

Item 40.

@Override 애노테이션을 일관되게 사용하라





INDEX

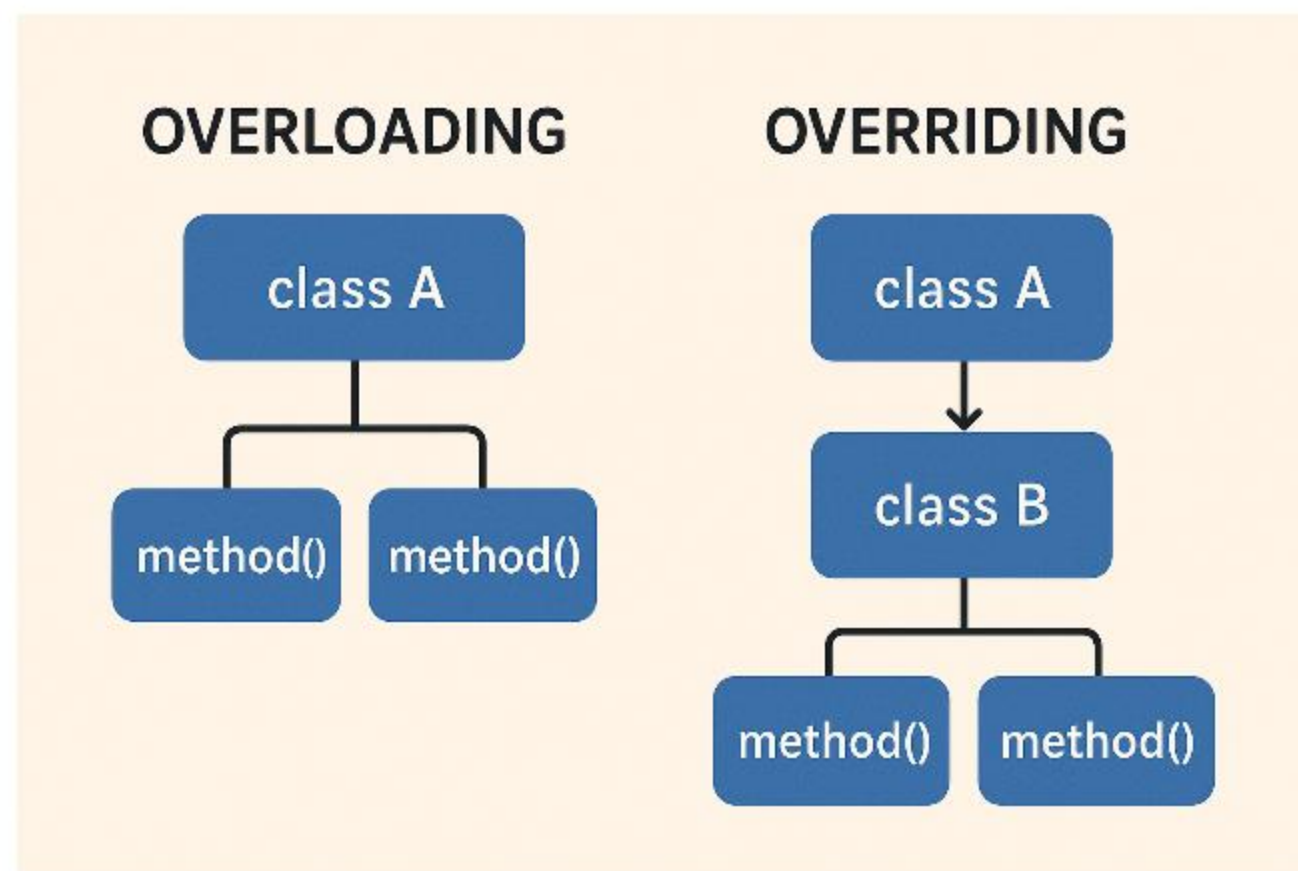
목차

1. 오버로딩, 오버라이딩 개념
2. @Override
3. Bigram 실습
4. 생략 및 명시



01

오버로딩, 오버라이딩 개념





```
1  // 매개변수 타입
2  public int add(int a, int b) {
3      return a + b;
4  }
5
6  public double add(double a, double b) {
7      return a + b;
8  }
9
10 // 매개변수 개수
11 public int add(int a, int b, int c) {
12     return a + b + c;
13 }
14
15 // 매개변수 순서
16 public double add(int a, double b) {
17     return a + b;
18 }
19
20 public double add(double b, int a) {
21     return a + b;
22 }
```



Overloading

같은 이름의 메서드를 매개변수의 타입이나 개수를 다르게 정의하는 것.

