Team B - 주식 거래 프로그램 명세서

포트폴리오 리스크 분석 시스템

1. 프로그램 개요

프로그램 기능

이 프로그램은 **투자자가 보유한 주식 포트폴리오의 위험도를 분석**하고, **각 종목의 가격 흐름을 그래프로 보여주는** 시스템입니다. 실시간으로 흐르는 장 시간(09:00~15:30) 동안 주가가 변동하며, 랜덤 이벤트가 발생하여 섹터별로 주가에 영향을 미칩니다.

핵심 시나리오

1. **프로그램 시작** → 장 시작 (09:00), 포트폴리오와 리스크 점수 자동 표시
2. **실시간 시간 흐름** → 시스템 시간이 흐르고 (현실 1분 = 시스템 1초), 랜덤 이벤트 발생
3. **이벤트 발생** → 특정 섹터(IT, AUTO, FINANCE 등)에 영향, 해당 섹터 주가 일괄 변동
4. **종목 선택** → 가격 변화를 그래프로 확인
5. **리스크 재계산** → 가격 변동성, 종목별 위험도, 보유 종목 수, 현금 비율 종합 분석
6. **장 마감** (15:30) → 거래 종료, 다음날로 이동 가능

---

2. 차별화 기능

1. 텍스트 기반 시세 그래프

- 콘솔 화면에서 주식 가격 변화를 **그래프로 시각화**
- priceHistory 데이터를 활용한 시각적 표현

2. 고도화된 리스크 점수 계산

4가지 요소를 종합한 리스크 분석:

1. **보유 종목 수** (분산 투자 정도)
2. **현금 비율** (유동성)
3. **가격 변동성(Volatility)** - priceHistory에서 표준편차 계산
4. **종목별 고유 위험도** - 코인 관련(High), 일반 주식(Medium), ETF(Low)

3. 실시간 장 시간 시스템

- **실제 주식 시장처럼 장 시간 개념** 도입
- 09:00 장 시작 → 15:30 장 마감
- **시간 가속**: 현실 1분 = 시스템 1초 (빠른 시뮬레이션)
- 장 시간 외에는 거래 불가, 다음날로 넘어가기 가능

