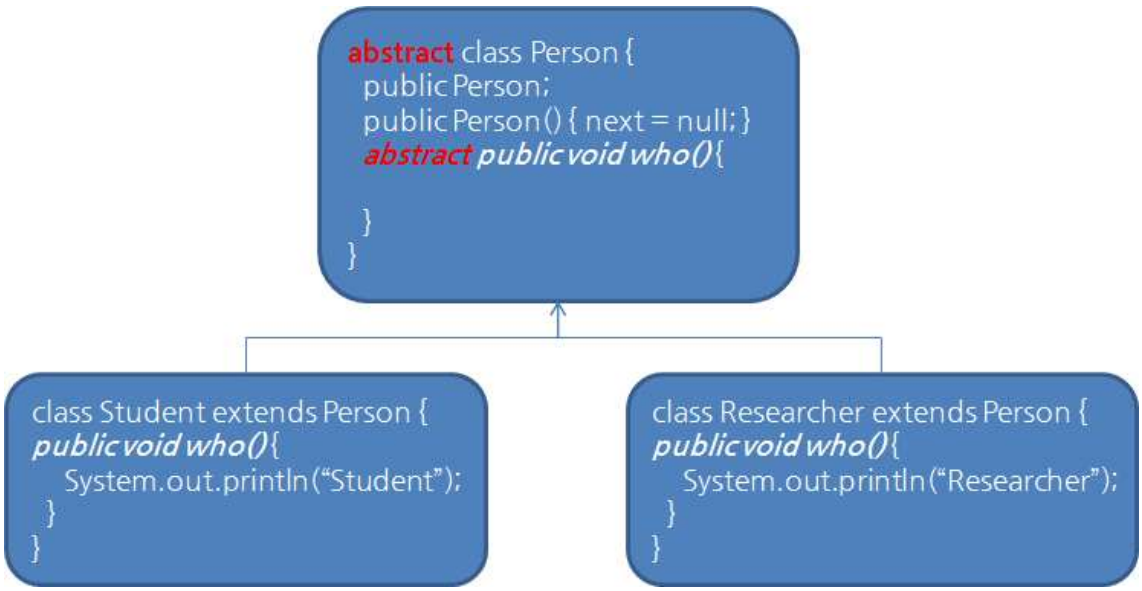
Week 6 : Interfaces

Part1. Abstract Class

- 추상클래스의 상속

추상 클래스를 상속받으면 상속받은 서브 클래스는 추상 클래스가 됨.
서브 클래스가 추상 클래스가 되지 않기 위해서는 추상 메소드를 모두 오버라이딩해야 함.



Part2. Interfaces

추상클래스의 한 종류로, 다중 상속이 가능하게 함.
implements 키워드는 인터페이스의 추상 메소드를 클래스에서 구현하는 것을 말함.

- 인터페이스의 특징

① 추상클래스와 다르게 반드시 추상 메소드와 상수만으로 구성
② 모든 메소드는 abstract public이며 생략 가능함.
③ 상수도 public static final을 생략하여 선언 가능함.
④ 인터페이스의 객체를 생성할 수 없음.
⑤ 다른 인터페이스에 상속될 수 있음.

⑥ 인터페이스도 레퍼런스 변수의 타입으로 사용 가능.

- 교재예제

```
interface CanFight {
        void fight();
}

interface CanSwim {
        void swim();
}

interface CanFly {
        void fly();
}

class ActionCharacter {
        public void fight() {
        System.out.println("fight");
        }
}

class Hero extends ActionCharacter implements CanFight, CanSwim, CanFly {
        public void swim() {
        System.out.println("swim");
        }

        public void fly() {
        System.out.println("fly");
        }
}

public class Adventure {
        public static void t(CanFight x) {
        x.fight();
        }

        public static void u(CanSwim x) {
        x.swim();
        }
```

```
        public static void v(CanFly x) {
        x.fly();
        }

        public static void w(ActionCharacter x) {
        x.fight();
        }

        public static void main(String[] args) {
        Hero h = new Hero();
        t(h); // Treat it as a CanFight
        u(h); // Treat it as a CanSwim
        v(h); // Treat it as a CanFly
        w(h); // Treat it as an ActionCharacter
        }
}
```

-교재 예제 : 상속을 포함한 인터페이스

```
interface Monster {
        void menace();
}

interface DangerousMonster extends Monster {
        void destroy();
}

interface Lethal {
        void kill();
}

class DragonZilla implements DangerousMonster {
        public void menace() {
        System.out.println("menace1");
        }

        public void destroy() {
        System.out.println("destory1");
        }
}
```

```
interface Vampire extends DangerousMonster, Lethal {
        void drinkBlood();
}

class VeryBadVampire implements Vampire {
        public void menace() {
        System.out.println("menace2");
        }

        public void destroy() {
        System.out.println("destroy2");
        }

        public void kill() {
        System.out.println("kill2");
        }

        public void drinkBlood() {
        System.out.println("drinkBlood2");
        }
}

public class HorrorShow {
        static void u(Monster b) {
        b.menace();
        }

        static void v(DangerousMonster d) {
        d.menace();
        d.destroy();
        }

        static void w(Lethal l) {
        l.kill();
        }

        public static void main(String[] args) {
        DangerousMonster barney = new DragonZilla();
        System.out.println("1");
```

```
            u(barney);
            System.out.println("2");
            v(barney);
            Vampire vlad = new VeryBadVampire();
            System.out.println("3");
            u(vlad);
            System.out.println("4");
            v(vlad);
            System.out.println("5");
            w(vlad);
            }
}
```

☞ 인터페이스를 만들 때, 다중 인터페이스 상속이 가능함. 인터페이스를 통한 다중 상속이 구현되었음을 확인 할 수 있음.

- 인터페이스 vs 추상클래스

추상클래스	인터페이스
- 일반 메소드 포함 가능 - 상수, 변수 필드 포함 가능 - 모든 서브 클래스에 공통된 메소드가 있는 경우에는 추상 클래스가 적합	- 모든 메소드가 추상 메소드 - 상수 필드만 포함 가능 - 다중 상속 지원