

# Programación Orientada a Objetos. Práctica 3.1

## Ejercicio 1. (proyecto prDatos)

### Características del ejercicio

En esta práctica, el alumno aprenderá a definir nuevas excepciones, tanto comprobadas como no-comprobadas, y las diferencias en el tratamiento y propagación de ambas (a diferencia de las excepciones no-comprobadas, las excepciones comprobadas deben ser anunciadas en caso de que un determinado método las lance o propague, o en otro caso el método está obligado a capturarla).

Así, el alumno definirá métodos que lancen excepciones, métodos que propaguen excepciones (es decir, métodos que reciban una excepción y no la capturan, sino que la propagan), y métodos que capturen excepciones y las traten, recuperándose del error. También definirá constructores y métodos que capturen excepciones lanzadas por el sistema.

- Trabaja con excepciones **no-comprobadas** y **comprobadas** (se deben anunciar o capturar).
- El método `calcMedia` lanza una excepción.
- El método `calcDesvTipica` propaga una excepción lanzada por `calcMedia`.
- El método `toString` captura excepciones y las trata, recuperandose del error.
- El método `setRango` captura las excepciones lanzadas por el sistema, y re-lanza otro tipo de excepción.
- El constructor captura las excepciones lanzadas por el sistema (`Double.parseDouble`), las trata y se recupera del error
- El programa principal recibe los datos de los argumentos, y en caso de que no sean correctos, captura la excepción, muestra un mensaje de error y termina la aplicación. Además, en caso de invocar a los métodos `calcMedia` o `calcDesvTipica`, tambien debe capturar las excepciones, y mostrar un mensaje de error.

### El diagrama de clases UML

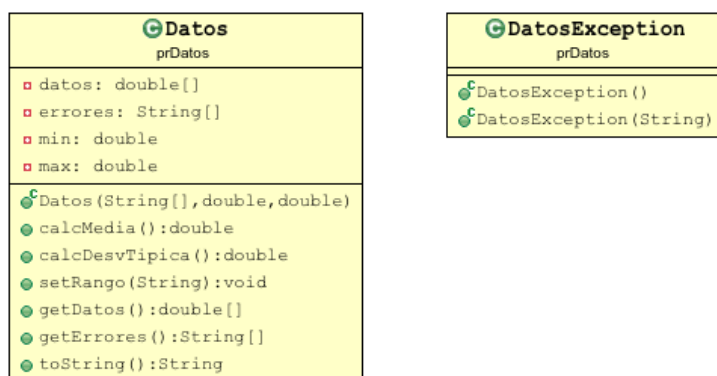


Figura 1: Diagrama de clases UML

### La excepción no-comprobada DatosException

Crea la excepción **no-comprobada** `DatosException` (del paquete `prDatos`) para manejar situaciones excepcionales que podrán producirse en las siguientes clases.

```

► DatosException()           // Constructor sin mensaje
► DatosException(String)     // Constructor con mensaje especificado

```

## La clase Datos

La clase `Datos` (del paquete `prDatos`) contiene un array con una secuencia de números reales, y también contiene un array con los posibles errores que se han producido en su construcción. Además, también contiene los valores `min` y `max` que serán utilizados en algunas operaciones.

```
► Datos(String[],double,double)
```

Construye el objeto, donde el primer parámetro contiene una secuencia de datos, desde donde se debe extraer el valor *double* de cada elemento (de tipo `String`) y almacenarlo en el array de datos, considerando que si algún elemento del array de entrada no se puede convertir a número real, entonces se añadirá al array de errores. Además, también recibe como parámetro los valores de los atributos `min` y `max`.

```
► calcMedia(): double
```

Calcula y devuelve la media aritmética correspondiente a los datos almacenados que se encuentren dentro del rango delimitado por los valores `min` y `max` (ambos inclusive). Lanza la excepción `DatosException` si no hay elementos dentro del rango especificado.

$$media = \frac{1}{n_{d_i}} \sum_i d_i : min \leq d_i \leq max$$

$$n_{d_i} = \#\{d_i : min \leq d_i \leq max\}$$

```
► calcDesvTipica(): double
```

Calcula y devuelve la desviación típica correspondiente a los datos almacenados que se encuentren dentro del rango delimitado por los valores `min` y `max` (ambos inclusive). Propaga la excepción `DatosException` lanzada por el método `calcMedia` si no hay elementos dentro del rango especificado.

