

4000번의 도전
드디어 찾아낸 반응 단백질



2012 인텔 국제과학경진대회 고든무어상 수상
(Intel ISEF 2012, Gordon E. Moore Award)



AN ANTONIO SPURS
처음에는 막막한 느낌이지만 시간이 지나면 감이 오거든요.



처음 배울때 감이 안나는게 뭐가 있을까요?



프로그래밍은 천재들만 할 수 있는것이 아니에요. 의지력이 필요할 뿐입니다.



Java 프로그래밍 수업내용 정리 및 실습 과제

책을 보거나 인터넷을 찾아보면서 배우고 거기에 저만의 방식을 추가했어요.



JACK ANDRAKA
IS CREATING A REVOLUTION



저는 여러분에게 컴퓨터를 통해 어떻게
세상을 바꿀 수 있는진 얘기하고 싶습니다

그러나 사실, 구글이나 위키피디아 등을 이용해
알고 싶은 모든 것을 찾을 수 있고

GOOGLE, WIKIPEDIA
AND STACKOVERFLOW

2020.04.22.수

B반 송명훈



15세 소년의 유일한 발명 도구는 “더 좋은 진단 키트를 만들고 싶었다.”
인터넷(internet)
- 잭 안드라카

목 차

1. 과제 리뷰 및 오답 정리
2. 수업 내용 정리
3. 실습화면 캡처
4. 연습문제 12-1
5. 연습문제 12-2
6. 연습문제 12-3
7. 연습문제 12-4
8. 연습문제 리뷰 내용 정리



1. 과제 리뷰 및 오답 정리

1. 과제 리뷰 및 오답 정리

```
public class PoisonKinoko extends Kinoko {
    private int poisonAttackCount = 5;

    public PoisonKinoko(char suffix) {
        super(suffix);
    }

    public int getPoissonAttackCount() {
        return poisonAttackCount;
    }

    @Override
    public void attack(Hero hero) {
        super.attack(hero);
        if(poisonAttackCount > 0) {
            System.out.println("추가로, 독 포자를 살포했다!");
            int poisonAttack = (int) (hero.getHp() / 5);
            hero.setHp(hero.getHp() - poisonAttack);
            System.out.println(poisonAttack + "포인트의 데미지");
            poisonAttackCount--;
        }
    }
}
```

PoisonKinoko class

불필요한 형 변환
⇒ 삭제

Attack보단 Damage가 좀 더 명확함
변수 이름 변경

```
public class PoisonKinoko extends Kinoko {
    private int poisonAttackCount;

    public PoisonKinoko(char suffix) {
        super(suffix);
        poisonAttackCount = 5;
    }

    public int getPoissonAttackCount() {
        return poisonAttackCount;
    }

    @Override
    public void attack(Hero hero) {
        super.attack(hero);
        if (poisonAttackCount > 0) {
            System.out.println("추가로, 독 포자를 살포했다!");
            int poisonDamage = hero.getHp() / 5;
            hero.setHp(hero.getHp() - poisonDamage);
            System.out.println(poisonDamage + "포인트의 데미지");
            poisonAttackCount--;
        }
    }
}
```



2. 수업 내용 정리

2. 수업 내용 정리 [추상클래스와 인터페이스]

추상클래스(abstract class) : 상세 부분이 미정인 메서드를 가진, 상속의 재료로 사용될 클래스

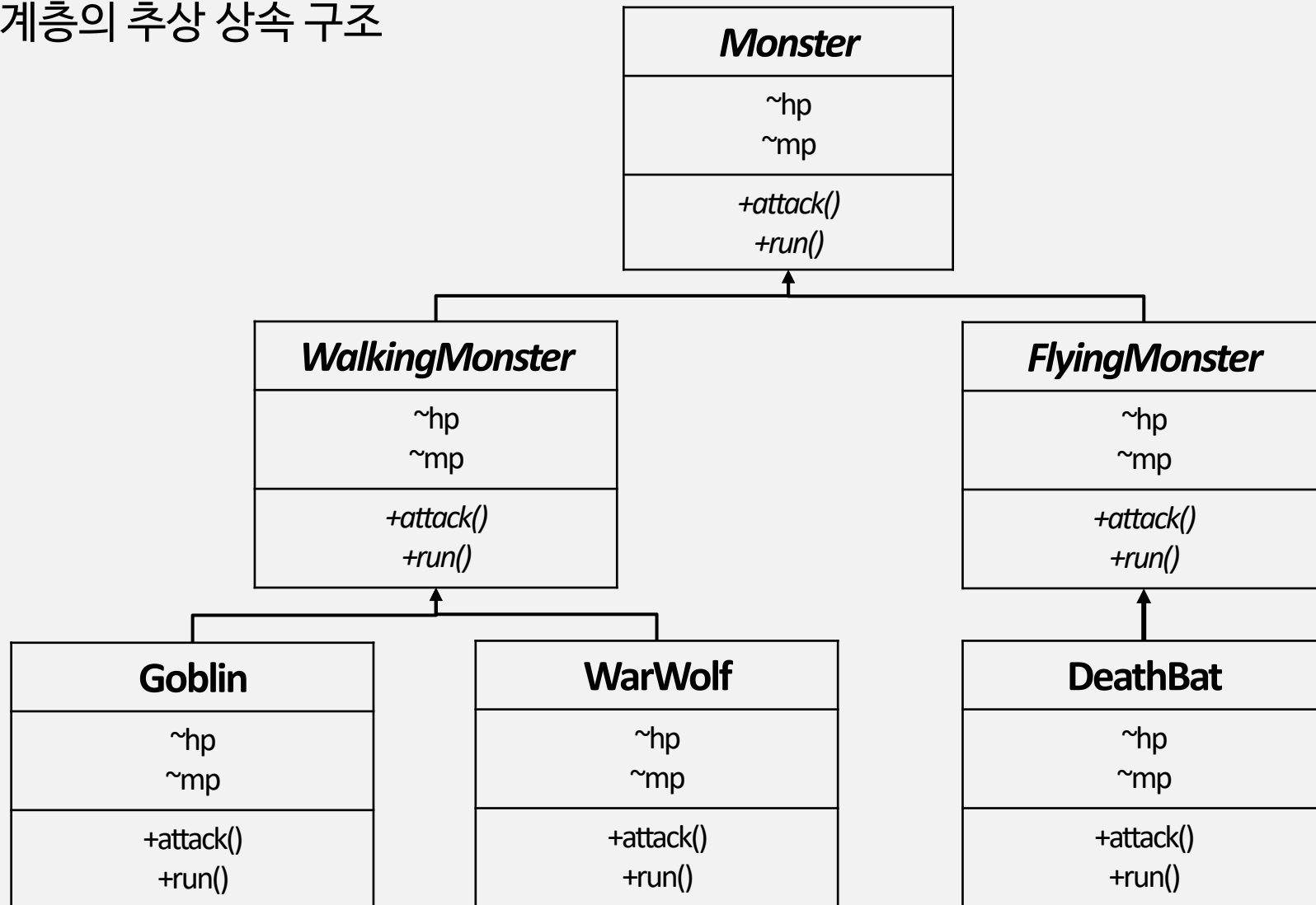
1. 인터페이스의 역할도 하면서 뭔가 구현체도 가지고 있는 자바의 돌연변이 같은 클래스. 혹자는 추상클래스는 인터페이스로 대체하는 것이 좋은 디자인이라고도 얘기함. [\[1\]](#)
2. 사용하는 목적은 추상 메서드(abstract method)가 포함된 클래스를 상속받는 자식 클래스가 반드시 추상 메서드를 구현하도록 하기 위함. 모듈처럼 중복되는 부분이나 공통적인 부분은 미리 다 만들어진 것을 사용하고, 이를 받아 사용하는 쪽에서는 자신에게 필요한 부분만을 재정의하여 사용함으로써 생산성이 향상되고 배포 등이 쉬워 지기 때문 [\[2\]](#)
3. 추상 메서드를 가지기 위해선 반드시 추상 클래스(abstract class)여야 한다.
 - 추상 메서드: 내용이 없는 메서드. 추상 메서드는 선언부만이 존재하고, 구현부는 작성되지 않아 중괄호({})도 붙이지 않음 [\[2\]](#)
 - 추상메서드 선언 문법 : abstract 반환타입 메서드명; (중괄호 없이 바로 세미콜론 붙여 줌) [\[2\]](#)
4. 추상 클래스는 new로 인스턴스화 금지. 또한 추상 클래스를 상속받으면 추상 메서드의 override 강제됨
5. 추상 클래스는 인터페이스화 할 수 없다.
6. 추상 클래스와 추상 메서드를 활용한 상속의 재료로서의 부모 클래스를 개발하면, 예기치 않은 인스턴스화나 Override의 미 구현 가능성이 없다. (human error 방지)
7. 추상클래스를 상속받는 추상클래스는 Override 강제가 없음

