제12장 추상클래스와 인터페이스 (abstract class, interface)

리스트 12-1 상세정의가 미정인 메소드를 가진 상속의 재료로 사용 될 클래스 Character


```java
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!");
    }
}
```

리스트 12-2 대책 1: attack() 메소드의 내부를 공백으로 두기


```java
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) {
    }
}
```

```java
public class Hero extends Character {
    // 미래의 개발자가 작성 할 코드
    @Override
    public void attack(Kinoko kinoko) {
        System.out.println(this.name + "의 공격");
        System.out.println("적에게 10포인트의 데미지를 주었다!");
        kinoko.hp -= 10;
    }
}
```

리스트 12-4 미래의 걱정 : 오버라이드 안 하는 것에 대한 걱정


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

```java
public class Main {
    public static void main(String[] args) {
        Hero hero = new Hero();
        Kinoko kinoko = new Kinoko();
        hero.attack(kinoko);    // 메소드는 호출 되지만..
    }
}
```

미래의 개발자에게 코멘트를 남긴다


```
/*
미래의 개발자님께
저는 Character 클래스 개발자인 오준석입니다.
이 클래스를 개발하는 시점에는 앞으로 이 클래스를 상속하여
작성 될 여러가지 직업 클래스가 몇 포인트의 데미지를 줄 것인가를
결정 하지 못했기 때문에, 아래의 메소드는 내용을 비워 두었습니다.
Character클래스를 상속하여 여러 직업 클래스를 작성할 때에는
attack() 의 내용을 꼭 오버라이드해서 사용해 주세요.
 */
public void attack(Kinoko kinoko) {
}
```

리스트 12-5 attack 의 철자가 틀렸다!!!


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

리스트 12-6 의도치 않게 new 로 생성되어 버림


```
1  public class Main {
2      public static void main(String[] args) {
3          // Hero 나 Wizard 가 아닌 Character를 new 해 버렸다
4          Character c = new Character();
5          Kinoko kinoko = new Kinoko('A');
6          // 오버라이드 하지 않았는데 아무 동작도 안 함
7          c.attack(kinoko);
8      }
9  }
```

new 되면 안되는 클래스


```
1
2   /*
3   미래의 개발자님께
4   저는 Character 클래스 개발자인 오준석입니다
5
6   이 클래스는 보통의 클래스 처럼 new 로 사용하는 클래스가 아닙니다
7   Hero 나 Wizard 등의 직업 클래스를 여러 사람이 만들 때에
8   조금이라도 편하게 하기 위해서, 모든 직업 클래스에 제공할 필드나
9   메소드를 모은 '상속의 재료' 입니다
10
11  이 클래스를 상속하여 필요한 필드나 메소드를 추가하는 것으로
12  여러가지 직업 클래스를 완성 시켜 주세요
13  역으로 말하면 이 Character 클래스는 그 자체로는 미완성인
14  클래스입니다
15  예를 들어 attack() 메소드는 내용이 확정되지 않아 공백입니다.
16  따라서, 이 클래스를 new 하여 실제로 이용하지 말아 주세요
17  버그가 발생하는 원인이 됩니다
18   */
19  public class Character {
```

추상 클래스

상속의 재료로 사용 되는 클래스

상세 부분이 미정의 된 클래스

리스트 12-7 추상 메소드 (abstract method)


```
1  public class Character {
2      String name;
3      int hp;
4      public void run() {
5          System.out.println(name + "은 도망쳤다");
6      }
7
8      public abstract void attack(Kinoko kinoko);
9  }
```

