Item 21

인터페이스는 구현하는 쪽을 생각해 설계하라

RAZEL

목차

1 인터페이스의 설계와 배경

2 디폴트 메서드의 도입

3 디폴트 메서드 도입으로 인한 문제

4 인터페이스를 설계할 때

Part 1

인터페이스의 설계와 배경

Part 1

인터페이스의 설계와 배경

interface

1996

Java 1.0

```java
public interface Movable {
    void move();
}
public class Car implements Movable {
    @Override
    public void move() {
        System.out.println("자동차가 이동합니다.");
    }
}
public class Robot implements Movable {
    @Override
    public void move() {
        System.out.println("로봇이 이동합니다.");
    }
}
```

구현해라 !

Part 1

인터페이스의 설계와 배경

```java
public interface Movable {
    void move();
}

public class Car implements Movable {
    @Override
    public void move() {
        System.out.println("자동차가 이동합니다.");
    }
}

public class Robot implements Movable {
    @Override
    public void move() {
        System.out.println("로봇이 이동합니다.");
    }
}
```

```java
public interface Movable {
    void move();
    void turn(); // 새로운 메서드 추가 🚨
}
```

컴파일 오류!
구현체는 Movable의 turn() 메서드를 구현해야 한다

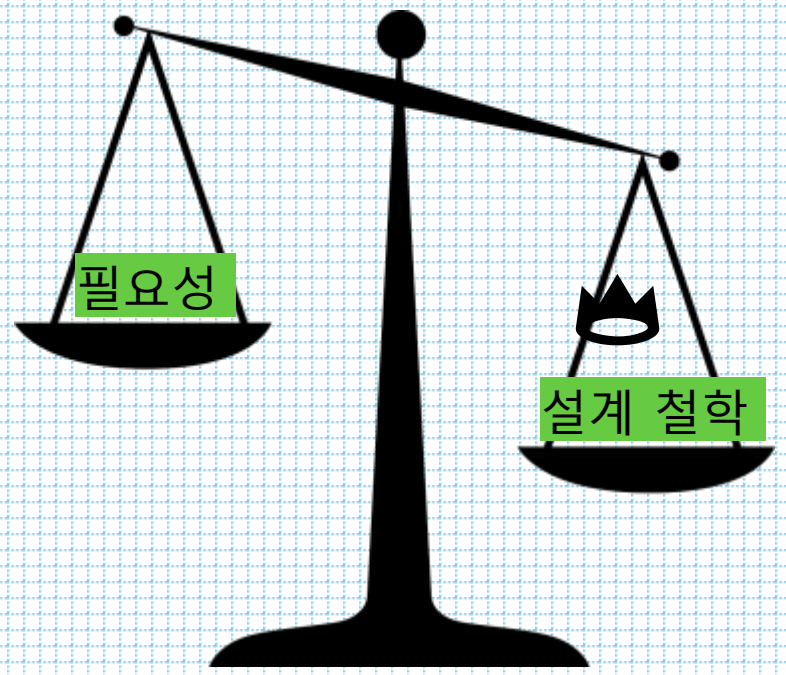
인터페이스가 변경 가능하다면?
➡기존 시스템이 깨질 위험이 존재

자바의 설계 철학: 보수적, 안정성 중시
➡한 번 정의된 인터페이스는 **불변**한다.
즉, 현재의 인터페이스에 새로운 메서드가 추가될 일이 영원히 없다.

Part 1

인터페이스의 설계와 배경

새로운 메서드를 추가하려면..?



하위 호환성 이슈 해결책

1. 새로운 인터페이스를 설계한다.
2. AbstractList와 같은 추상 클래스를 사용한다.

→ But, 유연성이 떨어진다!

