반복의 추상화

강 지 훈
jhkang@cnu.ac.kr

충남대학교 컴퓨터공학과

반복의 추상화

사용자가 리스트를 스캔 할 필요가 있다면?

구현과 무관하게:

- 리스트의 구현 방법을 몰라도, 아래의 표현이 가능한가?
  - 배열을 사용했는지, 아니면 연결리스트를 사용했는지?

```
int  pass = 0 ;
int  x = 0 ;
while ( x < list._size ) {
    if ( list._elements[x].score() >= 60 )
        pass++ ;
    x++ ;
}
```

```
int  pass = 0 ;
Node  x = list._head ;
while ( x != null ) {
    if ( x.element().score() >= 60 )
        pass++ ;
    x = x.next() ;
}
```

- 사용자는 차례대로 리스트의 원소의 값을 알고 싶을 뿐!

코드의 어느 부분이 문제일까?

- Capsule을 무력화 시킨 부분은?
  - List 객체 사용자가 List 객체 내부의 private instance variable에 접근!

```
int  pass = 0 ;
int  x = 0 ;
while ( x < list._size ) {
    if ( list._elements[x].score() >= 60 )
        pass++ ;
    x++ ;
}
```

```
int  pass = 0 ;
Node  x = list._head ;
while ( x != null ) {
    if ( x.element().score() >= 60 )
        pass++ ;
    x = x.next() ;
}
```

사용자는 이렇게 하고 싶을 뿐이다!

```
Element    e ;
List       list ; // 사용자는 이 list가 어떻게 구현되어 있는지와는 무관하게
                  // 리스트의 원소를 스캔 하려고 한다.
Iterator   iterator ; // Iterator는 반복을 추상화 한, 반복을 위한 class

int  pass = 0 ;
iterator = list.iterator() ;
while ( iterator.hasNextElement() ) {
    // 리스트의 원소를 얻어내어 사용
    e = iterator.nextElement() ;
    if ( e.score() >= 60 ) {
        pass++ ;
    }
}
```

```
int  pass = 0 ;
int  i = 0 ;
while ( i < list._size ) {
    if ( list._elements[i].score() >= 60 )
        pass++ ;
    i++ ;
}
```

```
int  pass = 0 ;
Node  x = list._head ;
while ( x != null ) {
    if ( x.element().score() >= 60 )
        pass++ ;
    x = x.next() ;
}
```

반복의 추상화

- 반복을 구현과 무관하게
  - 리스트의 원소들에 대한 순차 검색을, 구현에 독립적으로 실행
- 반복자 (Iterator)를 class로 정의
- 반복이 필요할 때마다 반복자 객체를 생성하여 사용

반복자 구현: 내부 클래스로

- 리스트를 위한 반복자를 효율적으로 구현하기 위해서는 리스트의 인스턴스 변수들에게 직접 접근할 수 있어야 한다.
- 하나의 리스트에 다른 목적의 여러 개의 반복자 객체를 둘 수 있을 필요가 있다.

- 반복자는 리스트 클래스의 내부 클래스 (inner class)로 선언한다.

Class "LinkedList<Element>.Iterator"

Class LinkedList⟨Element⟩.Iterator의 공개 함수

- public boolean hasNextElement() ;
  - 리스트의 다음 원소가 존재하는지를 알아낸다
- public Element nextElement() ;
  - 리스트의 다음 원소를 얻어낸다. 없으면 null을 얻는다.

생성자는 비공개 함수

- 생성자는?
  - private Iterator () ;
- LinkedList에 추가로 필요한 공개함수
  - public Iterator iterator() ;
    - 반복자를 얻어낸다.

LinkedList의 내부 클래스로 선언

```
pubic class LinkedList<Element>
{
    // LinkedList의 선언
    ……

    // Iterator 생성하여 얻기
    public Iterator iterator()
    {
        return  new Iterator() ;
    }

    // Inner Class "Iterator"의 선언
    public class Iterator
    {
        // 인스턴스 변수들
        ……
        private          Iterator () ; // 생성자
        public boolean   hasNextElement() ; // 다음 원소가 존재하는지를 알아낸다
        public Element   nextElement() ; // 다음 원소를 얻어낸다. 없으면 null을 얻는다.
    } // End of Iterator

} // End of LinkedList
```

