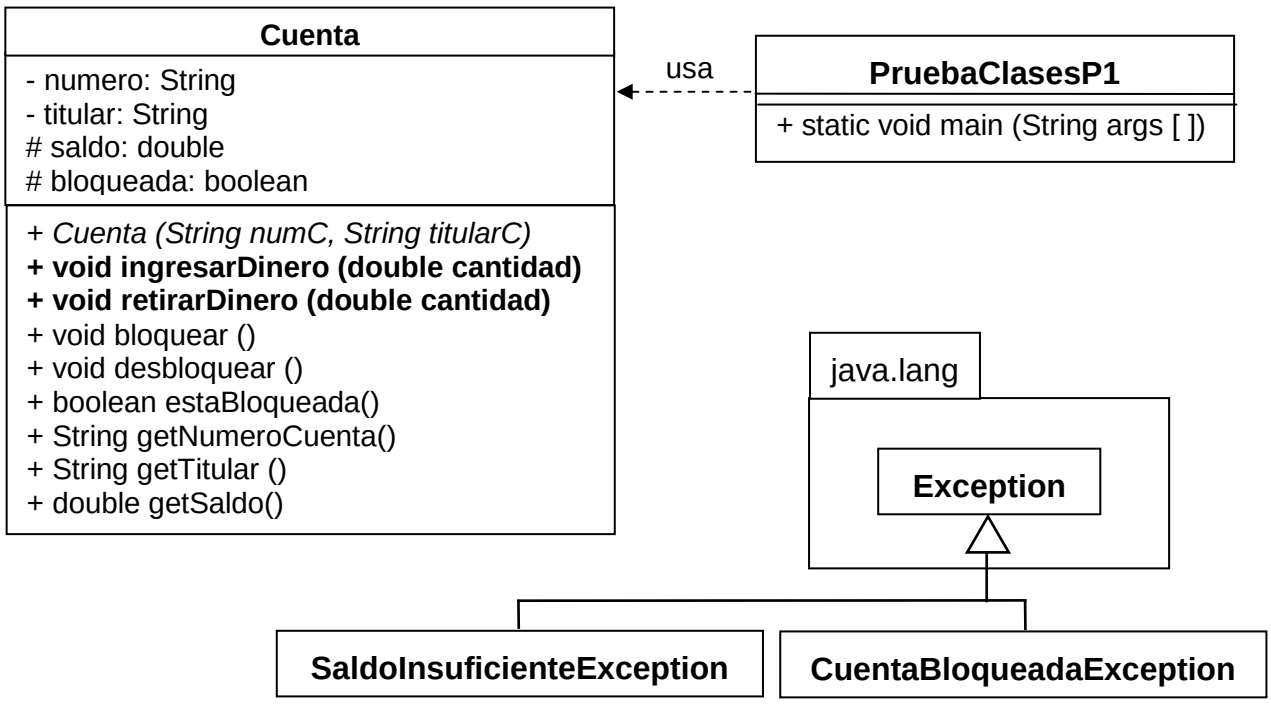
PROBLEMAS:

