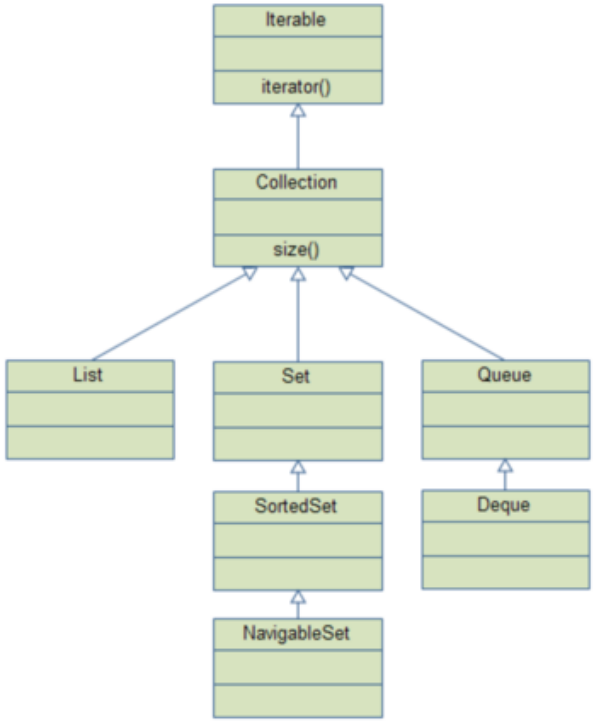
이펙티브 자바

Item 64 ~ 68

2024/06/25

발표자: 박지원

Item 64 객체는 인터페이스를 사용해 참조하라




```java
public interface Iterable<T> {
    /**
     * Returns an iterator over elements of type {@code T}.
     *
     * @return an Iterator.
     */
    Iterator<T> iterator();
}
```

왜 iterator를 상속하지 않았을까?

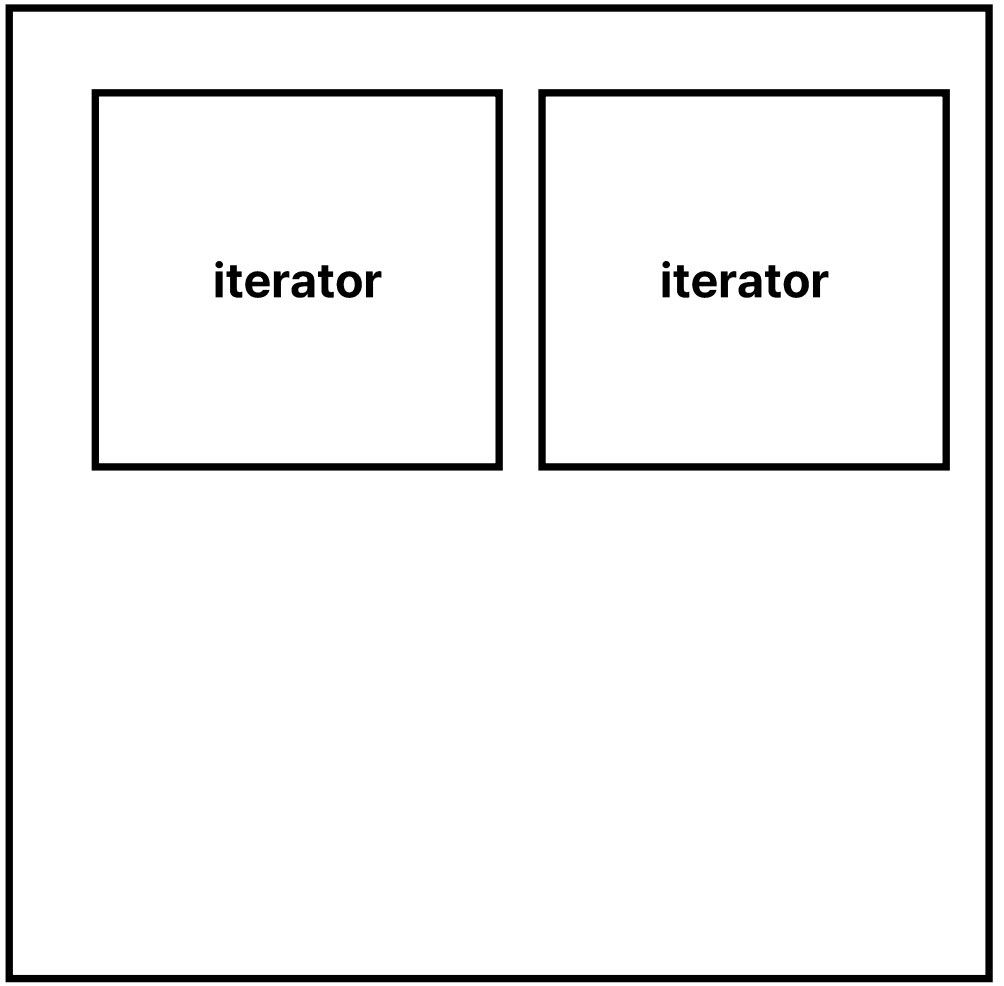
Item 64 객체는 인터페이스를 사용해 참조하라

```
@Override
boolean hasNext();

@Override
default void remove() {}

@Override
default void forEachRemaining() {}
```

VS



발표자: 박지원

Item 64 객체는 인터페이스를 사용해 참조하라

Iterable은 Iterator메소드의 구현을 강제한다.

