Técnicas Actuales de Programación 2023/2024 Convocatoria Extraordinaria – 27 junio

Responda al examen con un proyecto que lleve su nombre (elija Java with Ant -> Java Application).

Además de que el código sea razonablemente correcto y dé los resultados esperados, se valorará que esté limpio de restos de pruebas y de código anulado con comentarios, que esté bien formateado (Nebeans ayuda), y que incluya algún comentario donde se considere necesario.

Es posible invertir en los mercados bursátiles con programas que ejecutan compras y ventas de acciones automáticamente siguiendo una estrategia determinada. Antes de poner en marcha uno de estos programas es aconsejable simular su funcionamiento frente a los valores de cotización pasados y de este modo valorar si puede dar buenos resultados. La valoración de si los resultados son buenos, no puede realizarse simplemente observando si se consiguen ganancias, ya que pueden ser debidas a una tendencia alcista del mercado. La valoración de una estrategia debe hacerse por comparación con otras, y en último caso frente a una estrategia puramente aleatoria.

El ejercicio propuesto consiste en generar una herramienta muy simple para ejecutar una estrategia aleatoria sobre distintas compañías cotizadas.

Los datos

Disponemos de un archivo de texto index.txt que informa en cada una de sus líneas de la existencia de datos de una empresa mediante su ticker (identificador de mercado) y su denominación. Ambos datos están separados por un tabulador. Los datos de cada empresa se encuentran en un archivo de texto cuyo nombre coincide con el ticker y la extensión ".csv" (su formato es CSV, es decir, valores separados por comas).

Index.txt

```
QQQ     Invesco QQQ Trust, Series 1
TSLA    Tesla, Inc.
AAPL    Apple Inc.
…
```

TSLA.csv

```
Open,High,Low,Close,Volume
301.586669921875,311.1700134277344,283.8233337402344,310.0,151565700
304.73333740234375,317.086669921875,301.07000732421875,306.1333312988281,86595900
…
```

Los ficheros de cada empresa tienen una cabecera nombrando los cinco campos de cada línea y a continuación una línea por día (fecha indeterminada). En realidad, sólo usaremos las secuencias de precios "High" y "Low" además de la última cifra en la columna "Close".

Todos los ficheros están codificados en UTF-8.

Estos datos se pueden descargar como datos.zip o acceder directamente a ellos dentro de la carpeta https://gtts.ehu.es/German/Docencia/2324/TAP/examenes/data

La estrategia

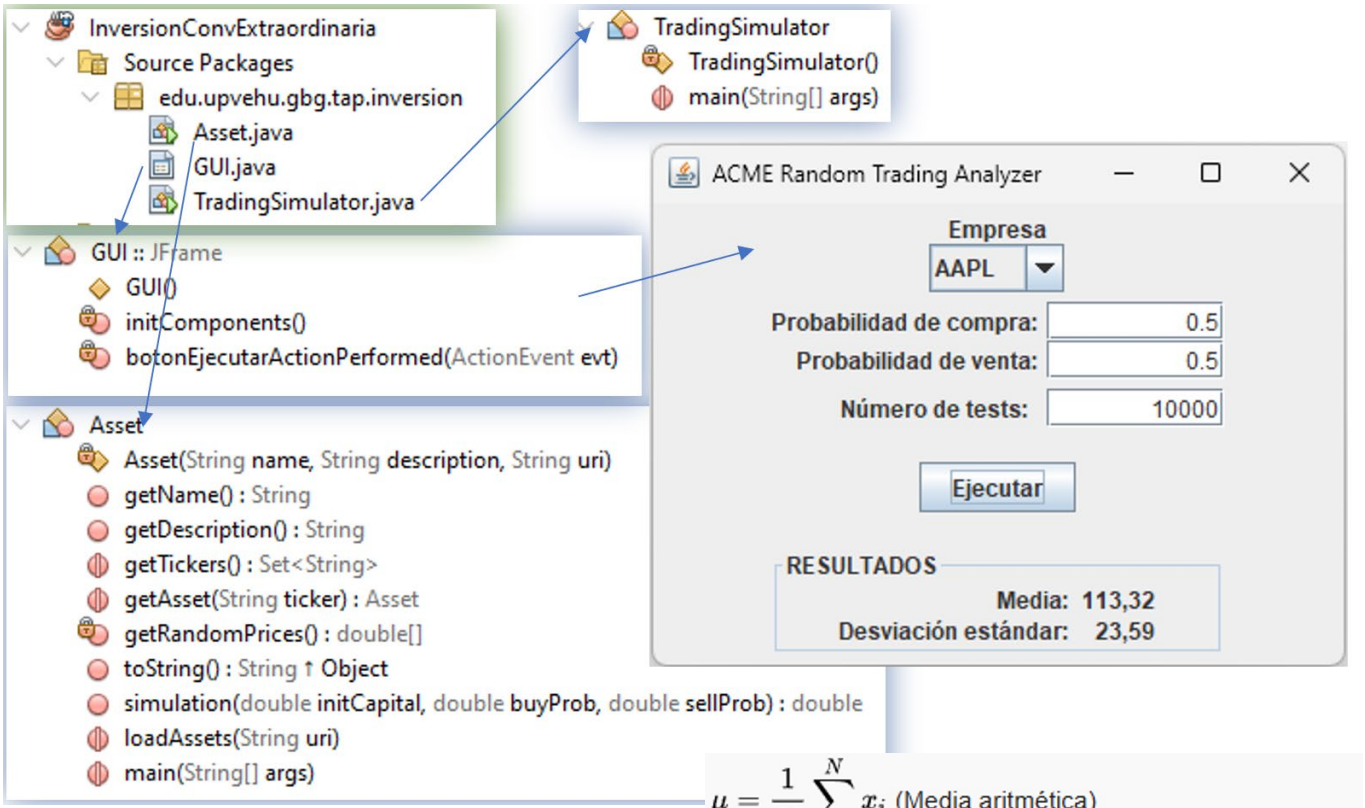
La estrategia a ejecutar será la siguiente:

- partimos de una cartera con 100 dólares y un stock de acciones vacío.
- Para cada día (línea de datos), si se tiene dinero (y no acciones) se hará una compra si un valor aleatorio supera un umbral preestablecido, y en caso contrario (si se tienen acciones y no dinero) se hará una venta si un valor aleatorio supera un umbral preestablecido (que puede ser diferente del de compra). El precio de compraventa será un valor aleatorio entre el mínimo y el máximo del día. Se pueden comprar/vender fracciones de acción, de modo que todo el capital estará volcado en dólares o en acciones en cada momento.
- Una vez hecho lo anterior para todos los días disponibles el resultado a mostrar es la cantidad de dólares en la cartera. Si está a cero porque hay stock de acciones, éstas se convierten en dólares al precio de cierre del último día.

La arquitectura de la solución

La aplicación debe proporcionar la funcionalidad mostrada en el GUI de la figura:

Para una empresa dada (`ComboBox`), al pulsar el botón, se ejecutará un número de veces indicado en un `TextField` la estrategia detallada en el apartado anterior, con las probabilidades de compra y de venta consignadas en los `TextFields` correspondientes. Como resultado se mostrará la media aritmética de los resultados obtenidos al aplicar la estrategia, así como su desviación típica muestral.



$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \text{ (Media aritmética)}$$

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2} \text{ (Desvío estándar muestral}$$

La arquitectura mostrada en la figura es una solución simple al problema:

- Arranca desde una clase de control (Llamémosla **TradingSimulator**) que lee el fichero index.txt y a continuación todos los ficheros de empresas generando, y almacenando en una colección, objetos de una clase que llamaremos **Asset**. Esto lo hace en realidad "encargándoselo" a la clase Asset llamando al método estático "`loadAssets(.)`" ya que es la responsable de todo lo que tiene que ver con las cotizaciones de acciones (assets).

- El GUI (`JFrame`) se autogenera (lo que conseguimos programar con la ayuda gráfica de Netbeans) y acto seguido da contenido al ComboBox con la lista de empresas pidiéndosela a la clase `Asset`. A partir de ahí, se limita a ejecutar un método asociado al botón cuando se pulse. Este método pide a `Asset` que ejecute la simulación tantas veces como se especifica en el interface, para terminar calculando y mostrando los valores resultantes.

- La clase `Asset`, como ya se ha indicado, se responsabiliza de todo lo que tiene que ver con los datos de cotizaciones bursátiles: proporciona el método para leer los ficheros y almacena todo en una colección de objetos `Asset` (cada uno de estos objetos corresponde a una empresa), y proporciona una serie de métodos de conveniencia para la tarea que ha de realizarse al pulsar el botón en el GUI. Como puede verse en la imagen, para realizar la simulación resulta conveniente hacerse con un array de precios "random", lo que puede ser conveniente programarlo como un método privado.

  Se añade en `Asset` un `main(.)` que permitirá ejecutar una prueba sencilla de la clase, independientemente del resto de la aplicación. Para ello se leerán los datos con `loadAssets(.)` y se mostrará un `Asset` cualquiera por pantalla (usando la implementación del `toString(.)` que presente el ticker, la descripción, y los valores máximo y mínimo de toda la serie de datos)

```
run-single:
AAPL [Apple Inc.] max: 199.62, min: 123.479805
BUILD SUCCESSFUL (total time: 1 second)
```