객체지향 프로그래밍 실습 과제

7주차. Polymorphism & Interfaces

과제 개요

6주차 실습과제를 업그레이드하여 상속을 적용해보자.

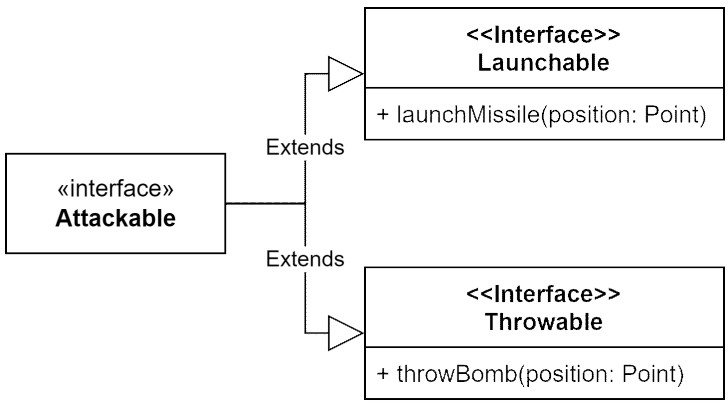
클래스 설명에 따로 제한사항이 주어지지 않는 한, 메서드 또는 필드변수를 자유롭게 선언 및 활용할 수 있다.

과제 세부사항

1. 세부 지시사항

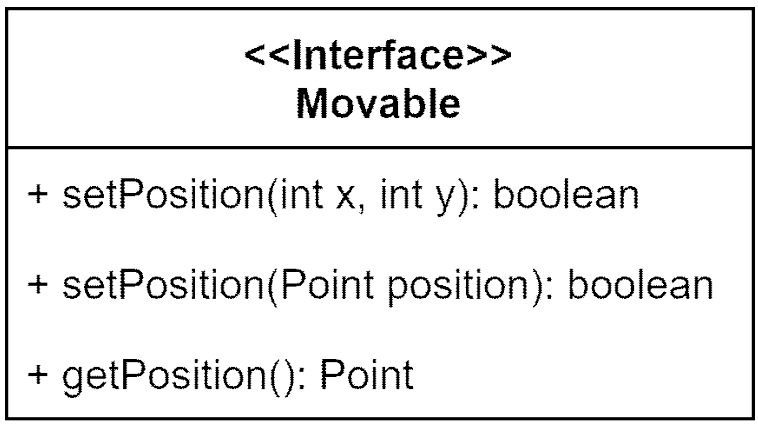
1. 패키지 assignment.hw06.drone을 생성하고, DroneTest 클래스를 제외한 모든 클래스를 assignment.hw06.drone에 위치시킨다. DroneTest 클래스는 assignment.hw06에 위치시킨다.
2. 하위클래스에서 상위클래스와 겹지는 이름의 메서드는 반드시 오버라이드 한다.
3. 오버라이드하는 메서드에는 반드시 @Override 어노테이션을 표기한다.
4. 아래 클래스 설명 중 회색으로 된 부분은 6주차 과제와 동일한 설명임을 의미한다.

2. Attackable interface (20%)



1. Attackable, Launchable, Throwable 인터페이스를 클래스 다이어그램처럼 구현한다.

3. Movable interface (20%)



1. Movable 인터페이스를 클래스 다이어그램처럼 구현한다.

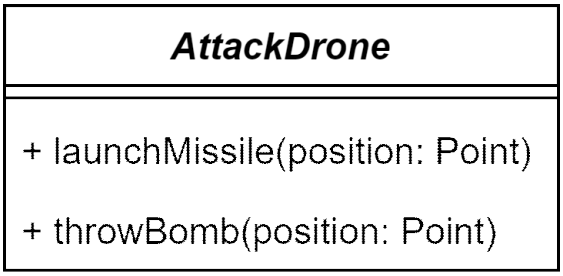
4. Drone abstract class (30%)

