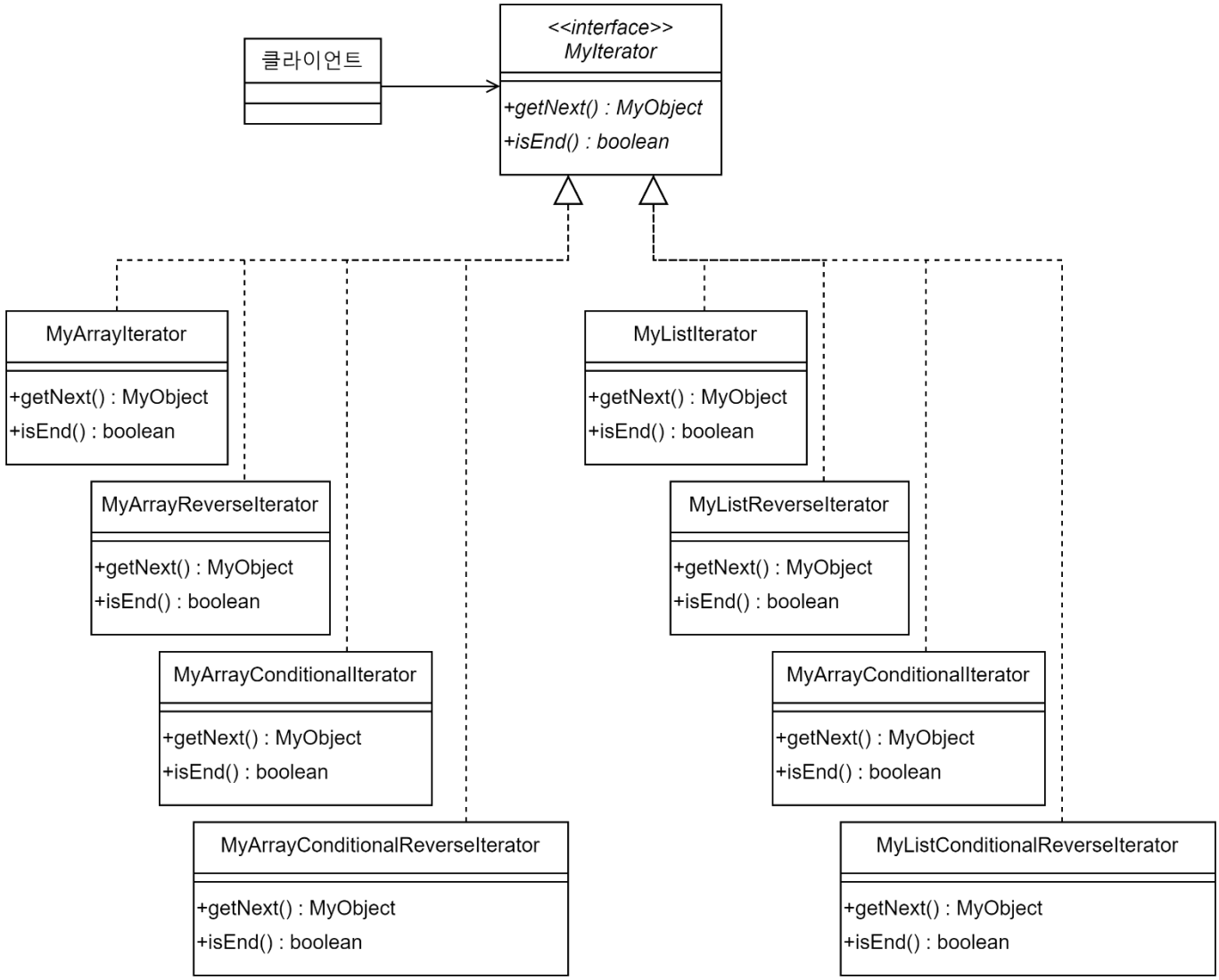
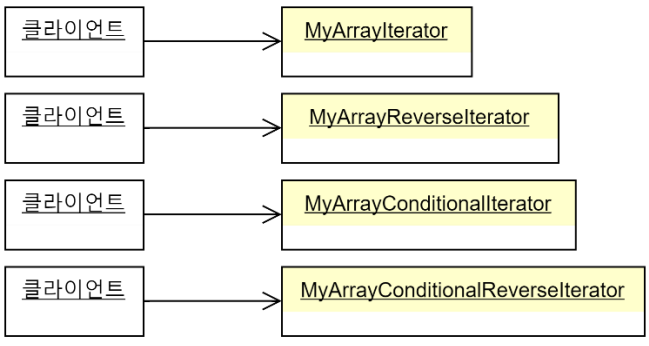
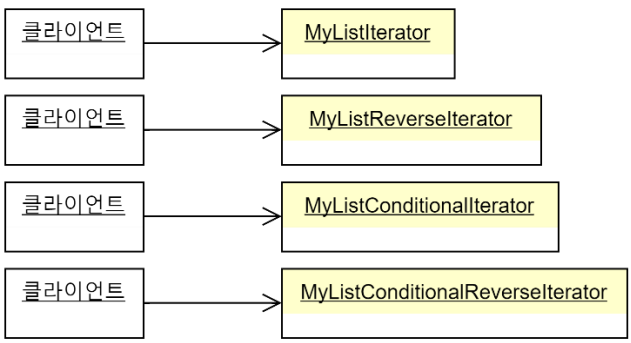
목차