연결 리스트를 위한 내부 클래스 Iterator 의 구현

LinkedList⟨Element⟩.Iterator: 인스턴스 변수

- 인스턴스 변수들

```
pubic class Iterator
{
    private Node    _nextNode ;
                    // 연결 체인에서 다음 원소를 소유하고 있는 노드
```

LinkedList⟨Element⟩.Iterator: 함수의 구현

```
private  Iterator () // 생성자
{
    this._nextNode = _head ;
}

public boolean  hasNextElement ()
{
    return (this._nextNode != null) ;
}

public Element  nextElement ()
{
    if (this._nextNode == null) {
        return null ;
    }
    else {
        Element e = this._nextNode.element() ;
        this._nextNode = this._nextNode.next() ;
        return  e ;
    }
}
```

```
Element e ;
int  pass = 0 ;
x = list._head ; // 생성자 Iterator()로
while ( x != null ) { // hasNextElement()로
      e = x.element() ; // nextElement()로
      if ( e.score() >= 60 )
            pass++ ;
      x = x.next() ; // nextElement()로
}
```

```
LinkedList<Element> scoreList ;
scoreList = new LinkedList() ;
.....

LinkedList<Element>.Iterator  iterator ;
iterator = scoreList.iterator() ;
Element e ;
int  pass = 0 ;
while ( iterator.hasNextElement() ) {
      e = iterator.nextElement() ;
      if ( e.score() >= 60 )
            pass++ ;
}
```

Class "ArrayList<Element>.Iterator"

Class ArrayList⟨Element⟩.Iterator의 공개 함수

- private Iterator() ;
  - 생성자
- public boolean hasNextElement() ;
  - 다음 원소가 존재하는지를 알아낸다
- public Element nextElement() ;
  - 다음 원소를 얻어낸다. 없으면 null을 얻는다.

ArrayList의 내부 클래스로 선언

```
pubic class ArrayList<Element>
{
    // ArrayList의 선언
    ……

    // ListIterator 생성하여 얻기
    public Iterator iterator()
    {
        return  new Iterator() ;
    }

    // Inner Class Iterator의 선언
    public class Iterator
    {
        // 인스턴스 변수들
        ……
        private            Iterator() ; // 생성자
        public boolean  hasNextElement() ; //다음 원소가 존재하는지를 알아낸다
        public Element  nextElement() ; // 다음 원소를 얻어낸다. 없으면 null을 얻는다.
    } // End of Iterator

} // End of ArrayList
```

배열 리스트를 위한 ArrayList<Element>.Iterator 의 구현

ArrayList⟨Element⟩.Iterator: 인스턴스 변수

- 인스턴스 변수들

```
private   class Iterator
{
    private int        _nextPosition ; // 배열에서의 다음 원소 위치
```

