Programación

7.2. Tarea UT07: Cuenta bancaria.

Pretendemos diseñar una pequeña aplicación para administrar las cuentas de una entidad bancaria. Sabiendo que las entidades financieras utilizan distintos tipos de cuenta, la herencia puede ser una solución para reutilizar código. La idea es diseñar una aplicación para administrar las cuentas corrientes y de ahorro de los clientes de una entidad financiera. Como ambas cuentas tienen bastantes cosas en común, hemos decidido agrupar éstas en una clase **CCuenta** de la cual derivaremos las cuentas específicas que vayan surgiendo. Sería absurdo crear objetos de **CCuenta**; más bien la intención es que se agrupe el código común que heredarán sus subclases, razón por la cual la declararemos abstracta. Pensemos entonces inicialmente en el diseño de la clase **CCuenta**. Después de un análisis de los factores que intervienen en una cuenta en general, llegamos a la conclusión de que los atributos y métodos comunes a cualquier tipo de cuenta son los siguientes:

Atributos	Significado
nombreTitular	Dato de tipo String que almacena el nombre del titular de la cuenta.
numCuenta	Dato de tipo String que almacena el número de cuenta.
Saldo	Dato de tipo double que almacena el saldo de la cuenta.
TipoDeInterés	Dato de tipo double que almacena el tipo de interés.

Método	Significado
CCuenta	Es el constructor de la clase. Inicia los datos nombre, cuenta, saldo y tipo de interés. Haciendo uso de los métodos de asignación de atributos. Define también un constructor por defecto sin código.
setNombre	Permite asignar el nombre.
getNombre	Retorna el nombre.
setCuenta	Asigna el número de cuenta.
getCuenta	Retorna el número de cuenta.
getSaldo	Retorna el saldo de la cuenta.
comisiones	Es un método abstracto sin parámetros que será redefinido en las subclases.
ingreso	Es un método que tiene un parámetro cantidad de tipo double que añade la cantidad especificada al saldo actual de la cuenta.
reintegro	Es un método que tiene un parámetro cantidad de tipo double que resta la cantidad especificada del saldo actual de la cuenta.
setTipoInteres	Asigna el tipo de interés.
getTipoInteres	Retorna el tipo de interés
intereses	Método abstracto. Calcula los intereses producidos

El diseño de la subclase **CCuentaAhorro** es el siguiente:

Además de los miembros heredados de la clase **CCuenta**, incorporamos:

Atributos	Significado

cuotaMantenimiento Dato de tipo double que almacena la comisión que cobrará la entidad bancaria por el mantenimiento de la cuenta.

Método	Significado
CCuentaAhorro	Es el constructor de la clase. Inicia loa atributos de la misma. Recuerda que los constructores no se heredarán.
setCuotaManten	Establece la cuota de mantenimiento de la cuenta
getCuotaManten	Devuelve la cuota de mantenimiento.
comisiones	Método que se ejecuta los días uno de cada mes para cobrar el importe al mantenimiento de la cuenta. El día se obtendrá usando el método: get(Calendar.DAY_OF_MONTH) de la clase GregorianCalendar (paquete java.util). La cuota de mantenimiento se cargará en la cuenta.
intereses	Método que permite calcular el importe correspondiente a los intereses mensuales producidos. . El día del mes se obtendrá haciendo uso de la clase GregorianCalendar. Acumulamos los intereses por mes sólo los días 1 de cada mes. Si el día no es primero de mes retornamos 0.0. Una vez calculados los intereses, se ingresan en la cuenta.

Vamos a aumentar la jerarquía de clases de la clase **CCuenta**, derivando una subclase denominada **CCcuentaCorriente** de la clase abstracta **CCuenta**. Así mismo diseñaremos una clase denominada **CCuentaCorrienteConIn** derivada de **CCuentaCorriente**.

La clase **CCuentaCorriente** es una nueva clase que hereda de la clase **CCuenta**. Por lo tanto, tendrá todos los miembros de su superclase, a los que añadiremos los siguientes:

Atributo	Significado
transacciones	Dato de tipo int que almacena el número de transacciones efectuadas sobre esa cuenta.
importePorTrans	Dato de tipo double que almacena el importe que la entidad bancaria cobrará por cada transacción.
transExentas	Dato de tipo int que almacena el número de transacciones gratuitas.