리스트 12-8 추상 메소드를 가지려면 반드시 추상 클래스 여야 한다


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

    public abstract void attack(Kinoko kinoko);
}
```

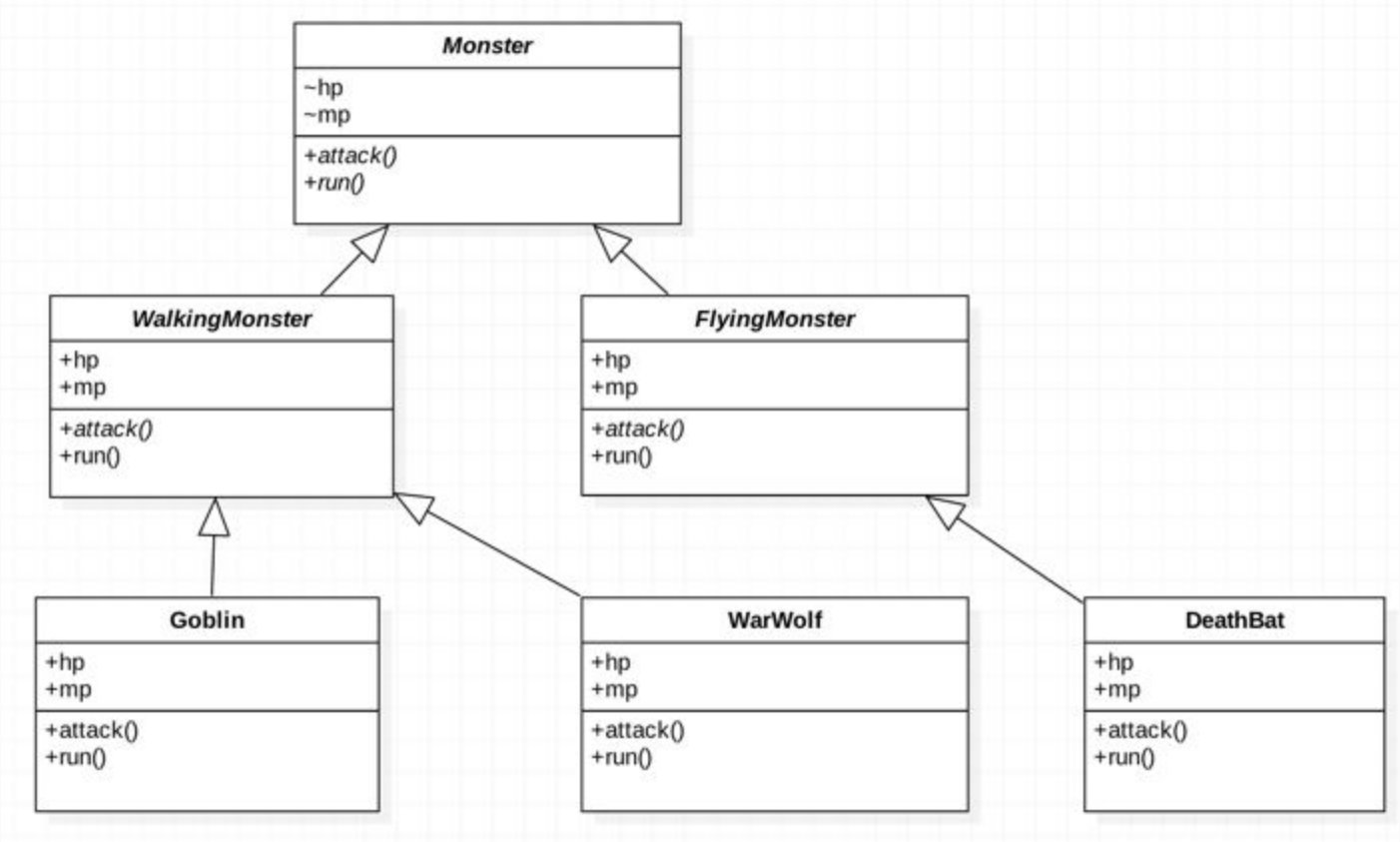
추상클래스의 제약 : 추상클래스는 new 에 의한 인스턴스화가 금지되어 있다


```
Character c = new Character();
```

리스트 12-9 오버라이드 강제


```
1  public class Dancer extends Character {
2
3      public void dance() {
4          System.out.println(this.name + "은 정열적으로 춤을 췄다");
5      }
6  }
```

다계층의 추상 상속 구조


Monster
~hp
~mp
+attack()
+run()
WalkingMonster
+hp
+mp
+attack()
+run()
FlyingMonster
+hp
+mp
+attack()
+run()
Goblin
+hp
+mp
+attack()
+run()
WarWolf
+hp
+mp
+attack()
+run()
DeathBat
+hp
+mp
+attack()
+run()

인터페이스

1. 모든 메소드는 추상 메소드 여야 한다
2. 필드를 가지지 않는다

리스트 12-11 추상클래스 -> 인터페이스


