객체지향 프로그래밍 실습 과제

6주차. Inheritance

과제 개요

5주차 실습과제를 업그레이드하여 상속을 적용해보자.

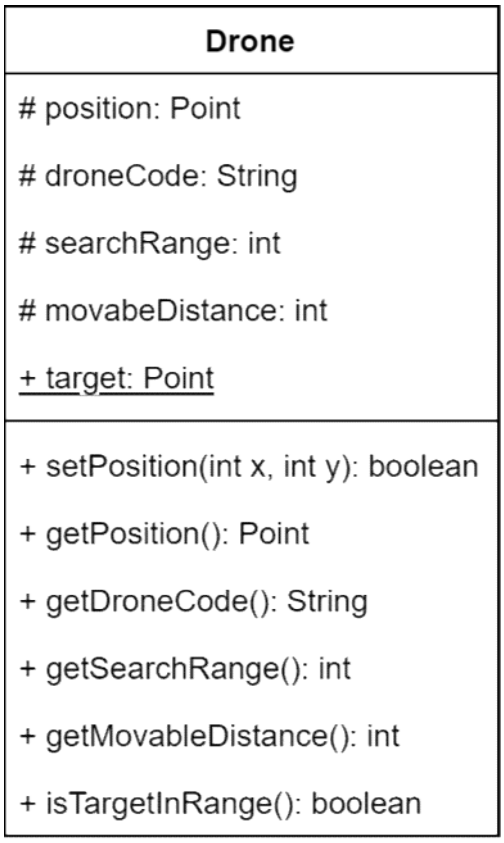
클래스 설명에 따로 제한사항이 주어지지 않는 한, 메서드 또는 필드변수를 자유롭게 선언 및 활용할 수 있다.

과제 세부사항

1. 세부 지시사항

1. 패키지 assignment.hw06.drone을 생성하고, DroneTest 클래스를 제외한 모든 클래스를 assignment.hw06.drone에 위치시킨다. DroneTest 클래스는 assignment.hw06에 위치시킨다.
2. GlobalHawk, Predator, Quadrotor 클래스에 대한 class diagram에서 Drone 클래스와 겹치는 이름의 메소드는 반드시 오버라이딩하여 구현한다.
3. 아래 클래스 설명 중 회색으로 된 부분은 5주차 과제와 동일한 설명임을 의미한다.

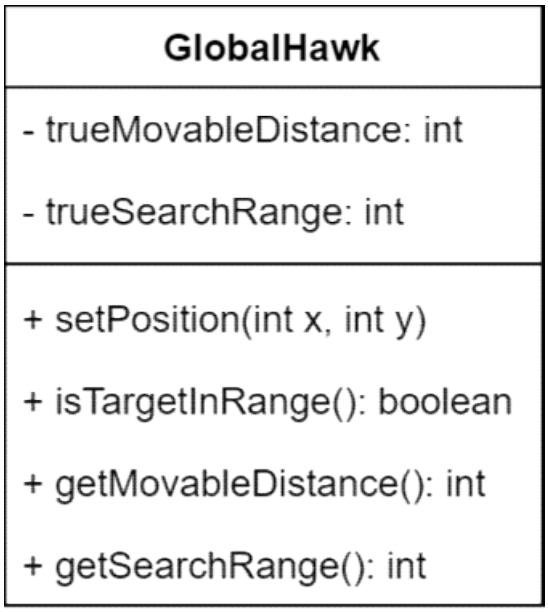
2. Drone class (30%)



1. 멤버의 접두사 #은 protected member를 의미한다.
2. position 멤버는 Drone의 현재 위치를 담고 있다.
3. droneCode는 Drone의 관제코드를 의미한다.
4. searchRange는 Drone의 적군 탐색 가능 범위를 의미한다.
5. movableDistance는 Drone이 한 번에 움직일 수 있는 최대 거리를 의미한다.

