

Item 41

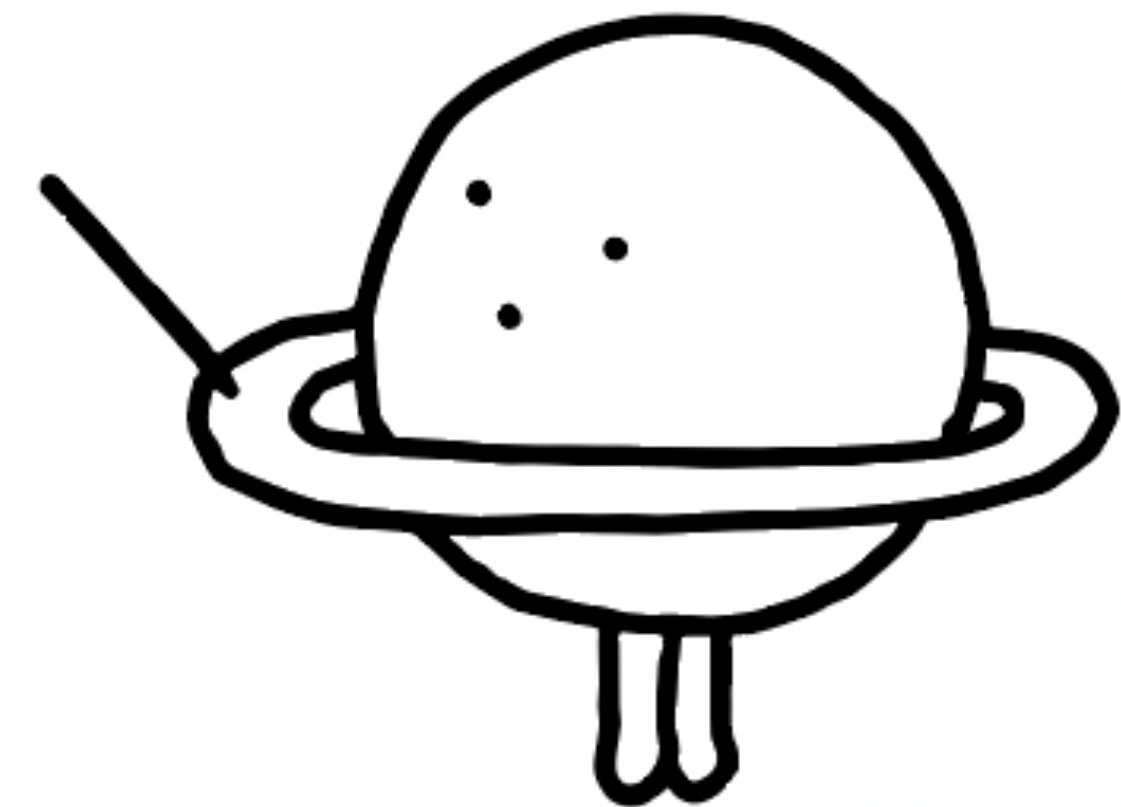
정의하려는 것이 타입이라면 마커 인터페이스를 사용하라

Contents

1. 마커 인터페이스

2. 마커 애노테이션

3. 마커 인터페이스 vs 마커 애노테이션



마커 인터페이스


클래스에게 특정 타입을 표시해주기 위한 인터페이스

마커 인터페이스




```
public interface Serializable {  
  
}
```

마커 인터페이스



```
public class Animal implements Serializable {  
    @Override  
    public String toString() {  
        return "Animal";  
    }  
}
```



```
public class Human {  
    @Override  
    public String toString() {  
        return "Human";  
    }  
}
```

마커 인터페이스

```
public class Main {  
  
    public static void main(String[] args) {  
        final Animal animal = new Animal();  
        final Human human = new Human();  
        outputStream(human);  
        outputStream(animal);  
    }  
  
    private static void outputStream(final Object object) {  
        try {  
            final FileOutputStream fileOutputStream = new FileOutputStream("wilson.csv");  
            new ObjectOutputStream(fileOutputStream).writeObject(object);  
        } catch (IOException e) {  
            System.out.println(object.toString());  
        }  
    }  
}
```

