TradeSystem

От спекуляций на бирже следует воздерживаться в двух случаях: если у вас нет денег, и если они у вас есть. (Марк Твен)

Вы - трейдер, который придумал несколько интересных стратегий торговли. Причем, каждую их этих стратегий можно описать алгоритмически, это вы и сделали.
Осталось написать сервис, который будет автоматически управлять этими стратегиями...

Немного контекста

Для простоты будем считать, что существует только один инструмент, которым вы торгуете(акции, облигации или шапки на рынке, сути не имеет).
В каждый момент времени, у этого инструмента есть цена. Также, ограничимся лишь одной ценой, никаких других параметров не будет.

Стретигии торговли соответственно могут либо покупать инструмент, либо продавать.

Покупая, вы создаете *позицию*. Позиция характеризуется двумя параметрами: цена входа позиции(`price`), объем позиции(`volume`).

Допустим, в данный момент времени цена актива 100. Вы покупаете этот актив по этой цене объемом 2.5. Таким образом, на эту позицию вы тратите `100 * 2.5 = 250` средств.
В общем случае, если цена актива `price`, а объем вашей позиции `volume`, то на открытие позиции вы тратите `price * volume` средств.

Также актив можно продавать. В этой задаче продажа актива, это по сути закрытие позиции. Закрыть можно только открытую позицию, причем полностью. Пример:
цена актива была `100`, размер позиции `2.5`. Вы потратили `100*2.5 = 250` средств. Когда цена стала `300`, вы решили эту позицию закрыть. Закрывая позицию, вы получите `300*2,5 = 750` средств. Итого, вы заработали `750 - 250 = 500` единиц.

Однако, если цена стала не `300`, а `20`, то, закрывая позицию, вы получите `20 * 2.5 = 50` средств. Итого `250 - 50 = 200` единиц убытка.

Задача

Ваша задача -- придумать стратегию торговли.

В тестирующей системе реализован класс

```
class AccountInterface {
public:
    virtual double GetAvailBalance() const = 0;
    virtual int OpenPosition(double positionSize) = 0;
    virtual void ClosePosition(int positionId) = 0;
};
```

```
GetAvailBalance()-- возвращает текущий размер доступных средств
OpenPosition(double positionSize) - открывает позицию по текущей цене
объемом positionSize. Возвращает уникальный индекс позиции.
ClosePosition(int positionId) - закрывает позицию с указанными индексом.
```

Будет создан объект этого класса и передан вашей программе.

Напишите абстрактный класс `ModuleBase`. Этот класс будет задавать интерфейс для любой торговой стратегии. У него должно быть два `public` метода:

```
void OnTick(double price); // Этот метод будет вызываться у вашей стратегии
всякий раз, когда приходит новая цена актива
void CloseAll(); // Этот метод будет вызываться у вашего актива, если
появится необходимость закрыть все позиции, которые открыла ваша стратегия
```

Для простоты сразу сделаем следующее ограничение: Вы знаете, что цена актива колеблется от 1 до 10, только в этом отрезке. Причем каждая новая котировка -- равновероятно берется именно из отрезка от 1 до 10. Актив изменяется исключительно по этим правилам.

Исходя из этого предположения, напишите класс `ModuleBorders`. Этот класс будет наследником `ModuleBase`

Конечно же, у него должно быть 2 `public` метода:

```
OnTick(double price);
CloseAll();
```

А также `public` конструктор:

```
ModuleBorders(AccountInterface* account);
```

Он принимает на вход объект аккаунта, а именно, указатель на него. Вашему модулю будет передан реализованный объект аккаунта, именно с помощью его методов вы сможете открывать / закрывать позиции, а также следить за текущим размером доступных средств.

Как отмечалось в начале задачи, у вас несколько торговых стратегий. Так вот, для управления этими стратегиями, реализуйте класс `ModulesRouter`

```
class ModulesRouter {
public:
    void OnTick(double price);
    void AddModule(ModuleBase* ptr);
```

```
    void CloseAll();
};
```

Этот класс хранит указатели на объекты торговых стратегий `ModuleBase*`. Стратегий может быть несколько(одна ваша, а также со стороны теструющей системы)

```
OnTick(double price); // Этот метод будет вызываться всякий раз, когда приходит новая котировка. Класс должен вызвать метод OnTick у всех модулей, которых он хранит
AddModule(ModuleBase* ptr); /// Этот метод позволяет добавить новый модуль в класс
CloseAll(); // Этот метод должен вызвать метод CloseAll у всех модулей, которых ваш класс хранит
```

Критерии оценивания

Тестирующая система создаст объект наследник класса `AccountInterface`. В этом объекте будет реализована вся логика `AccountInterface`. Также она создаст объект `ModulesRouter`, несколько торговых модулей `ModuleBase` и передаст их в `ModulesRouter`. Допустим, в системе реализованы классы:

```
class AccountInterface; // Система
class AccountChecker : public AccountInterface; // Система
class ModuleBase; // Ваш класс
class ModuleBorders : public ModuleBase; // Ваш класс
class ModuleJury : public ModuleBase; // Модуль от системы
class ModulesRouter; // Ваш класс
```

Тогда, один из запусков тестов может выглядеть так:

```
AccountChecker account;
ModulesRouter router;

ModuleBorders module1(&account);
ModuleJury module2(&account);

router.AddModule(&module1);
router.AddModule(&module2);

auto balanceBefore = account.GetAvailBalance();

for (int i = 0; i < /*Who knows*/; ++i) {
    double price = GetPrice();
    router.OnTick(price);
}
router.CloseAll();
```

```
auto balanceAfter = account.GetAvailBalance();

assert(balanceAfter > 2 * balanceBefore);
```

Ваше решение получит вердит `RE`, если она некорректно работает с балансом или позициями(нельзя закрыть несуществующую позицию или открыть позицию объемом больше, чем баланс аккаунта), либо просто содержит ошибки, независящие от задачи

Ваше решение получит вердикт `OK`, если вы напишите такую стратегию, которая будет не хуже авторской не больше чем в `100` раз, а именно, если `ModuleJury` заработал `x` единиц, то ваше решение должно получить хотя бы `0.01 * x`.

Гарантируется, что авторское решение использует простой подход к торговле в стратегии и не пользуется информацией о будущих котировках.