-

Método	Significado
CCuentaCorriente	Constructor de la clase. Inicia sus datos miembro, excepto transacciones que inicialmente vale 0.
decrementarTransacciones	Decrementa en 1 el número de transacciones.
setImportePorTrans	Establece el importe por transacción.
getImportePorTrans	Devuelve el importe por transacción.
setTransExentas	Establece el número de transacciones exentas.
getTransExentas	Devuelve el número de transacciones exentas.
ingreso	Añade la cantidad especificada al saldo actual de la cuenta e incrementa el número de transacciones. Dado que éste método se llama igual que otro definido en la superclase CCuenta, para acceder al método de la superclase y reutilizar su código tendremos que utilizar la palabra reservada super. Por ejemplo: super.ingreso(cantidad);
reintegro	Resta la cantidad especificada del saldo actual de la cuenta e incrementa el número de transacciones. Actuaremos igual que en ingreso para reutilizar el código de la superclase CCuenta.
comisiones	Se ejecuta los días uno de cada mes para cobrar el importe de las transacciones efectuadas que no estén exentas y pone el número de transacciones a cero.
intereses	Se ejecuta los días uno de cada mes para calcular el importe correspondiente a los intereses mensuales producidos y añadirlos al saldo. Hasta 3000 euros. al 0.5%. El resto al interés establecido. Retorna 0.0 para el resto de los días. Este ingreso no debe incrementar las transacciones

Procediendo de forma similar a como lo hemos hecho para las clases **CCuentaAhorro** y **CCuentaCorriente**, construimos a continuación la clase **CCuentaCorrienteConIn** (cuenta corriente con intereses) derivada de **CCuentaCorriente**.

Supongamos que este tipo de cuenta se ha pensado para que acumule intereses de forma distinta a los otros tipos de cuenta, pero para obtener una rentabilidad mayor respecto a **CCuentaCorriente**.

Digamos que se trata de una cuenta de tipo **CCuentaCorriente** que precisa un saldo mínimo de 3000 euros. para que pueda acumular intereses. Según esto **CCuentaCorrienteIn**, además de los miembros heredados, sólo precisa implementar sus constructores y variar el método intereses:

Método	Significado
CCuentaCorrienteConIn	Constructor de la clase.
intereses	Permite calcular el importe mensual correspondiente a los intereses producidos. Precisa un saldo mínimo de 3000 euros.

Importante:

Una subclase que redefina un método heredado sólo tiene acceso a su propia versión y a la publicada por su superclase directa. Por ejemplo, las clases **CCuenta** y **CCuentaCorriente** incluyen cada una su versión del método ingreso; y la subclase **CCuentaCorrienteConIn** hereda el método ingreso de **CCuentaCorriente**. Entonces, **CCuentaCorrientecConIn**, además de su propia versión, sólo puede acceder a la versión de su superclase directa por medio de la palabra **super** (en este caso ambas versiones son la misma), pero no puede acceder a la versión de su superclase indirecta **CCuenta** (**super.super** no es una expresión admitida por el compilador de Java).

Prueba el diseño de las clases con un fuente que contenga el módulo principal, por ejemplo:

```
public class Test
{
 public static void main(String[] args) throws IOException
 {
  CCuentaAhorro cliente01 = new CCuentaAhorro(
            "Angel Lillo", "111/6666", 10000, 3.5, 30);

  System.out.println(cliente01.getNombre());
  System.out.println(cliente01.getCuenta());
  System.out.println(cliente01.getSaldo());
  System.out.println(cliente01.getTipoDeInterés());
  System.out.println(cliente01.intereses());

  CCuentaCorrienteConIn cliente02 = new CCuentaCorrienteConIn();
  cliente02.setNombre("Ainhoa");
  cliente02.setCuenta("1234567890");
  cliente02.setTipoDeInterés(3.0);
  cliente02.setTransExentas(0);
```

```
    cliente02.setImportePorTrans(1.0);


    cliente02.ingreso(20000);
    cliente02.reintegro(10000);
    cliente02.intereses();
    cliente02.comisiones();
    System.out.println(cliente02.getNombre());
    System.out.println(cliente02.getCuenta());
    System.out.println(cliente02.getSaldo());


  }
}
```

Estado de la entrega

Estado de la entrega	Enviado para calificar
Estado de la calificación	Sin calificar
Fecha de entrega	lunes, 7 de marzo de 2022, 23:59
Tiempo restante	1 hora 2 minutos
Última modificación	lunes, 7 de marzo de 2022, 22:27
Archivos enviados	rodriguez.vasquez.yenniferjohanna.PRO.U7.7z7 de marzo de 2022, 22:27
Comentarios de la entrega	Comentarios (0)

◄ 7.1. Test UT07.

Ir a...

Dudas sobre la UT08 ►