마커 인터페이스

```
public final void writeObject(Object obj) throws IOException {  
    if (enableOverride) {  
        writeObjectOverride(obj);  
        return;  
    }  
    try {  
        writeObject0(obj, false);  
    } catch (IOException ex) {  
        if (depth == 0) {  
            writeFatalException(ex);  
        }  
        throw ex;  
    }  
}
```

마커 인터페이스

```
if (obj instanceof String) {
    writeString((String) obj, unshared);
} else if (cl.isArray()) {
    writeArray(obj, desc, unshared);
} else if (obj instanceof Enum) {
    writeEnum((Enum<?>) obj, desc, unshared);
} else if (obj instanceof Serializable) {
    writeOrdinaryObject(obj, desc, unshared);
} else {
    if (extendedDebugInfo) {
        throw new NotSerializableException(
            cl.getName() + "\n" + debugInfoStack.toString());
    } else {
        throw new NotSerializableException(cl.getName());
    }
}
```


마커 인터페이스

리팩토링 해볼까?

```
public final void writeObject(Object obj) throws IOException {  
    if (enableOverride) {  
        writeObjectOverride(obj);  
        return;  
    }  
    try {  
        writeObject0(obj, false);  
    } catch (IOException ex) {  
        if (depth == 0) {  
            writeFatalException(ex);  
        }  
        throw ex;  
    }  
}
```

마커 인터페이스

```
public final <T extends Serializable> void writeObject(T obj) throws IOException {  
    if (enableOverride) {  
        writeObjectOverride(obj);  
        return;  
    }  
    try {  
        writeObject0(obj, false);  
    } catch (IOException ex) {  
        if (depth == 0) {  
            writeFatalException(ex);  
        }  
        throw ex;  
    }  
}
```

마커 인터페이스



```
public class Parent {  
  
    @Override  
    public String toString() {  
        return getClass().getSimpleName();  
    }  
}
```



```
public interface NotChild  
{  
}
```

