

Ejercicios Excepciones

[Descargar estos ejercicios](#)

Índice

1. [Ejercicio 1](#)
2. [Ejercicio 2](#)
3. ☒ [Ejercicio 3](#)
4. ☒ [Ejercicio 4](#)
5. [Ejercicio 5](#)

Ejercicio 1

Crea un programa que tome un número **como parámetro en la línea de comandos**, y que muestre el logaritmo en base diez de dicho número. Para ello vamos a seguir los siguiente pasos:

1. Desde el **Main** se llamará a un método estático **CalculaLogaritmo** al que se le pase un **double** como argumento de entrada, calcule el logaritmo y devuelva el resultado del cálculo al método **Main**.
2. Compilar el programa y ejecutarlo de tres formas distintas:
 - Sin parámetros en la línea de comandos.
 - Poniendo un parámetro no numérico.
 - Poniendo un parámetro numérico.
3. Anotad las excepciones que se lanzan en cada caso (si se lanzan).
4. Modificar el código de el **Main** para que capture las excepciones producidas y muestre los errores correspondientes en cada caso.
5. Probad de nuevo el programa, comprobando que las excepciones son capturadas y tratadas.

Nota: para pasar los argumentos desde la línea de comandos puedes ver el siguiente tutorial.

[Paso de argumentos en línea de comandos](#)

Ejercicio 2

Partiendo del ejercicio anterior, ahora vamos a trabajar con los problemas del método logaritmo. La función de las BCL que calcula el logaritmo, no comprueba si el valor introducido es menor o igual que 0, el problema es que para estos valores la función logaritmo no está definida, así que tendremos que ser nosotros los que controlemos la entrada. Se pide:

1. Busca entre las excepciones de la BCL, alguna que consideres adecuada para lanzar. Por ejemplo, una que indique que a un método se le ha pasado un argumento ilegal.
2. Una vez elegida la excepción adecuada, añade el código (en el método logaritmo) para que en el caso de haber introducido un valor incorrecto se lance dicha excepción.
3. Probar el programa para comprobar lo que ocurre cuando lanzas la excepción.
4. Captura y trata la excepción en el Main.

✓ Ejercicio 3

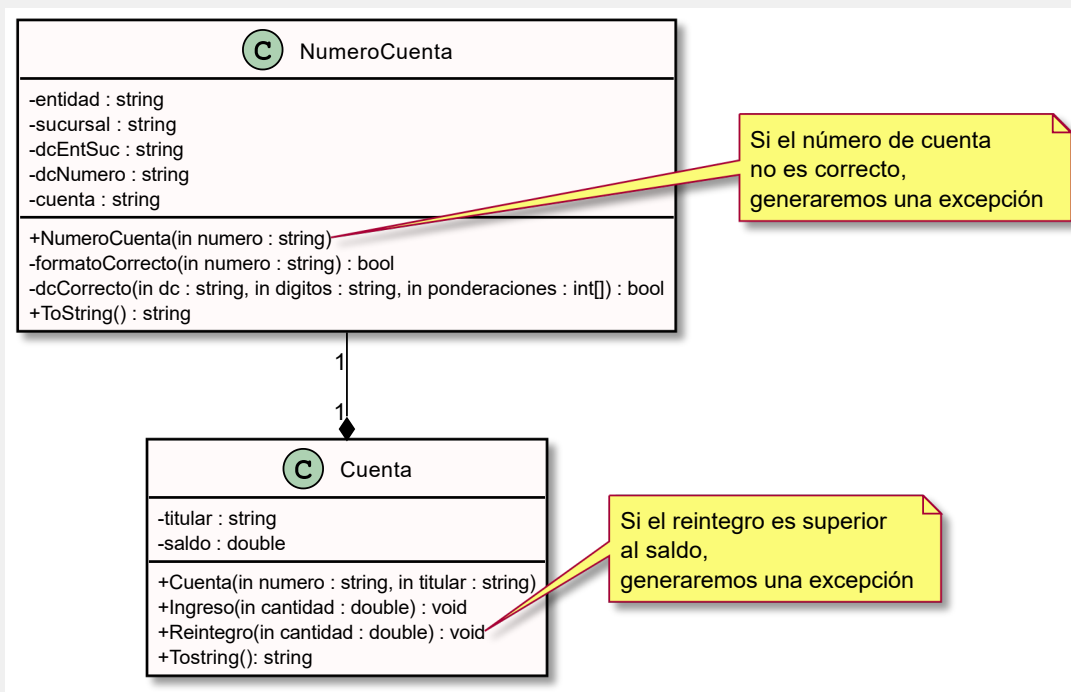
Modifica el programa anterior para crear nuestro propio tipo de excepción derivada de **Exception** . La excepción se llamará **ParametroNoValidoException** , y le definiremos 2 constructores:

1. Vacío.
2. Pasándole el mensaje de error que a su vez pasará a la clase base.

Deberás lanzar esta excepción en lugar de la escogida en el punto anterior. La capturarás donde capturabas la anterior.