4. 섹터 기반 이벤트 시스템

- 단순 랜덤 변동이 아닌 **이벤트 기반 주가 변동**

- 각 주식은 섹터(IT, AUTO, FINANCE 등)에 속함
- 랜덤 이벤트 발생 시 해당 섹터의 모든 주식에 일괄 영향

---

3. 전체 시스템 플로우

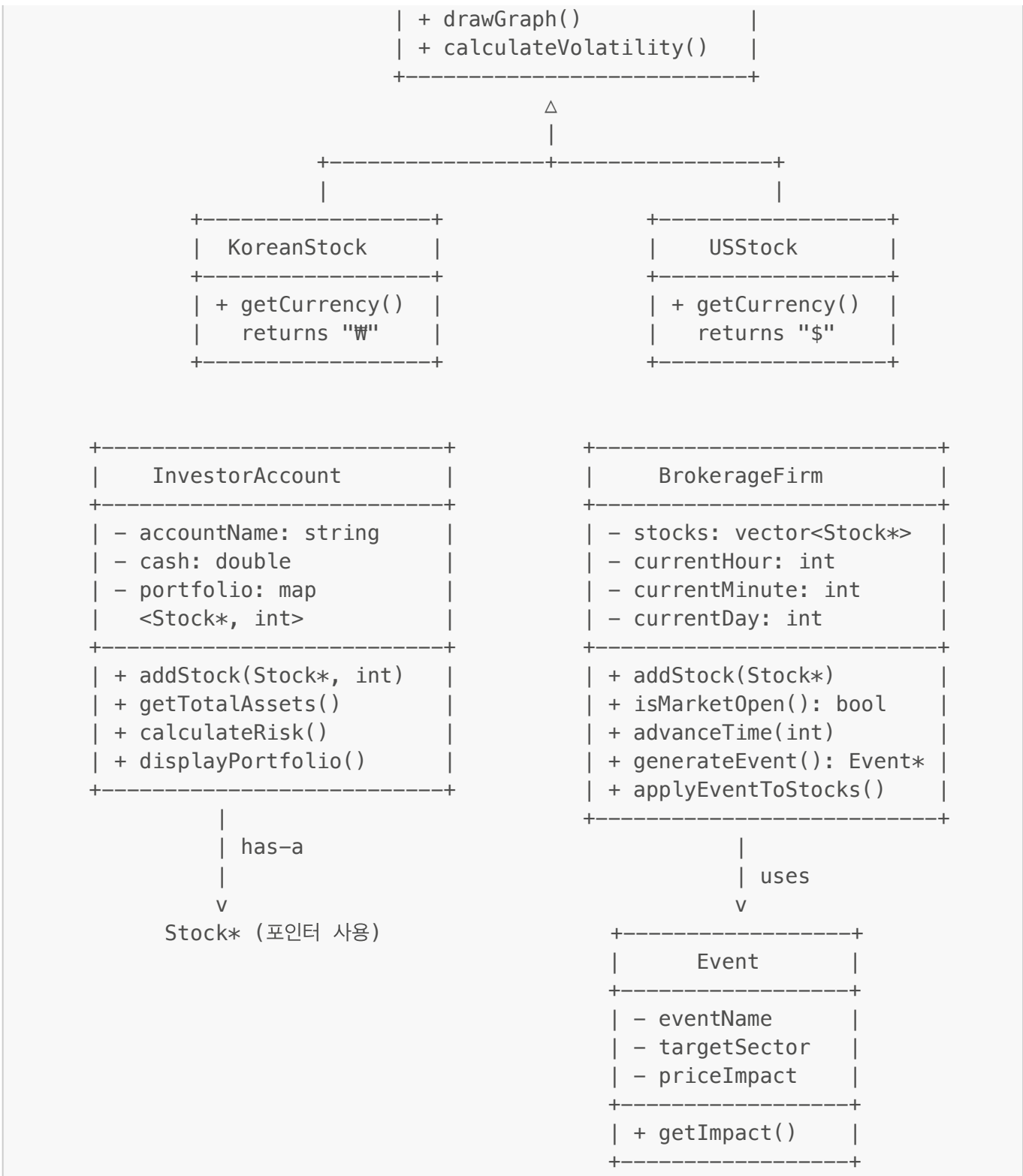
```
프로그램 시작
    ↓
장 시작 (09:00)
    ↓
포트폴리오 자동 표시 (리스크 점수 포함)
    ↓
┌──────────────────────────────┐
│  메뉴 선택                   │
│  1. 종목 그래프 보기         │
│  2. 시간 흘리기              │
│  3. 다음날로 이동            │
│  4. 종료                     │
└──────────────────────────────┘
    ↓
[시간 흐름 중]
- 실시간 시간 증가 (1초 = 1분)
- 랜덤 이벤트 발생 → 섹터별 주가 변동
- 변동성 업데이트 → 리스크 재계산
    ↓
15:30 도달 → 장 마감
    ↓
다음날 or 종료
```

---

4. 클래스 구조

클래스 다이어그램

```
                    +--------------------------+
                    |        <<abstract>>      |
                    |          Stock           |
                    +--------------------------+
                    | - name: string           |
                    | - currentPrice: double   |
                    | # priceHistory: vector   |
                    | - sector: string         |
                    | - riskLevel: double      |
                    +--------------------------+
                    | + Stock(...)             |
                    | + getCurrency(): string  |
                    |   (pure virtual)         |
                    | + updatePrice(double)    |
```



4.1 Stock (추상 클래스)

- **역할**: 모든 주식의 부모 클래스
- **자식 클래스**: KoreanStock, USStock
- **관계**: 상속 (is-a)
- **주요 멤버**:

    - 데이터: name, currentPrice, priceHistory, sector, riskLevel
    - 함수: getCurrency() [순수 가상], updatePrice(), drawGraph(), calculateVolatility()

4.2 KoreanStock / USStock

- **역할**: 통화별 주식 구현
- **차이점**: getCurrency() 반환값 ("₩" vs "$")

4.3 InvestorAccount

- **역할**: 투자자 계좌 및 포트폴리오 관리
- **관계**: Stock 객체를 구성 (has-a)
- **주요 멤버**:
    - 데이터: accountName, cash, portfolio (map<Stock*, int>)
    - 함수: addStock(), getTotalAssets(), calculateRisk(), displayPortfolio()

4.4 BrokerageFirm

- **역할**: 증권사 - 주식 관리 및 시간/이벤트 시스템
- **관계**: Stock 객체를 구성 (has-a), Event 객체를 사용 (uses)
- **주요 멤버**:
    - 데이터: stocks (vector), currentHour, currentMinute, currentDay
    - 함수: addStock(), isMarketOpen(), advanceTime(), generateEvent(), applyEventToStocks()

