

제13장 다형성

개발을 즐겁게 하는 다형성

다형성 (polymorphism)

어떤 것을 이렇게도 볼 수 있고, 저렇게도 볼 수 있는 것
대충 **통** 치는 것

예:

핸들이 있고, 오른 페달이 액셀, 왼쪽이 브레이크인 것 = 차, 그랜저, 버스 ...

세부적인 부분 부분은 다르지만, 어쨌든 대충 보면 그냥 차다

대충 통 치는 방법

선언을 대충 상위 개념으로 통 치고 new 는 하위 개념으로 한다.

애매한 선언 = new 상세 정의

```
Character character1 = new Character();  
Character character2 = new SuperHero();  
|
```

통 치기 실패

```
Character character = new SuperHero(); // OK
Sword sword = new Hero(); // 에러
Flower f = new Fish(); // 에러
Phone p = new Coffee(); // 에러
```

```
1 public interface Life {  
2 }
```

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Life life = new Wizard(); // 살아있는 것으로 통  
4     }  
5 }
```

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

```
1 public class Wizard extends Character {  
2     int mp;  
3  
4     @Override  
5     public void attack(Kinoko kinoko) {  
6         System.out.println(this.name + "의 공격!");  
7         System.out.println("적에게 3포인트의 데미지");  
8         kinoko.hp -= 3;  
9     }  
10  
11     public void fireball(Kinoko kinoko) {  
12         System.out.println(this.name + "는 불의 구슬을 맞았다!");  
13         System.out.println("적에게 20포인트의 데미지");  
14         kinoko.hp -= 20;  
15         this.mp -= 5;  
16     }  
17 }
```

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Wizard w = new Wizard();  
4         Kinoko k = new Kinoko();  
5         w.name = "레이나";  
6         w.attack(k);  
7         w.fireball(k);  
8     }  
9 }
```

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Wizard w = new Wizard();  
4         Character c = w;  
5         Kinoko k = new Kinoko();  
6         c.name = "스랄";  
7         c.attack(k);  
8         c.fireball(k); // 에러  
9     }  
10 }
```

attack() 이 가능한 이유?

fireball() 이 불가능한 이유?


```
1 public abstract class Monster {  
2     public void run() {  
3         System.out.println("몬스터는 도망쳤다");  
4     }  
5 }
```

```
1 public class Slime extends Monster {  
2     @Override  
3     public void run() {  
4         System.out.println("슬라임은 슬금슬금 도망쳤다");  
5     }  
6 }
```

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Slime slime = new Slime();  
4         Monster monster = new Slime();  
5         slime.run();  
6         monster.run();  
7     }  
8 }
```

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Character character = new Wizard();  
4         Wizard wizard = (Wizard) character;  
5     }  
6 }
```

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Character character = new Wizard();  
4         Hero hero = (Hero) character;  
5     }  
6 }
```

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Character character = new Wizard();  
4  
5         if (character instanceof Hero) {  
6             Hero hero = (Hero) character;  
7         } else {  
8             System.out.println("형변환 불가");  
9         }  
10    }  
11 }
```

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Hero h1 = new Hero();  
4         Hero h2 = new Hero();  
5         Thief t1 = new Thief();  
6         Wizard w1 = new Wizard();  
7         Wizard w2 = new Wizard();  
8         // 모험개시!  
9         // 우선 여관에 머물기  
10        h1.setHp(h1.getHp() + 50);  
11        h2.setHp(h2.getHp() + 50);  
12        t1.setHp(t1.getHp() + 50);  
13        w1.setHp(w1.getHp() + 50);  
14        w2.setHp(w2.getHp() + 50);  
15    }  
16 }
```

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Character[] characters = new Character[5];  
4         characters[0] = new Hero();  
5         characters[1] = new Hero();  
6         characters[2] = new Thief();  
7         characters[3] = new Wizard();  
8         characters[4] = new Wizard();  
9         // 모험개시!  
10        // 우선 여관에 머물기  
11        for (Character character : characters) {  
12            character.setHp(character.getHp() + 50);  
13        }  
14    }  
15 }
```



```
1 public class Hero extends Character {  
2  
3     @Override  
4     public void attack(Kinoko kinoko) {  
5         System.out.println(this.name + "의 공격!");  
6         System.out.println("적에게 10포인트의 데미지를 주었다!");  
7         kinoko.hp -= 10;  
8     }  
9  
10    public void attack(Goblin goblin) {  
11        System.out.println(this.name + "의 공격!");  
12        System.out.println("적에게 10포인트의 데미지를 주었다!");  
13        goblin.hp -= 10;  
14    }  
15  
16    // 아래에 슬라임용 작성  
17 }
```

```
1 public class Hero extends Character {  
2  
3     @Override  
4     public void attack(Monster monster) {  
5         System.out.println(this.name + "의 공격!");  
6         System.out.println("적에게 10포인트의 데미지를 주었다!");  
7         monster.hp -= 10;  
8     }  
9  
10 }
```



