

# 제10장 캡슐화

# 캡슐화 (encapsulation)

이제 클래스나 인스턴스를 이용하여 현실세계를 객체 지향 프로그램으로 자유롭게 개발 할 수 있게 되었다.

하지만, 실수로 속성을 덮어 쓰거나, 잘못된 조작 하는 등의 휴먼 에러 (human error) 를 완전히 없앨 수는 없다.

그래서 **Java** 에는 실수를 미연에 방지하는 “캡슐화” 라는 방법이 있다.

```

1  public class Hero {
2      int hp;
3      String name;
4      Sword sword;
5      static int money;
6
7      void bye() {
8          System.out.println("용자는 이별을 고했다");
9      }
10     void die() {
11         System.out.println(this.name + "는 죽었다");
12         System.out.println("Game Over");
13     }
14     void sleep() {
15         this.hp = 100;
16         System.out.println(this.name + "는 잠을 자고 회복했다!");
17     }
18     void attack(Kinoko enemy) {
19         System.out.println(this.name + "의 공격!");
20         System.out.println("괴물 버섯" + enemy.suffix + "로부터 2포인트의 반격을 받았다");
21         this.hp -= 2;
22         if (this.hp <= 0) {
23             this.die();
24         }
25     }
26 }


```

```
1 public class Inn {  
2     void checkIn(Hero hero) {  
3         hero.hp = -100;  
4     }  
5 }
```

```
1  public class King {  
2      void talk(Hero hero) {  
3          System.out.println("왕 : 우리 성에 어서오시오. 용사 " +  
4              hero.name + "이여");  
5          System.out.println("왕 : 긴 여행에 피로하겠군");  
6          System.out.println("왕 : 우선 성 아랫 마을을 보고 와도 좋소. " +  
7              "그럼 또 봅시다");  
8          hero.die();  
9      }  
10 }
```