1) 개요


클라이언트
<<interface>>
MyIterator
+getNext() : MyObject
+isEnd() : boolean
MyArrayIterator
+getNext() : MyObject
+isEnd() : boolean
MyListIterator
+getNext() : MyObject
+isEnd() : boolean
MyArrayReverseIterator
+getNext() : MyObject
+isEnd() : boolean
MyListReverseIterator
+getNext() : MyObject
+isEnd() : boolean
MyArrayConditionalIterator
+getNext() : MyObject
+isEnd() : boolean
MyArrayConditionalIterator
+getNext() : MyObject
+isEnd() : boolean
MyArrayConditionalReverseIterator
+getNext() : MyObject
+isEnd() : boolean
MyListConditionalReverseIterator
+getNext() : MyObject
+isEnd() : boolean


클라이언트
MyArrayIterator
클라이언트
MyArrayReverseIterator
클라이언트
MyArrayConditionalIterator
클라이언트
MyArrayConditionalReverseIterator


클라이언트
MyListIterator
클라이언트
MyListReverseIterator
클라이언트
MyListConditionalIterator
클라이언트
MyListConditionalReverseIterator

2) MyIterator.java

```java
package decorator.i1;

public interface MyIterator {
    MyObject getNext();
    boolean isEnd();
}
```

3) MyArrayIterator.java

```java
package decorator.i1;

class MyArrayIterator implements MyIterator {
    private final MyArray myArray;
    private int current;

    public MyArrayIterator(MyArray myArray) {
        this.myArray = myArray;
        this.current = 0;
    }

    @Override
    public MyObject getNext() {
        return this.myArray.get(current++);
    }

    @Override
    public boolean isEnd() {
        return current >= this.myArray.getCount();
    }
}
```

4) MyArrayReverseIterator.java

```java
package decorator.i1;

class MyArrayReverseIterator implements MyIterator {
    private final MyArray myArray;
    private int current;

    public MyArrayReverseIterator(MyArray myArray) {
        this.myArray = myArray;
        current = this.myArray.getCount() - 1;
    }

    @Override
    public MyObject getNext() {
        return this.myArray.get(current--);
    }

    @Override
    public boolean isEnd() {
        return current < 0;
    }
}
```

5) MyArrayConditionalIterator.java

```java
package decorator.i1;

import java.util.function.Predicate;

class MyArrayConditionalIterator implements MyIterator {
    private MyArray myArray;
    private int current;
    private Predicate<MyObject> predicate;
    private MyObject value;
    private boolean end;

    public MyArrayConditionalIterator(MyArray myArray, Predicate<MyObject> predicate) {
        this.myArray = myArray;
        this.current = 0;
        this.predicate = predicate;
        this.value = findNext();
        this.end = false;
    }

    private MyObject findNext() {
        while (!(current >= this.myArray.getCount())) {
            MyObject value = this.myArray.get(current++);
            if (predicate.test(value)) return value;
        }
        this.end = true;
        return null;
    }

    @Override
    public MyObject getNext() {
        MyObject r = value;
        value = findNext();
        return r;
    }

    @Override
    public boolean isEnd() {
        return end;
    }
}
```