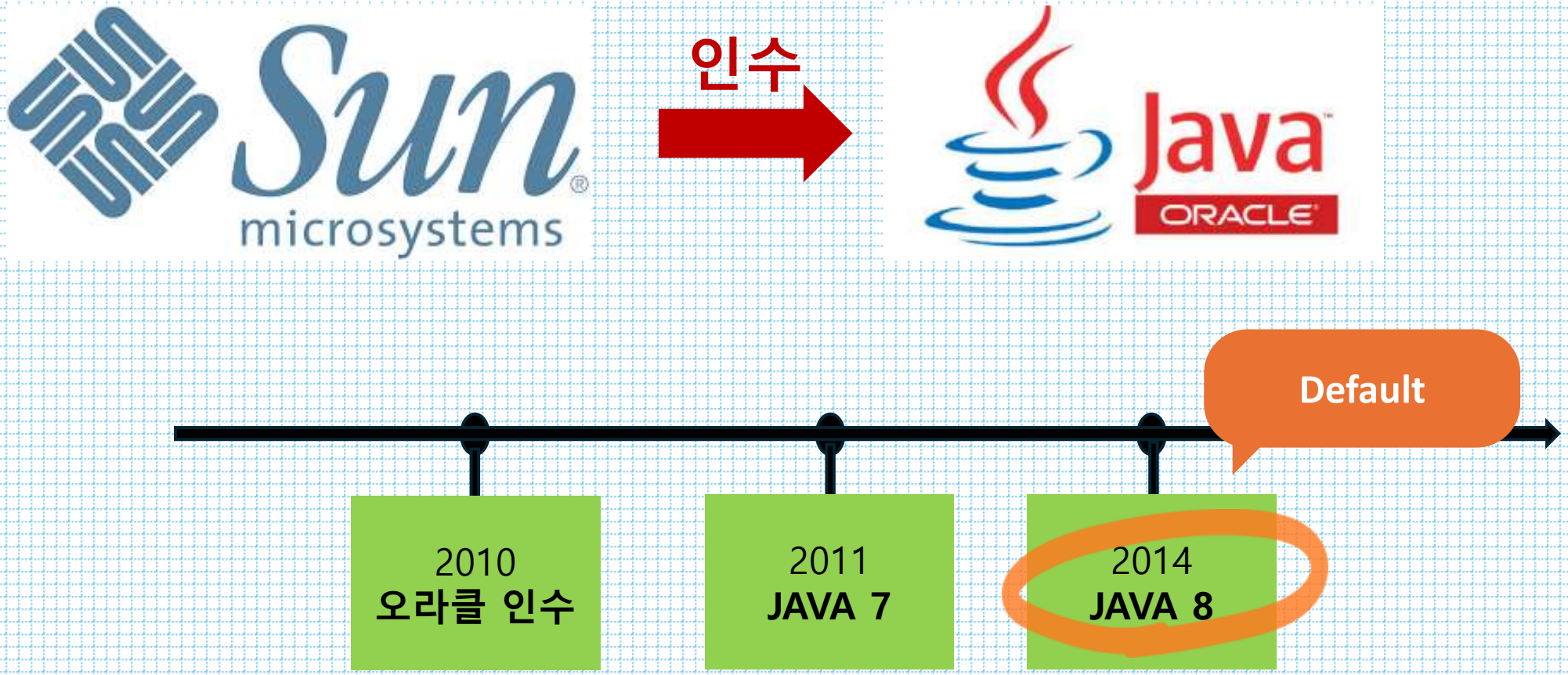
자바 8에서는...

Part 2

디폴트 메서드의 도입

Part 2

디폴트 메서드의 도입



Part 2

디폴트 메서드의 도입

Stream, Lambda

컬렉션 종류	구현체
List	ArrayList, LinkedList, Vector, Stack
Set	HashSet, LinkedHashSet, TreeSet
Queue	PriorityQueue, LinkedList, ArrayDeque
Map	HashMap, LinkedHashMap, TreeMap, Hashtable

→□이미 존재하는 구현체가 너무 많다

수정하지 않고 기능을 추가해야 한다!

Part 2

디폴트 메서드의 도입

```java
public interface Collection<E> {

    // 기본적인 메서드들
    boolean add(E e);
    boolean remove(Object o);
    int size();
    boolean isEmpty();
    void clear();
    boolean contains(Object o);
    Iterator<E> iterator();
}
// 자바 8에서 추가된 default 메서드
default void forEach(Consumer<? super E> action) {
    Objects.requireNonNull(action);
    for (E e : this) {
        action.accept(e);
    }
}
```

구현 없이
새로운 기능추가!