# 멤버에 대한 액세스 제어

## 접근 지정자 (access modifier)

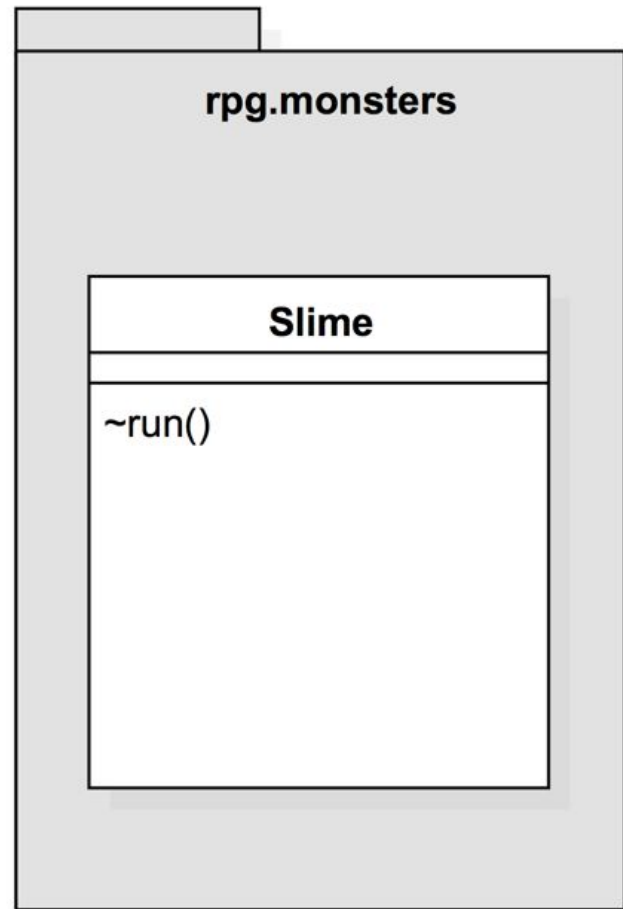
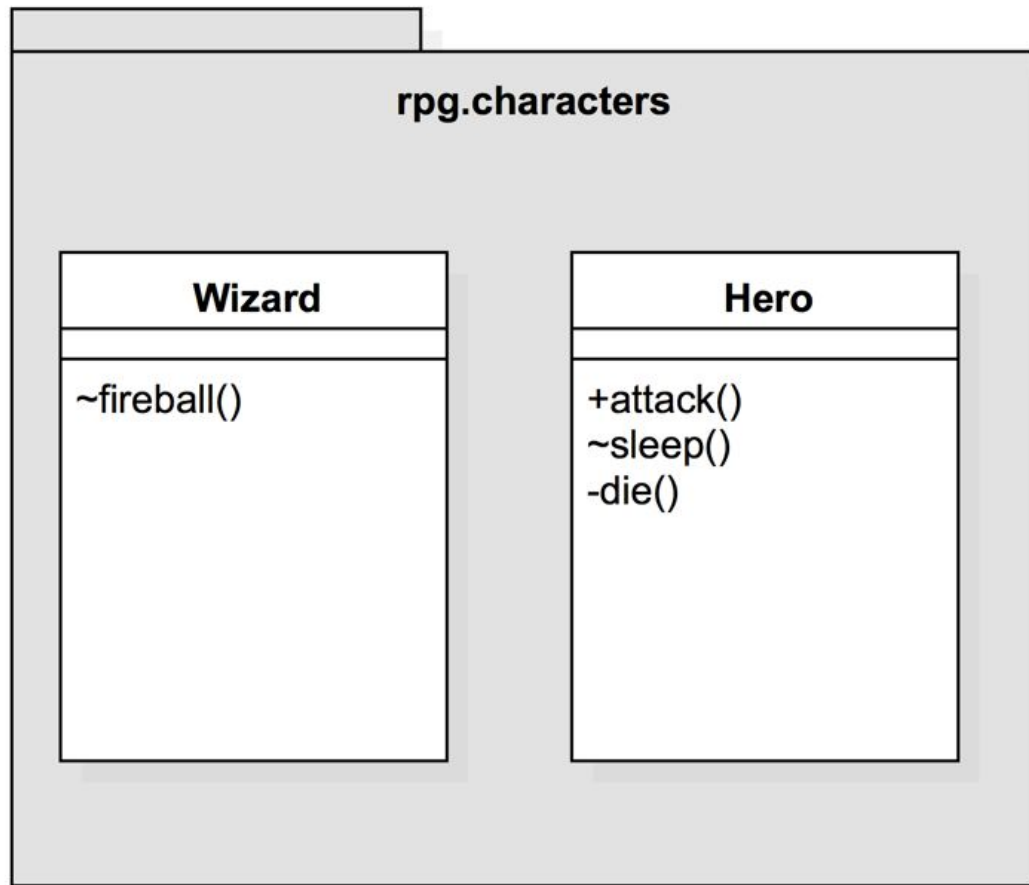
제한 범위	명칭	설정 방법	접근 가능한 범위
제한이 엄격	private	private	자기 자신의 클래스
	package private (default)	(아무것도 안 씀)	자신과 같은 패키지에 소속된 클래스
	protected	protected	자신과 같은 패키지에 소속되던지, 자신을 상속받은 자식 클래스
제한이 느슨	public	public	모든 클래스

```
1  public class Hero {  
2      private int hp;  
3      String name;  
4      Sword sword;  
5      static int money;  
6  
7      void sleep() {  
8          this.hp = 100;  
9          System.out.println(this.name + "는 잠을 자고  
10     }
```

```
1 public class Hero {  
2     private void die() {  
3         System.out.println(this.name + "는 죽었다");  
4         System.out.println("Game Over");  
5     }
```



```
1  public class Hero {
2      private void die() {
3          System.out.println(this.name + "는 죽었다");
4          System.out.println("Game Over");
5      }
6
7      public void attack(Kinoko enemy) {
8          System.out.println(this.name + "의 공격!");
9          System.out.println("괴물 버섯" + enemy.suffix
10         this.hp -= 2;
11         if (this.hp <= 0) {
12             this.die();
13         }
14     }
```



# 멤버에 관한 액세스 지정의 정석

- 필드는 전부 `private`
- 메소드는 전부 `public`

# 클래스에 대한 액세스 지정의 정석

- 별다른 이유가 없으면 `public`

# getter 와 setter

메소드를 경유한 필드 조작

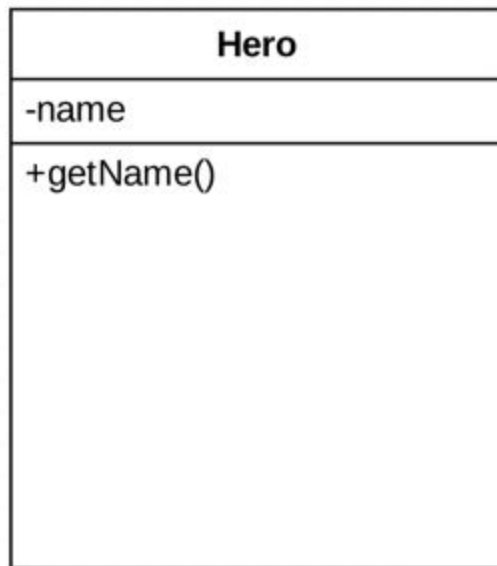
모든 필드를 **private** 로 지정 해 다른 클래스로부터 접근이 안 되도록 막는다.

