# 국내주식 자동매매 프로그램 설계서

| | |
|---|---|
| 생성됨 | @2025년 12월 3일 오후 10:49 |
| 태그 | 전략 알고리즘 구현 |

## 0. 전제 조건 & 공통 가정

- **대상 시장**: KOSPI / KOSDAQ 개별종목 (+ 필요 시 지수ETF)
- **상품**: 현물 위주, 공매도/파생은 1차 버전에서는 제외
- **주요 타임프레임**
  - 분봉: 1분 / 3분 / 5분
  - 일봉: 스윙 필터용
- **API/브로커**: 한국투자증권 KIS OpenAPI (또는 동급 API)

  → 주문/체결/잔고/시세 모듈은 브로커별 Adapter 패턴으로 교체 가능하게 설계
- **실행 환경**
  - 서버형(24시간 구동) Linux 클라우드 서버
  - Python

---

## 1. 시스템 전체 아키텍처

### 1-1. 모듈 구성

1. **MarketData 모듈**
   - 실시간 시세 수신 (틱/호가/분봉)
   - 과거 데이터 조회 (백테스트/리플레이용)
   - 데이터 캐싱 및 리샘플링 (틱 → 1분봉, 1분봉 → 3분봉 등)
2. **Strategy Engine (전략 엔진)**
   - 여러 개의 전략을 동시에 관리
   - 각 전략은 공통 인터페이스 사용:
     - `on_bar(symbol, bar)` : 새 봉(캔들) 도착 시 호출

    - `on_tick(symbol, tick)` : 틱 단위 사용 시
    - `on_order_update(order)` : 주문 상태 변경 콜백
    - `on_position_update(position)` : 포지션 변동 콜백
- 전략에서 생성한 "시그널"을 **Execution 모듈**에 전달

3. **Execution / Order 모듈 (주문 실행기)**
    - 전략은 "추상 주문"만 발행:
        - 예: `BUY 500주 @시장가`, `SELL_ALL @시장가`, `PLACE_STOP_LOSS`, ...
    - 실제 API 호출(시장가/지정가/조건부 등)로 변환
    - 주문 체결/취소/정정 관리 (주문ID 기반 상태 추적)
    - **주문 큐 & 속도 제한 관리**
        - 초당 주문 횟수 제한, API Rate Limit 관리(초당 10건 미만)
4. **Risk Manager (리스크 관리 모듈)**
    - 계좌 전체/전략별 리스크 관리
    - 기능:
        - 1종목당 최대 투자금 비율
        - 일일 최대 손실(계좌/전략 기준)
        - 일일 최대 이익 확정 후 "트레이딩 중지" 옵션
        - 종가 청산, 장중 손실 컷 등 글로벌 룰 적용
5. **Portfolio / Position 모듈**
    - 종목별 포지션, 평균단가, 미실현손익, 실현손익 관리
    - 전략별 포지션 분리 또는 통합(태깅 필요)

      → `position.tag = strategy_id`
6. **Logging & Monitoring**
    - 모든 시그널, 주문, 체결, 포지션 변화, 리스크 이벤트를 로그
    - Web 대시보드 or 간단한 GUI:
        - 전략별 당일 손익
        - 종목별 포지션
        - 주문 성공/실패 현황

- 에러/예외 알림 (Slack, 카톡, 이메일 등)

---

# 2. 공통 설계 원칙

## 2-1. 전략 인터페이스 (Pseudo-code)

```
class BaseStrategy:
    def __init__(self, config, broker, risk_manager, portfolio):
        self.config = config
        self.broker = broker
        self.risk = risk_manager
        self.portfolio = portfolio

    def on_bar(self, symbol, bar):
        """새 분봉/일봉 발생 시 호출"""
        raise NotImplementedError

    def on_tick(self, symbol, tick):
        """틱 기반 전략 필요 시 사용"""
        pass

    def on_order_update(self, order):
        """주문 체결/취소/거부 등"""
        pass

    def on_position_update(self, position):
        """포지션 변경 시"""
        pass
```

---

# 3. 대표 전략 설계 (4종)

> “많이 쓰이는” 범용 전략들을 의도적으로 골랐습니다.
> ① 추세추종 이평선 돌파, ② 볼린저 평균회귀,
> ③ 전일 고가 돌파 모멘텀, ④ VWAP 기반 장중 스캘핑.