```
1  public abstract class Creature {
2      public abstract void run();
3  }
```

```
1  public interface Creature {
2      public abstract void run();
3  }
```

```
1  public interface Creature {
2      void run();
3  }
```

인터페이스에서의 상수 선언


```
1  public interface Creature {
2      // public static final 이 생략
3      double PI = 3.14;
4
5      void run();
6  }
```

```
1  public interface CleaningService {
2      Shirt washShirt(Shirt shirt);
3      Towel washTowel(Towel towel);
4      Coat washCoat(Coat coat);
5  }
```

리스트 12-15 인터페이스의 구현


```java
public class SuwonCleaningService implements CleaningService {
    private String ownerName;   // 주인 이름
    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;
    }

    @Override
    public Coat washCoat(Coat coat) {
        return coat;
    }
}
```

인터페이스의 효과

1. 같은 인터페이스를 구현한 클래스들은 공통 메소드를 구현하도록 강제된다
2. 어떤 클래스가 인터페이스를 구현하고 있다면, 적어도 그 인터페이스에 정의된 메소드를 가지고 있다는 것이 보증된다

인터페이스의 특별 취급

- 다중상속의 효과를 낼 수 있음

리스트 12-16 인터페이스의 상속


```
1  public interface Human extends Creature {
2      void talk();
3      void watch();
4      void hear();
5
6      // 추가로, 부모 인터페이스로부터 run() 을 상속 받고 있음
7  }
```

implements 와 extends 의 사용방법 정리

부모	자식	keyward	다중 상속
class	class	extends	X
interface	class	implements	O
interface	interface	extends	O

리스트 12-17 extends와 implements 를 동시에 사용


```
1   public class Fool extends Character implements Human {
2       // Character로부터 hp나 getName()등의 메소드를 상속받고 있다
3       // Character로부터 상속 받은 추상 메소드 attack()
4       @Override
5       public void attack(Kinoko kinoko) {
6           System.out.println(getName() + "는 싸우지 않고 놀고 있다");
7       }
8
9       // 여기부터는 Human 의 공통 추상 메소드
10      @Override
11      public void talk() {...}
14
15      @Override
16      public void watch() {...}
19
20      @Override
21      public void hear() {...}
24
25      @Override
26      public void run() {...}
29  }
```

제 12장 정리

상속의 재료를 작성하는 개발자의 입장과 역할

- **다른 사람이 상속의 재료로 사용할 부모 클래스를 만드는 입장**의 개발자도 존재 한다
- **미래의 개발자가 효율 좋게 안심하고 이용할 수 있는 상속의 재료를 작성하는 것**이 그의 사명
- 그의 사명을 달성하기 위해, Java 에서는 추상 클래스나 인터페이스라는 도구를 제공

추상 클래스

- 내용이 정의되지 않고 **상세정의 미정인 메소드**에는 abstract 를 붙여서 추상메소드로 한다
- 추상메소드를 1개라도 포함한 클래스는 abstract 를 붙여 추상 클래스가 된다
- 추상 클래스는 인터페이스화 할 수 없다
- 추상 클래스와 추상 메소드를 활용한 **상속의 재료**로서의 부모클래스를 개발하면, 예기치 않은 인스턴스화나 오버라이드의 미 구현의 걱정이 없다

인터페이스

- 추상 클래스 중에, 기본적으로 추상메소드만 가지고 있는 것을 **인터페이스** 로서 특별 취급 한다
- 인터페이스에 선언된 메소드는 자동적으로 public abstract 가 되고, 필드는 public static final 이 된다
- 복수의 인터페이스를 부모로 두는 다중상속이 가능
- 인터페이스를 부모로 두는 자식 클래스 정의에 implements 를 사용

연습문제 12-1

어떤 회사에서 자산관리 프로그램을 만들려고 한다. 현시점에서 **컴퓨터**, **책** 을 표현하는, 다음과 같은 두개의

