Excepciones

Ejercicio 1

Crea un programa que tome un número como parámetro en la línea de comandos, y como salida muestra el logaritmo de dicho número.

El método **Main** llamara a un método estático al que se le pase el parámetro de entrada y nos calcule el logaritmo, este método devolverá el resultado a la función main.

1. Compilar el programa y ejecutadlo de tres formas distintas:
   - Sin parámetros.
   - Poniendo un parámetro no numérico.
   - Poniendo un parámetro numérico.
2. Anotad las excepciones que se lanzan en cada caso (si se lanzan).
3. Modificar el código de Main para que capture las excepciones producidas y muestre los errores correspondientes en cada caso.
4. Probad de nuevo el programa igual que en el caso anterior comprobando que las excepciones son capturadas y tratadas.

Ejercicio 2

Partiendo del ejercicio anterior, ahora vamos a tratar las excepciones del método logaritmo. La función de las BCL que calcula el logaritmo, comprueba si el valor introducido es menor o igual que 0, ya que para estos valores la función logaritmo no está definida. Se pide:

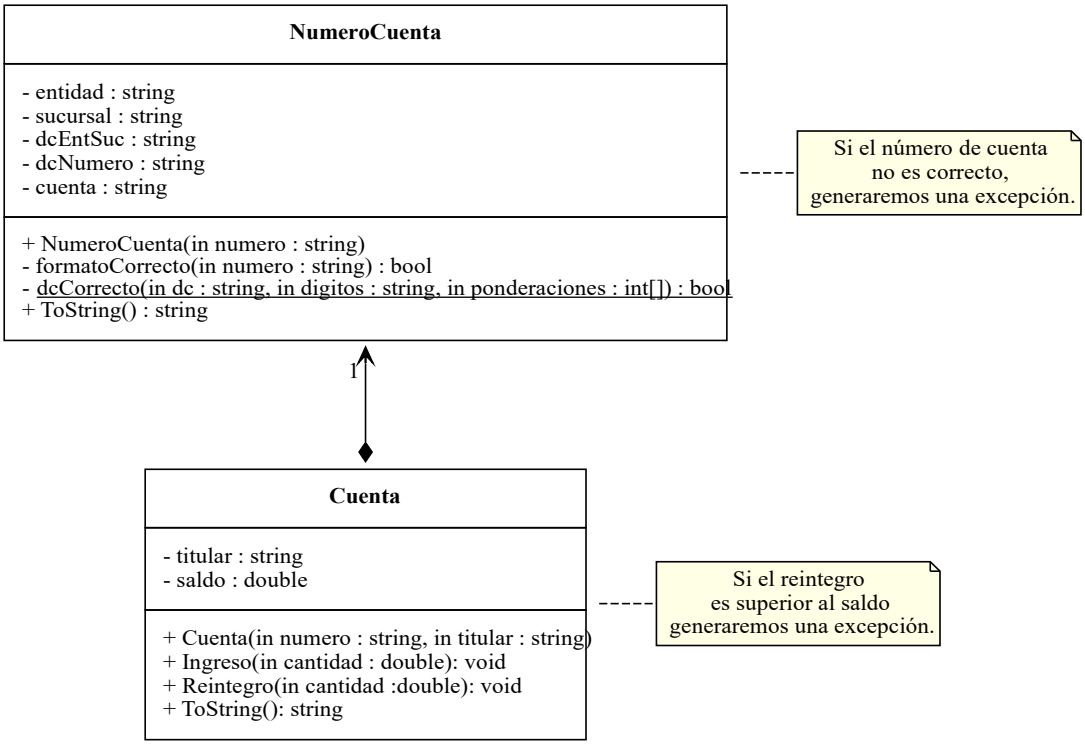
1. Busca entre las excepciones la más adecuada para lanzar en este caso, que indique que a un método se le ha pasado un argumento ilegal.
2. Una vez elegida la excepción adecuada, añadir código (en el método logaritmo) para que en el caso de haber introducido un parámetro incorrecto se lance dicha excepción.
3. Probar el programa para comprobar el efecto que tiene el lanzamiento de la excepción.
4. Captura la excepción en el Main.

Ejercicio 3

- Modifica el programa anterior para crear nuestro propio tipo de excepción derivada de **Exception**.
- Esta será lanzada en caso de introducir un valor no válido como parámetro. La excepción se llamará **ParametroNoValidoException**.
- Le definiremos 2 constructures:
  1. Vacío.
  2. Pasándole el mensaje de error que a su vez pasará a la clase base.
- Deberemos lanzar esta excepción en lugar de la escogida en el punto anterior.
- Capturala donde capturabas la anterior.