마커 인터페이스



```
public class ChildA extends Parent  
{
```



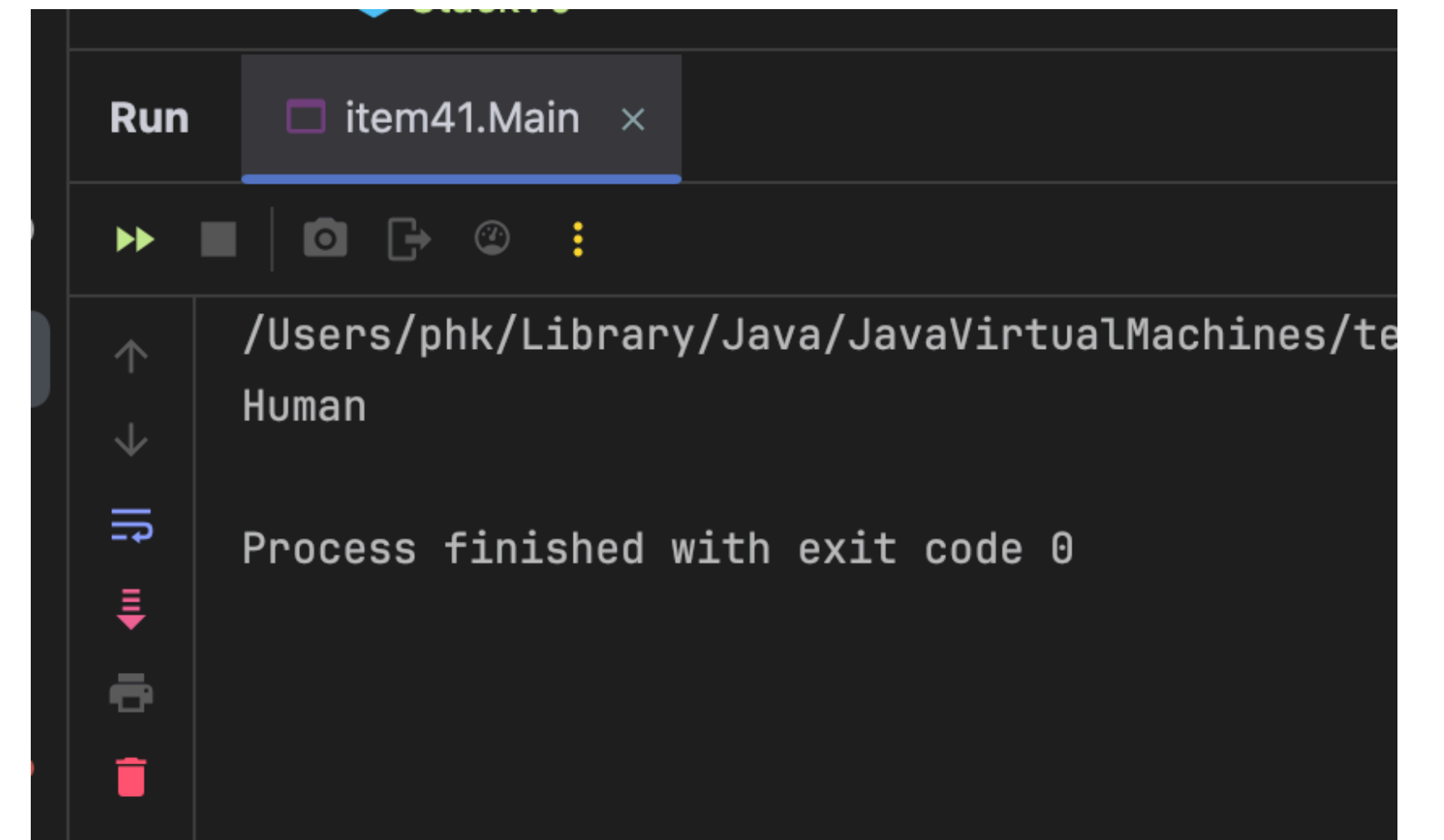
```
public class ChildB extends Parent implements NotChild  
{
```



```
public class ChildC extends Parent  
{
```

마커 인터페이스

```
public class Main {  
    public static void main(String[] args) {  
        final List<Parent> parents = List.of(new ChildA(), new ChildB(),  
new ChildC());  
        for (Parent parent : parents) {  
            if (parent instanceof NotChild) {  
                continue;  
            }  
            System.out.println(parent.toString());  
        }  
    }  
}
```