4.5 Event

- **역할**: 섹터별 주가 영향 이벤트
- **주요 멤버**:
    - 데이터: eventName, targetSector, priceImpact
    - 함수: getImpact()

클래스 간 관계

1. **상속**: Stock ← KoreanStock, USStock
2. **구성**: InvestorAccount has-a Stock*, BrokerageFirm has-a Stock*
3. **의존**: BrokerageFirm uses Event

---

5. 클래스 명세

5.1 Stock (추상 클래스)

역할: 모든 주식의 부모 클래스

구분	이름	타입	설명
Data Member	name	string	종목명
	currentPrice	double	현재가
	priceHistory	vector<double>	가격 이력 (최대 20개)

구분	이름	타입	설명
	sector	string	섹터 (IT, AUTO, FINANCE 등)
	riskLevel	double	종목 고유 위험도 (0.0~1.0)
Member Function	Stock(name, price, sector, risk)	-	생성자
	getCurrency()	string	순수 가상 함수 (₩ or $)
	updatePrice(newPrice)	void	가격 업데이트, priceHistory에 추가
	drawGraph()	void	그래프 출력
	calculateVolatility()	double	priceHistory 표준편차 계산

OOP 요소: 추상 클래스, 캡슐화, 다형성

5.2 KoreanStock / USStock

역할: 한국/미국 주식 (통화 구분)

구분	설명
KoreanStock::getCurrency()	"₩" 반환
USStock::getCurrency()	"$" 반환

OOP 요소: 상속, 다형성

5.3 InvestorAccount

역할: 투자자 계좌 및 포트폴리오 관리

구분	이름	타입	설명
Data Member	accountName	string	계좌 소유자
	cash	double	보유 현금
	portfolio	map<Stock*, int>	주식 포인터와 수량
Member Function	InvestorAccount(name, cash)	-	생성자
	addStock(stock, quantity)	void	포트폴리오에 주식 추가
	getTotalAssets()	double	총 자산 계산
	calculateRisk()	RiskReport	리스크 점수 (0~100)
	displayPortfolio()	void	포트폴리오 출력 (리스크 포함)

calculateRisk() 계산 요소:

1. 보유 종목 수
2. 현금 비율
3. 각 종목의 변동성 (calculateVolatility 활용)
4. 각 종목의 riskLevel

OOP 요소: 구성, 캡슐화

5.4 BrokerageFirm

역할: 증권사 - 전체 주식 관리, 시간 흐름, 이벤트 발생

구분	이름	타입	설명
Data Member	stocks	vector<Stock*>	거래 가능한 모든 주식
	currentHour	int	현재 시각 (시)
	currentMinute	int	현재 시각 (분)
	currentDay	int	현재 날짜
Member Function	addStock(stock)	void	주식 추가
	displayAllStocks()	void	전체 주식 목록 출력
	getStockByName(name)	Stock*	이름으로 주식 검색
	isMarketOpen()	bool	장 시간 체크 (09:00~15:30)
	advanceTime(seconds)	void	시간 경과 (1초 = 1분)
	generateEvent()	Event*	랜덤 이벤트 생성
	applyEventToStocks(event)	void	섹터별 가격 변동 적용
	updateAllPrices()	void	모든 주식 가격 업데이트

OOP 요소: 구성, 의존

5.5 Event

역할: 섹터별 주가 영향 이벤트

구분	이름	타입	설명
Data Member	eventName	string	이벤트 이름
	targetSector	string	영향받는 섹터
	priceImpact	double	가격 영향률 (예: +5% = 0.05)
Member Function	Event(name, sector, impact)	-	생성자
	getImpact()	double	영향률 반환

사용 예시: "IT 호황" 이벤트 발생 → IT 섹터 모든 주식 5% 상승

---

6. 세부 기능 명세

6.1 Stock::drawGraph()

함수 원형:

```
void Stock::drawGraph() const
```

목적: priceHistory 벡터 데이터를 콘솔 화면에 10단계 높이의 텍스트 그래프로 시각화

그래프 설정:

- Graph Height (Y축): 10단계
- Max Width (X축): 최근 20개 데이터로 제한
- 사용 문자:
  - 데이터 포인트: ●
  - 데이터 아래 채우기: |
  - Y축 경계: |
  - X축 경계: --
  - 축 교차점: +
- ANSI Color Code 사용하여 시각적 구분감 제공

