자료구조: 2022년 1학기 [강의]

반복의 추상화

강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터융합학부

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

- 사용자는, 구현관 무관하게 차례대로 리스트의 원소의 값을 알고 싶을 뿐!

코드의 어느 부분이 문제일까?

- Capsule 을 무력화 시킨 부분은?
  - List 객체 사용자가 List 객체 내부의 private instance variable 에 접근!

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

```
int  pass = 0 ;
LinkedNode  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 list 가 어떻게 구현되어 있는 지와는 무관하게
                  // 리스트의 원소를 스캔 하려고 한다.
Iterator   iterator ; // Iterator는 반복을 추상화 한, 반복을 위한 class
```

```
int  pass = 0 ;
iterator = list.iterator() ;
while ( iterator.hasNext() ) {
    // 리스트의 원소를 얻어내어 사용
    e = iterator.next() ;
    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 ;
LinkedNode  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 class / interface 로 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 객체 를 둘 수 있을 필요가 있다.

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

Class "LinkedList<T>.ListIterator"

Class LinkedList⟨T⟩.ListIterator 의 공개 함수

- public boolean hasNext() ;
  - 리스트의 다음 원소가 존재하는지를 알아낸다
- public T next() ;
  - 리스트의 다음 원소를 얻어낸다. 없으면 null 을 얻는다.

Inner Class 인 ListIterator 의 생성자는 비공개

- 생성자는 비공개:
  - private ListIterator () ;
- LinkedList<T> 에 추가로 필요한 공개함수
  - public ListIterator<T> iterator() ;
    - 반복자를 얻어낸다.

LinkedList〈T〉 의 내부 클래스로 선언

```
pubic class LinkedList<T>
{
    // LinkedList 의 선언
    ……

    // 공개함수 iterator(): 반복자 객체를 생성하여 얻기
    public ListIterator  iterator()
    {
        return  new ListIterator() ; // Inner class 의 생성자를 사용하여 반복자 객체 생성
    }

    // Inner Class "ListIterator"의 선언
    public class  ListIterator
    {
        // 인스턴스 변수들
        ……
        private          ListIterator () {…} ; // 생성자
        public boolean   hasNext() {…} ; // 다음 원소가 존재하는지를 알아낸다
        public T         next() {…} ; // 다음 원소를 얻어낸다. 없으면 null을 얻는다.
    } // End of Inner Class ListIterator

} // End of Outer Class LinkedList
```

연결 리스트를 위한 내부 클래스 ListIterator 의 구현

LinkedList⟨T⟩.ListIterator: 인스턴스 변수

- 인스턴스 변수들

```
pubic class  ListIterator
{
    private LinkedNode<T>  _nextNode ;
                // 연결 체인에서 다음 원소를 소유하고 있는 노드
```

LinkedList⟨T⟩.ListIterator: 함수의 구현

```
public class  ListIterator
{
      private LinkedNode<T>    _nextNode ;
            // 연결 체인에서 다음 원소를 소유하고 있는 노드

      // 생성자
      private  ListIterator () {
            this._nextNode = LinkedList<T>.this._head ;
      }

      // 공개함수
      public boolean  hasNext ()
      {
            return (this._nextNode != null) ;
      }
      public T  next ()
      {
            if ( this._nextNode == null ) {
                  return  null ;
            }
            else {
                  T  e = this._nextNode.element() ;
                  this._nextNode = this._nextNode.next() ;
                  return  e ;
            }
      }
} // End of Inner Class "ListIterator"
```

LinkedList⟨T⟩.ListIterator: 사용 예

```
public class  ListIterator
{
      private LinkedNode<T>    _nextNode ;
            // 연결 체인에서 다음 원소를 소유하고 있는 노드

      // 생성자
      private  ListIterator () {
            this._nextNode = LinkedList<T>.this._head ;
      }

      // 공개함수
      public boolean  hasNext ()
      {
            return (this._nextNode != null) ;
      }
      public T  next ()
      {
            if ( this._nextNode == null ) {
                  return  null ;
            }
            else {
                  T  e = this._nextNode.element() ;
                  this._nextNode = this._nextNode.next() ;
                  return  e ;
            }
      }
} // End of Inner Class "ListIterator"
```

```
Student  s ;
int  pass = 0 ;
LinkedNode<Student> x=studentList._head ;
                  // ListIterator 객체 생성으로
while ( x != null ) { // iterator.hasNext() 로
      s = x.element() ; // iterator.next() 로
      if ( s.score() >= 60 ) {
            pass++ ;
      }
      x = x.next() ; // iterator.next() 로
}
```

```
LinkedList<Student>  studentList
            = new LinkedList<Student>() ;
.....