$$\sigma = \sqrt{\frac{1}{n_{d_i}} \sum_i (d_i - media)^2 : min \leq d_i \leq max}$$

$$n_{d_i} = \#\{d_i : min \leq d_i \leq max\}$$

```
► setRango(String): void
```

Actualiza los valores de los atributos *min* y *max* a los valores que deberán ser extraídos del parámetro de tipo `String`, considerando que el primer valor se corresponde con *min*, el segundo valor se corresponde con *max*, y ambos valores se encuentran separados por el símbolo de *punto y coma* (;). Lanza la excepción `DatosException` en el caso de que se produzca *algún error* en la extracción de los dos valores.

```
► getDatos(): double[]
```

Devuelve el array de datos que contiene el objeto.

```
► getErrores(): String[]
```

Devuelve el array de errores que contiene el objeto.

```
► toString(): String // [Redefinición]
```

Devuelve la representación textual del objeto, en el siguiente formato (sin considerar los saltos de línea):

```
Min: 10.0, Max: 20.0,  
[ 5.0, 9.0, 10.0, 12.0, 13.0, 17.0, 20.0, 25.0 ],  
[ Pepe, María, Paco, Ana, Juan, Lola ],  
Media: 14.4, DesvTipica: 3.6110940170535577
```

En caso de que se produzcan errores a la hora de calcular la media aritmética o la desviación típica, se devolverá una representación textual similar al siguiente ejemplo:

```
Min: 0.0, Max: 4.0,  
[ 5.0, 9.0, 10.0, 12.0, 13.0, 17.0, 20.0, 25.0 ],  
[ Pepe, María, Paco, Ana, Juan, Lola ],  
Media: ERROR, DesvTipica: ERROR
```

## La aplicación PruebaDatos

Desarrolle una aplicación (en el paquete anónimo) que permita realizar una prueba de las clases anteriores. Para ello, se creará un objeto datos de la clase Datos, donde los valores necesarios para su creación se recibirán como argumentos del método `main`: valor *min*, valor *max* y los valores para rellenar el array de cadenas (para construirlo será necesario utilizar el método `copyOfRange` de la clase `Arrays`), y se mostrará por pantalla la representación del mismo. Posteriormente se invocará al método `setRango` del objeto datos pasándole como parámetro el valor "0;4" y se volverá a mostrar por pantalla la representación del objeto datos. Después se invocará de nuevo al método `setRango`, pero esta vez con el valor "15 25".

Si se produce algún error con los argumentos recibidos (no hay datos suficientes o alguno de los valores min o max no puede calcularse adecuadamente), la aplicación terminará mostrando por pantalla el mensaje de error correspondiente.

Ejecutaremos la aplicación tres veces:

- Para la primera ejecución se introducirán como argumentos al método `main` los siguientes datos:

```
10 20 5 9 Pepe 10 Maria 12 13 Paco 17 20 Ana 25 Juan Lola
```

La salida por pantalla será la siguiente (sin considerar los saltos de línea):

```
Min: 10.0, Max: 20.0,  
[ 5.0, 9.0, 10.0, 12.0, 13.0, 17.0, 20.0, 25.0 ],  
[ Pepe, María, Paco, Ana, Juan, Lola ],  
Media: 14.4, DesvTipica: 3.6110940170535577
```

```
Min: 0.0, Max: 4.0,  
[ 5.0, 9.0, 10.0, 12.0, 13.0, 17.0, 20.0, 25.0 ],  
[ Pepe, María, Paco, Ana, Juan, Lola ],  
Media: ERROR, DesvTipica: ERROR
```

Error en los datos al establecer el rango

- Para la segunda ejecución se introducirán como argumentos al método `main` los siguientes datos:

```
10
```

La salida por pantalla será la siguiente:

```
Error, no hay valores suficientes
```

- Para la tercera ejecución se introducirán como argumentos al método `main` los siguientes datos:

```
10 hola 5 9 Pepe 10 María 12 13 Paco 17 20 Ana 25 Juan Lola
```

La salida por pantalla será la siguiente (para mostrar las parte que aparece entre paréntesis hay que utilizar el método `getMessage`):

```
Error, al convertir un valor a número real (For input string: "hola")
```

## Estudio de las excepciones Comprobadas

Realice los cambios necesarios en las clases anteriormente especificadas para que la excepción `DatosException` sea ahora **comprobada**.