[1] <https://wikidocs.net/219>, 05-8추상클래스

[2] http://tcpschool.com/java/java_polymorphism_abstract

2. 수업 내용 정리 [추상클래스와 인터페이스]

5. 추상클래스를 상속받는 추상클래스는 Override가 강제되지 않는다.
6. 다계층의 추상 상속 구조



2. 수업 내용 정리 [추상클래스와 인터페이스]

인터페이스(Interface)

1. 다른 클래스를 작성할 때 기본이 되는 틀을 제공하면서, 다른 클래스 사이의 중간 매개 역할까지 담당하는 일종의 추상 클래스. 클래스나 추상 클래스의 다중 상속이 Java에서는 금지되어 있기 때문에 인터페이스가 다중상속의 역할을 담당 (복수의 인터페이스를 Super로 두는 방식) [\[3\]](#)
 1. 미구현 메소드는 인터페이스로 만드는 것이 좋다.
 2. 추상 클래스와 인터페이스를 둘 다 사용하는 객체는 Override도 추상 클래스와 인터페이스 모두 강제된다.
2. Java에서 추상 클래스는 추상 메서드뿐만 아니라 생성자, 필드, 일반 메서드도 포함 가능하지만, 인터페이스는 오직 추상 메서드와 상수만을 가질 수 있다. [\[3\]](#)
3. 인터페이스 문법 [\[3\]](#)
 - 접근제어자 interface 인터페이스 이름 {
 public static final 타입 상수이름 = 값;

 public abstract 메서드이름(Parameter);
...}
 - 인터페이스의 모든 필드는 public static final 이어야 하며, 모든 메서드는 public abstract 이어야 한다.
 - 이 부분은 모든 인터페이스에 공통적 적용되는 부분이므로 생략 가능
4. 인터페이스의 구현 [\[3\]](#)
 - class 클래스이름 implements 인터페이스 이름 {...}

2. 수업 내용 정리 [추상클래스와 인터페이스]

인터페이스(Interface)

5. 인터페이스의 규칙

- 하나의 클래스가 여러 개의 인터페이스를 구현 가능 [\[4\]](#)
- 인터페이스도 상속 가능 [\[4\]](#)
- 객체가 특정 인터페이스 사용 시, 해당 객체는 반드시 인터페이스의 메서드들을 구현해야 한다 (Override 강제).

6. 인터페이스의 장점 [\[3\]](#)

- 대규모 프로젝트 개발 시 일관되고 정형화된 개발을 위한 표준화 가능
- 클래스의 작성과 인터페이스의 구현을 동시에 진행할 수 있으므로, 개발 시간 단축 가능
- 클래스와 클래스 간의 관계를 인터페이스로 연결 시, 클래스마다 독립적인 프로그래밍 가능

7. 인터페이스를 사용해 구현한 인스턴스는 클래스의 객체이기도 하지만 인터페이스의 객체이기도 하다. (다형성, Polymorphism : 객체가 1개 이상의 자료형 타입을 갖게 되는 특성) [\[5\]](#)

Super	Sub	Keyword	다중 상속
class	class	extends	X
interface	class	implements	O
interface	interface	extends	O

[4] <https://opentutorials.org/module/2495/14142>

[5] <https://wikidocs.net/217>, 05-6. 인터페이스]

2. 수업 내용 정리 [추상클래스와 인터페이스]

추상 클래스와 인터페이스

- 추상 클래스
 - 1개 이상의 미 구현 메서드를 가지고 있다.
 - 상속은 한 개만 (다중 상속 불가)
 - field 존재.
- 인터페이스
 - 모든 메소드가 미구현
 - 다중 상속 가능
 - field 없음