ArrayList⟨T⟩.Iterator: 함수의 구현

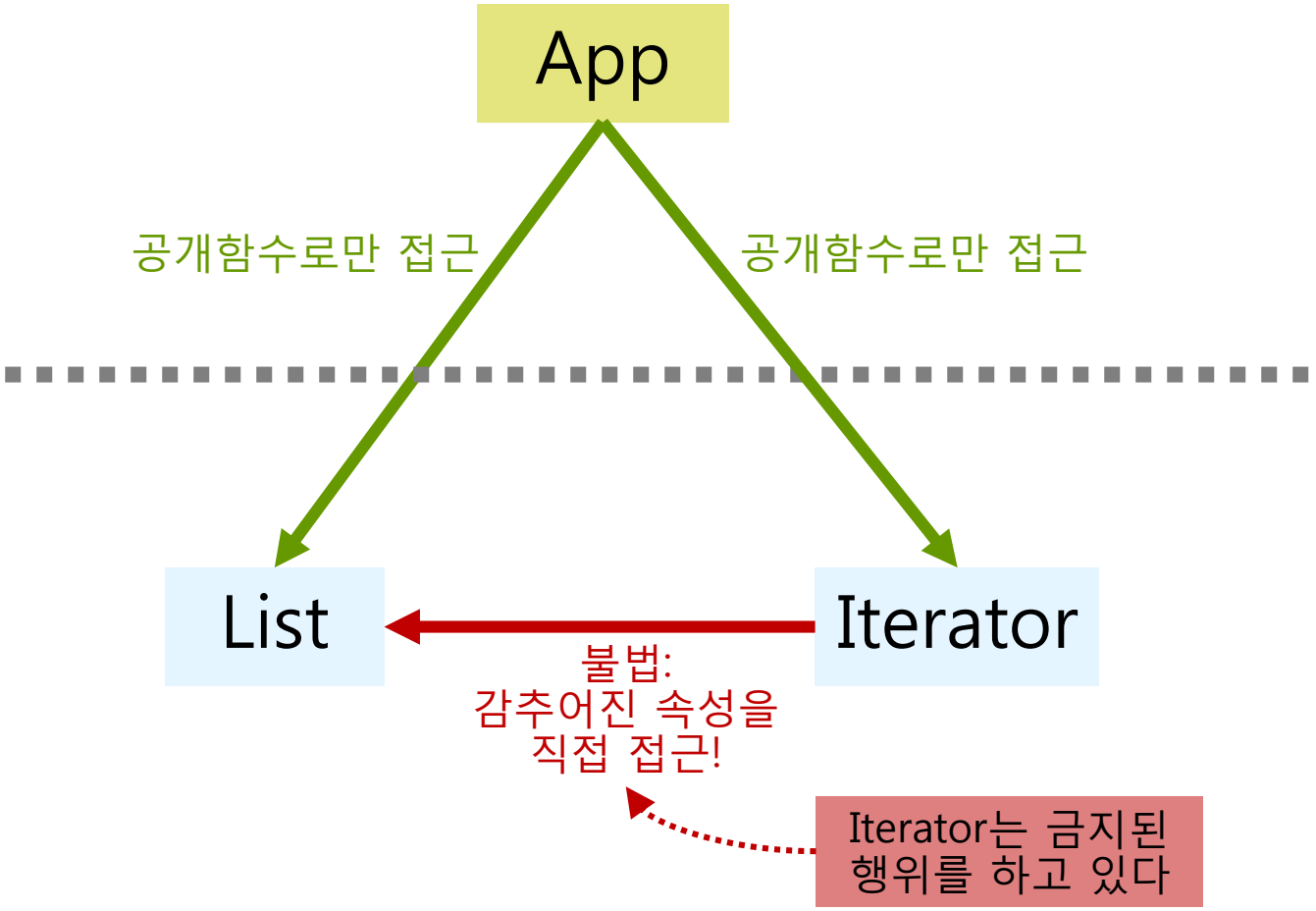
```
private  Iterator ()
{
    this._nextPosition = 0 ;
}

public boolean  hasNextElement ()
{
    return (this._nextPosition < ArrayList.this._size) ;
}

public Element  nextElement ()
{
    if ( this._nextPosition == ArrayList.this._size ) {
        return null ;
    }
    else {
        Element  nextElement =
            ArrayList.this._elements[this._nextPosition] ;
        this._nextPosition++;
        return  nextElement ;
    }
}
```

```
Element  e = null ;
int  pass = 0 ;
i = 0 ; // 생성자 Iterator()로
while ( i < list._size) {
                    // hasNextElement()로
      e = list._elements[i] ; // nextElement()로
      if ( e.score() >= 60 )
            pass++ ;
      i++; // nextElement()로
}
```