정규화 로직:

```
Max = max(priceHistory)
Min = min(priceHistory)
Range = Max - Min (단, Range <= 0이면 1.0으로 보정)

특정 가격 P의 그래프 상 높이:
h = (int)((P - Min) / Range * HEIGHT)
```

Y축 라벨링:

- 가독성을 위해 짝수 행에만 가격 표시
- 표시할 가격: `Price_i = Min + (Range * i / HEIGHT)`
- getCurrency() 활용: "₩" 또는 "$" 자동 표기

예외 처리:

- priceHistory가 비어있으면 "[No Data]" 출력 후 리턴
- 모든 가격이 동일할 경우 Range=1로 설정하여 0으로 나누기 오류 방지

출력 예시:

6.2 Stock::calculateVolatility()

함수 원형:

```
double Stock::calculateVolatility() const
```

목적: priceHistory를 이용해 일일 수익률(Daily Return)의 표준편차를 계산하여 변동성을 정량적으로 평가

계산 절차:

1. **유효성 체크**
   - priceHistory.size() < 2일 때 → 0.0 반환
2. **일일 수익률 계산**

```
수익률 r_i = (P_i - P_{i-1}) / P_{i-1}

P_i: i번째 날짜의 가격
r_i: i번째 일일 수익률
priceHistory의 길이가 n+1이면 수익률의 개수는 n개
```

3. **평균 수익률 계산**

```
평균 r̄ = (1/n) * Σ r_i
```

4. **분산 계산**

```
분산 σ² = (1/n) * Σ(r_i - r̄)²
```

5. **표준편차(변동성) 계산**

```
변동성 = √σ²
```

반환값:

- 변동성이 클수록 → 고위험 종목
- 변동성이 작을수록 → 저위험 종목

6.3 InvestorAccount::calculateRisk()

함수 원형:

```
RiskReport InvestorAccount::calculateRisk() const

struct RiskReport {
    double totalScore;  // 0~100
    string grade;       // "Low" / "Medium" / "High"
    string comment;     // 분석 코멘트
};
```

목적: 4가지 리스크 요소를 점수화하여 0~100점의 최종 리스크 점수 계산

4가지 리스크 요소 (각 0~25점):

1) 변동성 점수 (scoreVol)

```
maxVol = 0.05 (기준 최대 변동성 5%)
avgVol = (각 종목의 변동성 합) / 종목 수
ratioVol = avgVol / maxVol (0~1로 제한)
scoreVol = ratioVol × 25
```

2) 분산도 점수 (scoreDiversification)

```
H = 보유 종목 수
H_max = 10 (충분히 분산된 기준)

divRatio = {
    0.0,           H = 0
    H / 10.0,      0 < H < 10
```

```
    1.0,           H >= 10
}

scoreDiversification = (1.0 - divRatio) × 25
```

- 종목이 적을수록 점수 높음 (위험)

3) 고위험 종목 비율 점수 (scoreHighRisk)

```
고위험 종목 = calculateVolatility() > 0.05인 종목
H_high = 고위험 종목 개수

highRiskRatio = H_high / H
scoreHighRisk = highRiskRatio × 25
```

4) 현금 비율 점수 (scoreCash)

```
Total = 보유 주식 평가액 합 + Cash
cashRatio = Cash / Total

scoreCash = (1.0 - cashRatio) × 25
```

- 현금이 적을수록 점수 높음 (위험)

최종 점수:

```
totalScore = scoreVol + scoreDiversification + scoreHighRisk + scoreCash
```

리스크 등급:

- 0~33점: **Low** (저위험)
- 34~66점: **Medium** (중간 위험)
- 67~100점: **High** (고위험)

코멘트:

- Low: "포트폴리오가 전반적으로 안정적인 편입니다. 변동성과 고위험 종목 비중이 낮습니다."
- Medium: "일부 고위험 요소가 존재합니다. 종목 분산 및 현금 비율을 주기적으로 점검하는 것이 좋습니다."
- High: "포트폴리오의 변동성과 고위험 종목 비율이 매우 높습니다. 자산 배분 조정이 권장됩니다."