3. 실습화면 캡처

3. 실습 화면 캡처

12-1. 상세정의가 미정인 메서드를 가진 상속의 재료로 사용될 클래스 Character

```
public class Character {  
    String name;  
    int hp;  
    // 도망  
    public void run() {  
        System.out.println(name + "은 도망쳤다");  
    }  
    // 싸우기  
    public void attack(Kinoko kinoko) {  
        System.out.println(name + "의 공격!");  
        kinoko.hp -= ??;  
        System.out.println("적에게 ??포인트의 데미지를 주었다!");  
    }  
}
```

12-2. 대책 1: attack() 메서드의 내부를 공백으로 두기

```
public class Character {  
    String name;  
    int hp;  
    public void run() {  
        System.out.println(name + "은 도망쳤다");  
    }  
    // 대책 1 : 메소드 안을 비워 둔다  
    public void attack(Kinoko kinoko) {  
    }  
}  
  
public class Hero extends Character{  
    // 미래의 개발자가 작성할 코드  
    @Override  
    public void attack(Kinoko kinoko) {  
        System.out.println(this.name + "의 공격");  
        System.out.println("적에게 10포인트의 데미지를 주었다!");  
        kinoko.hp -= 10;  
    }  
}
```

3. 실습 화면 캡처

12-4. 미래의 걱정

: 오버라이드 안하는 것에 대한 걱정

```
public class Hero extends Character{  
    // attack() 을 오버라이드 해야 하지만 하지 않았다.  
}
```

```
public class GameMain {  
    public static void main(String[] args) {  
        Hero hero = new Hero();  
        Kinoko kinoko = new Kinoko( suffix: 'A');  
        hero.attack(kinoko);  
    }  
}
```

GameMain ×

"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-ja

Process finished with exit code 0

12-5. attack의 철자가 틀렸다!!

```
public class Hero extends Character{  
    public void atack(Kinoko kinoko) {  
        System.out.println(this.name + "의 공격!");  
        System.out.println("적에게 10포인트의 데미지를 주었다!");  
    }  
}
```

12-6. 의도치 않게 new 로 생성되어 버림

```
public class GameMain {  
    public static void main(String[] args) {  
        // Hero나 Wizard가 아닌 Character를 new 해 버렸다  
        Character c = new Character();  
        Kinoko kinoko = new Kinoko( suffix: 'A');  
        // 오버라이드 하지 않았는데 아무 동작도 안함  
        c.attack(kinoko);  
    }  
}
```

3. 실습 화면 캡처

12-7. 추상메서드(abstract method)

```
public class Character {  
    String name;  
    int hp;  
    public void run() {  
        System.out.println(name + "은 도망쳤다");  
    }  
  
    public abstract void attack(Kinoko kinoko);  
}
```

```
public abstract class Character {  
    String name;  
    int hp;  
    public void run() {  
        System.out.println(name + "은 도망쳤다");  
    }  
  
    public abstract void attack(Kinoko kinoko);  
}
```

12-8.

추상 메서드를
가지려면 반드시
추상 클래스여야 한다

12-9. 오버라이드 강제

```
public class Dancer extends Character {  
    public void dance() {  
        System.out.println(this.name + "은 정열적으로 춤을 췄다");  
    }  
}
```

12-11. 추상클래스 → 인터페이스

```
public abstract class Creature {  
    public abstract void run();  
}
```

```
public interface Creature {  
    public abstract void run();  
}
```

```
public interface Creature {  
    void run();  
}
```


3. 실습 화면 캡처

12-14. 세탁소

```
public interface CleaningService {  
    Shirt washShirt(Shirt shirt);  
    Towel washTowel(Towel towel);  
    Coat washCoat(Coat coat);  
}
```

12-15. 인터페이스의 구현

```
public class SuwonCleaningService implements CleaningService {  
    private String ownerName; // 주인이름  
    private String address; // 주소  
    private String phone; // 전화번호  
  
    @Override  
    public Shirt washShirt(Shirt shirt){  
        // 대형세탁기 15분  
        // 업무용 건조기 30분  
        // 스팀 다림질 5분  
        return shirt;  
    }  
  
    @Override  
    public Towel washTowel(Towel towel) {  
        return towel;  
    }  
  
    public Coat washCoat(Coat coat) {  
        return coat;  
    }  
}
```

12-16. 인터페이스의 상속

```
public interface Human extends Creature {  
    void talk();  
    void watch();  
    void hear();  
    // 추가로, 부모 인터페이스로부터 run()을 상속받고 있음  
}
```

```
public class Fool extends Character implements Human {  
    // Character로부터 hp나 getName()등의 메소드를 상속받고 있다  
    // Character로부터 상속 받은 추상 메소드 attack()  
    @Override  
    public void attack(Kinoko kinoko) {  
        System.out.println(getName() + "는 싸우지 않고 놀고 있다");  
    }  
    // 여기부터는 Human의 공통 추상 메소드  
    @Override  
    public void run() {...}  
  
    @Override  
    public void talk() {}  
  
    @Override  
    public void watch() {}  
  
    @Override  
    public void hear() {}  
}
```

12-17. extends와
implements 를
동시에 사용



4. 연습문제 12-1

4. 연습문제 12-1

12-1. 문제

- 어떤 회사에서 자산관리 프로그램을 만들려고 한다. 현시점에서 컴퓨터, 책 을 표현하는, 다음과 같은 두 개의 클래스가 있다

```
1  public class Book {
2      private String name;
3      private int price;
4      private String color;
5      private String isbn;
6
7      public Book(String name, int price, String color, String isbn) {
8          this.name = name;
9          this.price = price;
10         this.color = color;
11         this.isbn = isbn;
12     }
13
14     public String getName() { return name; }
17     public int getPrice() { return price; }
20     public String getColor() { return color; }
23     public String getIsbn() { return isbn; }
26 }
```

4. 연습문제 12-1

12-1. 문제

- 이후, 컴퓨터와 책 이외에도 여러가지 자산을 관리하고 싶은 경우에 유용한 유형자산(TangibleAsset)이라는 이름의 추상클래스를 작성 하시오. 또, Computer 나 Book 은 그 부모 클래스를 활용한 형태로 수정 하시오.

```
public class Computer {
    private String name;
    private int price;
    private String color;
    private String makerName;

    public Computer(String name, int price, String color, String makerName) {
        this.name = name;
        this.price = price;
        this.color = color;
        this.makerName = makerName;
    }

    public String getName() { return name; }
    public int getPrice() { return price; }
    public String getColor() { return color; }
    public String getMakerName() { return makerName; }
}
```