직접 iterator 사용!

```java
Collection<String> collection = new ArrayList<>();
collection.add("one");
collection.add("two");

Iterator<String> iterator = collection.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

```java
Collection<String> collection = new ArrayList<>();
collection.add("one");
collection.add("two");

// 자바 8에서 추가된 forEach 메서드 사용
collection.forEach(item -> System.out.println(item));
```

Part 2

디폴트 메서드의 도입

설계 철학은...?

'Trade Off'

기술과 요구사항이 변하게 되면서
새로운 방식을 필요로 하게 되었다.

Part 3

디폴트 메서드 도입으로 인한 문제

Part 3

디폴트 메서드 도입으로 인한 문제

자바 라이브러리의 디폴트 메서드는 코드 품질이 높고 범용적
➡하지만 모든 상황에서 잘 작동한다고 보장❌

디폴트 메서드는 그저 삽입될 뿐, 구현체에 대해 모름

```
default boolean removeIf(Predicate<? super E> filter) {
    Objects.requireNonNull(filter);
    boolean removed = false;
    for (Iterator<E> it = iterator(); it.hasNext(); ) {
        if (filter.test(it.next())) {
            it.remove();
            removed = true;
        }
    }
    return removed;
}
```

현존하는 모든 Collection 구현체와
잘 어우러질까..?

Part 3

디폴트 메서드 도입으로 인한 문제

java.util의 Collections.synchronizedCollection() ?

```
class Counter {
    private int count = 0;

    public void increment() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

여러 스레드가 동시에 increment()를 실행하면 count++ 연산이 올바르게 실행될까..?

A 스레드: count = 5 + 1 = 6
B 스레드: count = 5 + 1 = 6

ㄴ덮어 씌워짐!

기댓값: count = 7
결과값: count = 6

- 여러 개의 스레드가 같은 데이터(공유 자원)를 동시에 수정하면 데이터가 꼬이거나, 원하지 않는 결과가 나온다.
- **동기화(Synchronization)** 는 이런 문제를 막기 위해 한 번에 하나의 스레드만 특정 코드를 실행하도록 제한하는 것

Part 3

디폴트 메서드 도입으로 인한 문제

```java
class Counter {
    private int count = 0;

    public void increment() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

```java
class Counter {
    private int count = 0;

    public synchronized void increment() {
        count++;  // 동기화된 메서드: 하나의 스레드만 실행 가능
    }

    public int getCount() {
        return count;
    }
}
```

synchronized 키워드는
해당 객체의 락을 가져와서(lock) 실행

Part 3

디폴트 메서드 도입으로 인한 문제

```java
public static <T> Collection<T> synchronizedCollection(Collection<T> c) {
    return new SynchronizedCollection<>(c);
}

static class SynchronizedCollection<E> implements Collection<E>, Serializable {
    final Collection<E> c;  // 원본 컬렉션
    final Object mutex;     // 🔥 동기화에 사용할 락 객체

    SynchronizedCollection(Collection<E> c) {
        this.c = Objects.requireNonNull(c);
        this.mutex = this; // 기본적으로 자기 자신을 락으로 사용
    }

    SynchronizedCollection(Collection<E> c, Object mutex) {
        this.c = Objects.requireNonNull(c);
        this.mutex = mutex; // 🔥 사용자 지정 락 가능
    }

    public int size() {
        synchronized (mutex) { return c.size(); }
    }
```

기존 인터페이스의 메서드는 구현됨!

```java
public interface Collection<E> {

    // 기본적인 메서드들
    boolean add(E e);
    boolean remove(Object o);
    int size();
    boolean isEmpty();
    void clear();
    boolean contains(Object o);
    Iterator<E> iterator();
}
```

Part 3

디폴트 메서드 도입으로 인한 문제

```
public int size() {
    synchronized (mutex) { return c.size(); }
}
```

removeIf 메서드는
동기화에 대하여 모른다...!