6. target은 Drone이 추적해야 할 목표의 위치로써, 모든 객체가 같은 target을 가진다.
7. 필드변수를 초기화 할 수 있도록 생성자를 구현한다.
8. Drone 클래스의 getters, setters는 diagram에 주어진 메서드 외에 구현하지 않는다.
9. setPosition(int x, int y)은 드론을 위치 (x, y)로 이동시키기 위한 기능이다. 만약 현재 위치와 이동하려는 위치 사이의 거리가 movableDistance보다 크다면 false를 반환하고 현재 위치를 변경하지 않는다(움직이지 않는다). 반대로 둘 사이의 거리가 movableDistance 보다 작거나 같다면 true를 반환하고 현재 위치를 메서드 인자 x와 y값으로 변경한다. DroneUtil.calculateDistance() 메서드를 활용한다.
10. isTargetInRange()는 Drone의 searchRange 범위 내에 target이 위치해 있는지 여부를 반환한다. DroneUtil.calculateDistance() 메서드를 활용한다.

3. GlobalHawk class (20%)



1. GlobalHawk 클래스가 Drone 클래스를 상속하도록 구현한다.
2. 멤버변수를 초기화 할 수 있도록 생성자를 구현한다.
3. GlobalHawk 드론은 movableDistance에서 2배 보너스를 받는다. 이를 trueMovableDistance에 저장한다. 따라서 movableDiatance가 4일 경우, trueMovableDistance는 8이되며 GlobalHawk 드론이 한 번에 움직일 수 있는 최대거리는 8이 된다.
4. GlobalHawk 드론은 searchRange에서 3배 보너스를 받는다. 이를 trueSearchRange에 저장한다. 따라서 searchRange가 4일 경우, trueSearchRange는 12가 된다. (수정)
5. getMovableDistance()는 trueMovableDistance를 반환한다.
6. getSearchRange()는 trueSearchRange를 반환한다.
7. setPosition(int x, int y) 메서드에서 한 번에 갈 수 있는 거리는 trueMovableDistance를 따른다.
8. isTargetInRange() 메서드에서 search range는 trueSearchRange를 따른다.

4. Predator class (20%)

Predator
- trueMovableDistance: int - trueSearchRange: int
+ setPosition(int x, int y) + isTargetInRange(): boolean + getMovableDistance(): int + getSearchRange(): int

1. Predator 클래스가 Drone 클래스를 상속하도록 구현한다.
2. 멤버변수를 초기화 할 수 있도록 생성자를 구현한다.
3. Predator 드론은 movableDistance에서 3배 보너스를 받는다. 이를 trueMovableDistance에 저장한다. 따라서 movableDiatance가 4일 경우, trueMovableDistance는 12가 되며 Predator 드론이 한 번에 움직일 수 있는 최대거리는 12가 된다. (수정)
4. Predator 드론은 searchRange에서 4배 보너스를 받는다. 이를 trueSearchRange에 저장한다. 따라서 searchRange가 4일 경우, trueSearchRange는 16이 된다. (수정)
5. getMovableDistance()는 trueMovableDistance를 반환한다.
6. getSearchRange()는 trueSearchRange를 반환한다.
7. setPosition(int x, int y) 메서드에서 한 번에 갈 수 있는 거리는 trueMovableDistance를 따른다.
8. isTargetInRange() 메서드에서 search range는 trueSearchRange를 따른다.

5. Quadrotor class (20%)

Quadrotor
- trueMovableDistance: int
+ setPosition(int x, int y) + getMovableDistance(): int

1. Quadrotor 클래스가 Drone 클래스를 상속하도록 구현한다.
2. 멤버변수를 초기화 할 수 있도록 생성자를 구현한다.

3. Predator 드론은 movableDistance에서 0.5배 보너스를 받는다. 이를 trueMovableDistance에 저장한다. 따라서 movableDiatance가 4일 경우, trueMovableDistance는 2가 되며 Predator 드론이 한 번에 움직일 수 있는 최대거리는 2가 된다. (만약 결과값이 나누어떨어지지 않을 경우, 정수가 되도록 반올림한다)
4. getMovableDistance()는 trueMovableDistance를 반환한다.
5. setPosition(int x, int y) 메서드에서 한 번에 갈 수 있는 거리는 trueMovableDistance를 따른다.