6.4 InvestorAccount::displayPortfolio()

출력 형식:

```
==========================================
 투자자 포트폴리오 요약
==========================================

[1] 종목명 : NVDA
 - 현재 가격 : 182.40 달러
 - 보유 수량 : 10주
 - 변동성(Volatility) : 0.0213 (2.13%)

[2] 종목명 : TSLA
 - 현재 가격 : 252.70 달러
 - 보유 수량 : 5주
 - 변동성(Volatility) : 0.0461 (4.61%)

------------------------------------------
 포트폴리오 통계
------------------------------------------
- 평균 변동성 : 0.0337 (3.37%)
- 전체 보유 종목 수 : 2개
- 고위험 종목 비율 : 50%
- 현금 비율 : 28%

------------------------------------------
 리스크 분석 결과
------------------------------------------
- 총 리스크 점수(0~100) : 62.4점
- 리스크 등급 : 중간 위험(Medium)
- 종합 코멘트 :
 일부 고위험 요소가 존재합니다. 종목 분산 및 현금 비율을
 주기적으로 점검하는 것이 좋습니다.

==========================================
```

6.5 BrokerageFirm::isMarketOpen()

함수 원형:

```
bool BrokerageFirm::isMarketOpen() const
```

목적: 현재 시스템 시간이 장시간 내(09:00~15:30)에 있는지 확인

반환값:

- true: 장 중
- false: 장 마감

로직:

```
currentHour > 9 and currentHour < 15 → true
currentHour == 9 and currentMinute >= 0 → true
currentHour == 15 and currentMinute <= 30 → true
나머지 → false
```

6.6 BrokerageFirm::advanceTime(int systemSeconds)

함수 원형:

```
void BrokerageFirm::advanceTime(int systemSeconds)
```

목적: 시간 가속 (현실 1초 = 시스템 1분). 장 중 주가 업데이트 및 이벤트 발생 처리

매개변수:

- systemSeconds: 현실 시간으로 경과한 시간 (초 단위)

시간 변환:

```
marketMinutes = systemSeconds
(설정 규칙: 현실 1초 = 시장 1분)
```

처리 절차:

1. **시간 갱신**

   ```
   currentMinute에 marketMinutes 더하기
   60분 넘으면 currentHour 증가
   24시 초과하면 currentDay 증가, currentHour = 0
   ```

2. **장 중 체크**

   ```
   isMarketOpen() 호출
   - true: updateAllPrices() + generateEvent()/applyEventToStocks() 실행
   - false: 시간만 흐르고 주가 업데이트 및 이벤트 발생 로직 건너뜀
   ```

6.7 BrokerageFirm::generateEvent()

함수 원형:

```
Event* BrokerageFirm::generateEvent()
```

목적: 장 중일 때 정해진 확률에 기반하여 랜덤 이벤트 객체 생성

반환값:

- Event*: 생성된 Event 객체의 포인터
- nullptr: 이벤트 미발생 시

처리 절차:

1. **장 중 체크**

   ```
   isMarketOpen() == false → nullptr 반환
   ```

2. **이벤트 발생 확률**

   ```
   advanceTime() 호출당 7% 확률로 이벤트 발생 시도
   ```

3. **이벤트 선택**
   - 정의된 이벤트 리스트에서 무작위 선택
   - 선택된 정보로 Event 객체 동적 생성하여 포인터 반환

이벤트 종류 리스트:

IT 섹터

이벤트명	priceImpact 범위
AI 투자 열풍	+3% ~ +5%
반도체 공급 부족	+2% ~ +4%
빅테크 규제 강화	-3% ~ -5%

AUTO 섹터

이벤트명	priceImpact 범위
전기차 보조금 증가	+2% ~ +4%
글로벌 자동차 판매 감소	-3% ~ -5%
배터리 가격 하락	+1% ~ +3%

FINANCE 섹터

이벤트명	priceImpact 범위
금리 인상 발표	-2% ~ -4%
은행 실적 호조	+2% ~ +3%
금융 규제 완화	+1% ~ +3%

6.8 BrokerageFirm::applyEventToStocks(Event* event)

함수 원형:

```
void BrokerageFirm::applyEventToStocks(Event* event)
```

목적: generateEvent()를 통해 생성된 이벤트 객체를 입력받아, 이벤트가 지정하는 특정 섹터의 모든 주식 가격에 변동률 적용

매개변수:

- event: 적용할 Event 객체의 포인터 (targetSector와 priceImpact 포함)

처리 절차:

1. **이벤트 정보 추출**

   ```
   targetSector = event->targetSector
   priceImpact = event->priceImpact
   ```