```
default boolean removeIf(Predicate<? super E> filter) {
    Objects.requireNonNull(filter);
    boolean removed = false;
    for (Iterator<E> it = iterator(); it.hasNext(); ) {
        if (filter.test(it.next())) {
            it.remove();
            removed = true;
        }
    }
    return removed;
}
```

만약 여러 스레드가 공유하는 컬렉션을 다루는 상황에서 synchronizedCollection을 사용중이고, removeIf 메서드를 호출한다면..?

➡ConcurrentModificationException 발생

Part 3

디폴트 메서드 도입으로 인한 문제

1 java.util의 Collections.synchronizedCollection

```
Collection<String> syncCollection = Collections.synchronizedCollection(new ArrayList<>());

synchronized(syncCollection) {
    syncCollection.removeIf(e -> e.equals("A"));
}
```

명시적 동기화

2 아파치 커먼즈 라이브러리의 collection.SynchronizedCollection

```
@Override
public synchronized boolean removeIf(Predicate<? super E> filter) {
    return super.removeIf(filter);
}
```

오버라이드

→□디폴트 메서드를 호출하기 전에 필요한 작업 진행
→□구현한 인터페이스의 디폴트 메서드를 재정의

But, 제3의 기존 컬렉션 구현체들은 수정될 기회가 없었고 여전히 수정✕

Part 4

인터페이스를 설계할 때

Part 4

인터페이스를 설계할 때

기존의 인터페이스에 메서드를 추가하는 경우

1. 반드시 필요하지 않으면 기존 인터페이스에 디폴트 메서드로 새 메서드를 추가하지 않는다.

2. 추가하려는 디폴트 메서드가 기존 구현체와 충돌하는 지 확인한다.

새로운 인터페이스를 만드는 경우

디폴트 메서드는 표준적인 메서드 구현을 제공하는 데 유용한 수단으로,
인터페이스를 쉽게 구현해 활용하게 해준다.

But, 기존 클래스와의 호환성을 고려해서 활용해야 한다.

디폴트 메서드는

- 인터페이스로부터 메서드 제거 용도❌
- 기존 메서드의 시그니처를 수정하는 용도❌

Part 4

인터페이스를 설계할 때

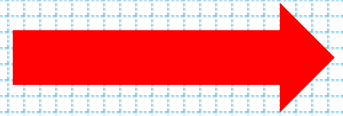
새로운 인터페이스라면...?
→릴리스 전에 반드시 테스트 할 것!

```java
interface CoffeeMaker {
    void brewCoffee();
}
```

```java
class DripCoffee implements CoffeeMaker {
    @Override
    public void brewCoffee() {
        System.out.println("Dripping coffee...");
    }
}
```

```java
class Espresso implements CoffeeMaker {
    @Override
    public void brewCoffee() {
        System.out.println("Brewing espresso...");
    }
}
```

```java
class AeroPress implements CoffeeMaker {
    @Override
    public void brewCoffee() {
        System.out.println("Making AeroPress coffee...");
    }
}
```



```java
class Cafe {
    private CoffeeMaker coffeeMaker;

    public Cafe(CoffeeMaker coffeeMaker) {
        this.coffeeMaker = coffeeMaker;
    }

    public void serveCoffee() {
        System.out.println("Serving coffee at Cafe...");
        coffeeMaker.brewCoffee();
    }
}

public class Main {
    public static void main(String[] args) {
        CoffeeMaker dripCoffee = new DripCoffee();
        CoffeeMaker espresso = new Espresso();
        CoffeeMaker aeroPress = new AeroPress();

        Cafe cafe = new Cafe(dripCoffee);
        cafe.serveCoffee();

        cafe = new Cafe(espresso);
        cafe.serveCoffee();

        cafe = new Cafe(aeroPress);
        cafe.serveCoffee();
    }
}
```

summary

1

- 한번 정의된 인터페이스는 불변
- 하위 호환성 이슈 해결책은
  - 새로운 인터페이스 설계
  - 추상 클래스 사용

2

- 자바 8에 Stream과 lambda 추가
- 수정 없이 신기능 추가 필요
- 자바의 설계 철학과 Trade Off

3

- 디폴트 메서드는 삽입될 뿐 구현체에 대해 모름
- 모든 상황에 디폴트 메서드가 적용❌
- 해결책도 자바 플랫폼 라이브러리에 한함.

4

- 디폴트 메서드를 사용할 때 신중할 것
- 릴리스 전 반드시 테스트 할 것
- 릴리스 전에 결함을 찾아내는 게 최선

Thank you