1. Abre un nuevo proyecto llamado '*T5P1 – Cuenta'* en el que vamos a implementar el siguiente diagrama de clases:



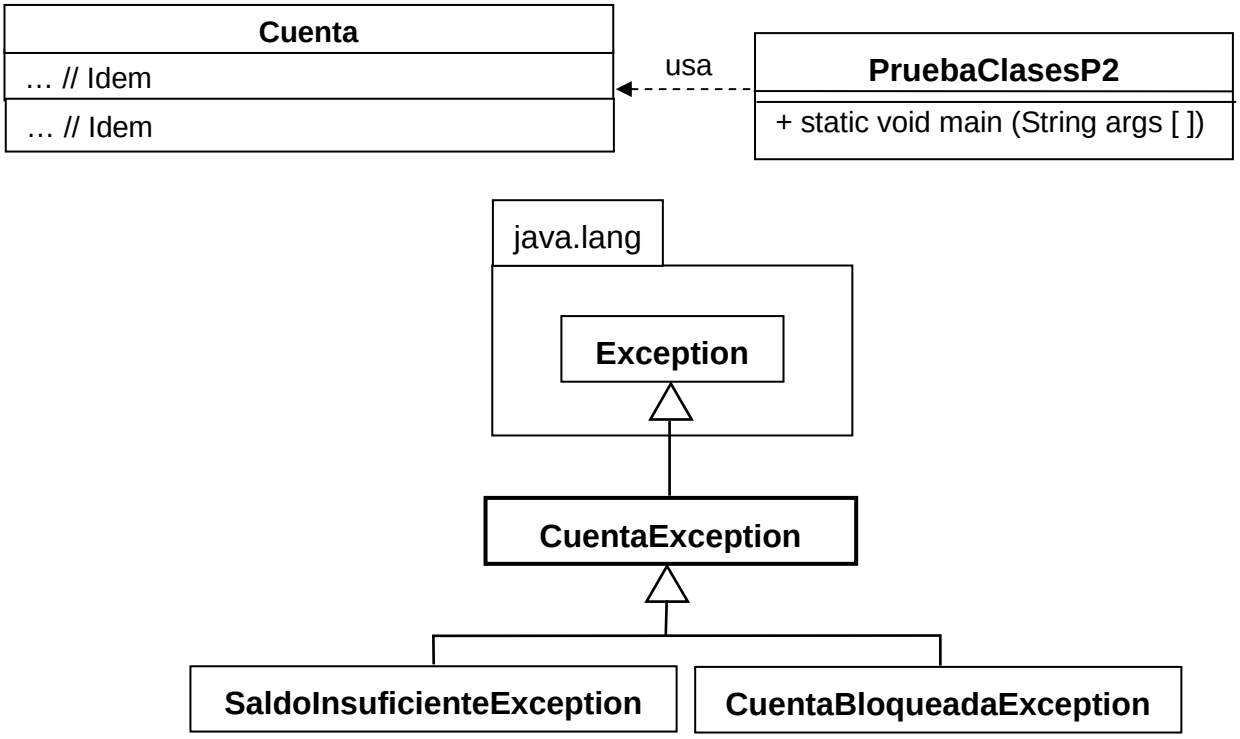
Teniendo en cuenta que:

- Si la cuenta está bloqueada los métodos *ingresarDinero* y *retirarDinero* lanzan una excepción *CuentaBloqueadaException.*
- Si el método *retirarDinero* desea extraer de la cuenta una cantidad superior al saldo existente se lanzará una *SaldoInsuficienteException*.
- Ambas excepciones heredan de la clase *Exception* y contienen el constructor sin parámetros.
- Todas las clases desarrolladas se ubicarán en el paquete por defecto.

La clase *PruebaClasesP1* debe:

- Crear un objeto cuenta y a continuación bloquearla.
- Intentar ingresar 2000 euros en la cuenta y capturar la excepción que pueda lanzarse.
- Desbloquear la cuenta.
- Intentar las siguientes operaciones como un bloque, capturando las posibles excepciones que puedan lanzarse: ingresar 2000 €, retirar 100 €, imprimir el saldo y retirar 3000 € *(todo en mismo bloque try)*.

2. Abre un nuevo proyecto llamado '*T5P2 – CuentaException'* en el que vamos a modificar diagrama de clases del problema anterior de manera que:



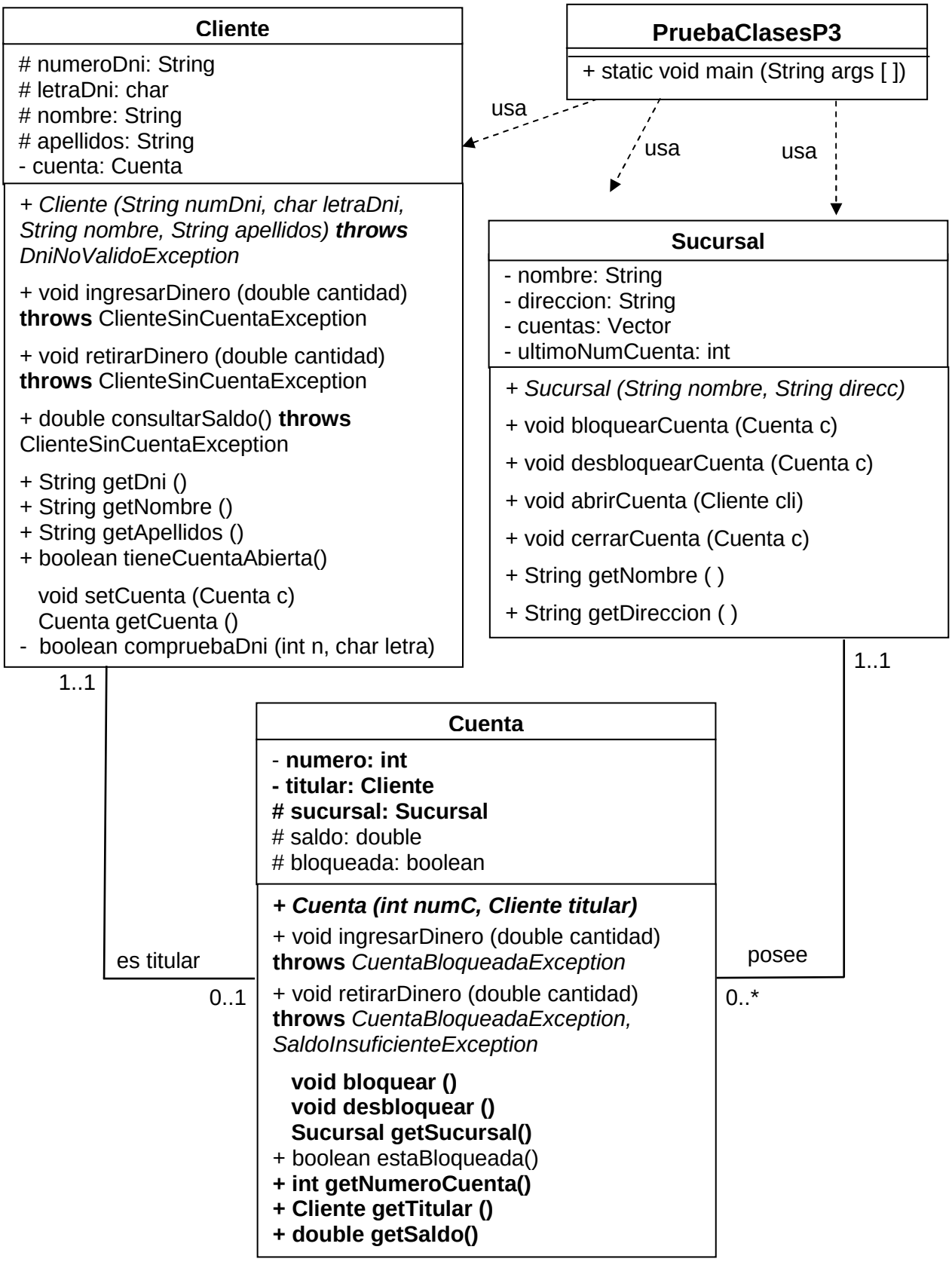
Para ello debemos:

- Importar las clases *Cuenta*, *SaldoInsuficienteException* y *CuentaBloqueadaException* del proyecto anterior.
- Crear la clase *CuentaException* con su constructor sin parámetros.
- Modificar las dos subclases para que ahora hereden de *CuentaException*.