Ejercicio 4

Crea un programa a partir del siguiente diagrama de clases en UML



Me permitirña recoger datos de una cuenta bancaria, creando los siguientes métodos y comprobando que funcionan.

El método privado de NumeroDeCuenta **formatoCorrecto**, comprobará el formato del número de cuenta con expresiones regulares y si es correcto rellenará los campos del objeto.

El método Reintegro, lanzará una exception **SaldoInsuficienteException**, cuando se intente retirar una cantidad y no haya suficiente saldo.

El método NumeroCuenta recibirá un string con el número de cuenta y si esta no se corresponde con un número de cuenta correcto también se lanzara una excepción **NumeroCuentaIncorrecto**.

Para comprobar que el número de cuenta es correcta deberemos primero comprobar que la cadena cumple con el Patrón correspondiente (usando expresiones regulares) y después comprobaremos que los dígitos de control son los correctos (Ver página siguiente).

Nota: Debes gestionar en el programa principal las excepciones lanzadas.

¿Cómo calcular los dígitos de control de una cuenta corriente?

El código cuenta corriente es un número de 20 cifras. Las cuatro primeras de la izquierda identifican a la Entidad, las cuatro siguientes, la Sucursal, luego vienen dos dígitos de control y las diez últimas corresponden al número de la cuenta corriente.

- El primero es el dígito de control de Entidad/Sucursal.
- El segundo es el dígito de control del número de la cuenta corriente.

El siguiente ejemplo nos enseñará a calcularlos…

Supongamos ahora `2085` el código de una hipotética entidad y `0103` el código de una de sus sucursales.

Para calcular el dígito de control de Entidad/Sucursal se realiza la operación:

```
2*4 + 0*8 + 8*5 + 5*10 + 0*9 + 1*7 + 0*3 + 3*6 = 123
```

Es decir, cada una de las cifras de la entidad, seguidas de la sucursal, se han ido multiplicando por `4, 8, 5, 10, 9, 7, 3, 6` y luego se han sumado estos resultados.

Ahora realizaremos la operación `11 - 123 % 11` el resultado será un número entre 1 y 11. Si el número es menor que 10 será ya el valor del DC. Pero si es 10 el DC será 1 y si es 11 será 0. En este caso `11 - 123 % 11 = 9` que será el DC de Entidad/sucursal.

Para el DC del número de una cuenta. Si este es, por ejemplo, el 0300731702, para calcular su dígito de control se realiza la operación:

```
0*1 + 3*2 + 0*4 + 0*8 + 7*5 + 3*10 + 1*9 + 7*7 + 0*3 + 2*6 = 141
```

Es decir, cada una de las cifras del número de la cuenta, leídas de izquierda a derecha, se han ido multiplicando por `1, 2, 4, 8, 5, 10, 9, 7, 3, 6` y luego se han sumado estos resultados.

Realizaremos la misma operación que antes `11 - 121 % 11 = 2` que será el DC de número de cuenta.

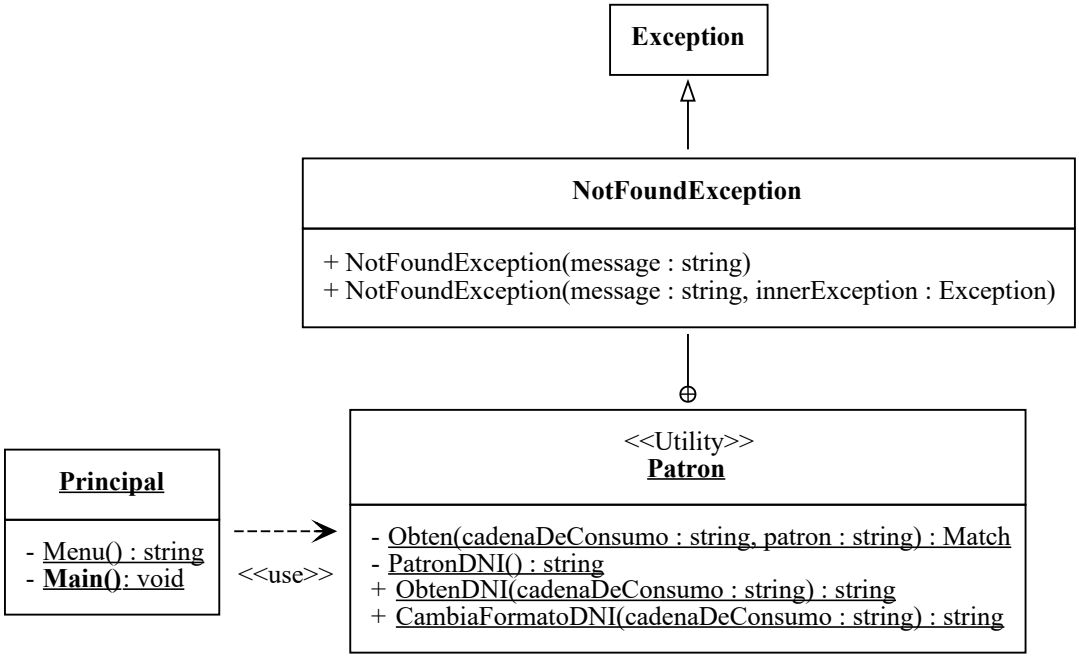
Algunas cuentas correctas para poder probar son:

```
2085 0103 92 0300731702
2100 0721 09 0200601249
0049 0345 31 2710611698
2100 1162 43 0200084482
2100 1516 05 0200306484
2100 0811 79 0200947329
9182 5005 04 0201520831
```