6. DroneManager class

DroneManager
- droneList: List<Drone>
+ addDrone(Drone drone) + removeDrone(String droneCode) + removeDrone(int index) + getDrone(String droneCode): Drone + getDrone(int index): Drone + getDroneList(): List<Drone> + setTarget(int positionX, int positionY)

1. 업그레이드 사항이 없으므로 5주차 과제와 동일한 코드를 사용하도록 한다.
2. Drone을 여러 개 담을 수 있도록 리스트를 필드로 갖는다.
3. addDrone(Drone drone) 메서드는 drone 객체를 droneList에 추가하는 역할을 수행한다.
4. removeDrone(...) 메서드는 droneList에 추가된 drone 객체를 삭제하는 역할을 수행한다.
    1) removeDrone(String droneCode)은 droneList에서 droneCode가 동일한 객체를 리스트에서 삭제한다.
    2) removeDrone(int index)는 droneList에서 index 번째에 위치한 객체를 리스트에서 삭제한다.
5. getDrone(...)는 droneList에 있는 drone객체 중 하나를 반환하는 역할을 수행한다.
    1) getDrone(String droneCode)는 droneCode가 동일한 객체를 droneList에서 찾아, 이를 반환한다.
    2) getDrone(int index)는 droneList에서 index 번째에 위치한 객체를 반환한다.
6. setTarget() 메서드는 Drone.target 클래스변수의 값을 재설정한다.

7. Point class

Point
- positionX: int - positionY: int
+ setPosition(x: int, y: int) + getPositionX(): int + getPositionY(): int

1. 업그레이드 사항이 없으므로 5주차 과제와 동일한 코드를 사용하도록 한다.
2. Point는 2차원 좌표계를 표현하기 위한 클래스로써, positionX와 positionY 값을 가진다.
3. 필드변수를 초기화 할 수 있도록 생성자를 구현한다.

8. DroneUtil class

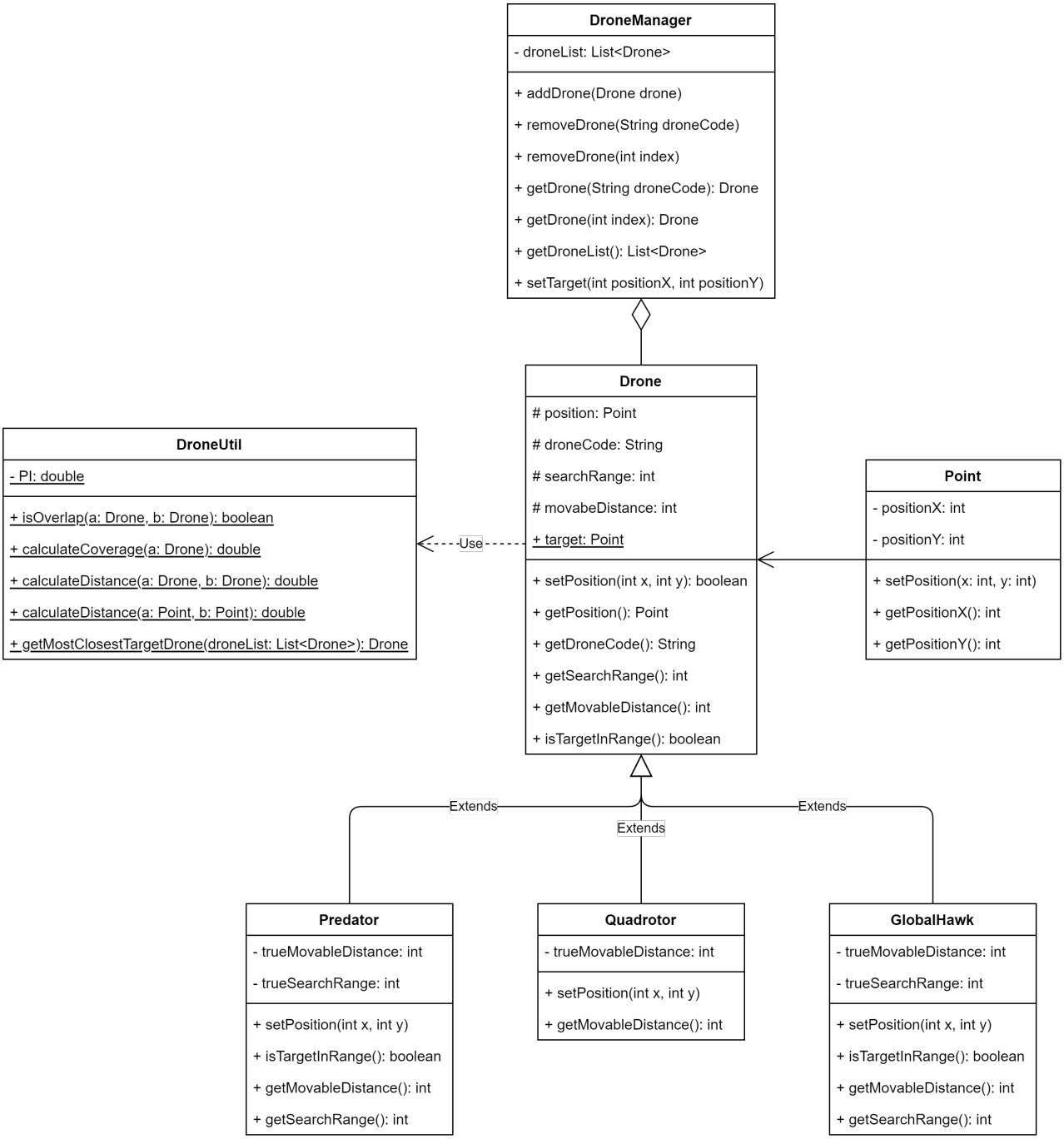
DroneUtil
- PI: double
+ isOverlap(a: Drone, b: Drone): boolean + calculateCoverage(a: Drone): double + calculateDistance(a: Drone, b: Drone): double + calculateDistance(a: Point, b: Point): double + getMostClosestTargetDrone(droneList: List<Drone>): Drone