각 전략은 **같은 엔진** 안에서 파라미터만 바꿔서 여러 종목에 적용 가능하게 설계합니다.

## 전략 1. 단기 이평선 골든크로스 추세추종

### 1-1. 개요

- **목적**: 단기 상승 추세에 초기에 진입해, 추세가 꺾이기 전까지 보유.
- **타임프레임**: 5분봉 (또는 15분봉)
- **매매 스타일**: 데이 + 1~2일 스윙 가능 (종가청산 옵션)

### 1-2. 입력 데이터

- 5분봉 OHLCV (최소 60봉 이상)
- 일봉 20일 이평 (종목 필터용: 중기 상승장만 매매)
- 종목 유니버스: 시가총액 상위 n개 + 거래대금 상위 n개

### 1-3. 진입 조건 (Long Only)

1. **추세 필터 (일봉)**
   - 20일 이동평균선이 우상향 ( `MA20_today > MA20_yesterday` )
   - 종가가 MA20 위 ( `close_daily > ma20_daily` )
2. **단기 골든크로스 (5분봉)**
   - 단기 MA: 5봉, 중기 MA: 20봉
   - `MA5_prev <= MA20_prev` 이고, `MA5_now > MA20_now` 이면 골든크로스 발생
3. **거래량 필터**
   - 현재 5분봉 거래량 > 최근 20봉 평균 거래량 * k (예: 1.5)

→ 모든 조건 만족 시 **매수 시그널** 발생.

### 1-4. 청산 조건

- **손절**: 진입가 대비 -2% 도달 시 시장가 손절
- **이익 실현**:
  - 1차 목표: +3% 도달 시 절반 청산
  - 나머지 절반은 **트레일링 스탑**:
    - 최고가에서 -1.5% 하락 시 청산
- **시간 청산**:

- 당일 종가에 포지션이 남아 있으면 강제 청산 옵션 선택 가능

## 1-5. 포지션/리스크 관리

- 한 종목당 계좌 평가액의 최대 5%
- 동일 전략에서 동시 보유 종목 최대 5개
- 일일 전략 손실 -5% 초과 시 해당 전략 중지

## 1-6. 의사코드

```
class MovingAverageTrendStrategy(BaseStrategy):
    def on_bar(self, symbol, bar):
        bars = self.get_bars(symbol, timeframe="5m", lookback=60)
        daily = self.get_bars(symbol, timeframe="1d", lookback=30)

        if len(bars) < 20 or len(daily) < 20:
            return

        # 일봉 추세 필터
        ma20_daily_now = daily.close[-20:].mean()
        ma20_daily_prev = daily.close[-21:-1].mean()
        if not (ma20_daily_now > ma20_daily_prev and daily.close[-1] > ma20_
daily_now):
            return

        # 5분봉 MA 계산
        ma5_now = bars.close[-5:].mean()
        ma5_prev = bars.close[-6:-1].mean()
        ma20_now = bars.close[-20:].mean()
        ma20_prev = bars.close[-21:-1].mean()

        volume_now = bars.volume[-1]
        avg_vol20 = bars.volume[-20:].mean()

        position = self.portfolio.get_position(symbol)

        # 진입
        if position is None:
```

```
            golden_cross = (ma5_prev <= ma20_prev) and (ma5_now > ma20_n
ow)
            vol_ok = volume_now > avg_vol20 * self.config["vol_k"]

            if golden_cross and vol_ok:
                qty = self.calc_position_size(symbol)
                if self.risk.can_open_new_position(symbol, qty):
                    self.broker.buy_market(symbol, qty, tag=self.config["id"])

        # 청산 로직 (포지션 있을 때)
        else:
            pnl_pct = (bar.close - position.avg_price) / position.avg_price * 100
            max_price = max(position.max_price, bar.high)

            # 손절
            if pnl_pct <= -self.config["stop_loss_pct"]:
                self.broker.sell_market(symbol, position.qty, tag=self.config["id"])
                return

            # 1차 익절
            if (not position.partial_taken) and pnl_pct >= self.config["take_profit1
_pct"]:
                half = position.qty // 2
                self.broker.sell_market(symbol, half, tag=self.config["id"])
                position.partial_taken = True

            # 트레일링 스탑
            drawdown_from_high = (bar.close - max_price) / max_price * 100
            if drawdown_from_high <= -self.config["trail_stop_pct"]:
                self.broker.sell_market(symbol, position.qty, tag=self.config["id"])
```

