Programación Avanzada I. Práctica 3.1

Tema 3. Gestión de Excepciones

Ejercicio 1. (proyecto prDatos, paquete datos)

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, recuperándose del error.
- El método `setRango` captura las excepciones lanzadas por el sistema, y *relanza* 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.

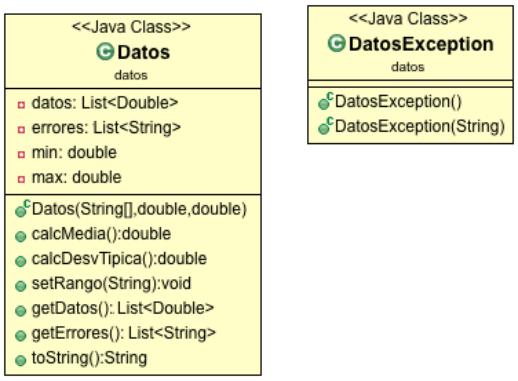

Figura 1: Diagrama de clases UML

El diagrama de clases UML

La excepción no-comprobada `DatosException`

Crea la excepción **no-comprobada** `DatosException` (del paquete `datos`) 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 `datos`) contiene sendas listas (del tipo `List`), una con una secuencia de números reales y otra con los posibles errores que se han producido en su construcción. Además, también incluye 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 la lista de datos, considerando que si algún elemento del array de entrada no se puede convertir a número real, entonces se añadirá a la lista de errores. Además, también recibe como parámetro los valores de los atributos `min` y `max`.[1]

- `calcMedia(): double`

[1] Para calcular el valor numérico representado en un `String` se puede utilizar el *método de clase* `parseDouble` de la clase `Double`, el cual lanza la excepción `NumberFormatException` en caso de error.

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` con el mensaje "No hay datos en el rango especificado" 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 excepcion `DatosException` lanzada por el método `calcMedia` si no hay elementos dentro del rango especificado.[2]

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

- `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` (con el mensaje "Error en los datos al establecer el rango") en el caso de que se produzca *algún error* en la extracción de los dos valores.[3]

- `getDatos(): List<Double>`

  Devuelve el array de datos que contiene el objeto.

- `getErrores(): List<String>`

  Devuelve el array de errores que contiene el objeto.

- `toString(): String` *// @Redefinición*

[2] Para calcular el cuadrado de un número se puede utilizar el *método de clase* `pow` de la clase `Math`, y para calcular la raíz cuadrada se puede emplear el *método de clase* `sqrt` de la clase `Math`.

[3] Para buscar la posición donde se encuentra el carácter `';'` en un `String`, se puede utilizar el método `indexOf`. Además, para obtener un determinado substring se puede utilizar el método `substring`.

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

```
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. Obsérvese que debe haber al menos tres valores de entrada, si no hubiese suficiente se informará con un mensaje "Error, no hay valores suficientes". Los dos primeros valores deben ser numéricos, si no fuese el caso se dará el mensaje "Error, al convertir un valor a número real (N)", siendo N el primer valor con formato incorrecto.[4]

Posteriormente se enviará el mensaje `setRango` al objeto `datos` con argumento `"0;4"`, y se volverá a mostrar por pantalla la representación del objeto `datos`. Después se enviará un nuevo mensaje `setRango`, pero esta vez con argumento `"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`

[4]Para saber como pasar valores como argumentos al programa principal, desde el entorno de programación, consulte la Guía del Entorno de Desarrollo de Java (Eclipse).

La salida por pantalla será la siguiente (sin los saltos de línea de cada representación del objeto datos, es decir, en tres líneas):

```
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 la parte que aparece entre paréntesis hay que utilizar el método `getMessage`):

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

Ejercicio 2. (proyecto prDatos, paquete datos2)

Estudio de las excepciones Comprobadas

Crea una copia del paquete `datos` y llámalo `datos2` (con las clases incluidas), y crea una copia de la clase `PruebaDatos` denominada `PruebaDatos2` en el paquete anónimo.

Además, cambia la clase `PruebaDatos2` para que ahora utilice las clases del paquete `datos2`.

En las nuevas clases `DatosException` y `Datos` del paquete `datos2`, y en la clase distinguida `PruebaDatos2`, realiza los cambios necesarios para que la excepción `DatosException` (del paquete `datos2`) sea ahora **comprobada**.