Técnicas Actuales de Programación 2023/2024
Examen Final - Enero

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. En el GUI se valorará el redimensionamiento razonable.

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 puede resultar interesante simular su funcionamiento frente a los valores de cotización pasados y de este modo valorar si puede dar buenos resultados. El ejercicio propuesto tiene que ver con estas simulaciones (si bien de una manera muy simplificada en muchos aspectos).

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 los precios al cierre de cada sesión (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 arquitectura de la solución

La aplicación debe arrancar por un main(.) en una clase de control (Llamémosla **TradingSimulator**) que lea 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,** para finalizar arrancando un **GUI** del que hablaremos más adelante.

Para hacer los análisis que nos interesan plantearemos dos interfaces:

MarketSignaling, con un método que permita generar una lista de "señales" (objetos de una enumeración **Signal** con valores {BUY, SELL, HOLD}) asociadas a cada día. De esta forma podremos fabricar distintas clases que implementen el interfaz y que informen con diferentes criterios en qué días es conveniente comprar, vender, o no hacer nada.

```
Signal[] getSignals(Asset asset);
```

TradingStrategy, con un método que, para un saldo inicial en la cuenta de inversión, permite simular de principio a fin la ejecución de compras y ventas dada una señalización del mercado, y obtener el saldo final que será la suma del valor de las acciones (si las hay en cartera el finalizar) y el saldo en cuenta.

```
double simulate(Asset asset,
                MarketSignaling signaling,
                double initialBalance);
```

Y en concreto implementaremos las siguientes clases para probar señalizaciones y estrategias:

AlwaysBuySignaling: Siempre dice BUY.

RandomSignaling: Indica cada día de un modo aleatorio BUY, SELL y HOLD con igual probabilidad.

BuyAtRiseSellAtFallSignaling: Con un parámetro "N" indica BUY cada día que sea el último de una secuencia de N o más subidas ininterrumpidas, e indica SELL cada día que sea el último de una secuencia de N o más bajadas ininterrumpidas. El resto son HOLD.

BuyAtFallSellAtRaiseSignaling: Es la estrategia opuesta a la anterior: con un parámetro "N" indica BUY cada día que sea el último de una secuencia de N o más bajadas ininterrumpidas, e indica SELL cada día que sea el último de una secuencia de N o más subidas ininterrumpidas. El resto son HOLD.

NaiveStrategy: Inicialmente avanza a lo largo de los días y en cuanto encuentra una señal BUY invierte todo el dinero en cuenta en comprar acciones (suponemos que se pueden comprar fracciones de acción y de ese modo invertir todo el dinero disponible en cuenta). A partir de ahí sigue avanzando y cuando encuentra una señal SELL vende todas las acciones.

Dejamos al GUI para más adelante, para hablar primero de los primeros resultados a obtener

Primeros resultados a obtener y sugerencia de desarrollo

Con todo el material planteado, vamos a hacer algo que permite comprobar que "las cosas" van funcionando sin llegar a hacer el GUI (y esto también será código evaluable).

a) En **Asset** haremos un main() que genere un objeto de un asset cualquiera (p.ej. "AAPL") e imprima por pantalla el ticker, la descripción, y los valores inicial máximo , mínimo y final de la serie de valores de cierre (tal y como se ve en el GUI más adelante)

