item 40

애너테이션을 일관되게 사용하라

유현우

CONTENTS

CONTENTS

오버로딩(Overloading)

```java
public class View {
  public void print(String description) {
    System.out.println("description = " + description);
  }
  public void print(int count) {
    System.out.println("count = " + count);
  }
}
```

같은 이름의 메서드 여러개를 가지면서 매개변수의 유형과 개수가 다르도록 하는 기술

오버로딩(Overloading)

compile error

```
public class View {
    no usages
    public void print(String description) {
        System.out.println("description = " + description);
    }
    no usages
    public int print(String description) {
        System.out.println("description = " + description);
        return -1;
    }
}
```

메서드 시그니처는 메서드 이름과 매개변수 리스트 조합 (반환값 X)

오버라이딩(Overriding)

```java
public abstract class View {
  abstract void print(String description);
}
```

```java
class ViewImpl extends View {
  @Override
  void print(String description) {
    System.out.println("description = " + description);
  }
}
```

```java
class ViewImpl extends View {
  void print(String description) {
    System.out.println("description = " + description);
  }
}
```

상위 클래스가 가지고 있는 메서드를 하위 클래스가 재정의해서 사용

상위 타입에 메서드 이름이 같은 메서드가 있을 때

파라미터가 전부 틀리다면 오버로딩으로 동작하고 전부 동일하다면 오버라이딩이 된다.

CONTENTS

@override

```java
public abstract class View {
  abstract void print(String description);
}
```

```java
class ViewImpl extends View {
  @Override
  void print(String description) {
    System.out.println("description = " + description);
  }
}
```

상위 타입의 메서드를 재정의했음을 뜻하는 애너테이션

@override 예제

Object.java

```
See Also:       hashCode(),
                java.util.HashMap
163 public boolean equals(Object obj) {
164     return (this == obj);
165 }
166
```

```java
public class Bigram {
    private final char first;
    private final char second;

    public Bigram(char first, char second) {
        this.first  = first;
        this.second = second;
    }

    public boolean equals(Bigram b) {
        return b.first == first && b.second == second;
    }

    public int hashCode() {
        return 31 * first + second;
    }
}
```

@override 예제

```
public static void main(String[] args) {
  Set<Bigram> s = new HashSet<>();
  for (int i = 0; i < 10; i++)
    for (char ch = 'a'; ch <= 'z'; ch++)
      s.add(new Bigram(ch, ch));
  System.out.println(s.size());
}
```

ASCII 코드표 (0-127)

97	61	a	111	6F	o
98	62	b	112	70	p
99	63	c	113	71	q
100	64	d	114	72	r
101	65	e	115	73	s
102	66	f	116	74	t
103	67	g	117	75	u
104	68	h	118	76	v
105	69	i	119	77	w
106	6A	j	120	78	x
107	6B	k	121	79	y
108	6C	l	122	7A	z
109	6D	m			
110	6E	n			

@override 예제

```java
public static void main(String[] args) {
  Set<Bigram> s = new HashSet<>();
  for (int i = 0; i < 10; i++)
    for (char ch = 'a'; ch <= 'z'; ch++)
      s.add(new Bigram(ch, ch));
  System.out.println(s.size());
}
```

ASCII 코드표 (0-127)

97	61	a	111	6F	o
98	62	b	112	70	p
99	63	c	113	71	q
100	64	d	114	72	r
101	65	e	115	73	s
102	66	f	116	74	t
103	67	g	117	75	u
104	68	h	118	76	v
105	69	i	119	77	w
106	6A	j	120	78	x
107	6B	k	121	79	y
108	6C	l	122	7A	z
109	6D	m			
110	6E	n			

Set은 중복을 허용하지 않기 때문에 260이 출력되어야 하는데 2600이 출력된다.

@override 예제

Object.java

```
See Also:       hashCode(),
                java.util.HashMap
163 public boolean equals(Object obj) {
164     return (this == obj);
165 }
166
```

```java
public class Bigram {
    private final char first;
    private final char second;

    public Bigram(char first, char second) {
        this.first  = first;
        this.second = second;
    }

    public boolean equals(Bigram b) {
        return b.first == first && b.second == second;
    }

    public int hashCode() {
        return 31 * first + second;
    }
}
```

매개변수 타입이 달라서 상속한 equals와 별개인 equals 메서드를 정의한 꼴이 되었다.