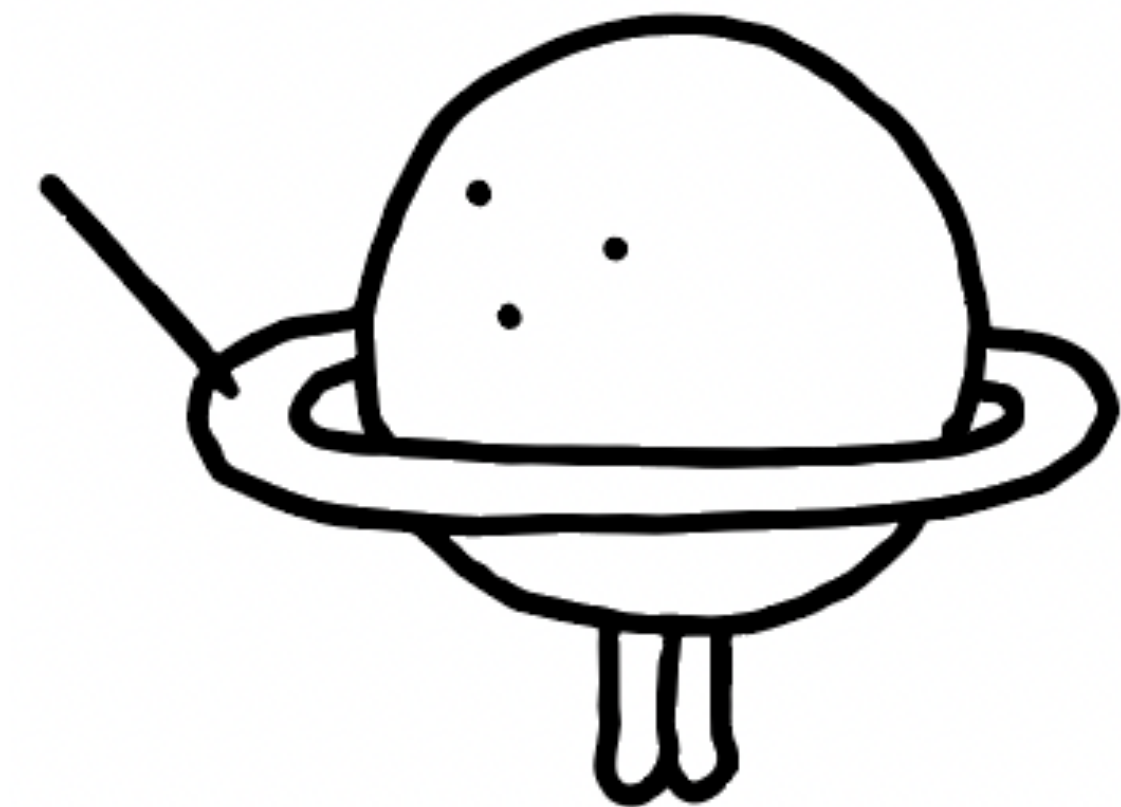
```
Run    item41.Main x  
▶ ■ | 📷 📄 🕒 ⋮  
↑ /Users/phk/Library/Java/JavaVirtualMachines/te  
↓ Human  
⏮  
⏭ Process finished with exit code 0  
⏮  
🗑
```

Contents

1. 마커 인터페이스

2. 마커 애노테이션

3. 마커 인터페이스 vs 마커 애노테이션



마커 애노테이션

표시를 통해 컴파일 또는 런타임에서 특별한 처리가 필요함을 알리는 키워드

마커 애노테이션

```
@Component  
public class SpringBean  
{  
}
```

@Target({ElementType.TYPE})

- 클래스, 인터페이스, Enum, 다른 애노테이션에 적용 가능

```
@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
@Indexed  
public @interface Component {  
    String value() default "";  
}
```

@Retention(RetentionPolicy.RUNTIME)

- 런타임까지 유지 되어 리플렉션으로 접근 가능

마커 애노테이션

```
@Test
void test() {
}
```

```
@Target({ElementType.ANNOTATION_TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@API(
    status = Status.STABLE,
    since = "5.0"
)
@Testable
public @interface Test {
}
```

@Target({ElementType.ANNOTATION_TYPE, ElementType.METHOD})

- 다른 애노테이션에 적용 가능
- 메서드에 적용 가능

@Retention(RetentionPolicy.RUNTIME)

- 런타임까지 유지 되어 리플렉션으로 접근 가능

마커 애노테이션

RetentionPolicy	설명	유지 범위	주요 용도	예시
SOURCE	소스 코드에만 존재	컴파일러까지만 유지	컴파일 타임 검사	@Override, @SuppressWarnings
CLASS	클래스 파일에 포함	바이트코드까지 유지 (런타임 X)	바이트코드 처리/분석	Lombok(@Getter, @Setter)
RUNTIME	런타임까지 유지	JVM 실행 중에도 접근 가능	리플렉션 기반 처리	@Autowired, @Entity