4. 연습문제 12-1: TangibleAsset & Book class

TangibleAsset abstract class

```
3 public abstract class TangibleAsset{
4     private String name;
5     private int price;
6     private String color;
7
8     public TangibleAsset(String name, int price, String color) {
9         this.name = name;
10        this.price = price;
11        this.color = color;
12    }
13
14    public String getName() {
15        return name;
16    }
17
18    public int getPrice() {
19        return price;
20    }
21
22    public String getColor() {
23        return color;
24    }
25
26    public abstract void printScript();
27 }
```

Book class

```
3 public class Book extends TangibleAsset {
4     private String isbn;
5
6     public Book(String name, int price, String color, String isbn) {
7         super(name, price, color);
8         this.isbn = isbn;
9     }
10
11    public String getIsbn() {
12        return isbn;
13    }
14
15    @Override
16    public void printScript() {
17        String totalClassName = Book.class.getName();
18        String[] totalClassNameSplit = totalClassName.split(regex: "\\.");
19        String className = totalClassNameSplit[1];
20        System.out.println("클래스 이름: " + className);
21    }
22 }
```

4. 연습문제 12-1: Computer class

Computer class

```
3 public class Computer extends TangibleAsset {
4     private String makerName;
5
6     public Computer(String name, int price, String color, String makerName) {
7         super(name, price, color);
8         this.makerName = makerName;
9     }
10
11     public String getMakerName() {
12         return makerName;
13     }
14
15     @Override
16     public void printScript() {
17         String totalClassName = Computer.class.getName();
18         String[] totalClassNameSplit = totalClassName.split(regex: "\\.");
19         String className = totalClassNameSplit[1];
20         System.out.println("클래스 이름: " + className);
21     }
22 }
```


4. 연습문제 12-1 : TestMain class

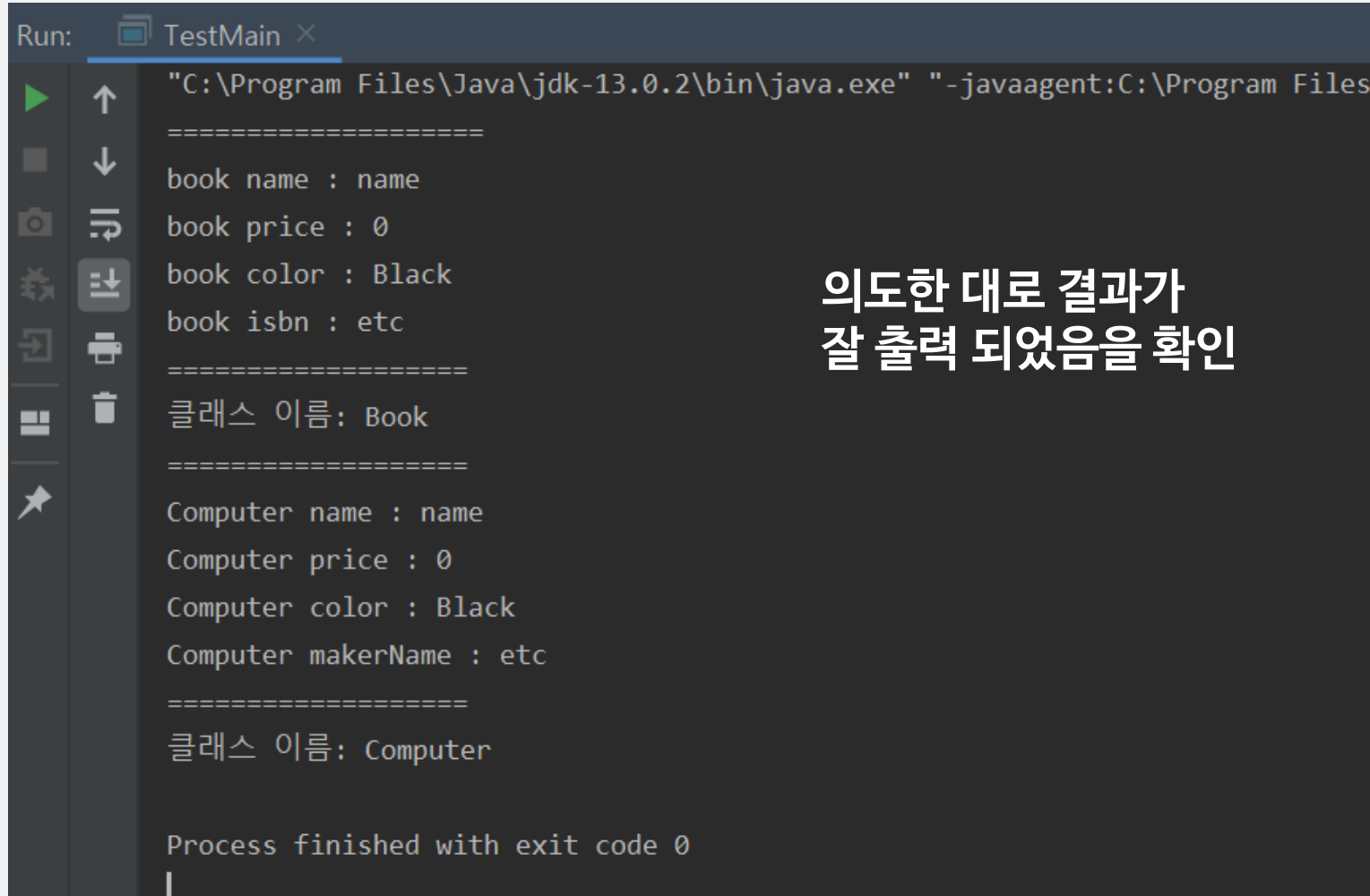
TestMain class

```
public class TestMain {  
    static void printStatus(Book book) {  
        System.out.println("=====");  
        System.out.println("book name : " + book.getName());  
        System.out.println("book price : " + book.getPrice());  
        System.out.println("book color : " + book.getColor());  
        System.out.println("book isbn : " + book.getIsbn());  
        System.out.println("=====");  
    }  
  
    static void printStatus(Computer computer) {  
  
        System.out.println("=====");  
        System.out.println("Computer name : " + computer.getName());  
        System.out.println("Computer price : " + computer.getPrice());  
        System.out.println("Computer color : " + computer.getColor());  
        System.out.println("Computer makerName : " + computer.getMakerName());  
        System.out.println("=====");  
    }  
}
```

main method

```
public static void main(String[] args) {  
    String name = "name";  
    int price = 0;  
    String color = "Black";  
    String etc = "etc";  
  
    Book book = new Book(name, price, color, etc);  
    Computer computer = new Computer(name, price, color, etc);  
  
    printStatus(book);  
    book.printScript();  
    printStatus(computer);  
    computer.printScript();  
}
```