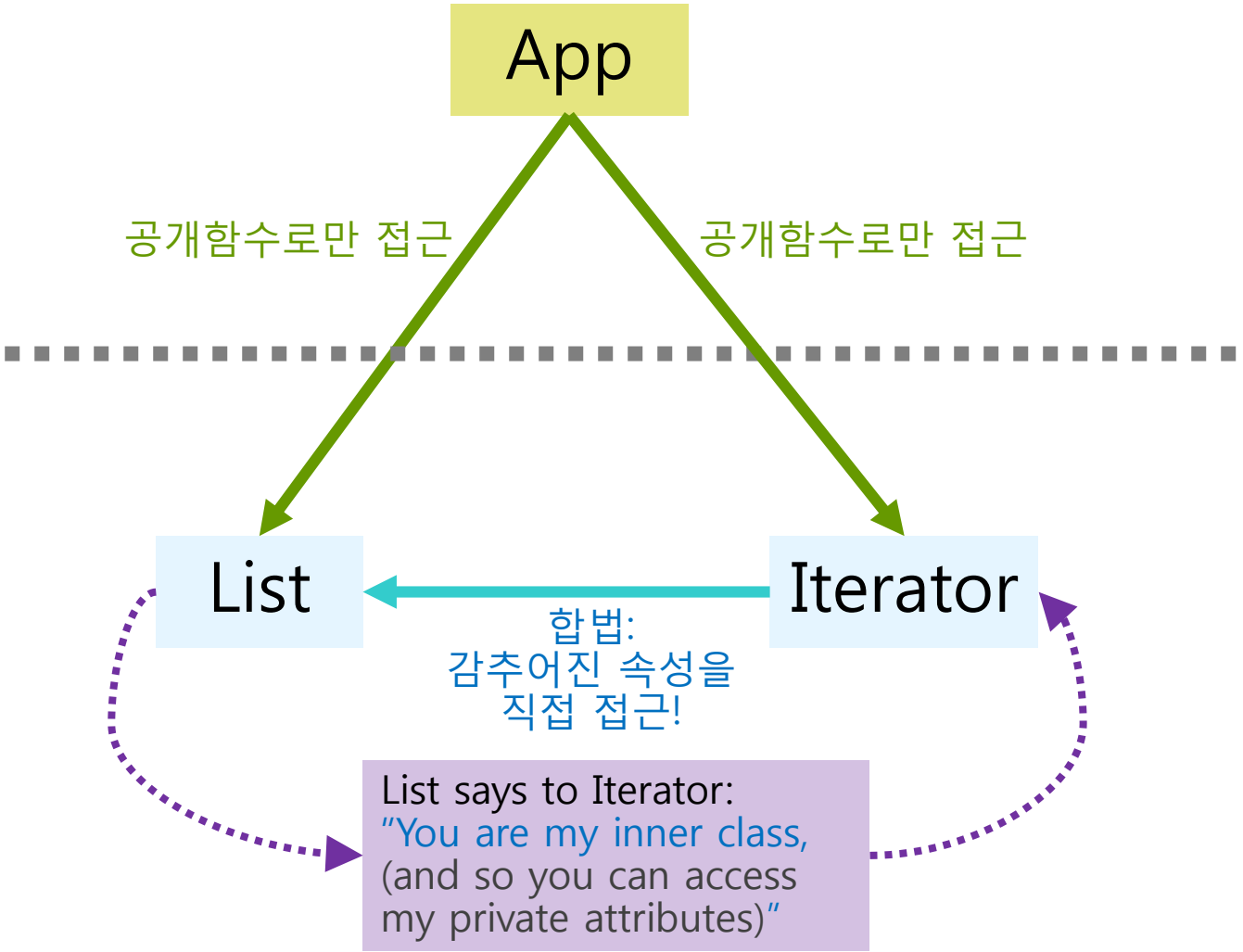
```
ArrayList<Element> scoreList ;
scoreList = new List() ;
.....

ArrayList<Element>.Iterator  iterator ;
iterator = scoreList.iterator() ;
Element  e = null ;
int  pass = 0 ;
while ( iterator.hasNextElement() ) {
      e = iterator.nextElement() ;
      if ( e.score() >= 60 )
            pass++ ;
}
```

반복자는 왜 리스트의 내부 클래스로?

반복자는 왜 리스트의 내부 클래스로?



반복자는 왜 리스트의 내부 클래스로?



해결책: C

- Iterator를 구현하기 위해서는, 어쩔 수 없이, List의 감추어진(private) 속성을 접근해야 한다.
  - 이것은 감추어진 속성의 정의에 어긋나는 행위이다.
- C 프로그램에서는 이것을 통제할 아무런 수단이 없다.
  - 컴파일러의 도움을 받을 수 없다.
  - 지금까지 객체지향적 방법의 구현에 관해서 그랬듯이, 프로그램 작성자가 알아서 통제한다.
  - 구현자의 관점에서, "Iterator의 구현은 private 속성을 직접 접근할 수 밖에 없는 예외적인 경우"이다.

해결책: C++

- C++ 프로그램에서는 컴파일러의 도움을 받을 수 있다.
  - List는 Iterator를 friend class 로 선언하면 된다.
  - List의 속성 중에서 friend class인 Iterator가 접근하게 되는 속성은 protected 로 선언한다.

해결책: Java

- Java 에서도 컴파일러의 도움을 받을 수 있다
- Iterator를 List의 내부 클래스로 선언
  - List의 감추어진(private) 인스턴스 변수들을 사용할 수 있으므로, 효율적인 구현이 가능
  - 하나의 리스트에, 필요에 따라 동시에 여러 개의 반복자 객체를 생성할 수 있다.