---

## 전략 2. 볼린저 밴드 기반 평균회귀 (Mean-Reversion)

### 2-1. 개요

- **목적**: 과도하게 눌린 구간(하단 밴드 이탈)을 매수 후 평균 회귀를 노림.
- **타임프레임**: 15분봉

- **매매 스타일**: 데이/1박2일 스윙

## 2-2. 입력 데이터

- 15분봉 기준:
  - 20봉 단순 이동평균 (MA20)
  - 20봉 표준편차
  - 볼린저 밴드 상/하단 (± 2σ)

## 2-3. 진입 조건 (Long Only)

1. 현재 종가가 **하단 밴드보다 1% 이상 아래**
   - `close < lower_band * 0.99`
2. 직전 10봉 동안 가격이 횡보/완만한 상승 (추세적 하락장 제외)
   - 예: `MA20` 기울기가 -1% 이상 급락하지 않을 것
3. 거래량 급증이 아닌 **점진적 감소 후 첫 반등**

   → 만족하기 어려우면 1차 버전에서는 거래량 필터 생략 가능

## 2-4. 청산 조건

- **기본 목표**: 중심선(MA20) 도달 시 전량 청산
- **손절**: 진입가 대비 -1.5% (평균회귀 전략은 손절 타이트하게)
- **시간 청산**: 3일 이상 보유 시 무조건 청산

## 2-5. 포지션/리스크

- 종목당 계좌의 3% (추세전략보다 작은 비중)
- 전략 전체 동시 보유 종목 10개 이내

## 2-6. 의사코드 (핵심 로직만)

```
class BollingerMeanReversion(BaseStrategy):
    def on_bar(self, symbol, bar):
        bars = self.get_bars(symbol, timeframe="15m", lookback=60)
        if len(bars) < 20:
            return
```

```
        ma20 = bars.close[-20:].mean()
        std20 = bars.close[-20:].std()
        upper = ma20 + 2 * std20
        lower = ma20 - 2 * std20
        close = bars.close[-1]

        position = self.portfolio.get_position(symbol)

        # 진입
        if position is None:
            # 하단 밴드 과대 하회
            if close < lower * 0.99:
                qty = self.calc_position_size(symbol, risk_pct=self.config["risk_pc
t"])
                if self.risk.can_open_new_position(symbol, qty):
                    self.broker.buy_market(symbol, qty, tag=self.config["id"])
            return

        # 청산
        pnl_pct = (close - position.avg_price) / position.avg_price * 100

        # 손절
        if pnl_pct <= -self.config["stop_loss_pct"]:
            self.broker.sell_market(symbol, position.qty, tag=self.config["id"])
            return

        # 평균 회귀 도달
        if close >= ma20:
            self.broker.sell_market(symbol, position.qty, tag=self.config["id"])
```

---

## 전략 3. 전일 고가 돌파 모멘텀 (Breakout)

## 3-1. 개요

- **목적**: 강한 모멘텀 종목에서 “전일 고가 돌파” 순간의 힘을 따라가는 단타/데이 전략.
- **타임프레임**: 1분 또는 5분봉

- **매매 스타일**: 당일 청산 (Overnight 금지)

## 3-2. 입력 데이터

- 전일 OHLC
- 당일 1분/5분봉 실시간
- 거래대금 랭킹

## 3-3. 진입 조건

1. 당일 시초가가 전일 종가 대비 +2% 이상 (갭상승 종목 위주)
2. 현재가가 **전일 고가를 상향 돌파** + 돌파봉의 거래량이 이전 20봉 평균의 2배 이상
3. 돌파 후 1~2봉 동안 전일 고가 위에서 유지 (페이크 돌파 방지)

