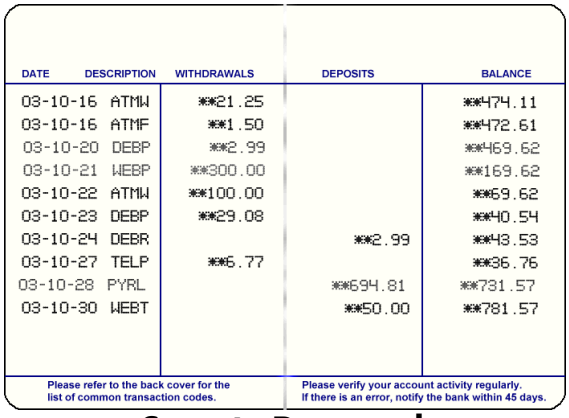
1.1.- Ejercicio 1: uso de cuentas bancarias.



DATE	DESCRIPTION	WITHDRAWALS	DEPOSITS	BALANCE
03-10-16	ATMW	**21.25		**474.11
03-10-16	ATMF	**1.50		**472.61
03-10-20	DEBP	**2.99		**469.62
03-10-21	WEBP	**300.00		**169.62
03-10-22	ATMW	**100.00		**69.62
03-10-23	DEBP	**29.08		**40.54
03-10-24	DEBR		**2.99	**43.53
03-10-27	TELP	**6.77		**36.76
03-10-28	PYRL		**694.81	**731.57
03-10-30	WEBT		**50.00	**781.57

Please refer to the back cover for the list of common transaction codes.

Please verify your account activity regularly. If there is an error, notify the bank within 45 days.

La clase **`CuentaBancaria`** implementa un modelo simplificado de cuenta para gestionar saldos, descubiertos, ingresos, extracciones, transferencias, embargos, etc. Es importante que le eches un vistazo a la **documentación javadoc** de esta clase antes de empezar a trabajar con ella. Así podrás ver qué tipo de información contiene, qué constructores proporciona, qué restricciones se le pueden aplicar, cuáles son los métodos disponibles, etc. Recuerda que tendrás que hacer el **`import`** correspondiente para poder usar esta clase en tu programa.

Para practicar con la creación y uso de objetos de este tipo, vamos a trabajar con **tres cuentas bancarias**: una cuenta que llamaremos "*privada*", otra cuenta que llamaremos "*conjunta*" y una tercera cuenta que llamaremos "*familiar*". En los tres casos se tratará de objetos instancia de la clase **`CuentaBancaria`**.

En el proyecto de trabajo que se proporciona para esta tarea dispones de un programa **`Ejercicio01`** que te servirá como base y donde tendrás que llevar a cabo las siguientes acciones paso a paso:

1. **Declarar tres variables referencia** a objetos instancia de la clase **`CuentaBancaria`**. Podrías llamarlas, por ejemplo: **`cuentaPrivada`**, **`cuentaConjunta`** y **`cuentaFamiliar`**.
2. **Instanciar tres objetos de la clase `CuentaBancaria`** a los cuales apuntarán cada una de las variables anteriores. Pero antes, haremos un par de intentos con errores para comprobar que funciona el lanzamiento de excepciones por parte de sus constructores:
    1. **Intentar crear una cuenta con fecha no válida** (**01/09/2027**). Recuerda que para instanciar un objeto de una determinada clase debes invocar a un constructor de esa clase mediante el

operador **`new`**. Dado que en este caso sabes que la llamada al constructor va a fallar seguro, no tendrás más remedio que usar un bloque **`try-catch`** para capturar la excepción y gestionar ese error mostrando por pantalla el mensaje de error que te proporcione la excepción. Si no lo hace tu código, lo hará la máquina virtual de Java y tu programa se detendrá abruptamente ante el error, cosa que nunca debe suceder. Tu código siempre debe tener previsto este tipo de posibles errores.

2. **Intentar crear otra cuenta con un saldo no válido** (**-200.00 euros**). Nuevamente, tendrás que usar otro bloque **`try-catch`** para capturar la excepción que se pueda producir al invocar al constructor para evitar que tu programa "aborte". No olvides mostrar el texto del error por pantalla.
3. **Crear una cuenta válida con un saldo inicial de 1000.00 euros, con fecha de creación a 1 de julio de 2021 y -200.00 euros de límite de descubierto**. Utiliza para ello el **constructor de tres parámetros**. Lo que devuelva el operador **`new`** al invocar al constructor es lo que tendrás que asignar a la variable referencia con la cual podrás manipular el objeto. En este caso será la variable **`cuentaPrivada`**.
4. **Crear una cuenta válida con un saldo inicial de 200.00 euros y con fecha de creación a 1 de julio de 2021**, sin indicar nada más. Utiliza para ello el **constructor de dos parámetros**. En este caso, utilizarás la variable **`cuentaConjunta`** para apuntar a ese nuevo objeto recién creado tras la llamada al constructor.
5. **Crear una cuenta válida con los valores por omisión**. Utiliza para ello el **constructor sin parámetros**. En este caso, utilizarás la variable **`cuentaFamiliar`** para apuntar al nuevo objeto creado. Recuerda que cuando decimos que una variable de tipo referencia va a "apuntar" a un objeto significa simplemente que va a almacenar en una variable (de tipo referencia, obviamente) aquello que devuelva el operador **`new`** tras la invocación al constructor (es decir, una referencia a la zona de memoria donde se encuentran realmente los atributos del objeto).

3. **Obtener la siguiente información de la cuenta privada y mostrarla por pantalla**. Para ello tendrás que usar algunos de sus métodos *getter*:
    1. **Identificador** de cuenta.
    2. **Fecha de creación** de la cuenta.
    3. **Límite de descubierto** de la cuenta. Con **dos decimales**.

    4. Si la cuenta está **embargada** o no.
    5. Si la cuenta está en **descubierto** o no.
    6. El **número de días** que lleva la cuenta abierta.
4. Llevar a cabo las siguientes **operaciones sobre las cuentas**:
    1. **Ingresar 100.00 euros** en la **cuenta familiar**.
    2. **Extraer 100.00 euros** de la **cuenta conjunta**.
    3. **Transferir 1100.00 euros** de la **cuenta privada a la familiar**.
5. Muestra **información la información sobre el estado final de cada una las cuentas** que proporciona el método **`toString`**):
    1. Estado final de la **cuenta privada**.
    2. Estado final de la **cuenta conjunta**.
    3. Estado final de la **cuenta familiar**.

IMPORTANTE: uso de `System.out.printf`

Para la realización de **este ejercicio y el resto de ejercicios de esta tarea**, siempre que tengas que mostrar algo por pantalla, deberás hacerlo usando la herramienta de E/S **`System.out.printf`** en lugar de **`System.out.println`** o **`System.out.print`**. Es decir, tendrás que usar la salida con formato utilizando los indicadores de formato **`%s`**, **`%d`**, **`%f`**, etc.

El único caso en el que podrás usar **`print`** o **`println`** será cuando vayas a escribir por pantalla un literal de cadena (un texto entre comillas), sin mostrar el contenido de ninguna variable. En estos casos el uso de **`printf`** no ofrece ninguna ventaja adicional.