1. 업그레이드 사항이 없으므로 5주차 과제와 동일한 코드를 사용하도록 한다.
2. DroneUtil은 Drone과 Point와 관련하여 여러 가지 연산을 지원하기 위한 클래스이다.
3. isOverlap() 메서드는 Drone a와 b의 searchRange가 겹치는지 아닌지를 반환한다.
4. calculateCoverage() 메서드는 searchRange를 반지름으로 가지는 원의 넓이를 반환한다.
5. calculateDistance()는 Drone a와 b 또는 Point a와 b 사이의 거리를 반환한다.
6. getMostClosestTargetDrone() 메서드는 droneList에 담겨있는 Drone 중에서 target과 가장 가까운 거리에 위치한 Drone 객체를 반환한다.
7. PI 변수는 3.14159265359로 초기화 한다.
8. DroneUtil 클래스의 모든 필드와 메서드는 static 타입이다.
9. DroneUtil 클래스 외부에서 DroneUtil 클래스의 메서드를 호출할 때에는 static import를 활용한다.

9. DroneTest class (10%)

1. main 메소드를 생성하여 아래 사항이 작동할 수 있도록 한다.
2. 아래 출력 예시와 같은 기능을 수행할 수 있도록 구현한다.

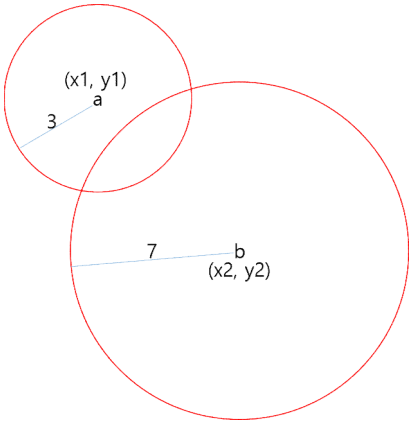
전체 클래스에 대한 Class Diagram



주의 사항

1. droneCode가 중복되지 않는다고 가정한다.
2. 소스코드가 들어있는 Eclipse 프로젝트 폴더와 실행결과 캡처 사진을 압축하여 제출한다.
3. 압축파일 명은 "학번_이름_HW05"으로 한다.
4. Java code convention(Camel case 등)을 준수하고 간단한 주석을 작성한다.
5. 프로그램 종료(8번 선택지)를 입력하기 전까지 프로그램은 계속 실행되고 있어야 한다.
6. Eclipse Preference에서 텍스트파일 인코딩을 UTF-8로 설정한다. (미흡 시 감점)