컴파일 타임에 어떤 메서드가 호출될지 결정.

- 매개변수 타입/개수/순서 영향이 있다.
- 리턴 타입은 영향이 없다.

```

1  class Parents {
2      void print() {
3          System.out.println("일반 출력");
4      }
5  }
6
7  class Child1 extends Parents {
8
9      @Override
10     protected void print() {
11         System.out.println("자식 출력");
12     }
13 }
14
15 class Child2 extends Parents {
16
17     @Override
18     public void print() {
19         System.out.println("자식 출력");
20     }
21 }
22
23 class Child3 extends Parents {
24
25     @Override
26     private void print() {
27         System.out.println("자식 출력");
28     }
29 }

```

Overriding

상속 관계에서 부모 클래스의 메서드를 자식 클래스가 재정의 하는 것.

런타임에 어떤 메서드가 호출될지 결정.

- 상속 관계에서 사용
- 메서드 이름, 매개변수, 리턴 타입이 모두 동일
- 접근 제어자는 부모랑 같거나 더 넓은 범위
- **@Override 애노테이션** 사용

```
@Override  
private void print() {  
    System.out.println("자식 출력");  
}
```

접근 제어자

부모랑 같거나 더 넓은 범위만
지정 가능



02

@Override



@Override



```
1  class Parents {
2      void print() {
3          ...
4      }
5  }
6
7  class Child1 extends Parents {
8
9      @Override
10     protected void print() {
11         ...
12     }
13 }
14
15 class Child2 extends Parents {
16
17     @Override
18     public void print() {
19         ...
20     }
21 }
```



@Override

Java에서 메서드 오버라이딩을 명시적으로 나타내는 애노테이션.

부모 클래스나 인터페이스에서 정의한 메서드를 자식 클래스가 재정의할 때 사용

- **사전 방지**: 메서드 이름, 매개변수, 반환 타입 등을 검사하여 실수 방지

- **명확성**: 상속받은 메서드를 재정의했음을 표시

인터페이스



```
1  interface InterfaceExample {  
2      void print(String message);  
3  }  
4  
5  class InterfaceExampleChild1 implements InterfaceExample {  
6      @Override  
7      public void print(String message) {  
8          System.out.println(message);  
9      }  
10 }
```

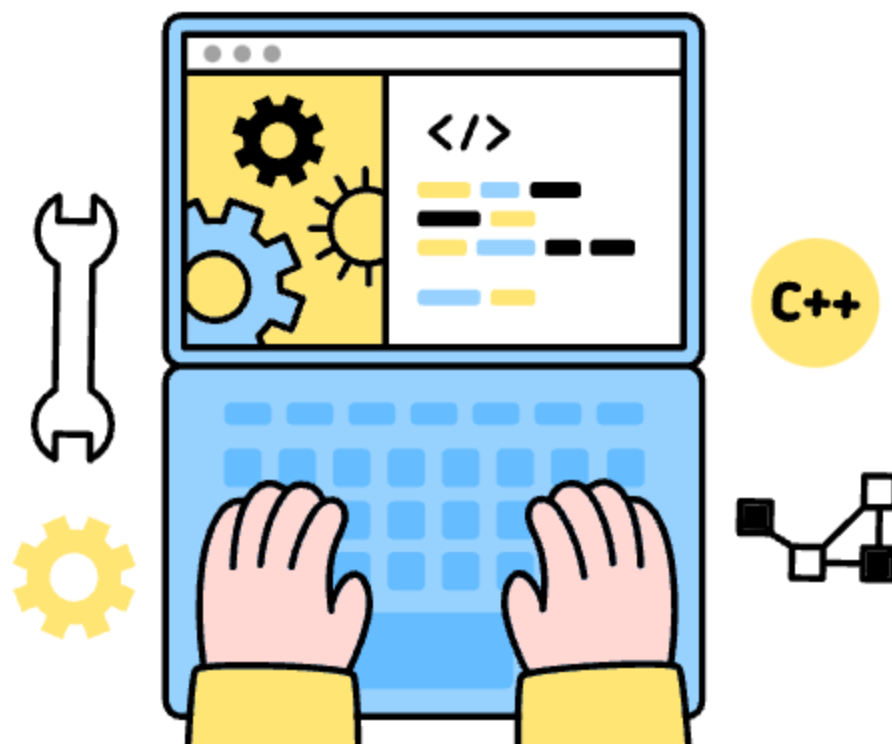
추상 클래스



```
1  public class abstractExample {  
2      public void print(String message) {  
3          System.out.println("부모" + message);  
4      }  
5  }  
6  
7  class abstractChild1 extends abstractExample {  
8      @Override  
9      public void print(String message) {  
10         System.out.println("자식1" + message);  
11     }  
12 }
```