Drone
- position: Point - droneCode: String - searchRange: int - movabeDistance: int - target: Point
+ setTarget(x: int, y: int): boolean + setTarget(target: Point): boolean + getTarget(): Point + setPosition(x: int, y: int): boolean + setPosition(position: Point): boolean + getPosition(): Point + getDroneCode(): String + getSearchRange(): int + getMovableDistance(): int + isTargetInRange(): boolean + *toString: String*

1. Drone 클래스는 Abstract class 이다.
2. Drone 클래스가 Movable을 구현하도록 한다.
3. Static 필드인 target를 제외한 모든 필드를 초기화 할 수 있도록 생성자를 작성한다.
4. toString() 메서드를 abstract로 선언한다.
5. position 멤버는 Drone의 현재 위치를 담고 있다.
6. droneCode는 Drone의 관제코드를 의미한다.
7. searchRange는 Drone의 적군 탐색 가능 범위를 의미한다.
8. movableDistance는 Drone이 한 번에 움직일 수 있는 최대 거리를 의미한다.
9. target은 Drone이 추적해야 할 목표의 위치로써, 모든 객체가 같은 target을 가진다.
10. Drone 클래스의 getters, setters는 diagram에 주어진 메서드 외에 구현하지 않는다.
11. 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.
12. isTargetInRange()는 Drone의 searchRange 범위 내에 target이 위치해 있는지 여부를 반환

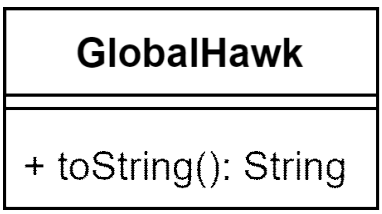
한다. DroneUtil.calculateDistance() 메서드를 활용한다.

5. AttackDrone abstract class (20%)



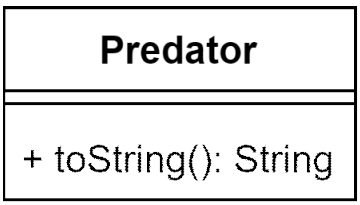
1. AttackDrone이 Drone 클래스를 상속하도록 한다.
2. AttackDrone이 Attackable를 구현하도록 한다.
3. launchMissile(Point position) 메서드는 position 위치에 미사일을 발사하는 기능이다. 만약 드론의 현재 위치와 position 사이의 거리가 searchRange보다 멀다면 발사할 수 없다는 메시지를 출력한다. 만약 거리가 searchRange보다 가깝다면 미사일을 발사했다는 메시지를 출력한다.

6. GlobalHawk class (20%)



1. GlobalHawk 클래스가 AttackDrone을 상속하도록 한다.
2. 상위 클래스의 멤버변수를 초기화 할 수 있도록 생성자를 구현한다.
3. super() 키워드로 상위 클래스의 생성자에 인자를 넘겨줄 때, movableDistance는 **2배**가 되도록 하며 searchRange는 **3배**가 되도록 넘겨준다. 즉, 생성자의 movableDistance 매개변수에 3을 입력하면, getMovableDistance()의 반환 값은 6이 되어야 한다.

7. Predator class (20%)



1. Predator 클래스가 AttackDrone을 상속하도록 한다.
2. 상위 클래스의 멤버변수를 초기화 할 수 있도록 생성자를 구현한다.
3. super() 키워드로 상위 클래스의 생성자에 인자를 넘겨줄 때, movableDistance는 **3배**가 되도록 하며 searchRange는 **4배**가 되도록 넘겨준다. 즉, 생성자의 movableDistance 매개변수에 3을 입력하면, getMovableDistance()의 반환 값은 9가 되어야 한다.