✓ Ejercicio 4

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



La aplicación permitirá recoger datos de una cuenta bancaria. Deberás crear los siguientes métodos y comprobar que funcionan correctamente. Prueba la aplicación con más de un número de cuenta y captura las posibles excepciones en el programa principal.

- El constructor **NumeroCuenta** recibirá un string con el número de cuenta completo y si este no se corresponde con un número de cuenta correcto se lanzara una excepción **NumeroCuentaIncorrectoException** . Esta excepción se puede lanzar por dos motivos:

1. Porque no cumple el patrón de un número de cuenta (para ello llamará al método `formatoCorrecto`)
 2. Porque la cuenta no se corresponde con los dígitos de control (en este caso usara `dcCorrecto`).
- El método privado de `NumeroDeCuenta` , `formatoCorrecto` . Comprobará el formato del número de cuenta con expresiones regulares y si **el formato** es correcto rellenará los campos del objeto. En caso contrario lanzará una excepción `NumeroCuentaIncorrectoException` , indicando que el formato de la cuenta no cumple las condiciones necesarias.
 - El método privado `dcCorrecto` , comprobará si el dígito de control corresponde a la cuenta, devolviendo false en caso contrario. Para ver como se hace esto, consulta la Nota y la Pista posterior.
 - El método `Reintegro` , lanzará una exception `SaldoInsuficienteException` , cuando se intente retirar una cantidad y no haya suficiente saldo.

El código cuenta corriente es un número de **20** cifras. Las **4** primeras de la izquierda identifican a la **Entidad**, las **4** siguientes, la **Sucursal**, luego vienen **2 dígitos de control** y las **10** 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**:

- Realizaremos 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 los números del array de ponderaciones **4, 8, 5, 10, 9, 7, 3, 6** y luego se han sumado estos resultados.
- Ahora realizaremos la operación **11 - (resultado % 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 calcular el dígito de control de **número de cuenta corriente**:

- 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 - (141 % 11) = 2** que será el DC de número de cuenta.

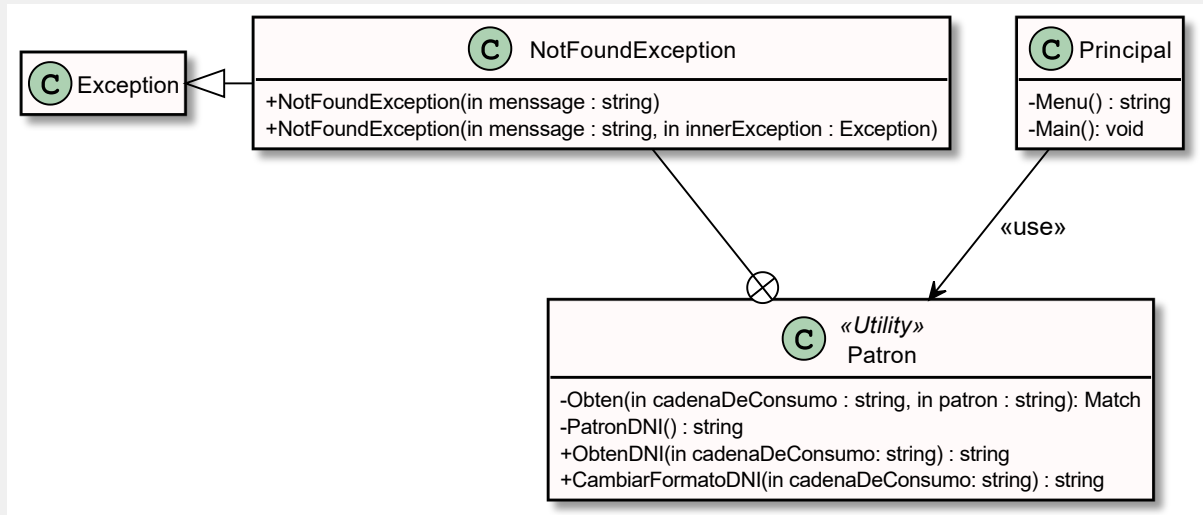
Algunas cuentas para poder probar son:

```
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
```

 **Pista:** El método **dcCorrecto** calcula el dígito de control de una string con N números a partir de un array de ponderaciones de también N números. Y después comprueba el número resultante, con el DC pasado como argumento, el método devolverá **true** si los dígitos son iguales, **false** en caso contrario. Por tanto, este método será llamada en dos ocasiones: para los 8 número anteriores a los 2 dígitos de control, y para los 10 números posteriores. Usando cada vez el array de ponderaciones correspondiente que se describen arriba.

Ejercicio 5

Vamos a realizar un programa que produzca una cadena de excepciones hasta un punto de recuperación, en el cual mostrará las excepciones que se han ido desencadenando. 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...
- Obten** : 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, guión 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 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 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.