```java
public class Book {
    private String name;
    private int price;
    private String color;
    private String isbn;

    public Book(String name, int price, String color, String isbn) {
        this.name = name;
        this.price = price;
        this.color = color;
        this.isbn = isbn;
    }

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

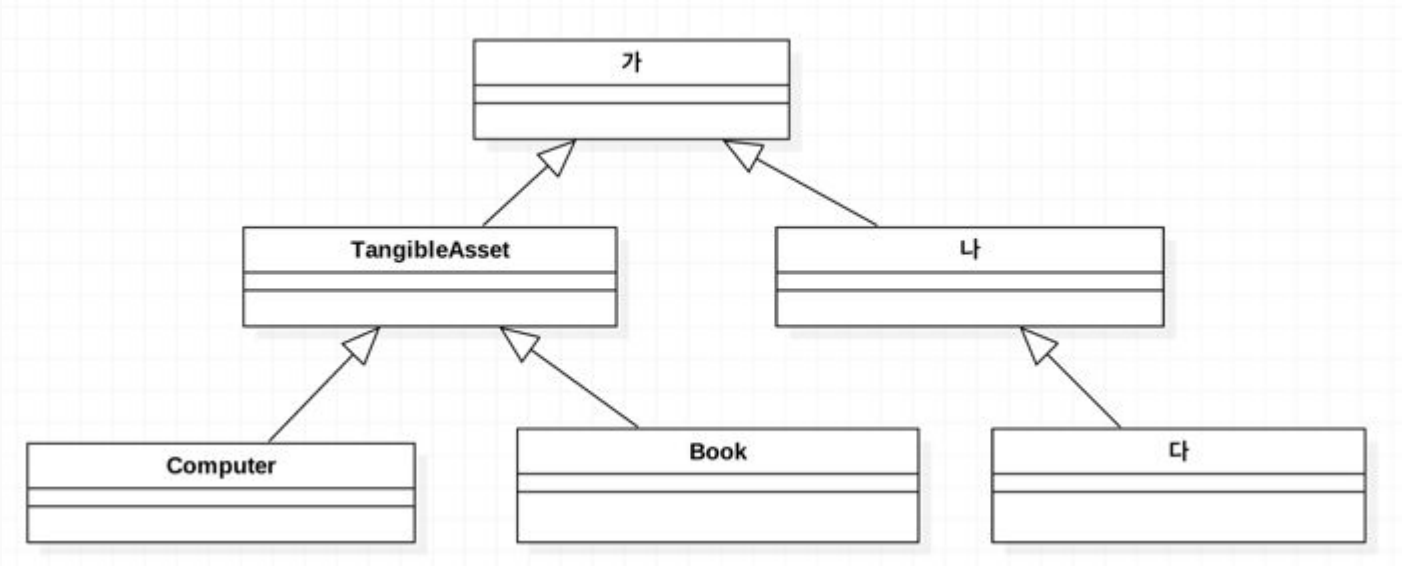
연습문제 12-1 (계속)


```java
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; }
}
```

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.

연습문제 12-2

문제 12-1 의 회사에서, 형태가 없는 무형자산(IntangibleAsset) 도 관리하려고 생각하고 있다.
무형자산에는, 예를들어 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.

연습문제 12-3

자산인지 아닌지 따지지 말고, 형태가 있는 것 (Thing) 이면, **무게**가 있다

그래서, double 형으로 무게를 얻을 수 있는 getter 메소드 getWeight() 와 setter 메소드 setWeight(무게) 를 가지는 인터페이스 Thing 을 만드시오

연습문제 12-4

유형자산 (TangibleAsset) 은, 자산 (Asset) 의 일종이며, 형태가 있는 것 (Thing) 의 일종이기도 하다.

이 정의에 맞도록 TangibleAsset 의 소스 코드를 수정하시오.

이 때, TangibleAsset 에 필드나 메소드의 추가가 필요하다면, 적당히 추가하시오.