Interface "ListIterator"

반복자(Iterator)는 왜 인터페이스로?

- 배열리스트에서의 반복자를 위한 공개함수와 연결리스트에서의 반복자를 위한 반복자의 공개함수들은 동일
- 동일한 의미의 공통되는 공개함수를 인터페이스로 선언
  - 동일한 기능으로서 무엇이 필요한지를 미리 정의
    - 인터페이스가 추가된 클래스에 어떤 함수들을 구현해야만 하는지를 알 수 있다.
  - 따라서, 코드 관리를 효율적으로 할 수 있다

인터페이스 ListIterator의 공개함수

```
pubic interface  ListIterator<Element>
{
    public boolean hasNextElement() ;
        // 다음 원소가 존재하는지를 알아낸다
    public Element  nextElement() ;
        // 다음 원소를 얻어낸다. 없으면 null을 얻는다.
} // End of ListIterator
```

연결리스트에서의 인터페이스 사용과 구현

LinkedList<Element>.Iterator 에서의 인터페이스 사용

```
pubic class LinkedList<Element>
{
    // LinkedList의 선언
    ……

    // Iterator 생성하여 얻기
    public Iterator iterator()
    {
        return  new Iterator() ;
    }

    // Inner Class Iterator의 선언
    private class Iterator implements ListIterator<Element>
    {
        // 인스턴스 변수들
        ……
        private          Iterator () {...} ; // 생성자
        public boolean   hasNextElement() {...} ; //다음 원소가 존재하는지를 알아낸다
        public Element   nextElement() {...} ; // 다음 원소를 얻어낸다. 없으면 null을 얻는다.
    } // End of Iterator

} // End of LinkedList
```

LinkedList⟨Element⟩.Iterator: 함수의 구현

```
private  Iterator ()
{
    this._nextNode = LinkedList.this._head ;
}

public boolean  hasNextElement ()  // 인터페이스의 공개함수 구현
{
    return (this._nextNode != null) ;
}

public Element  nextElement () // 인터페이스의 공개함수 구현
{
    if (this._nextNode == null) {
        return null ;
    }
    else {
        Element  nextElement =
            this._nextNode.element() ;
        this._nextNode = this._nextNode.next() ;
        return  nextElement ;
    }
}
```

```
LinkedList<Element> scoreList ;
scoreList = new LinkedList() ;
.....

LinkedList<Element>.Iterator  iterator ;
iterator = scoreList.iterator() ;
Element element ;
pass = 0 ;
while ( iterator.hasNextElement() ) {
    element = iterator.nextElement() ;
    if ( element.score() >= 60 )
        pass++ ;
}
```

배열리스트에서의 인터페이스 사용과 구현

ArrayList의 내부 클래스로 선언

```
pubic class ArrayList<Element>
{
    // ArrayList의 선언
    ......

    // Iterator 생성하여 얻기
    public Iterator iterator()
    {
        return  new Iterator() ;
    }

    // Class Iterator의 선언
    private class Iterator implements ListIterator<Element>
    {
        // 인스턴스 변수들
        ......
        private          Iterator () {...}; // 생성자
        public boolean   hasNextElement() {...} ; //다음 원소가 존재하는지를 알아낸다
        public Element   nextElement() {...} ; // 다음 원소를 얻어낸다. 없으면 null을 얻는다.
    } // End of Iterator

} // End of ArrayList
```

ArrayList⟨Element⟩.Iterator: 생성자의 구현

```
private  ArrayListIterator ()
{
    this._nextPosition = 0 ;
}

public boolean  hasNextElement ()
{
    return (this._nextPosition < ArrayList.this._size) ;
}

public Element  nextElement ()
{
    if (this._nextPosition == ArrayList.this._size) ) {
        return null ;
    }
    else {
        Element  nextElement =
            ArrayList.this._elements[this._nextPosition] ;
        this._nextPosition++;
        return  nextElement ;
    }
}
```

```
ArrayList<Element> scoreList ;
scoreList = new ArrayList() ;
.....

List<Element>.Iterator  iterator ;
iterator = scoreList.iterator() ;
Element  element ;
pass = 0 ;
while ( iterator.hasNextElement() ) {
      element = iterator.nextElement() ;
      if ( element.score() >= 60 )
            pass++ ;
}
```

"Iterator" [끝]