@override 예제

```java
@Override
public boolean equals(Bigram b) {
  return b.first == first && b.second == second;
}

@Override
public int hashCode() {
  return 31 * first + second;
}
```

```java
@Override
public boolean equals(Object o) {
  if (!(o instanceof Bigram b)) {
    return false;
  }
  return b.first == first && b.second == second;
}

@Override
public int hashCode() {
  return 31 * first + second;
}
```

@Override 애너테이션을 선언해주면 컴파일 타임에 오류를 찾을 수 있다.

@override를 선언하지 않을 때

- equals에 문제가 있음에도 컴파일에는 성공한다.
- 이전 equals 메서드의 문제처럼 오버라이딩이 아닌 오버로딩이 되서 문제를 파악하기 힘들 수 있다.

상위 클래스의 메서드를 재정의하려는 모든 메서드에는
@Override 애너테이션을 달자.

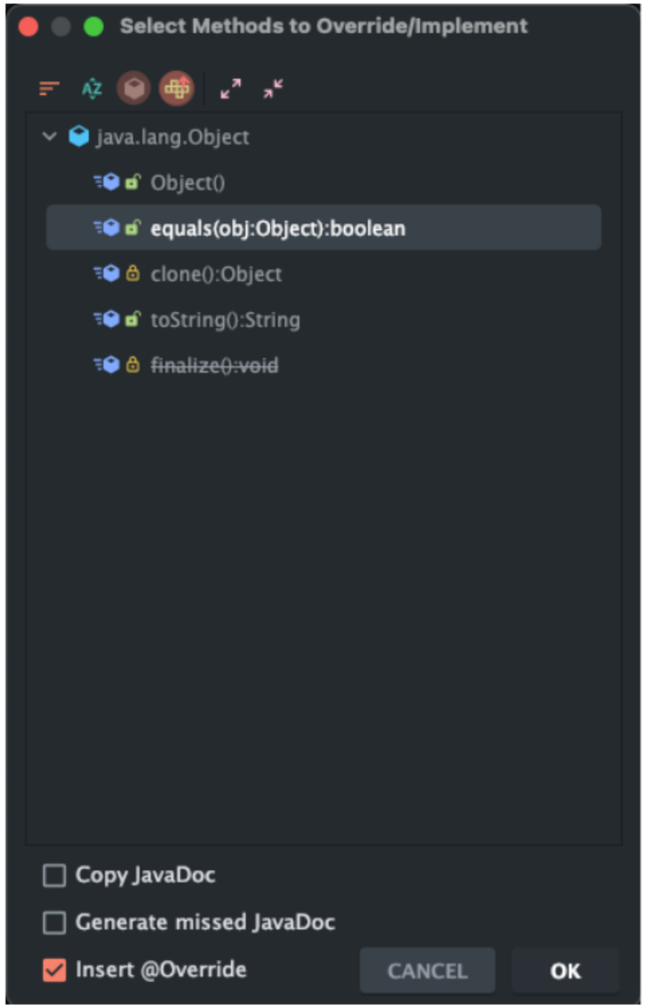
예외 사항

```
1 usage   1 inheritor
abstract class Bird {
    no usages
    abstract void fly();
}

no usages
class Parrot implements Bird {
}
```

```
1 usage   1 implementation
interface Flyable {
    no usages
    void fly();
}

no usages
class Bird implements Flyable {
}
```

구체 클래스인데 구현하지 않은 추상 메서드가 남아 있다면 컴파일러가 알려주기 때문이다.

예외 사항




```java
@Override
public boolean equals(Object obj) {
  return super.equals(obj);
}
```

대부분 IDE는 재정의할 메서드 선택하면 자동으로 @Override를 붙여준다.

CONTENTS

default 메서드와 @Override

```java
interface Vehicle {
  default void ride() {
    System.out.println("ride");
  }
}

class Bike implements Vehicle {
  @Override
  public void ride() {
    System.out.println("bike");
  }
}
```

default 메서드를 재정의할 때 @Override를 시그니처가 올바른지 재차 확신할 수 있다.

정리

- 상위 클래스나 상위 인터페이스의 메서드를 재정의 하는 모든 메서드에 @Override를 다는 것이 좋다.

- Set 인터페이스는 Collection 인터페이스를 확장했지만 새로 추가한 메서드는 없기 때문에 모든 메서드 선언에 @Override를 달아 실수로 추가한 메서드가 없음을 보장했다.

- 재정의한 모든 메서드에 @Override 애너테이션을 의식적으로 달자.