

# Patterns : Iterator

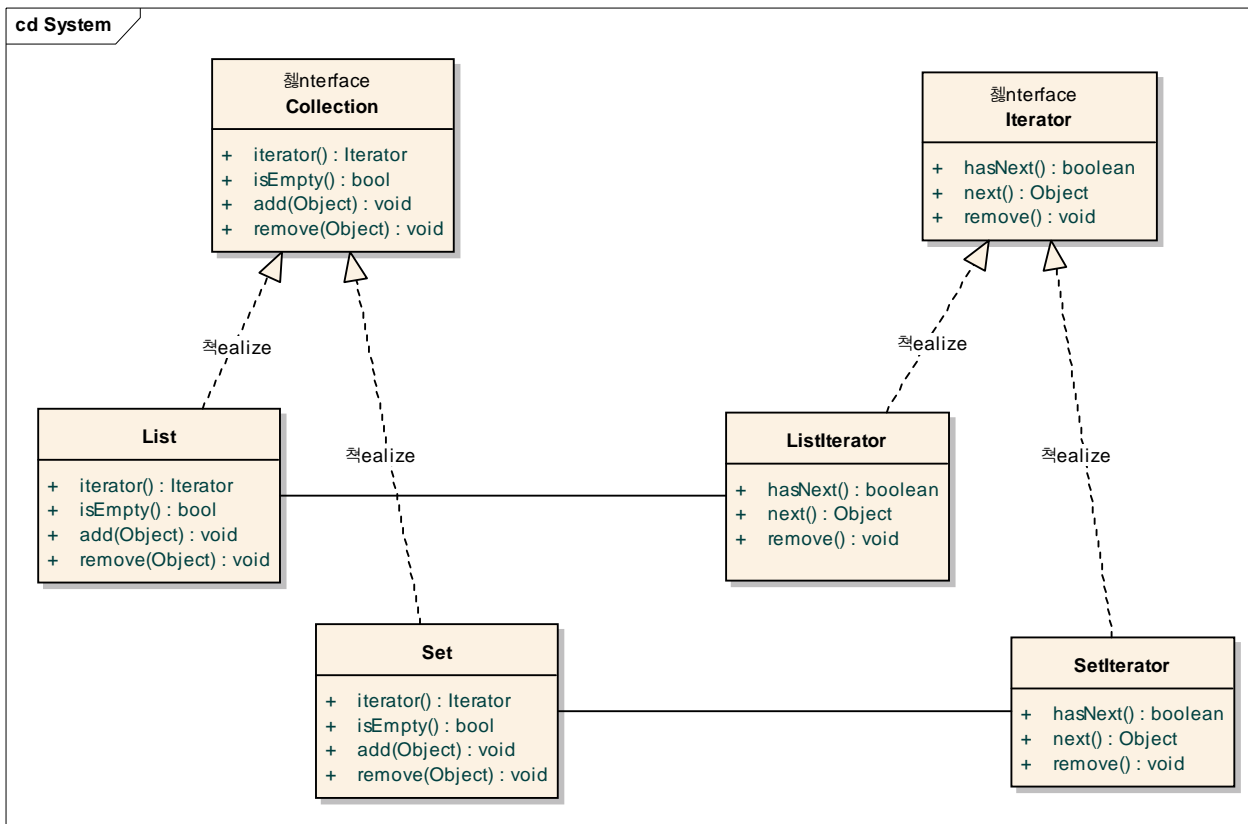
Byungjoon Lee@NCP Team.ETRI

[bjlee@etri.re.kr](mailto:bjlee@etri.re.kr)

<http://www.buggymind.com>

# Iterator Pattern

- One of the most simple design pattern



<< The Most widely-known "External" Iterator Pattern >>

# Requirements

- Make the client use collection objects in a consistent way
- Does below function needs to be modified when a new type of collection object is passed as an argument? (*No, of course*)

```
public void foo(Collection c) {  
    for ( Iterator i = c.iterator(); i.hasNext(); ) {  
        Bar bar = (Bar) i.next();  
        System.out.println("bar = " + bar.toString());  
    }  
}
```

# How to use Iterator Pattern

1. Make a concrete collection class by implementing the *Collection* interface
  - For example, *MyCollection*
2. Make a concrete iterator class by implementing the *Iterator* interface
  - For example, *MyIterator*
3. Modify the code of `MyCollection.iterator()` to return the `MyIterator` object

```
public class MyCollection {  
    ...  
    public Iterator iterator() {  
        return new MyIterator(this);  
    }  
    ...  
}
```

# How to Implement Iterator Class

- Considerations
  - Where to put iteration logics
    - In the collection class?
      - In this case, only location information is maintained by iterator class
    - In the iterator class?
      - In this class, the implementation of the iterator class will not be independent of the collection class codes
  - What if a modification to the items saved within a collection object is done?
    - Iterator invalidation is needed?
    - “Robust Iterator”
      - Iterator survives the modification on a collection