LinkedList<Student>.ListIterator  iterator
            = studentList.iterator() ;
Student  s = null ;
int  pass = 0 ;
while ( iterator.hasNext () ) {
      s = iterator.next () ;
      if ( s.score() >= 60 ) {
            pass++ ;
      }
}
```

Class "ArrayList<T>.ListIterator"

Class ArrayList⟨T⟩.ListIterator 의 공개 함수

- private ListIterator() ;
  - 생성자
- public boolean hasNext() ;
  - 다음 원소가 존재하는지를 알아낸다
- public T next() ;
  - 다음 원소를 얻어낸다. 없으면 null 을 얻는다.

ArrayList 의 내부 클래스로 선언

```
pubic class  ArrayList<T>
{
    // ArrayList 의 선언
    ......

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

    // Inner Class "ListIterator" 의 선언
    public class ListIterator
    {
        // 인스턴스 변수들
        ......
        private          ListIterator() {...} ; // 생성자
        public boolean   hasNext() {...} ; //다음 원소가 존재하는지를 알아낸다
        public T         next() {...} ; // 다음 원소를 얻어낸다. 없으면 null 을 얻는다.
    } // End of Inner Class ListIterator

} // End of Outer Class ArrayList
```

배열 리스트를 위한 ArrayList<T>.ListIterator 의 구현

ArrayList⟨T⟩.ListIterator: 인스턴스 변수

- 인스턴스 변수들

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

ArrayList⟨T⟩.ListIterator: 함수의 구현

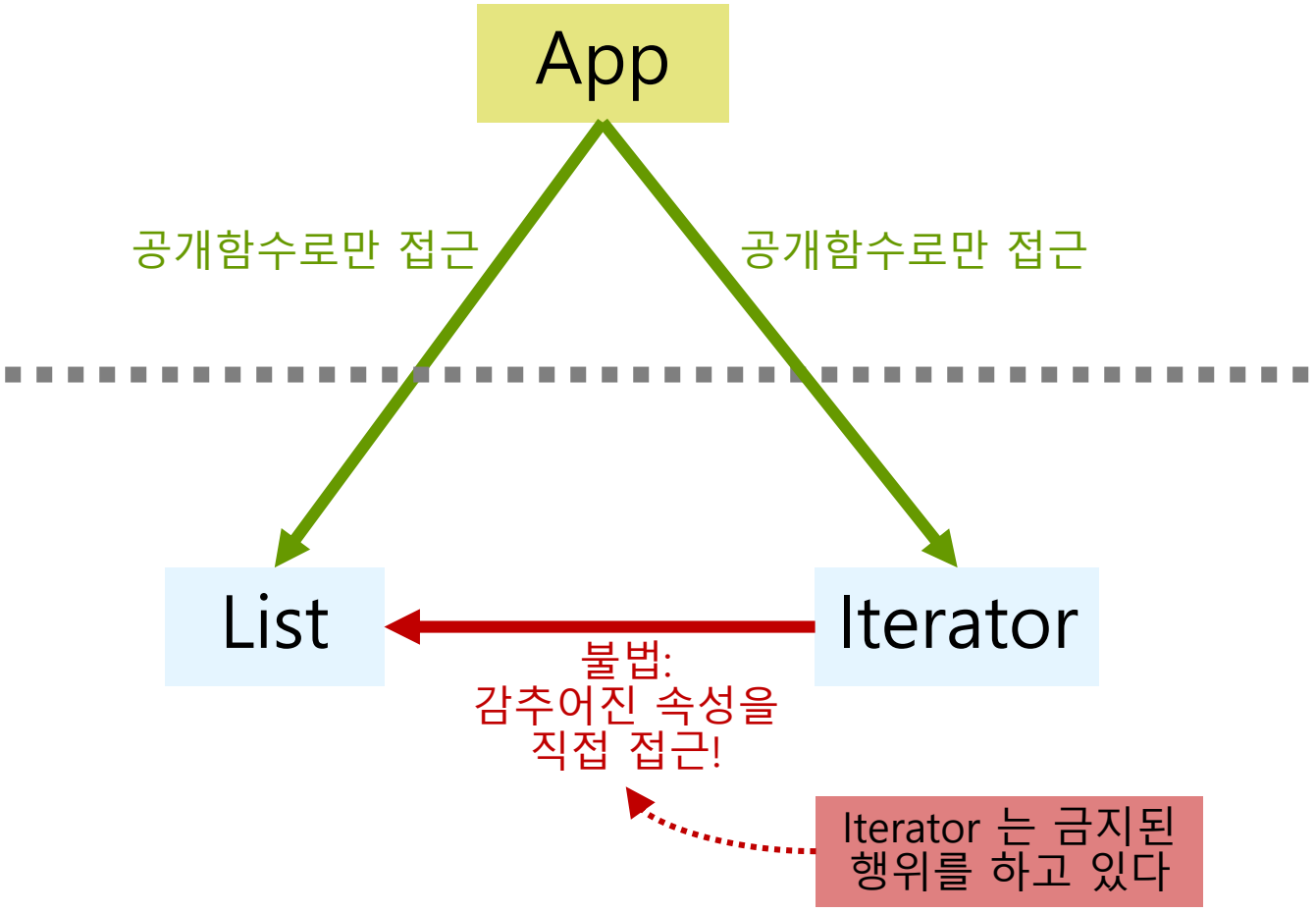
```
private   ListIterator ()
{
      this._nextPosition = 0 ;
}