03

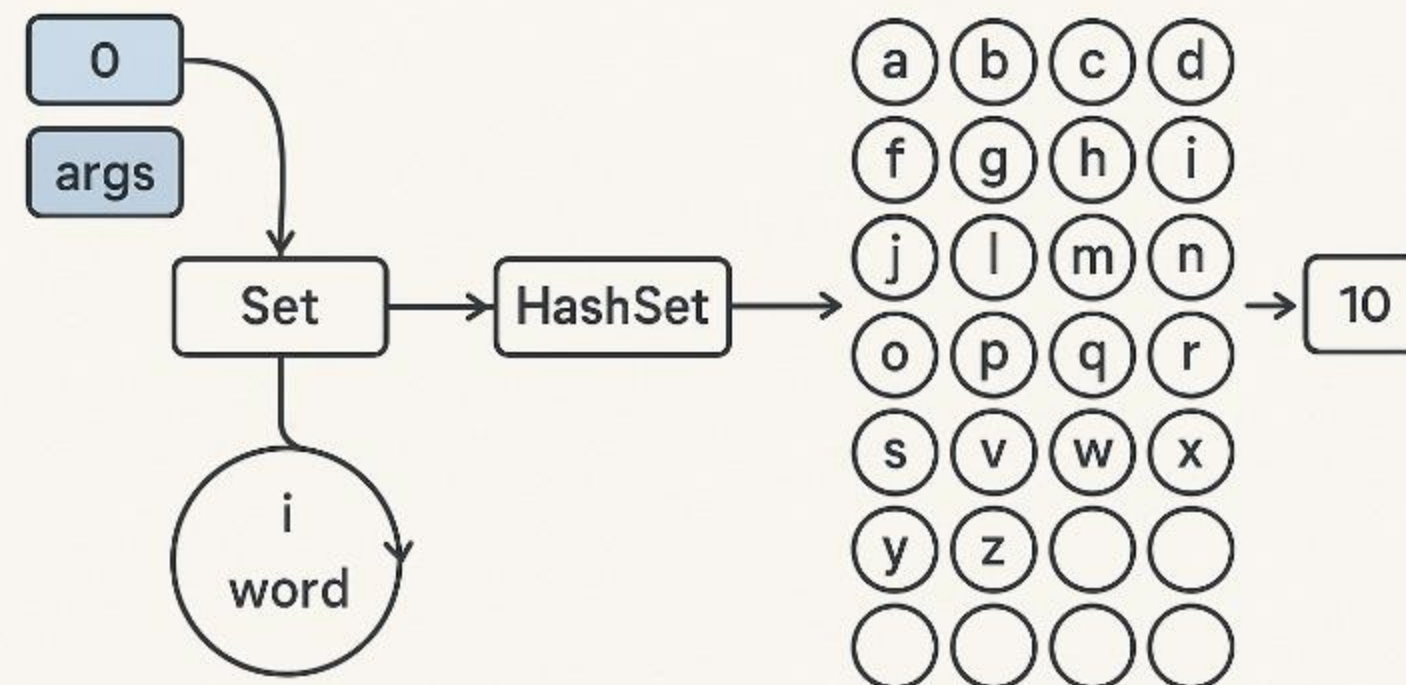
Bigram 실습





```
1 public class BigramExample {
2     private final char word;
3
4     public BigramExample(char word) {
5         this.word = word;
6     }
7
8     public boolean equals(BigramExample b) {
9         return b.word == word;
10    }
11
12    public int hashCode() {
13        return 31 * word;
14    }
15
16    public static void main(String[] args) {
17        Set<BigramExample> s = new HashSet<>();
18        for (int i = 0; i < 10; i++)
19            for (char word = 'a'; word <= 'z'; word++)
20                s.add(new BigramExample(word));
21        System.out.println(s.size());
22    }
23 }
```

BigramExample Algorithm



참고 ☒

영어 알파벳은 총 26개




260



260? 26 값을 기대했는데?