4. 연습문제 12-1 : main method 실행 결과



```
Run: TestMain ×
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files
=====
book name : name
book price : 0
book color : Black
book isbn : etc
=====
클래스 이름: Book
=====
Computer name : name
Computer price : 0
Computer color : Black
Computer makerName : etc
=====
클래스 이름: Computer

Process finished with exit code 0
```

의도한 대로 결과가
잘 출력 되었음을 확인

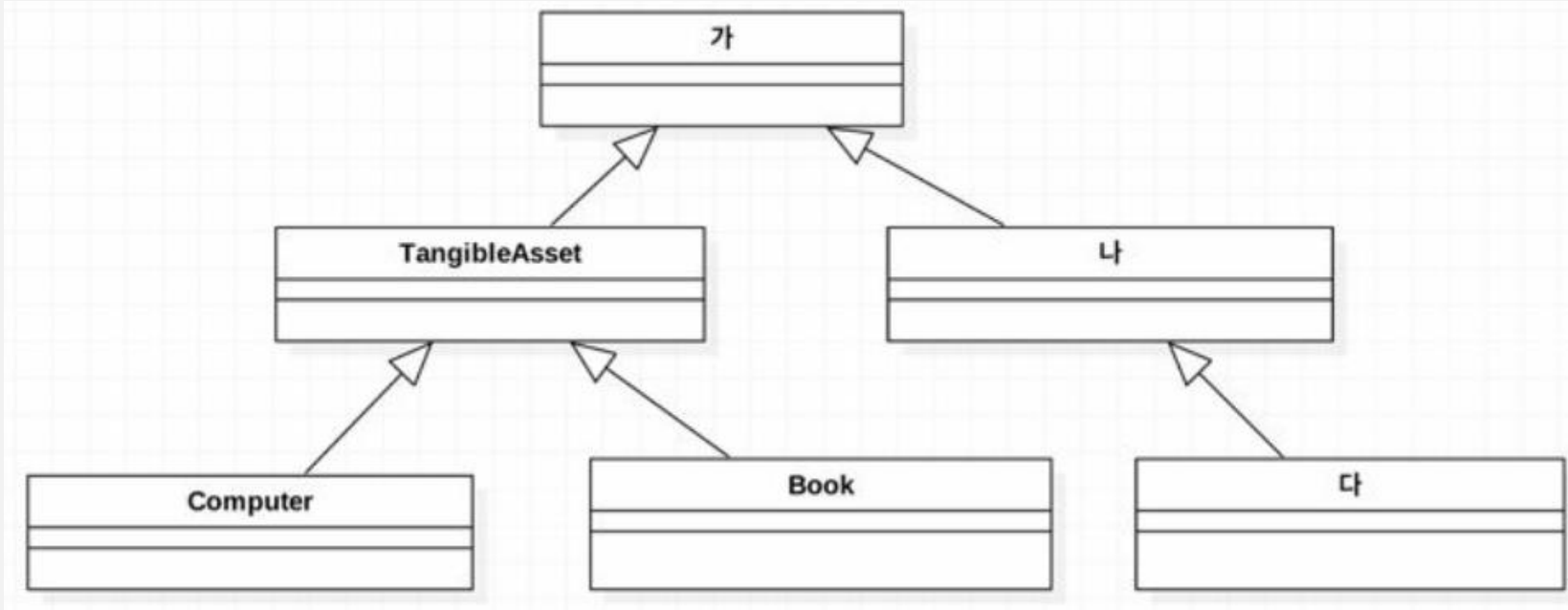


5. 연습문제 12-2

5. 연습문제 12-2

12-2. 문제

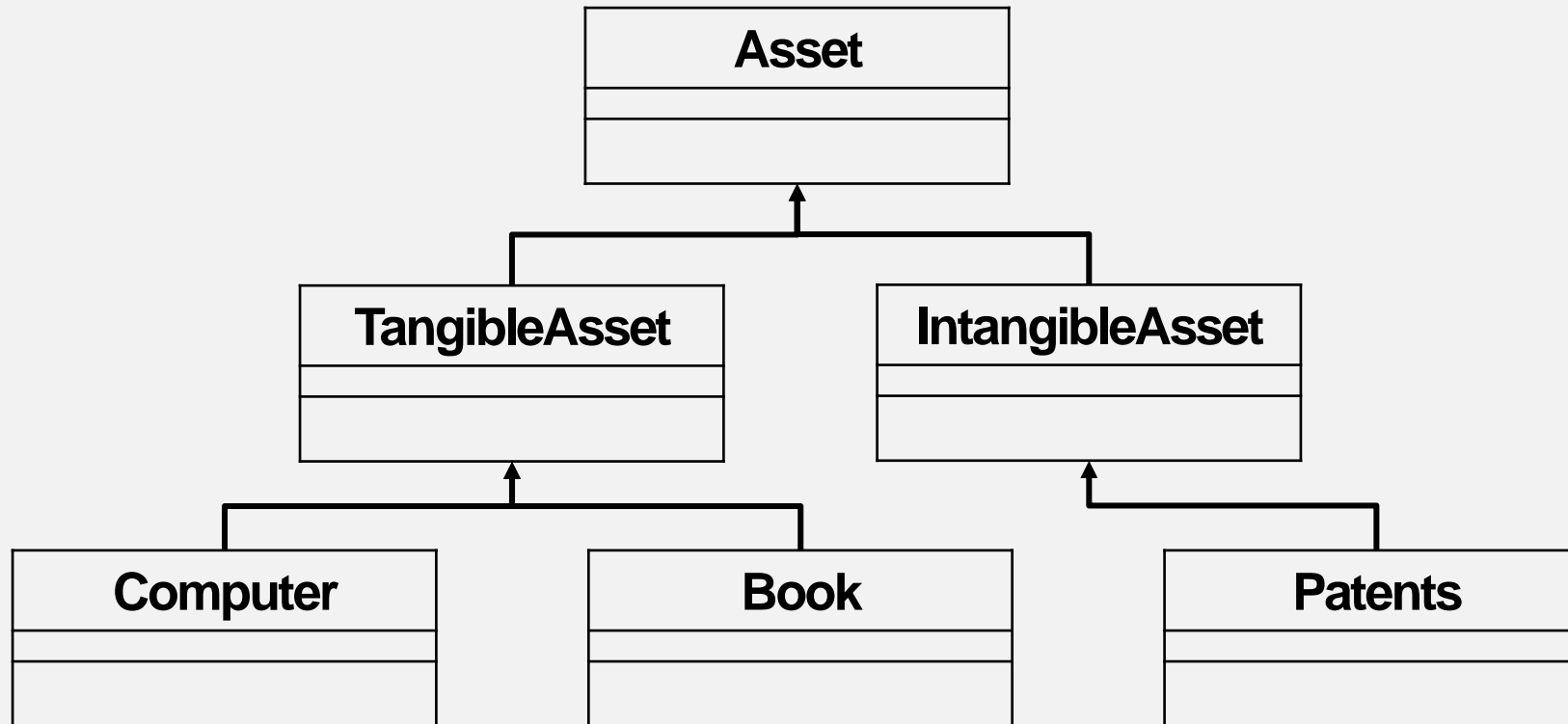
- 문제 12-1 의 회사에서, 형태가 없는 무형자산(IntangibleAsset) 도 관리하려고 생각하고 있다. 무형자산에는 , 예를 들어 특허권(Patent) 등이 있다. 무형자산도 유형자산도 자산(Asset)의 일종이다. 이것을 전제로 다음의 상속도의 가, 나, 다 부분의 클래스명을 생각 해 보시오.