public boolean   hasNext ()
{
      return (this._nextPosition < ArrayList.this.size()) ;
}

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

```
Student   s = null ;
int   pass = 0 ;
i = 0 ; // ListIterator 객체 생성으로
while ( i < studentList.size() ) {
            // iterator.hasNext() 로
      student = studentList._elements[i];
            // iterator.next() 로
      if ( s.score() >= 60 ) {
            pass++ ;
      }
      i++; // iterator.next() 로
}
```

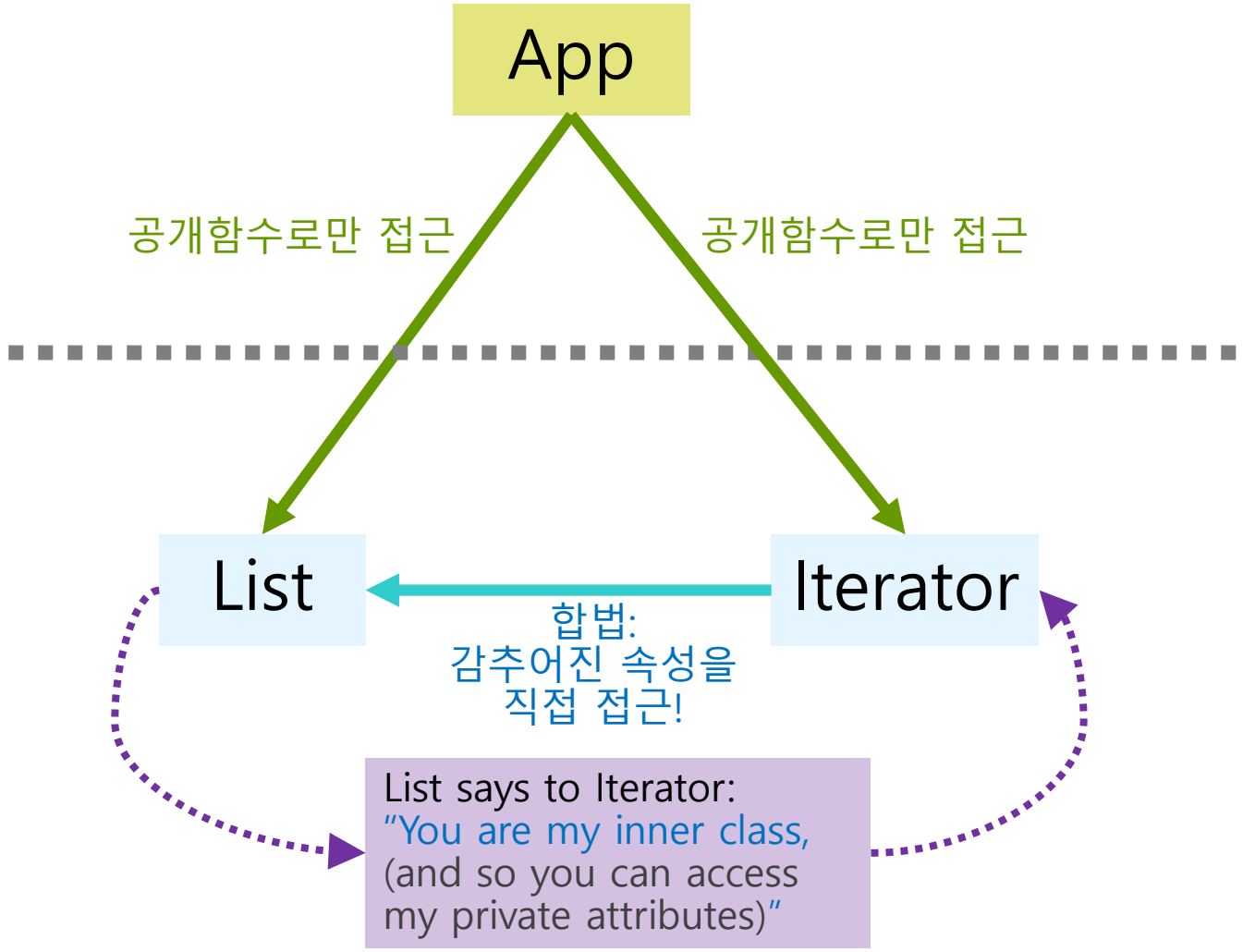
```
ArrayList<Student> studentList
            = new ArrayList<Student>() ;
.....
ArrayList<Student>.ListIterator iterator
            = studentList.iterator() ;
Student   s = null ;
int   pass = 0 ;
while ( iterator.hasNext() ) {
      s = iterator.next () ;
      if ( s.score() >= 60 ) {
            pass++ ;
      }
}
```