메소드를 통해서 접근 하도록 클래스를 설계하는 것이 기본.

```
1 public class King {  
2     void talk(Hero hero) {  
3         System.out.println("우리 나라에 어서오세요. 용사 " +  
4             hero.name + "이여");  
5     }  
6 }
```

```
1 public class Hero {  
2     private String name;  
3  
4     public String getName() {  
5         return name;  
6     }  
7 }
```

```
1 public class King {  
2     void talk(Hero hero) {  
3         System.out.println("우리 나라에 어서오세요. 용사 " +  
4             hero.getName() + "이여");  
5     }  
6 }
```





```
1  public class Hero {  
2      private String name;  
3  
4      public String getName() {  
5          return name;  
6      }  
7  
8      public void setName(String name) {  
9          this.name = name;  
10     }
```

```
1 public class Hero {  
2     String name;  
3 }
```

```
1 public class Hero {  
2     private String name;  
3  
4     public String getName() {  
5         return name;  
6     }  
7  
8     public void setName(String name) {  
9         this.name = name;  
10    }
```

# getter / setter 의 메리트

1. Read Only, Write Only 필드의 실현
2. 필드의 이름 등, 클래스의 내부 설계를 자유롭게 변경 가능
3. 필드로의 액세스를 검사 가능

```
String name;

public void setName(String name) {
    if (name == null) {
        throw new IllegalArgumentException("이름은 null이 아니어야 함");
    }
    if (name.length() <= 1) {
        throw new IllegalArgumentException("이름이 너무 짧음");
    }
    if (name.length() >= 8) {
        throw new IllegalArgumentException("이름이 너무 깊");
    }
    this.name = name;
}
```

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

# 클래스에 대한 액세스 제어

클래스의 액세스 제어의 지정방법과 범위

이름	기술 방법	접근 가능한 범위	제한
<code>package private</code>	(아무것도 쓰지 않음)	자신과 같은 패키지에 소속된 클래스	엄격
<code>public</code>	<code>public</code>	모든 클래스	느슨

비 `public` 클래스의 특징

1. 클래스의 이름은 소스 파일명과 달라도 된다
2. 1개의 소스 파일에 여러개 선언해도 된다

# 캡슐화에 대한 생각

## 메소드로 필드를 보호

## 캡슐화의 개요

- 캡슐화를 하여 멤버나 클래스로의 접근을 제어할 수 있음
- 특히, 필드에 “현실세계에서 있을 수 없는 값”이 들어가지 않도록 제어

## 멤버에 대한 접근 지정

- **private** 지정된 멤버는, 동일 클래스내에서만 접근 가능
- **package private** (아무것도 지정 안된) 지정된 멤버는, 동일 패키지내의 클래스에서만 접근 가능.
- **public** 지정된 멤버는, 모든 클래스에서 접근 가능

## 클래스에 대한 접근 지정

- **package private** (연산자 없음) 으로 선언 된 클래스는, 동일 패키지내의 클래스에서만 접근 가능.
- **public** 지정된 클래스는, 모든 클래스에서 접근 가능

## 캡슐화의 정석

- 클래스는 **public**, 메소드는 **public**, 필드는 **private** 로 지정
- 필드에 접근하기 위한 메소드로서 **getter** 나 **setter**를 준비
- **setter** 내부에서는 인수의 타당성 검사를 수행



다음 2개의 클래스 “Wizard (마법사)”, “Wand (지팡이)” 의 모든 필드와 메소드에 대해, 캡슐화의 정석에 따라 접근 지정자를 추가하시오. (Wizard 클래스의 컴파일 에러는 일단 무시하시오)

```
1 public class Wand {  
2     String name;    // 지팡이의 이름  
3     double power;   // 지팡이의 마력  
4 }
```

```
1 public class Wizard {  
2     int hp;  
3     int mp;  
4     String name;  
5     Wand wand;  
6  
7     void heal(Hero hero) {  
8         int basePoint = 10;    // 기본회복 포인트  
9         int recovPoint = (int) (basePoint * this.wand.power);    // 지팡이에 의한 증폭  
10        hero.setHp(hero.getHp() + recovPoint);    // 용사의 HP를 회복  
11    }
```

## 연습문제 10-2

문제 10-1 에서 작성한 **Wand** 클래스와 **Wizard** 클래스의 모든 필드에 대해, 정석에 따라 **getter** 메소드를 **setter** 메소드를 작성하시오.

그리고, **Wizard** 클래스의 **heal** 메소드에서 발생하는 컴파일 에러를 해결하시오.

**setter** 메소드에 대해서 인수의 타당성 검사는 하지 않아도 됨.

## 연습문제 10-3

문제 10-2 에서 작성한 **Wand** 클래스와 **Wizard** 클래스의 각 **setter** 메소드에 대해, 아래의 4가지 규칙에 따라 인수의 타당성 검사를 추가하십시오.

부정한 값이 설정 될 경우에는 “**throw new IllegalArgumentException(“에러메세지”);**” 를 기술하고 프로그램을 중단 시킵니다.

1. 마법사나 지팡이의 이름은 **null** 일 수 없고, 반드시 **3**문자 이상이어야 한다
2. 지팡이의 마력은 **0.5** 이상 **100.0** 이하여야 한다.
3. 마법사의 지팡이는 **null** 일 수 없다.
4. 마법사의 **MP**는 **0** 이상이어야 한다.
5. **HP**가 음수가 되는 상황에서는 대신 **0**을 설정 되도록 한다. (에러 아님)