8. Quadrotor class (20%)

Quadrotor
+ toString(): String

1. Predator 클래스가 Drone을 상속하도록 한다.
2. 상위 클래스의 멤버변수를 초기화 할 수 있도록 생성자를 구현한다.
3. super() 키워드로 상위 클래스의 생성자에 인자를 넘겨줄 때, movableDistance는 **0.5배**가 되도록하며 searchRange는 1**배**가 되도록 넘겨준다. 즉, 생성자의 movableDistance 매개변수에 3을 입력하면, getMovableDistance()의 반환 값은 2가 되어야 한다.

9. 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 사항은 없으나, 여러 클래스들의 구성이 업데이트 되었으므로 작동이 되지 않는 부분은 수정하여 사용한다.
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.

10. Point class

Point
- positionX: int - positionY: int
+ getPositionX(): int + getPositionY(): int

1. Point 클래스를 immutable 하게 사용하고자 한다. 따라서 setter를 삭제하였다.
2. Point는 2차원 좌표계를 표현하기 위한 클래스로써, positionX와 positionY 값을 가진다.
3. 필드변수를 초기화 할 수 있도록 생성자를 구현한다.

11. DroneUtil class

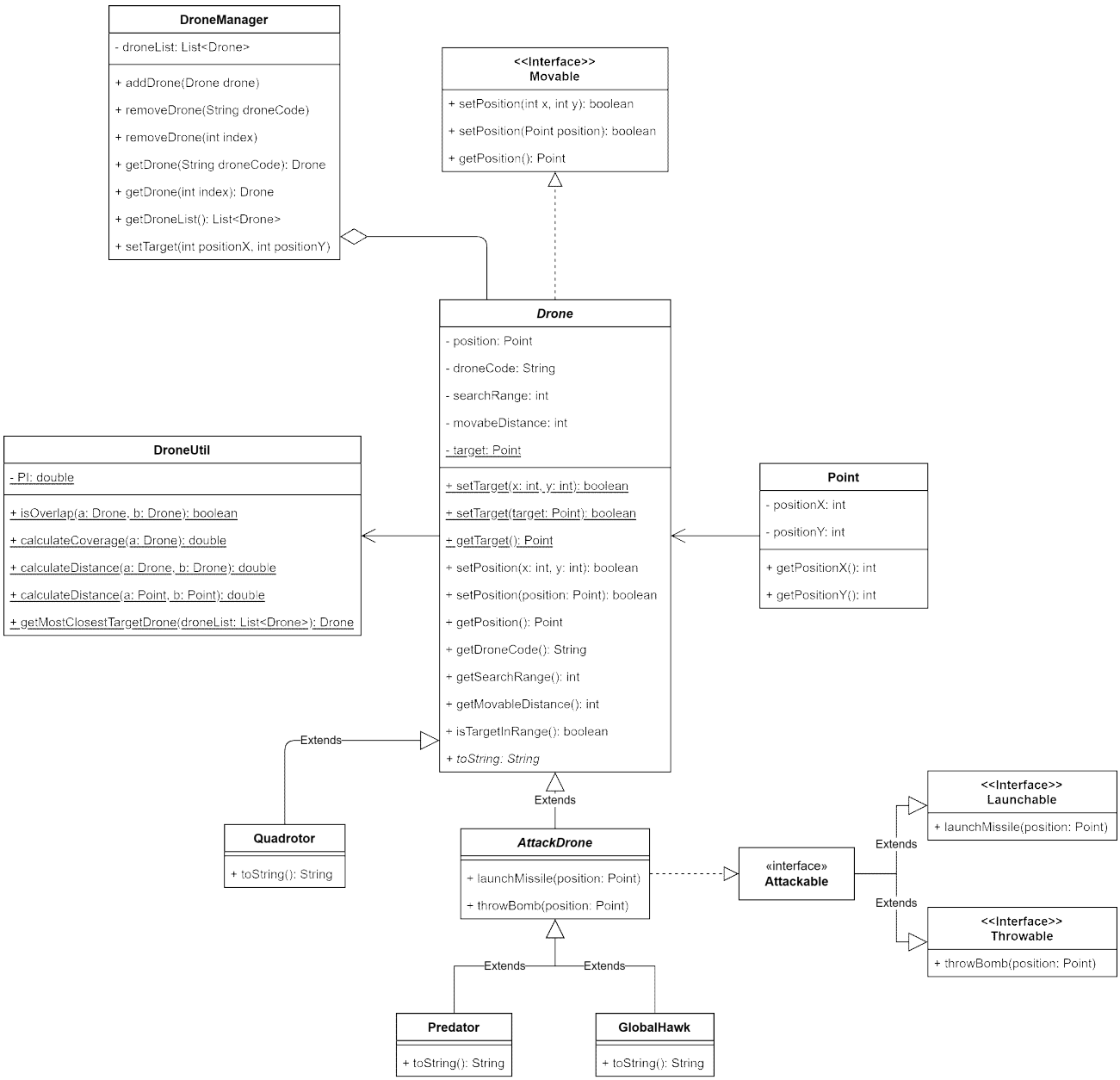
DroneUtil
- PI: double
+ isOverlap(a: Drone, b: Drone): boolean + calculateCoverage(a: Drone): double + calculateDistance(a: Drone, b: Drone): double + calculateDistance(a: Point, b: Point): double + getMostClosestTargetDrone(droneList: List<Drone>): Drone