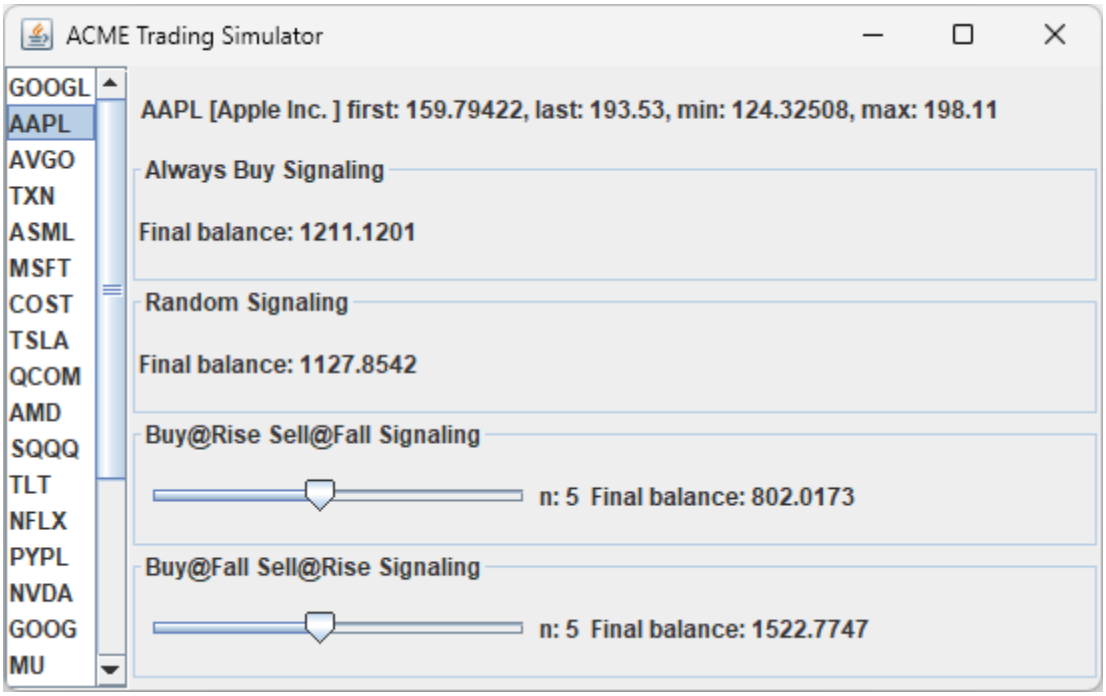
b) En **NaiveStrategy** hacemos un main() que genere también un objeto de un asset cualquiera y aplique la estrategia con todas las señalizaciones (Las dos últimas dependen de "N" por lo que se lanzarán para N en [1,10]) mostrando el balance final (fijaremos el balance inicial a 1000).

La sugerencia de desarrollo es (1) escribir **Asset** y probarla como se indica en el apartado a); (2) seguir con **MarketSignaling**, **Signal**, **TradingStrategy**, **AlwaysBuySignaling** y **NaiveStrategy** y probar según el apartado b) limitado a **AlwaysBuySignaling**; (3) o bien desarrollar el resto de señalizaciones y seguir con el GUI, o comenzar con el GUI para probar todo lo disponible y completar después las señalizaciones pendientes.

El GUI

El GUI será una ayuda gráfica para obtener los mismos resultados vistos en el punto b) anterior, pero facilitándonos seleccionar el Asset y pudiendo seleccionar un valor de N entre 1 y 10.

Un **posible** aspecto es el de la imagen. La lista de la izquierda (un componente JList) permite seleccionar un asset, de modo que a la derecha se muestra su descripción (mencionada en el apartado a) anterior) y las 4 señalizaciones, donde las que dependen de un parámetro permiten manejarlo con un JSLider.



Ayudas:

- la lista es inicializada por el GUI con unos valores por defecto. Podemos borrarlos o dejarlos, pero en todo caso pondremos los nuestros (los tickers de los assets) inmediatamente después de la llamada a **initComponents()** en el constructor del GUI. Para ello basta con darle el array de strings a la **JList** mediante el método **setListData()**
- Para atender a un cambio en la selección pondremos un listener mediante <laLista>.**getSelectionModel().addListSelectionListener(**elListener**)**. Un **SelectionListener** requiere escribir el método **void valueChanged(ListSelectionEvent e)**, donde podemos pedirle a la lista la **String** seleccionada con **getSelectedValue()**