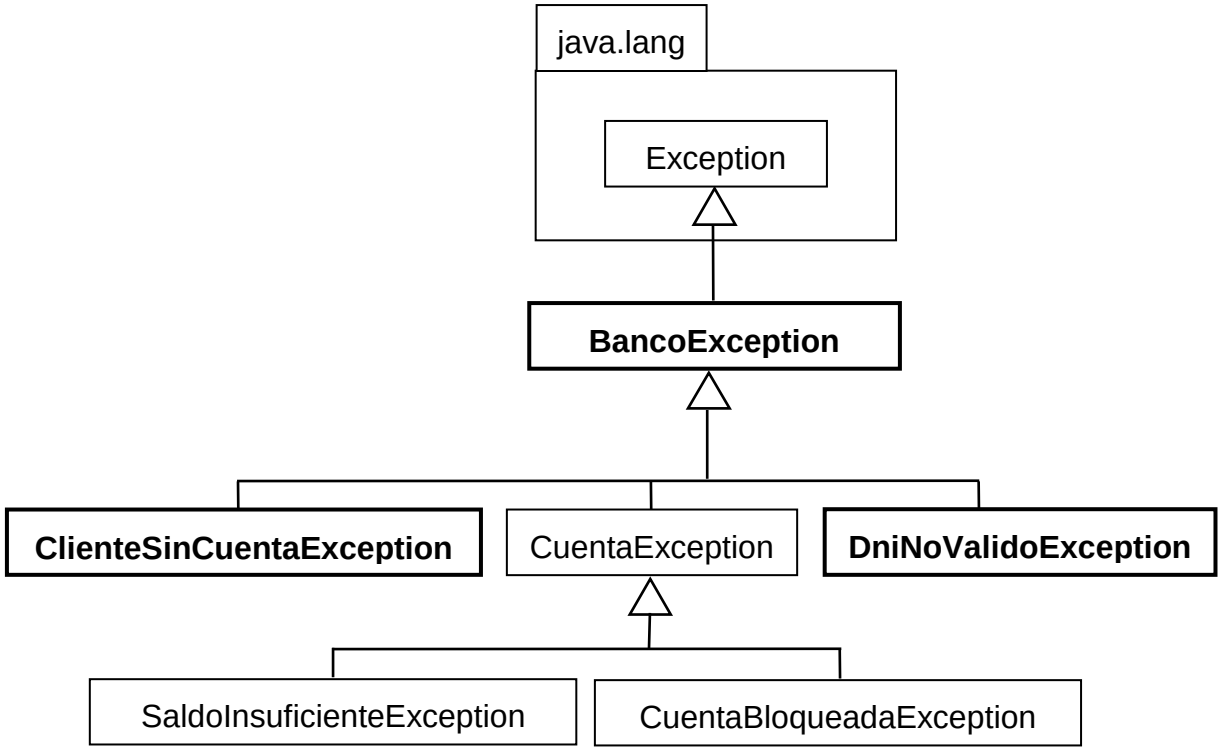
La clase *PruebaClasesP2* debe:

- Crear un objeto cuenta.
- Intentar las siguientes operaciones como un bloque: ingresar 2000 €, retirar 100 €, imprimir el saldo y retirar 3000 €, pero en este caso capturaremos sólo las excepciones *SaldoInsuficienteException* y *CuentaException*.

3. Abre un nuevo proyecto llamado '*T5P3 – Banco*' en el que vamos a implementar el siguiente diagrama de clases:



Con la siguiente jerarquía de excepciones:



Para ello debemos:

- Importar las clases *CuentaException*, *SaldoInsuficienteException*, *CuentaBloqueadaException* y *Cuenta* del proyecto anterior.
- Crear las clases *BancoException, ClienteSinCuentaException* y *DniNoValidoException* con su constructor sin parámetros.
- Modificar la clase *CuentaException* para que herede de *BancoException.*
- Crear la clase *Cliente* teniendo en cuenta que:
    - El constructor debe lanzar una *DniNoValidoException* si el método privado *compruebaDni* devuelve falso. El código de dicho método se encuentra al final del enunciado del problema.
    - Los métodos *ingresarDinero* y *retirarDinero* lanzan una *Cliente-SinCuentaException* si el cliente todavía no ha abierto una cuenta. En el caso de que el cliente posea cuenta realizan el ingreso o la retirada de dinero de la cuenta, capturando las excepciones que puedan elevarse.
    - El método *getDni* debe devolver la cadena: *númeroDni-letraDni.*
    - Los métodos *getCuenta* y *setCuenta* tiene visibilidad de paquete.