1. 업그레이드 사항은 없으나, 여러 클래스들의 구성이 업데이트 되었으므로 작동이 되지 않는 부분은 수정하여 사용한다.
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.

12. 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_HW07”으로 한다.
4. Java code convention(Camel case 등)을 준수하고 간단한 주석을 작성한다.
5. 프로그램 종료(10번 선택지)를 입력하기 전까지 프로그램은 계속 실행되고 있어야 한다.
6. Eclipse Preference에서 텍스트파일 인코딩을 UTF-8로 설정한다. (미흡 시 감점)

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. 미사일 발사
9. 폭탄 투하
10. 프로그램 종료
선택: 1

drone type: GlobalHawk
drone code: GH1
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. 미사일 발사
9. 폭탄 투하
10. 프로그램 종료
선택: 1

drone type: Quadrotor
drone code: QR1
position x: 7
position y: 9
search range: 5
movable distance: 7
```

드론을 추가할 때 drone type을 받아야 한다. drone type이 'GlobalHawk'일 경우에는 GlobalHawk 클래스를, 'Predator'일 경우에는 Predator 클래스를, 'Quatrotor'일 경우에는 Quatrotor 클래스의 인스턴스를 생성해야 한다. 또한 movable distance와 search range를 받을 수 있어야 한다. default drone type은 삭제되었다.

```
drone type: GlobalHawk  drone code:        GH1
x: 10  y:  5  search range: 15  movable distance: 14

drone type: Quadrotor  drone code:        QR1
x:  7  y:  9  search range: 5  movable distance: 4
```

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
```

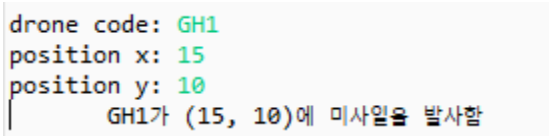
거리가 멀지 않아서 이동할 수 있는 거리라면, 해당 위치로 움직이도록 구현해야 한다.

```
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. 미사일 발사
9. 폭탄 투하
10. 프로그램 종료
선택: 8

drone code: QR1
       QR1는 Attack Drone이 아닙니다!

1. 드론 추가
2. 드론 삭제
3. 드론 위치 변경
4. 드론 조회
```

AttackDrone이 아닌 객체는 미사일 발사 또는 폭탄 투하를 할 수 없도록 한다.

```
drone code: GH1
position x: 100
position y: 100
       GH1는 (100, 100)와 멀리 떨어져 있습니다.
```

GH1이 발사 위치와 멀리 떨어진 경우.



발사위치가 search range 이내인 경우

제출 기한

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