- 또한, (가) 에 들어가는 추상 클래스를 개발하고, 이 클래스를 상속하도록 TangibleAsset 를 수정하시오.

5. 연습문제 12-2: 상속도

상속도



5. 연습문제 12-2: Asset & TangibleAsset Class

Asset abstract class

```
3 public abstract class Asset {  
4     private String name;  
5     private int price;  
6  
7     public Asset(String name, int Price) {  
8         this.name = name;  
9         this.price = price;  
10    }  
11  
12    public String getName() {  
13        return name;  
14    }  
15  
16    public int getPrice() {  
17        return price;  
18    }  
19    public abstract void printScript();  
20 }
```

- 무형 자산의 경우, 형태가 없으므로 Color field가 존재할 수 없다.
- ∴ 무형 자산과 유형 자산의 공통분모인 name, price field를 추출해 Asset의 필드로 선언해 줌
- 추상 메서드인 printScript는 Asset Class에 선언.

5. 연습문제 12-2: Asset & revised TangibleAsset Class

Tangible abstract class before extended Asset

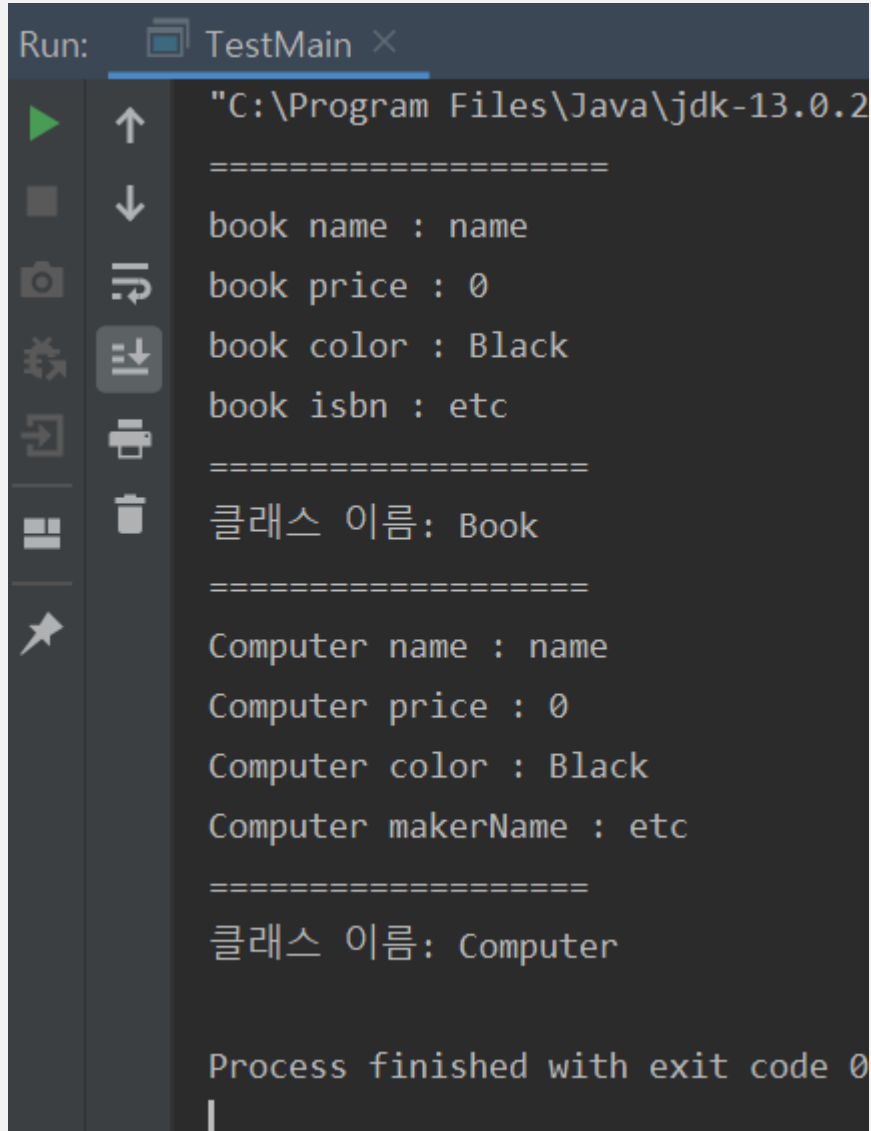
```
3 public abstract class TangibleAsset{
4     private String name;
5     private int price;
6     private String color;
7
8     public TangibleAsset(String name, int price, String color) {
9         this.name = name;
10        this.price = price;
11        this.color = color;
12    }
13
14    public String getName() {
15        return name;
16    }
17
18    public int getPrice() {
19        return price;
20    }
21
22    public String getColor() {
23        return color;
24    }
25
26    public abstract void printScript();
27 }
```

revised TangibleAsset abstract Class after

```
3 public abstract class TangibleAsset extends Asset{
4     private String color;
5
6     public TangibleAsset(String name, int price, String color) {
7         super(name, price);
8         this.color = color;
9     }
10
11    public String getColor() {
12        return color;
13    }
14 }
```