Iterator는 컴포넌트

```
final class LinkedKeyIterator extends LinkedHashIterator
    implements Iterator<K> {
    public final K next() { return nextNode().getKey(); }
}

final class LinkedValueIterator extends LinkedHashIterator
    implements Iterator<V> {
    public final V next() { return nextNode().value; }
}

final class LinkedEntryIterator extends LinkedHashIterator
    implements Iterator<Map.Entry<K,V>> {
    public final Map.Entry<K,V> next() { return nextNode();
}
```

발표자: 박지원

Item 64 객체는 인터페이스를 사용해 참조하라

226

An iterator is stateful. The idea is that if you call `Iterable.iterator()` twice you'll get *independent* iterators - for most iterables, anyway. That clearly wouldn't be the case in your scenario.

For example, I can usually write:

```
public void iterateOver(Iterable<String> strings)
{
    for (String x : strings)
    {
         System.out.println(x);
    }
    for (String x : strings)
    {
         System.out.println(x);
    }
}
```

That should print the collection twice - but with your scheme the second loop wo
terminate instantly.

```
Set<Integer> set=new HashSet<>();
for(int i=1;i<=10;i++)
    set.add(i);
Iterator<Integer> iterator1 = set.iterator();
while(iterator1.hasNext()){
    Integer next = iterator1.next();
    System.out.println("next1 = " + next);
}
while(iterator1.hasNext()){
    Integer next = iterator1.next();
    System.out.println("next1 = " + next);
}

Iterator<Integer> iterator2 = set.iterator();
while(iterator2.hasNext()){
    Integer next = iterator2.next();
    System.out.println("next2 = " + next);
}
```

발표자: 박지원

Item 64 객체는 인터페이스를 사용해 참조하라

```
Map<Integer, String> map = new LinkedHashMap<>();

/**
 * push element
 */

Iterator<Map.Entry<Integer, String>> iterator1 = map.entrySet().iterator();
while (iterator1.hasNext()) {
    System.out.println("iterator1.next() = " + iterator1.next());
}

Iterator<Integer> iterator2 = map.keySet().iterator();
while (iterator2.hasNext()) {
    System.out.println("iterator2.next() = " + iterator2.next());
}
```

발표자: 박지원

Item 64 객체는 인터페이스를 사용해 참조하라

적합한 인터페이스가 있다면 인터페이스를 선언하자

이미 관례적으로 사용되고 있다.

```
// 좋은 예. 인터페이스를 타입으로 사용했다.
Set<Son> sonSet = new LinkedHashSet<>();

// 나쁜 예. 클래스를 타입으로 사용했다.
LinkedHashSet<Son> sonSet = new LinkedHashSet<>();
```

발표자: 박지원

Item 64 객체는 인터페이스를 사용해 참조하라

원래의 클래스가 인터페이스의 일반 규약 이외에 특별한 기능을 제공하며, 주변 코드가 해당 기능에 기대어 동작한다면, 새로운 클래스도 동일한 기능을 제공해야 한다.

대표적으로 순서를 보장하는 경우(Linked)

유지보수 측면

발표자: 박지원

Item 64 객체는 인터페이스를 사용해 참조하라

적합한 인터페이스가 없다면 당연히 클래스를 참조하자

String,BigInteger

단일 값을 가지는 필드는 클래스를 참조하자

자료구조는 인터페이스를 참조하자

발표자: 박지원

Item 64 객체는 인터페이스를 사용해 참조하라

Queue 구현체로 LinkedList를 사용한다

PriortyQueue를 Queue로 선언하면 안될까?(LIFO 위배)

```
Queue<Integer> queue1=new LinkedList<>();
Queue<Integer> queue2=new PriorityQueue<>(Collections.reverseOrder());
```

발표자: 박지원

Item 65 리플렉션 보다는 인터페이스를 사용해라

리플렉션은 접근자의 제한없이 클래스와 필드에 접근가능하다.