반복자는 왜 리스트의 내부 클래스로?

반복자는 왜 리스트의 내부 클래스로?



반복자는 왜 리스트의 내부 클래스로?



해결책: C

- Iterator 를 구현하기 위해서는, 어쩔 수 없이, List 의 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, "Iterator 의 구현은 private 속성을 직접 접근할 수 밖에 없는 예외적인 경우"이다.

해결책: C++

- C++ 프로그램에서는 컴파일러의 도움을 받을 수 있다.
  - List 는 Iterator 를 friend class 로 선언하면 된다.
  - List 의 속성 중에서 friend class 인 Iterator 가 접근하게 되는 속성은 protected 로 선언한다.

해결책: Java

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

Interface “Iterator”

반복자 (Iterator) 는 왜 인터페이스로?

- 배열 리스트에서의 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 Iterator 의 공개함수

```
pubic interface  Iterator<T>
{
    public boolean    hasNext () ;
       // 다음 원소가 존재하는지를 알아낸다
    public T              next () ;
       // 다음 원소를 얻어낸다. 없으면 null을 얻는다.

} // End of interface "Iterator"
```

연결리스트에서의 인터페이스 사용과 구현

LinkedList<T>.ListIterator 에서의 인터페이스 사용

```
pubic class LinkedList<T>
{
    // LinkedList의 선언
    ......

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

    // Inner Class "ListIterator" 의 선언
    private  class ListIterator  implements Iterator<T>
    {
        // 인스턴스 변수들
        ......
        private          ListIterator () {...} ; // 생성자
        public boolean   hasNext () {...} ; //다음 원소가 존재하는지를 알아낸다
        public T         next () {...} ; // 다음 원소를 얻어낸다. 없으면 null을 얻는다.
    } // End of ListIterator

} // End of LinkedList
```

LinkedList⟨T⟩.ListIterator 에서의 인터페이스 사용

```
pubic class LinkedList<T>
{
    // LinkedList의 선언
    ......

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

    // Inner Class "ListIterator" 의 선언
    private  class ListIterator  implements Iterator<T>
    {
        // 인스턴스 변수들
        ......
        private          ListIterator () {...} ; // 생성자
        public boolean   hasNext () {...} ; //다음 원소가 존재하는지를 알아낸다
        public T         next () {...} ; // 다음 원소를 얻어낸다. 없으면 null을 얻는다.
    } // End of ListIterator

} // End of LinkedList
```

Why "private"?

LinkedList⟨T⟩.ListIterator: 함수의 구현

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

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

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

반복자의 type 으로 인터페이스 사용.
Class 이름 "LinkedList" 를 사용하지 않고 있다.

```
LinkedList<Student> studentList ;
studentList = new LinkedList<Student>() ;
.....

Iterator<Student> iterator ;
iterator = studentList.iterator() ;
Student s = null ;
pass = 0 ;
while ( iterator.hasNext() ) {
    s = iterator.next() ;
    if ( s.score() >= 60 ) {
        pass++ ;
    }
}
```

ArrayList 에서의 인터페이스 사용과 구현

ArrayList의 내부 클래스로 선언

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

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

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

} // End of ArrayList
```

ArrayList〈T〉.ListIterator: 생성자의 구현

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

public boolean  hasNext ()
{
    return (this._nextPosition < ArrayList.this.size()) ;
}

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

```
ArrayList<Student> studentList ;
studentList = new ArrayList() ;
.....

Iterator<Student> iterator ;
iterator = studentList.iterator() ;
Student  s ;
pass = 0 ;
while ( iterator.hasNext() ) {
    s = iterator.next() ;
    if ( s.score() >= 60 )
        pass++ ;
    }
}
```

End of

"Abstraction of Iteration"