## 3-4. 청산 조건

- **손절**: 전일 고가 재하향 이탈 시 or 진입가 대비 -2%
- **목표가**:
  - 1차: +3% (50% 청산)
  - 2차: 장 마감 전까지 보유하되, VWAP 아래로 종가 형성 시 청산

## 3-5. 포지션/리스크

- 종목당 계좌의 4% 이내
- 동시 보유 최대 3종목 (매우 집중적인 데이 전략)

## 3-6. 의사코드 핵심

```
class PreviousHighBreakout(BaseStrategy):
    def on_bar(self, symbol, bar):
        daily = self.get_bars(symbol, timeframe="1d", lookback=2)
        if len(daily) < 2:
            return

        prev_high = daily.high[-2]
        prev_close = daily.close[-2]
        today_open = daily.open[-1]
```

```
        bars = self.get_bars(symbol, timeframe="1m", lookback=50)
        if len(bars) < 20:
            return

        position = self.portfolio.get_position(symbol)
        close = bars.close[-1]
        volume_now = bars.volume[-1]
        avg_vol20 = bars.volume[-20:].mean()

        # 진입
        if position is None:
            gap_up = (today_open - prev_close) / prev_close * 100 >= self.config
["gap_pct"]
            breakout = (bars.high[-1] > prev_high) and (close > prev_high)
            vol_ok = volume_now > avg_vol20 * self.config["vol_k"]

            if gap_up and breakout and vol_ok:
                qty = self.calc_position_size(symbol)
                if self.risk.can_open_new_position(symbol, qty):
                    self.broker.buy_market(symbol, qty, tag=self.config["id"])
            return

        # 청산
        pnl_pct = (close - position.avg_price) / position.avg_price * 100

        # 손절: 전일 고가 이탈 or 손실 -2%
        if close < prev_high or pnl_pct <= -self.config["stop_loss_pct"]:
            self.broker.sell_market(symbol, position.qty, tag=self.config["id"])
            return

        # 익절
        if (not position.partial_taken) and pnl_pct >= self.config["take_profit1_p
ct"]:
            half = position.qty // 2
            self.broker.sell_market(symbol, half, tag=self.config["id"])
            position.partial_taken = True

        # 종가 청산 (엔진에서 장 종료 직전 한 번 호출)
```

```
        if self.is_near_close():
            self.broker.sell_market(symbol, position.qty, tag=self.config["id"])
```

---

## 전략 4. VWAP 기반 장중 스캘핑 (기관/세력 추종 느낌)

### 4-1. 개요

- **목적**: 장중 VWAP 위/아래에서의 매수/매도 압력을 이용한 단기 스캘핑
- **타임프레임**: 1분봉
- **매매 스타일**: 당일 안에서 짧은 트레이드 여러 번

### 4-2. 입력 데이터

- 시작 시점부터 현재까지의 누적 거래량/거래대금 → 실시간 VWAP 계산
- 1분봉 OHLCV

### 4-3. 진입 조건 (Long)

1. 현재가가 VWAP을 아래에서 위로 돌파 (캔들 종가 기준)
2. 돌파봉 거래량이 당일 평균 1분 거래량의 1.5배 이상
3. 직전 10분간 가격이 VWAP 아래에서 머물렀다가 처음으로 돌파

### 4-4. 청산 조건

- 손절: VWAP 재하향 이탈 + 1틱 여유
- 목표: +1% ~ +1.5% (짧은 스캘핑이므로 작은 목표)