- Crear la clase *Sucursal* de modo que cumpla con las especificaciones:
    - La propiedad *cuentas* es de la clase Vector, ésta se comporta como una tabla pero que es capaz de redimensionarse según haga falta. La usaremos para almacenar las cuentas abiertas de la sucursal. La clase vector se halla en el paquete *java.util* (es necesario importarlo para que compile) y se detalla al final del enunciado.
    - El atributo *ultimoNumCuenta* representa el último número de cuenta que se ha usado para abrir una cuenta.
    - El *constructor* debe inicializar a cero el atributo *ultimoNumCuenta* y crear el objeto *cuentas*.
    - Los métodos *bloquearCuenta* y *desbloquearCuenta* simplemente invoca a los métodos respectivos del objeto Cuenta que reciben como parámetro.
    - El método *abrirCuenta* comprueba que el cliente recibido no tenga ya una cuenta abierta. Si es así, emite un mensaje describiendo la situación. En caso contrario, creará una nueva cuenta con el cliente y el próximo número de cuenta disponible en la sucursal. A continuación añadirá la nueva cuenta a las ya existentes y se la asignará al cliente.
    - El método *cerrarCuenta* comprueba que el cliente tenga una cuenta abierta. Si no la tiene se muestra un mensaje y termina la función. En el caso de que sí posea una cuenta, la recuperamos, ponemos a *null* la cuenta del objeto Cliente y, a continuación, la borramos de las cuentas existentes en la sucursal.
- Modificar la clase *Cuenta* para que:
    - Incorpore o modifique los nuevos atributos *numero*, *titular y sucursal* y los nuevos métodos *getNumeroCuenta*, *getTitular* y *getSucursal.*
    - Modificar el constructor para que acepte los nuevos tipos de parámetros *numC* y *titular.*
    - Los métodos *bloquear, desbloquear* y *getSucursal* tienen visibilidad de paquete.
- Por último, vamos a crear la clase *PruebaClasesP3* de forma que realice las siguientes operaciones:
    - Crear un objeto *Sucursal*.
    - Instanciaremos dos objetos de la clase Cliente con dos DNI válidos (28748205-E y 28748206-T) y mostraremos un mensaje por cada objeto creado con éxito. Hay que capturar las excepciones que pudieran lanzarse.
    - Si se consiguieron crear los objetos correctamente entonces el primer cliente ingresa 200 € y el segundo 600 €, mostrando un

mensaje por cada ingreso realizado con éxito. Hay que capturar las excepciones que pudieran lanzarse.

- A continuación el primer cliente intenta sacar 300 € de su cuenta y el segundo 600 €. Mostramos un mensaje que nos informe del saldo de las cuentas de cada uno de los clientes.
- Ahora cerramos la cuenta del segundo cliente.
- Ejecutar el programa y depurarlo.
- Hacer las modificaciones necesarias para que se lancen las distintas excepciones, observando el camino que sigue el programa una vez lanzadas.

```
private boolean compruebaDni (String num, char letra) {
        String tablaLetras = "TRWAGMYFPDXBNJZSQVHLCKE";
        int i, calculo=0, pos;

        // El número de cifras debe ser 8, si es
        // menor se debe rellenar con ceros
        if (num.length() != 8)
              return false;
        else
        {
              for(i=0; i<8; i++)
                    calculo = calculo * 10 + (num.charAt(i)-'0');
              pos = calculo % 23;
              if (tablaLetras.charAt(pos) == letra)
                    return true;
              else
                    return false;
        }
}
```

java.util.Vector

Extracto de los constructores:

```
// Crea un objeto capaz de albergar 10 objetos.
public Vector( );
```

Extracto de los métodos:

```
// Añade un objeto (del tipo que sea) detrás del último existente
public void addElement (Object e);

// Busca el elemento que se le pasa como parámetro. Si lo encuentra
// lo borra y devuelve cierto. En caso contrario devuelve falso
public boolean removeElement (Object e);

// Devuelve la capacidad de almacenamiento actual del vector
public int capacity();

// Devuelve el número de elemento actual del vector
public int size();
```