2. **주식 순회 및 적용**

   ```
   stocks 벡터를 순회하면서:
   - stock->sector == targetSector인지 확인
   - 일치 시:
     새로운 가격 = 현재 가격 × (1 + priceImpact)
     stock->updatePrice(새로운 가격) 호출
   ```

출력 예시:

```
[10:30] 이벤트 발생!
→ AI 투자 열풍 (IT 섹터 +4.2%)

가격 변동:
- 삼성전자 (IT): 70,000₩ → 72,940₩ (+4.2%)
- Apple (IT): $180.5 → $188.1 (+4.2%)
- 현대차 (AUTO): 180,000₩ (변동 없음)
```

7. OOP 요소

상속 (Inheritance)

- Stock (추상 클래스) ← KoreanStock, USStock
- 순수 가상 함수(getCurrency) 사용
- 공통 기능은 부모에, 차이점은 자식에

설계 이유:

- 한국/미국 주식은 모두 주식이지만 통화가 다름
- 공통 기능(가격 업데이트, 그래프)은 부모에, 차이점(통화)은 자식에

다형성 (Polymorphism)

- Stock* 포인터로 KoreanStock/USStock 모두 참조
- getCurrency() 호출 시 **런타임에 결정**

```
Stock* stock = new KoreanStock(...);
stock->getCurrency();  // "₩" (다형성!)
```

중요성:

- InvestorAccount의 portfolio는 Stock* 포인터 사용
- 한국/미국 주식 구분 없이 동일하게 관리 가능

구성 (Composition)

- InvestorAccount **has-a** Stock* (map으로 관리)
- BrokerageFirm **has-a** Stock* (vector로 관리)
- BrokerageFirm **uses** Event

설계 이유:

- 계좌는 주식을 "소유"함 (강한 연관)
- 증권사는 주식을 "관리"함
- 이벤트는 필요할 때만 생성하여 "사용"

캡슐화 (Encapsulation)

- private/protected/public 접근 제어
- 데이터 멤버는 private/protected, 외부 접근은 getter/setter만 제공
- priceHistory는 protected → 자식 클래스만 접근

중요성:

- 데이터 무결성 보장 (직접 수정 불가)
- 변경사항 발생 시 함수만 수정하면 됨

추상화 (Abstraction)

- 복잡한 리스크 계산 → calculateRisk() 하나로 추상화
- 그래프 출력 로직 → drawGraph() 하나로 추상화
- 이벤트 적용 → applyEventToStocks() 하나로 추상화

중요성:

- 사용자는 내부 구현 몰라도 함수 호출만으로 기능 사용
- 복잡도 숨기고 인터페이스만 노출

---

8. 구현 참고 사항

초기 데이터 설정

구현 시 다음 샘플 데이터로 초기화해주세요:

생성할 주식:

- 한국 주식: 삼성전자, 현대차, LG화학
- 미국 주식: Apple, Tesla

샘플 priceHistory (그래프 테스트용):

- 삼성전자: [65000, 67000, 68500, 69000, 70000, 71000, 70500, 69500, 70000]
- Apple: [170.0, 172.5, 175.0, 177.5, 180.0, 182.0, 181.0, 179.5, 180.5]
- Tesla: [230.0, 235.0, 240.0, 238.0, 242.0, 245.0, 247.0, 246.0, 245.3]

샘플 포트폴리오:

- 계좌명: "투자자"
- 초기 현금: 1,000,000원
- 보유 종목: 삼성전자 10주, Apple 5주

메인 메뉴 구조

```
프로그램 실행 중:
  1. 포트폴리오 자동 표시 (리스크 점수 포함)
  2. 메뉴 표시
  3. 사용자 선택 처리
  4. 종료까지 반복
```

시간 시스템 참고사항

- 외부 타이머를 사용하여 주기적으로 advanceTime() 호출
- 변환: 현실 1초 = 시장 1분
- 장 시간(09:00~15:30) 동안에만 이벤트 발생 가능
- 장 시간 외에는 시간만 흐름

- **중요**: 한국 주식과 미국 주식의 거래 시간은 현실과 달리 프로그램 내에서 동기화되어 동일한 시간(09:00~15:30)에 거래됩니다. 실제로는 한국 장과 미국 장의 시간이 다르지만, 이 프로그램에서는 시뮬레이션의 단순화를 위해 모든 주식이 같은 시간대에 거래되는 것으로 가정합니다.

명세서 버전: 1.0 **작성일**: 2025.11.28 **팀**: B