Ejercicio 5 Ampliación

Vamos ha realizar un programa que produzca una cadena de excepciones hasta un punto de recuperación, en el cual mostrará las excepciones que se han ido desencadenado.

Para realizar el ejercicio implementaremos el siguiente diagrama de clases...



Clase Patrón:

- Definirá una clase anidada **NotFoundException** que heredará de Exception y que redefinirá los constructores que ves en el diagrama.
- Será una clase estática que contendrá los siguientes métodos estáticos de utilidad…
  - **Obtén**: Método estático privado que devolverá el primer match del patrón en la cadena de consumo y generará un NotFoundException si success es igual a false con el mensaje "***No se han obtenido coincidencias.***"
  - **PatronDNI**: Método estático privado que devolverá el patrón para buscar un DNI con los siguientes grupos etiquetados...
    - 8 dígitos.
    - Un separador que podrá ser espacio, guion o nada.
    - Una letra mayúscula o minúscula.
  - **ObtenDNI**: Método estático público que busca la primera ocurrencia de un dni en la cadena de consumo usando los métodos privados definidos anteriormente.

    Además, capturará cualquier posible excepción y la lanzará una nueva NotFoundException (encadenando a la excepción original) con el mensaje "***Imposible de obtener un DNI.***"
  - **CambiaFormatoDNI**: Método estático público que busca la primera ocurrencia

de un dni en la cadena de consumo usando los métodos privados definidos anteriormente.

Posteriormente reemplazará usando Regex.Replace la ocurrencia del DNI encontrada con un nuevo DNI con el formato <8 digitos><Letra mayúscula>

Además, capturará cualquier posible excepción y la lanzará una nueva NotFoundException (encadenando a la excepción original) con el mensaje "***No se ha podido cambiar el formato del DNI.***"

Clase Principal:

- Define el método Menu() que devolverá una cadena con las siguientes opciones de menú.
  - 1- Introduce texto.
  - 2- Muestra texto.
  - 3- Obten DNI.
  - 4- Cambia formato DNI.
  - ESC- Salir.
  - Selecciona una opción
- Define el método principal Main donde gestionará el bucle del menú de tal manera que si se produce una excepción mostrará el mensaje de todas las excepciones que se hayan podido encadenar y volverá a mostrar el menú.

  Ejemplo de ejecución:

```
<Opciones menú> (si pulsamos 4)
No se ha podido cambiar el formato del DNI.
El valor no puede ser nulo.
Nombre del parámetro: input

<Opciones menú> (si pulsamos 1 e introducimos “Mi DNI es 21455678”)

<Opciones menú> (pulsamos 3)
Imposible de obtener un DNI.
No se han obtenido coincidencias.

<Opciones menú> (si pulsamos 1 e introducimos “Mi DNI es 21455678-h”)

<Opciones menú> (si pulsamos 2 → “Mi DNI es 21455678-h”)

<Opciones menú> (si pulsamos 3 → “El DNI encontrado es 21455678-h”)

<Opciones menú> (si pulsamos 4 → “El DNI se ha formateado con éxito.”)

<Opciones menú> (si pulsamos 2 → “Mi DNI es 21455678H”)
```