

USO DE EXCEPCIONES.

1. Como hemos visto en el 1º Cuatrimestre cuando dividimos un número entre 0 se produce una excepción. ¿Qué clase de excepción es? ¿Qué excepción se produce? Vamos a capturar dicha excepción para que no detenga nuestro programa y cuando dividamos un número entre 0 que tan sólo salga un aviso por pantalla del estilo “No se puede dividir entre 0”.

- 1.1. En esta ocasión vamos a crear una clase llamada Division con un único método llamado “dividir” dado dos números de entrada (numerador y denominador). Como hemos visto anteriormente, si el denominador
- 1.2. Crea una clase “RangoDivisionPermitida” que genere el mensaje “Rango de números no permitidos” y al llamar al método dividir haz que si el número no se encuentra en el rango de 0 a 100 lance dicha excepción previamente creada “RangoDivisionPermitida”. Por su parte, en el Main captura dicha excepción y muestra el mensaje que genera.

2. Desde el inicio del curso hemos trabajado mucho con la clase Scanner y más concretamente el método “nextInt()” para leer un número que el usuario escribía por terminal. ¿Qué pasa cuando nuestro programa espera un número entero y utilizamos “nextInt()” y el usuario por error o desconocimiento escribe un texto, por ejemplo, “uno”? A continuación, evita el programa se detenga y captura la excepción, en caso de que el usuario escribe un dato que no es válido por ejemplo “1hh2” el programa muestre un mensaje por terminal de “Formato de entrada incorrecto”.

- 2.1. Si vemos las excepciones que se pueden producir este método vemos que son varias.
Haz que salga el “formato de entrada incorrecto” únicamente cuando el usuario introduce una entrada incorrecta por ejemplo “1hh2” con la clase excepción que le corresponde, y luego captura las restantes excepciones mediante una excepción genérica y muestra por pantalla “Se ha capturado otro tipo de problema, cuidado”.

nextInt

```
public int nextInt(int radix)
```

Scans the next token of the input as an int. This method will throw `InputMismatchException` if the next token cannot be converted to an int.

If the next token matches the `Integer` regular expression defined above then the token is converted into an int value. If specific negative prefixes and suffixes were present, and passing the resulting string to `Integer.parseInt` with the radix.

Parameters:

`radix` - the radix used to interpret the token as an int value

Returns:

the int scanned from the input

Throws:

`InputMismatchException` - if the next token does not match the `Integer` regular expression, or is out of range


`NoSuchElementException` - if input is exhausted

`IllegalStateException` - if this scanner is closed

- 2.2. En esta ocasión además de mostrar los mensajes por terminal vuelve a solicitar nuevamente, hasta que el usuario introduzca un número válido y no se produzca ninguna excepción por culpa del formato de entrada.

3. Siguiendo con el uso de la clase Scanner al utilizar el método “nextDouble()”, como sabes con el método nextDouble el usuario puede escribir un número decimal, un error muy común es que el usuario no se sepa si tiene que escribir “2.2” o “2,2”. ¿Qué sucede cuando el usuario introduce “2,2” en lugar de “2.2”?

- 3.1. Utilizando una excepción como hicimos anteriormente, sería una opción para evitar que el programa terminase al producirse dicha incidencia y por ejemplo solicitarle nuevamente un número. Pero, cabe otra posibilidad:

1. Utiliza el método "nextLine()" de la clase Scanner para leer un String.
 2. Conviértelo a un double:  `parseDouble(String s) : double - Double`
 3. Escribe "2.2" y "2,2".
 4. Si no es capaz de convertirlo a un double ¿Qué excepción se produce?
 5. Captura dicha excepción y en caso de que se produzca cambia los caracteres correspondientes en el String para que el método parseDouble no genere excepción.
 6. Programa una opción "finally" que muestre por pantalla el número double obtenido.
4. Vamos a crear un formulario de registro de nuevos socios de un gimnasio. Para ello crearemos una clase llamada "Cliente" con los siguientes atributos:
- Nombre
 - email
 - edad

Esta clase aparte de los getters&setters y toString, dispondrá de un único constructor donde se le pasan todos los parámetros para crear el objeto completo y lanzará una serie de excepciones en caso de que los datos sean incorrectos.

Para gestionar la entrada correcta de los datos y creación del registro utilizaremos una serie de Excepciones que propias, así pues, crearemos las siguientes excepciones:

- CampoVacio. Muestra el texto "Campo de Datos Vacio".
- FormatoDatoIncorrecto. Muestra el texto "Formato de Dato incorrecto".
- DetectadoMenorEdad. Muestra el texto "Detectado el registro de un menor de edad".

Tanto CampoVacio y FormatoDatoIncorrecto deben ser capturados sí o sí, mientras que DetectadoMenorEdad no puede ser capturado o no.

A continuación, en el constructor determinaremos si los datos introducidos para crear el objeto son válidos y si no se generarán las excepciones correspondientes:

- Algún campo está vacío: Lanza la excepción "CampoVacio"
- En nombre hay algún carácter que no sea una letra, es decir, un número o símbolo extraño que no sea un espacio " ": Lanza la excepción "FormatoDatoIncorrecto".
- El campo email no tiene un @ en su interior: Lanza la excepción "FormatoDatoIncorrecto".
- La edad no es un número: Lanza la excepción "FormatoDatoIncorrecto".
- La edad es menor que 0 o mayor que 150: Lanza la excepción "FormatoDatoIncorrecto".
- La edad es menor que 18 años: Lanza la excepción "DetectadoMenorEdad"

Creará una clase de main de prueba donde utilice la clase "Cliente" para testearla.