- 추상 클래스를 상속받는 추상 클래스의 경우 추상 메서드 오버라이드를 강제 받지 않음
- ∴ 기존의 추상 메서드 printScript는 코드에서 삭제

4. 연습문제 12-2 : main method 실행 결과



```
Run: TestMain x
"C:\Program Files\Java\jdk-13.0.2
=====
book name : name
book price : 0
book color : Black
book isbn : etc
=====
클래스 이름: Book
=====
Computer name : name
Computer price : 0
Computer color : Black
Computer makerName : etc
=====
클래스 이름: Computer

Process finished with exit code 0
```

- Assat과 TangibleAssat class 변경사항 적용
- Book, Computer, TestMain class와 main method 는 변경사항 없음
- 출력 결과 동일함을 확인



6. 연습문제 12-3

6. 연습문제 12-3

12-3. 문제

- 자산인지 아닌지 따지지 말고, 형태가 있는 것 (Thing) 이면, 무게가 있다. 그래서, double 형으로 무게를 얻을 수 있는 getter 메서드 `getWeight()` 와 setter 메서드 `setWeight(무게)` 를 가지는 인터페이스 Thing 을 만드시오

- 답:

```
public interface Thing {  
    double getWeight();  
    void setWeight(double weight);  
}
```



7. 연습문제 12-4

7. 연습문제 12-4

12-4. 문제

- 유형자산 (TangibleAsset)은, 자산(Asset)의 일종이며, 형태가 있는 것(Thing)의 일종이기도 하다.
- 이 정의에 맞도록 TangibleAsset의 소스코드를 수정하시오
- 이 때, TangibleAsset에 필드나 메소드의 추가가 필요하다면, 적당히 추가하시오.

7. 연습문제 12-4 : Assat class & Thing interface

- Assat class는 코드 변화가 없고, Thing은 연습문제 12-3에서 상기 정의한 인터페이스를 사용

```
3 public abstract class Asset {  
4     private String name;  
5     private int price;  
6  
7     public Asset(String name, int Price) {  
8         this.name = name;  
9         this.price = price;  
10    }  
11  
12    public String getName() {  
13        return name;  
14    }  
15  
16    public int getPrice() {  
17        return price;  
18    }  
19    public abstract void printScript();  
20 }
```

```
3 public interface Thing {  
4     double getWeight();  
5     void setWeight(double weight);  
}
```

- implement 되고 있음이 표시

7. 연습문제 12-4 : revised TangibleAssat Class

- Thing 인터페이스를 implements하게끔 수정한 TangibleAssat class

```
3 public abstract class TangibleAsset extends Asset{
4     private String color;
5
6     public TangibleAsset(String name, int price, String color) {
7         super(name, price);
8         this.color = color;
9     }
10
11     public String getColor() {
12         return color;
13     }
14 }
```

Thing 인터페이스 implements
이전의 TangibleAsset 클래스

```
3 public abstract class TangibleAsset extends Asset implements Thing {
4     private String color;
5     private double weight;
6
7     public TangibleAsset(String name, int price, String color, double weight) {
8         super(name, price);
9         this.color = color;
10        this.weight = weight;
11    }
12
13    public String getColor() {
14        return color;
15    }
16
17    @Override
18    public double getWeight() {
19        return weight;
20    }
21
22    @Override
23    public void setWeight(double weight) {
24        this.weight = weight;
25    }
26 }
```

Thing 인터페이스의 추상 메서드를 override
하기 위해 double weight 필드 선언

생성자의 parameter에
weight 필드 추가

Thing 인터페이스에서 override한
getWeight 메서드와 setWeight 메서드

Thing 인터페이스
implements 한
TangibleAsset 클래스

7. 연습문제 12-4 : revised Book Class

- TangibleAssat 클래스의 생성자가 수정됨에 따라 Book 클래스의 생성자도 수정

Thing 인터페이스 implements
이전의 Book 클래스

```
public class Book extends TangibleAsset {  
    private String isbn;  
  
    public Book(String name, int price, String color, String isbn) {  
        super(name, price, color);  
        this.isbn = isbn;  
    }  
  
    public String getIsbn() {  
        return isbn;  
    }  
  
    @Override  
    public void printScript() {  
        String totalClassName = Book.class.getName();  
        String[] totalClassNameSplit = totalClassName.split(regex: "\\.");  
        String className = totalClassNameSplit[1];  
        System.out.println("클래스 이름: " + className);  
    }  
}
```

Thing 인터페이스 implements
이후의 Book 클래스

```
public class Book extends TangibleAsset {  
    private String isbn;  
  
    public Book(String name, int price, String color, String isbn, double weight) {  
        super(name, price, color, weight);  
        this.isbn = isbn;  
    }  
  
    public String getIsbn() {  
        return isbn;  
    }  
  
    @Override  
    public void printScript() {  
        String totalClassName = Book.class.getName();  
        String[] totalClassNameSplit = totalClassName.split(regex: "\\.");  
        String className = totalClassNameSplit[1];  
        System.out.println("클래스 이름: " + className);  
    }  
}
```

생성자의 parameter에
weight 필드 추가

7. 연습문제 12-4 : revised Computer Class

- TangibleAssat 클래스의 생성자가 수정됨에 따라 Computer 클래스의 생성자도 수정

Thing 인터페이스 implements
이전의 Computer 클래스

```
public class Computer extends TangibleAsset {  
    private String makerName;  
  
    public Computer(String name, int price, String color, String makerName) {  
        super(name, price, color);  
        this.makerName = makerName;  
    }  
  
    public String getMakerName() {  
        return makerName;  
    }  
  
    @Override  
    public void printScript() {  
        String totalClassName = Computer.class.getName();  
        String[] totalClassNameSplit = totalClassName.split(regex: "\\.");  
        String className = totalClassNameSplit[1];  
        System.out.println("클래스 이름: " + className);  
    }  
}
```

Thing 인터페이스 implements
이후의 Computer 클래스

```
public class Computer extends TangibleAsset {  
    private String makerName;  
  
    public Computer(String name, int price, String color, String makerName, double weight) {  
        super(name, price, color, weight);  
        this.makerName = makerName;  
    }  
  
    public String getMakerName() {  
        return makerName;  
    }  
  
    @Override  
    public void printScript() {  
        String totalClassName = Computer.class.getName();  
        String[] totalClassNameSplit = totalClassName.split(regex: "\\.");  
        String className = totalClassNameSplit[1];  
        System.out.println("클래스 이름: " + className);  
    }  
}
```