Set<E> 동작 흐름

1. hashCode() 호출해 **같은 해시값을 가지는지 확인**
2. equals() 호출해 **논리적으로 같은 객체인지 비교**
3. 둘 다 같으면 "이미 존재하는 값" 판단



```
1  public class BigramExample {
2      private final char word;
3
4      public BigramExample(char word) {
5          this.word = word;
6      }
7
8      public boolean equals(BigramExample b) {
9          return b.word == word;
10     }
11
12     public int hashCode() {
13         return 31 * word;
14     }
15
16     public static void main(String[] args) {
17         Set<BigramExample> s = new HashSet<>();
18         for (int i = 0; i < 10; i++)
19             for (char word = 'a'; word <= 'z'; word++)
20                 s.add(new BigramExample(word));
21         System.out.println(s.size());
22     }
23 }
```



```
1 public class BigramExample {
2     private final char word;
3
4     public BigramExample(char word) {
5         this.word = word;
6     }
7
8     public boolean equals(BigramExample b) {
9         return b.word == word;
10    }
11
12    public int hashCode() {
13        return 31 * word;
14    }
15
16    public static void main(String[] args) {
17        Set<BigramExample> s = new HashSet<>();
18        for (int i = 0; i < 10; i++)
19            for (char word = 'a'; word <= 'z'; word++)
20                s.add(new BigramExample(word));
21        System.out.println(s.size());
22    }
23 }
```



```
1 @Override
2 public boolean equals(BigramExample b) {
3     return b.word == word;
4 }
```



```
1 @Override
2 public boolean equals(Object o) {
3     if (o == null || getClass() != o.getClass()) {
4         return false;
5     }
6     BigramExample that = (BigramExample) o;
7     return word == that.word;
8 }
```

@Override

publ

}

메서드는 상위 클래스의 메서드를 재정의하지 않습니다

메서드 'equals'을(를) 새 인터페이스로 추출 ↵ ↵ 추가 액션... ↵ ↵

java.lang

```
1 public class BigramExample {
2     private final char word;
3
4     public BigramExample(char word) {
5         this.word = word;
6     }
7
8     @Override
9     public boolean equals(Object o) {
10         if (o == null || getClass() != o.getClass()) {
11             return false;
12         }
13         BigramExample that = (BigramExample) o;
14         return word == that.word;
15     }
16
17     public int hashCode() {
18         return 31 * word;
19     }
20
21     public static void main(String[] args) {
22         Set<BigramExample> s = new HashSet<>();
23         for (int i = 0; i < 10; i++)
24             for (char word = 'a'; word <= 'z'; word++)
25                 s.add(new BigramExample(word));
26         System.out.println(s.size());
27     }
28 }
29
```



260

04

생략 및 명시

@Override 생략

```
public void  
method() {  
  
    /* do something  
    */  
}
```

@Override 명시

```
@Override  
public void {  
    method()  
    /*  
    do somemething  
    */  
}
```

"구현하려는 인터페이스에 디폴트 메서드가 없음을
안다면 이를 구현한 메서드에서는 @Override를
생략해 코드를 조금 더 깔끔히 유지해도 좋다."

클래스 상속



```
1  public class SampleClass {  
2      public void print(String message) {  
3          System.out.println(message);  
4      }  
5  }  
6  
7  class SampleChild extends SampleClass {  
8      @Override  
9      public void print(String message) {  
10         super.print(message);  
11     }  
12 }
```

인터페이스



```
1  public interface SampleInterface {  
2      void print(String message);  
3  }  
4  
5  class SampleInterfaceChild implements SampleInterface {  
6      public void print(String message) {  
7          System.out.println(message);  
8      }  
9  }
```



생략하면 성능상 관여가 있을까?

생략



```
1 class SampleInterfaceChild implements SampleInterface {
2     public void print(String message) {
3         System.out.println(message);
4     }
5 }
```

명시



```
1 class SampleInterfaceChild implements SampleInterface {
2     @Override
3     public void print(String message) {
4         System.out.println(message);
5     }
6 }
```



```
1 class SampleInterfaceChild implements SampleInterface {
2     SampleInterfaceChild();
3     Code:
4         0: aload_0
5         1: invokespecial #1
6         4: return
7
8     public void print(java.lang.String);
9     Code:
10        0: getstatic      #7
11        3: aload_1
12        4: invokevirtual #13
13        7: return
14 }
15
```



```
1 class SampleInterfaceChild implements SampleInterface {
2     SampleInterfaceChild();
3     Code:
4         0: aload_0
5         1: invokespecial #1
6         4: return
7
8     public void print(java.lang.String);
9     Code:
10        0: getstatic      #7
11        3: aload_1
12        4: invokevirtual #13
13        7: return
14 }
15
```




**여러분은 생략, 명시 어떤 것을
선호하시겠습니까?**

"추상 클래스나 인터페이스에서는 상위 클래스나 상위
인터페이스의 메서드를 재정의하는 모든 메서드에
`@Override`를 다는 것이 좋다."

Item 40.

**@Override 애노테이션을
일관되게 사용하라**