7. isOverlap()메서드는 아래처럼 searchRange가 각각 3, 7인 Drone의 원이 겹치는 경우를 true로 판정한다.



출력 예시

```
1. 드론 추가
2. 드론 삭제
3. 드론 위치 변경
4. 드론 조회
5. 타겟 지정
6. 타겟과 가장 가까운 드론 검색
7. 드론 상호 조회
8. 프로그램 종료
선택: 1

drone type: default
drone code: Lightning
position x: 10
position y: 5
search range: 5
movable distance: 7

1. 드론 추가
2. 드론 삭제
3. 드론 위치 변경
4. 드론 조회
5. 타겟 지정
6. 타겟과 가장 가까운 드론 검색
7. 드론 상호 조회
8. 프로그램 종료
선택: 1

drone type: GlobalHawk
drone code: Thunder
position x: 1
position y: 35
search range: 7
movable distance: 8
```

드론을 추가할 때 drone type을 받아야 한다. drone type이 'default'일 경우에는 Drone 클래스를, 'GlobalHawk'일 경우에는 GlobalHawk 클래스를, 'Predator'일 경우에는 Predator 클래스를,

'Quatrotor'일 경우에는 Quatrotor 클래스의 인스턴스를 생성해야 한다. 또한 movable distance를 받을 수 있어야 한다.

```
drone type: default  drone code:  Lightning
x: 10  y:  5  search range: 5  movable distance: 7

drone type: GlobalHawk  drone code:    Thunder
x:  1  y: 35  search range: 21  movable distance: 16
```

4번을 선택했을 때의 모습이다.

```
drone type: default  drone code:  Lightning
x: 10  y:  5  search range: 5  movable distance: 7

drone type: GlobalHawk  drone code:    Thunder
x:  1  y: 35  search range: 21  movable distance: 16


1. 드론 추가
2. 드론 삭제
3. 드론 위치 변경
4. 드론 조회
5. 타겟 지정
6. 타겟과 가장 가까운 드론 검색
7. 드론 상호 조회
8. 프로그램 종료
선택: 3

drone code: Lightning
position x: 100
position y: 100
거리가 너무 멀어서 이동할 수 없습니다.

1. 드론 추가
2. 드론 삭제
3. 드론 위치 변경
4. 드론 조회
5. 타겟 지정
6. 타겟과 가장 가까운 드론 검색
7. 드론 상호 조회
8. 프로그램 종료
선택: 4

drone type: default  drone code:  Lightning
x: 10  y:  5  search range: 5  movable distance: 7

drone type: GlobalHawk  drone code:    Thunder
x:  1  y: 35  search range: 21  movable distance: 16
```

드론 위치를 변경할 때의 모습이다. 거리가 멀어서 이동할 수 없다면 움직이지 않도록 구현해야 한다.

```
drone code: Lightning
position x: 10
position y: 10
정상적으로 이동하였습니다.

1. 드론 추가
2. 드론 삭제
3. 드론 위치 변경
4. 드론 조회
5. 타겟 지정
6. 타겟과 가장 가까운 드론 검색
7. 드론 상호 조회
8. 프로그램 종료
선택: 4

drone type: default  drone code:  Lightning
x: 10  y: 10  search range: 5  movable distance: 7

drone type: GlobalHawk  drone code:    Thunder
x:  1  y: 35  search range: 21  movable distance: 16
```

거리가 멀지 않아서 이동할 수 있는 거리라면, 해당 위치로 움직이도록 구현해야 한다.

제출 기한

1. 일요일 23:59 까지: 100%
2. 월요일 23:59 까지: 70%
3. 이외: 0%