생성자의 parameter에
weight 필드 추가

7. 연습문제 12-4 : TestMain class 및 main method 수정

revised printStatus method

```
public class TestMain {  
    static void printStatus(Book book) {  
        System.out.println("=====");  
        System.out.println("book name : " + book.getName());  
        System.out.println("book price : " + book.getPrice());  
        System.out.println("book color : " + book.getColor());  
        System.out.println("book isbn : " + book.getIsbn());  
        System.out.println("book weight : " + book.getWeight() + " kg");  
        System.out.println("=====");  
    }  
  
    static void printStatus(Computer computer) {  
  
        System.out.println("=====");  
        System.out.println("Computer name : " + computer.getName());  
        System.out.println("Computer price : " + computer.getPrice());  
        System.out.println("Computer color : " + computer.getColor());  
        System.out.println("Computer makerName : " + computer.getMakerName());  
        System.out.println("Computer weight : " + computer.getWeight() + " kg");  
        System.out.println("=====");  
    }  
}
```

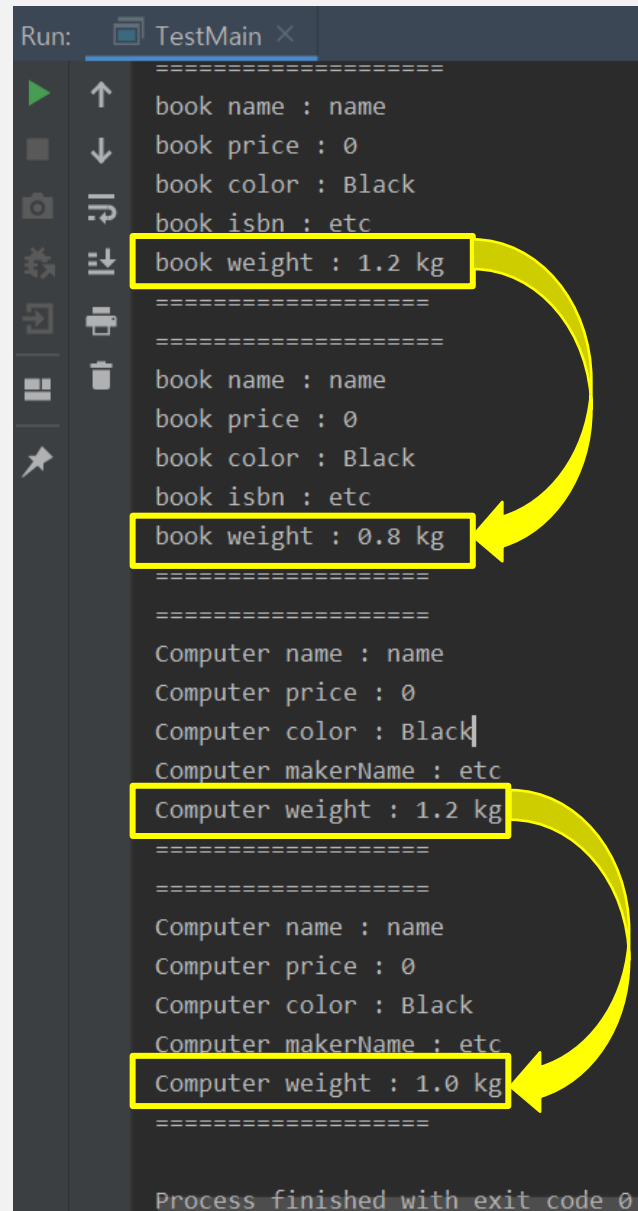
revised main method

```
public static void main(String[] args) {  
    String name = "name";  
    int price = 0;  
    String color = "Black";  
    String etc = "etc";  
    double weight = 1.2;  
  
    Book book = new Book(name, price, color, etc, weight);  
    Computer computer = new Computer(name, price, color, etc, weight);  
  
    printStatus(book);  
    book.setWeight(0.8);  
    printStatus(book);  
    printStatus(computer);  
    computer.setWeight(1.0);  
    printStatus(computer);  
}
```

book의 weight
1.2 → 0.8 로 변경

computer의 weight
1.2 → 1.0 로 변경

7. 연습문제 12-4 : main 동작 결과



```
Run: TestMain x
=====
book name : name
book price : 0
book color : Black
book isbn : etc
book weight : 1.2 kg
=====
book name : name
book price : 0
book color : Black
book isbn : etc
book weight : 0.8 kg
=====
Computer name : name
Computer price : 0
Computer color : Black
Computer makerName : etc
Computer weight : 1.2 kg
=====
Computer name : name
Computer price : 0
Computer color : Black
Computer makerName : etc
Computer weight : 1.0 kg
=====
Process finished with exit code 0
```

The screenshot shows the output of a program named 'TestMain'. The output is divided into four sections by separator lines. The first two sections are for 'book' objects, and the last two are for 'Computer' objects. In each section, the 'weight' attribute is highlighted with a yellow box. A yellow arrow points from the 'book weight : 1.2 kg' box to the 'book weight : 0.8 kg' box. Another yellow arrow points from the 'Computer weight : 1.2 kg' box to the 'Computer weight : 1.0 kg' box. The program ends with 'Process finished with exit code 0'.

의도한 대로 결과가
출력된 것을 확인함



8. 과제 리뷰 반영

1. 과제 리뷰 및 오답 정리

Assat class 수정

```
3 public abstract class Asset {
4     private String name;
5     private int price;
6
7     public Asset(String name, int Price) {
8         this.name = name;
9         this.price = price;
10    }
11
12    public String getName() {
13        return name;
14    }
15
16    public int getPrice() {
17        return price;
18    }
19    public abstract void printScript();
20 }
```

오타 수정

```
3 public abstract class Asset {
4     private String name;
5     private int price;
6
7     public Asset(String name, int price) {
8         this.name = name;
9         this.price = price;
10    }
11
12    public String getName() { return name; }
13
14    public int getPrice() { return price; }
15
16    public void setName(String name) { this.name = name; }
17
18    public void setPrice() { this.price = price; }
19
20    public void setPrice(int price) { this.price = price; }
21
22    public abstract void printScript();
23 }
```

getter와 setter는 선언해두는게 정석!

setter 추가!!