마커 애노테이션

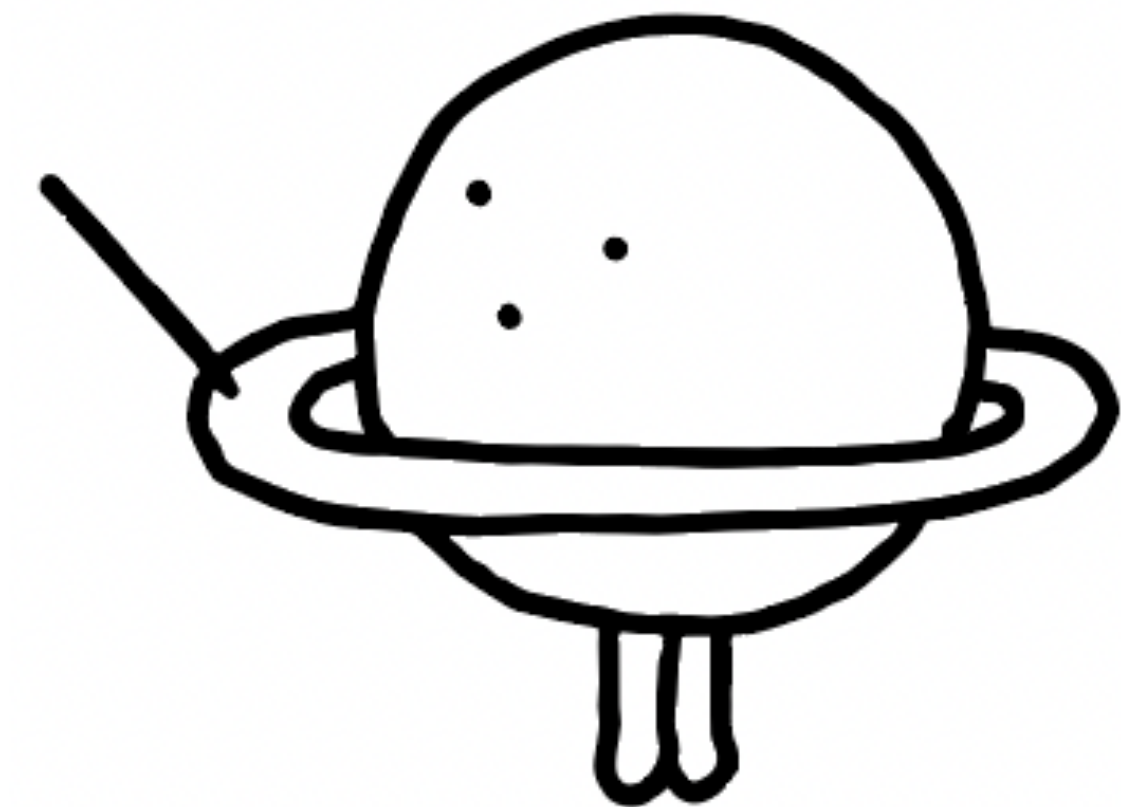
ElementType	적용 대상	설명	예시
TYPE	클래스, 인터페이스, 열거형	타입 선언에 사용	@Component, @Entity
METHOD	메서드	메서드 선언에 사용	@Override, @GetMapping
FIELD	필드(멤버 변수)	클래스의 필드에 사용	@Autowired, @Column
PARAMETER	메서드 매개변수	메서드 파라미터에 사용	@PathVariable, @RequestParam
CONSTRUCTOR	생성자	생성자 선언에 사용	@Autowired 생성자 주입
ANNOTATION_TYPE	애노테이션	다른 애노테이션 정의에 사용	@Target, @Retention

Contents

1. 마커 인터페이스

2. 마커 애노테이션

3. 마커 인터페이스 vs 마커 애노테이션



마커 인터페이스 vs 마커 애노테이션

✓ 특정 타입으로 지정해야 한다.



마커 인터페이스

✓ 프레임워크의 지원을 받아야 한다.



마커 애노테이션

Q & A