1. 컴파일 타임의 에러가 모두 런타임으로 넘어간다.
2. 구현양이 적지 않다. 유지보수 측면
3. 비용이 높은 API

스프링에서 주입과 관련된 로직 처리

Item 65 리플렉션 보다는 인터페이스를 사용해라

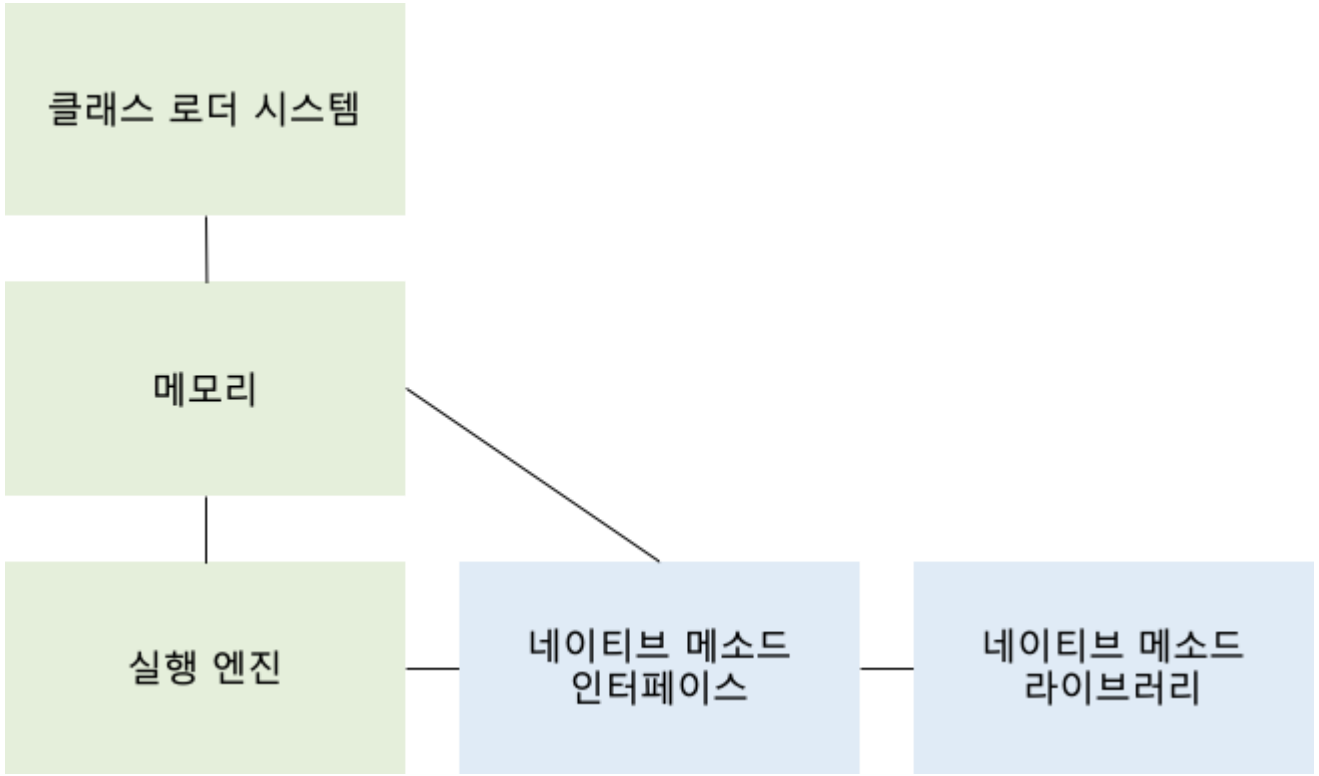
전부 처리...

```
try {
    s = cons.newInstance();
} catch (InvocationTargetException e) {
    // 생성자가 예외를 던졌을 때
    e.printStackTrace();
} catch (InstantiationException e) {
    // 클래스를 인스턴스화할 수 없을 때
    e.printStackTrace();
} catch (IllegalAccessException e) {
    // 생성자에 접근할 수 없을 때
    e.printStackTrace();
} catch (ClassCastException e) {
    // Set을 구현하지 않은 클래스일 때
    e.printStackTrace();
}
```

Item 66 네이티브 메서드는 신중히 사용하라

Thread.currentThread

사용할 수 있겠나..



발표자: 박지원

Item 67 최적화는 신중히하라

성능이 좋은 프로그램 < 안정적인 프로그램

성능을 제한하는 설계를 피하라(프로토콜, 데이터 포맷)

성능을 위해 API를 왜곡해선 안된다.

Item 68 일반적으로 통용되는 명명 규칙을 따르라

철자 규칙

com.springproject (역전구조, 점으로 구분)

각 요소는 8자이하(한시적 약어 혀용 awt)

Item 68 일반적으로 통용되는 명명 규칙을 따르라

T: 임의 타입

E: 컬렉션 원소

K, V: 맵의 키와 값

X: 예외

R: 반환 타입

Item 68 일반적으로 통용되는 명명 규칙을 따르라

객체를 생성할 수 없는 경우 복수형 명사(Collections)

메서드는 동사구

타입변환 메소드 toType

객체의 뷰 변환 asType (asList)

발표자: 박지원