5. Trading System Design 101

In this test, you are going to create a mini trading system.
You will need to code the following class:

```
class TradingStrategy
```

```
class OrderManager:
```

These two classes will be derived from the *AppBase* class.
You do **not** need to create the *AppBase* class.
AppBase is here to provide the function to start and stop the *TradingStrategy* and the *OrderManager*.

TradingStrategy class

The TradingStrategy class will need to have at least these 2 functions below:

```
def send_order(self,symbol, price, quantity, side)
```

```
def handle_order_from_market(self, order_execution)
```

The *TradingStrategy* class will create order by using the function *send_order*.
An order will be defined as a dictionary below (this is an example):

```
order={'strategyname' : "bob",
       'symbol' : "EUR/USD",
       'side' : OrderSide.BUY,
       'price': 12,
       'quantity' : 100,
       'order_id' : 0,
       'state' : OrderState.OPEN}
```

strategyname is the name of the trading strategy.
symbol is the symbol traded
order_id is a unique identifier for an order for a given strategy
state is the state of an order

An order can have 3 different states:

An order can have 3 different states:

```
class OrderState:
    OPEN = 1
    FILLED = 2
    CANCELLED =3
```

The side can have the values below:

```
class OrderSide:
    BUY = 1
    SELL = 2
```

The order_id will be incremented by 1 each time you will send an order. The id will start by 0. This order_id is specific to the trading strategy.
2 different trading strategies can have a same order ids.

The function *handle_order_from_market* updates the order states from a response from the market.

OrderManager class

The OrderManager class will have at least two functions:

```
def handle_order_from_ts(self,order):
```

```
def handle_order_from_market(self, order_execution):
```

The *function handle_order_from_ts* will take care of the orders sent by the trading strategy. Each order will be stored in a list of orders that you will call *orders*.
Additionally, a new key will be added to the dictionary representing the order. This new key is called *external_order_id* and will contain the specific unique identifier at the order manager level.
Each time the order manager receive an order from the trading strategy, it will increment this external_order_id by 1.

The function *handle_order_from_market* will update the order state from the market response.
Each time, you receive a market response, you should update the position of every trading strategies.
You will need to have a data structure called positions keeping track of the positions of all the symbol / strategy.

OrderNotFoundException exception

For the classes above:
handle_order_from_market will raise an exception if the order id (for the trading strategy) or the external order id (for the order manager) is not found.

MarketSimulator class

You will be given the class

```
class MarkerSimulator
```

This simulator will fill any orders having an even external_order_id value .
If it is a odd value, the order will be cancelled.
You do not need to code this function.

6. Trade Manager

You will create a class TradeManager.
This class will handle trade transaction.
It will take into account the position, the pnl for each symbol and the cash balance.

This class will have the following methods:
- update_balance updating the balance for each transation
- update_position updating the position in number of shares for each transaction
- update_pnl updating the pnl for each transation and for each symbol
- repr_balance which return a string displaying the balance
- repr_position which returns a string displaying the positions per symbol
- repr_pnl which returns a string displaying the pnl per symbol

example:
Balance:100310.0
Positions:EUR/USD 0,USD/CAD 0
PNLs:EUR/USD 10.0,USD/CAD 300.0

- set_balance which will just reset the balance to a given number. If the balance is negative it will raise an exception NegativeBalanceError

We will proceed step by step and you will code each part of this transaction separately.