50
La clase **Principal** implementa la funcionalidad correctamente.	1,50
La **información de un titular** no es almacenada en objetos de la clase **Persona** que a su vez son almacenados dentro de los objetos de cuenta bancaria.	-1,00
Se utiliza el polimorfismo y la ligadura dinámica para trabajar con las cuentas bancarias y funciona correctamente.	2,00
No se han incluido **comentarios** en las clases tal y como se ha pedido en el enunciado.	-1,00
No se han incluido **comentarios** apropiados en el **programa principal** describiendo el funcionamiento de éste.	-1,00
No se ha entregado el informe explicativo.	-1,00
El programa principal **no** es capaz de crear objetos de alguno de los tres tipos de cuentas bancarias solicitados (**cuenta de ahorro, cuenta corriente personal** o **cuenta corriente de empresa**).	-5,00
Alguna de las opciones de menú pedidas en el enunciado (menú del programa principal) **no** funciona correctamente.	-1,00 por cada opción
Total (máximo)	**10,00**

Dado que algunos criterios de puntuación son negativos, podría suceder que el balance final fuera negativo. En tal caso la puntuación final será simplemente de 0,00. Si el balance final es positivo y mayor que 10,00, la puntuación final quedará como 10,00.

Recursos necesarios para realizar la Tarea.

- Ordenador personal.
- JDK y JRE de Java SE 11 o posterior.
- Netbeans IDE 11 o posterior.
- Conexion a Internet.

Recomendaciones

- Antes de comenzar a escribir código, lee con detenimiento varias veces el enunciado.
- Trata de entender la jerarquía de clases que se muestra en la especificación, es fundamental para comprender de qué clases e interfaces se compone la aplicación.
- Ves probando la aplicación a medida que implementas. Es fundamental asegurarnos de que las partes que implementamos funcionan para poder hacer comprobaciones en nuevas partes de la aplicación.
- Un buen punto de partida es implementar la estructura de clases, comenzando por la parte superior.