```
1 public class Main {  
2     public static void main(String[] args) {  
3         // 타입은 Monster 로 통 치기  
4         Monster[] monsters = new Monster[3];  
5         monsters[0] = new Slime();           // run 재정의  
6         monsters[1] = new Goblin();          // run 재정의  
7         monsters[2] = new DeathBat();        // run 재정의  
8  
9         // 동작은 안에 담긴 객체를 따름  
10        for (Monster monster : monsters) {  
11            monster.run();  
12        }  
13    }  
14 }
```

인스턴스를 애매하게 통치기

- 상속에 의한 **is-a** 관계가 성립한다면, 인스턴스를 부모 클래스 타입의 변수에 대입할 수 있다
- 부모 클래스 타입 변수에 대입하는 것으로, 통 칠 수 있음

상자의 타입 과 내용의 타입 의 역할

- 어떤 멤버를 이용할 수 있는가는 상자의 타입이 결정한다
- 멤버가 어떻게 움직이는지는 내용의 타입이 결정한다

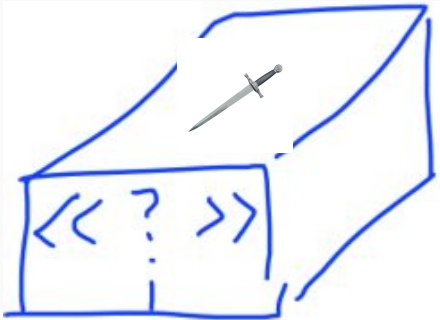
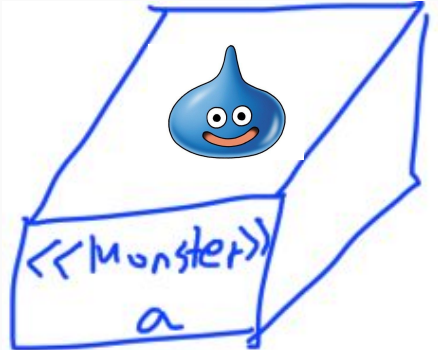
취급 변경

- 캐스트 연산자를 이용하면, 강제 대입이 가능
- 부정한 대입이 발생할 경우, **ClassCastException** 이 발생

다형성

- 다른 인스턴스를 동일시하여, 부모 클래스 타입의 배열에 담을 수 있다
- 마찬가지로, 부모 클래스 타입의 인수나 리턴 값을 이용하여, 다른 클래스를 모아서 처리 가능
- 동일시 취급 해도, 각각의 인스턴스는 각 클래스의 정의를 따르고 다른 동작을 한다.

빈칸에 들어갈 적절한 클래스명을 정하십시오

	(1)	(2)
코드	<code>Item i = new Sword();</code>	<code>____ a = new ____();</code>
이미지	<p>____ 인스턴스</p> 	<p>____ 인스턴스</p> 
해설문	<p>____ 를 생성했지만 어쨌든 ____ 로 보임</p>	<p>Slime 을 생성했지만 어쨌든 ____ 로 보임</p>

```

1  public class A extends Y {
2      @Override
3      public void a() { System.out.println("Aa"); }
6      @Override
7      public void b() { System.out.println("Ab"); }
10     public void c() { System.out.println("Ac"); }
13 }

```

```

1  public class B extends Y {
2      @Override
3      public void a() { System.out.println("Ba"); }
6      @Override
7      public void b() { System.out.println("Bb"); }
10     public void c() { System.out.println("Bc"); }
13 }

```

```

1  public abstract class Y implements X {
2      public abstract void a();
3      public abstract void b();
4  }

```

```

1  public interface X {
2      void a();
3  }

```

이런 클래스가 선언되어 있다.
다음 물음에 답하십시오

1. **X obj = new A();** 로 A인스턴스를 생성한 후, 변수 **obj**에서 호출할 수 있는 메소드를 **a()**, **b()**, **c()** 중에 골라보시오
2. **Y y1 = new A();**
Y y2 = new B(); 로 A와 B의 인스턴스를 생성한 후
y1.a();
y2.a(); 를 실행했을 때에 화면에 표시되는 내용을 말하십시오.

연습문제 13-3

문제 13-2 에서 이용한 **A**클래스나 **B**클래스의 인스턴스를 각각 1개씩 생성하여, 요소의 수가 2개인 1차원 배열에 차례로 담는다.

그 후에 배열의 요소를 루프로 차례대로 꺼내 각각의 인스턴스의 **b()** 메소드를 호출 하여야 한다. 이상을 전제로 다음 물음에 답하시오.

1. 배열변수의 타입으로 무엇을 사용하여야 하는가
2. 위에서 설명하고 있는 프로그램을 작성하시오

연습문제 13-4

- `UserRepository` interface를 생성하세요. 이 인터페이스는 `CRUD(Create, Read, Update, Delete)` 기능을 갖도록 메소드를 선언합니다.
- `UserCsvFileRepository` 클래스에서 사용할 데이터 구조를 정의하세요. 예를 들어, 사용자 정보를 저장하는 `User` 클래스를 정의하고 `UserCsvFileRepository` 클래스에서 이를 사용하세요.
- `UserRepository` interface를 구현한 `UserCsvFileRepository` 클래스를 생성하세요. `UserCsvFileRepository` 클래스는 `csv` 파일로부터 데이터를 읽고 쓰는 기능을 갖도록 구현합니다.
- `UserCsvFileRepository` 클래스의 `create`, `read`, `update`, `delete` 메소드를 구현하세요.