### 4-5. 포지션/리스크

- 건당 계좌의 1~2% 수준의 작은 비중
- 하루에 종목당 최대 3회 진입 제한

### 4-6. VWAP 계산 및 의사코드

```
class VWAPScalping(BaseStrategy):
    def on_bar(self, symbol, bar):
        bars = self.get_bars(symbol, timeframe="1m", lookback=200)
        if len(bars) < 5:
```

```
            return

        typical_price = (bars.high + bars.low + bars.close) / 3
        cum_pv = (typical_price * bars.volume).cumsum()
        cum_vol = bars.volume.cumsum()
        vwap_series = cum_pv / cum_vol
        vwap_now = vwap_series.iloc[-1]

        close = bars.close[-1]
        prev_close = bars.close[-2]

        position = self.portfolio.get_position(symbol)

        # 진입
        if position is None:
            # 아래 → 위 VWAP 돌파
            crossed_up = (prev_close < vwap_series.iloc[-2]) and (close > vwap
_now)
            if crossed_up:
                qty = self.calc_position_size(symbol, risk_pct=self.config["risk_pc
t"])
                if self.risk.can_open_new_position(symbol, qty):
                    self.broker.buy_market(symbol, qty, tag=self.config["id"])
            return

        # 청산: VWAP 이탈 or 목표수익
        pnl_pct = (close - position.avg_price) / position.avg_price * 100
        if close < vwap_now or pnl_pct >= self.config["take_profit_pct"]:
            self.broker.sell_market(symbol, position.qty, tag=self.config["id"])
```

# 4. 리스크 / 자금관리 공통 설계

## 4-1. 계좌/전략 레벨 제한

- **계좌 레벨**
  - 일간 최대 손실: -X% 도달 시 모든 전략 중지, 모든 포지션 정리 옵션

    - 일간 최대 이익: +Y% 도달 후 신규 포지션 진입 금지 (이익 보호 모드)
- **전략 레벨**
    - 전략별 일간 손실/이익 제한
    - 전략별 동시 보유 종목 수 제한
    - 전략별 최대 레버리지(현물 기준이면 단순 비중)

### 4-2. 포지션 사이징 공통 함수

```
def calc_position_size(self, symbol, risk_pct=None):
    # risk_pct: 계좌 평가금 대비 이 포지션에 할당할 비율
    account_value = self.portfolio.get_account_value()
    if risk_pct is None:
        risk_pct = self.config["default_risk_pct"]  # 예: 0.03 (3%)

    alloc = account_value * risk_pct
    price = self.market_data.get_last_price(symbol)
    qty = int(alloc // price)  # 국내주식 1주 단위
    return max(qty, 1)
```

---

## 5. 백테스트 & 실거래 전환 설계

### 5-1. 백테스트 엔진 요구사항

1. **동일 전략 코드** 재사용
    - `MarketData`, `Broker` 인터페이스만 실거래/백테스트에서 다르게 구현
2. **슬리피지 & 수수료 모델**
    - 매수/매도 시 체결가격에 ±슬리피지 적용
    - 증권사 수수료 + 거래세 반영
3. **리포트 출력**
    - 전략/전체 기준
    - 수익曲선, MDD, 승률, 기대수익, 연복리(연환산), 샤프비율 등

### 5-2. 실거래 전환 체크리스트

- 모의투자 계좌로 최소 1~2개월 실시간 테스트
- 주문 실패/체결 지연, API 오류에 대한 예외처리
  - 재시도 로직
  - 주문 거부 시 Slack/카톡 알림
- 서버 재시작 시 상태 복구 로직
  - 기존 포지션/주문 정보를 브로커에서 다시 읽어와 동기화

---

# 6. 안티그래비티 개발용 정리 (요약)

1. **코어 프레임워크**
   - Strategy Engine + MarketData + Broker Adapter + RiskManager + Portfolio
2. **전략 4종 구현**
   - `MovingAverageTrendStrategy` (단기 이평 골든크로스 추세추종)
   - `BollingerMeanReversion` (볼린저 평균회귀)
   - `PreviousHighBreakout` (전일 고가 돌파 모멘텀)
   - `VWAPScalping` (VWAP 기반 장중 스캘핑)
3. **공통 요소**
   - 파라미터는 모두 Config 파일(JSON/YAML)로 외부화
   - 전략ID(`config["id"]`)로 주문/포지션 태깅
   - 리스크/포지션 사이징은 공통 모듈 사용

---