Predicate<MyObject> predicate
조건식 객체
보통 조건식 객체는 lambda expression으로 구현된다

조건식을 만족하는 항목만 순방향으로 탐색하기 위한 iterator

6) MyArrayConditionalReverseIterator.java

```java
package decorator.i1;

import java.util.function.Predicate;

class MyArrayConditionalReverseIterator implements MyIterator {
    private MyArray myArray;
    private int current;
    private Predicate<MyObject> predicate;
    private MyObject value;
    private boolean end;

    public MyArrayConditionalReverseIterator(MyArray myArray, Predicate<MyObject> predicate) {
        this.myArray = myArray;
        this.current = this.myArray.getCount() - 1;
        this.predicate = predicate;
        this.value = findNext();
        this.end = false;
    }

    private MyObject findNext() {
        while (!(current < 0)) {
            MyObject value = this.myArray.get(current--);
            if (predicate.test(value)) return value;
        }
        this.end = true;
        return null;
    }

    @Override
    public MyObject getNext() {
        MyObject r = value;
        value = findNext();
        return r;
    }

    @Override
    public boolean isEnd() {
        return end;
    }
}
```

조건식을 만족하는 항목만 역방향으로 탐색하기 위한 iterator

7) MyListIterator.java

```java
package decorator.i1;

public class MyListIterator implements MyIterator {
    protected MyList myList;
    protected MyList.Node current;

    public MyListIterator(MyList myList) {
        this.myList = myList;
        this.current = myList.dummy.next;
    }

    @Override
    public MyObject getNext() {
        MyObject r = current.data;
        current = current.next;
        return r;
    }

    @Override
    public boolean isEnd() {
        return current == myList.dummy;
    }

}
```

8) MyListReverseIterator.java

```java
package decorator.i1;

public class MyListReverseIterator implements MyIterator {
    private MyList myList;
    private MyList.Node current;

    public MyListReverseIterator(MyList myList) {
        this.myList = myList;
        this.current = myList.dummy.prev;
    }

    @Override
    public MyObject getNext() {
        MyObject r = current.data;
        current = current.prev;
        return r;
    }

    @Override
    public boolean isEnd() {
        return current == myList.dummy;
    }

}
```

9) MyListConditionalIterator.java

```java
package decorator.i1;

import java.util.function.Predicate;

class MyListConditionalIterator implements MyIterator {
    private MyList myList;
    private MyList.Node current;
    private Predicate<MyObject> predicate;
    private MyObject value;
    private boolean end;

    public MyListConditionalIterator(MyList myList, Predicate<MyObject> predicate) {
        this.myList = myList;
        this.current = myList.dummy.next;
        this.predicate = predicate;
        this.value = findNext();
        this.end = false;
    }

    private MyObject findNext() {
        while (!(current == myList.dummy)) {
            MyObject value = current.data;
            current = current.next;
            if (predicate.test(value)) return value;
        }
        this.end = true;
        return null;
    }

    @Override
    public MyObject getNext() {
        MyObject r = value;
        value = findNext();
        return r;
    }

    @Override
    public boolean isEnd() {
        return end;
   }
}
```

10) MyListConditionalReverseIterator.java

```java
package decorator.i1;

import java.util.function.Predicate;

class MyListConditionalReverseIterator implements MyIterator {
    private MyList myList;
    private MyList.Node current;
    private Predicate<MyObject> predicate;
    private MyObject value;
    private boolean end;

    public MyListConditionalReverseIterator(MyList myList, Predicate<MyObject> predicate) {
        this.myList = myList;
        this.current = myList.dummy.prev;
        this.predicate = predicate;
        this.value = findNext();
        this.end = false;
    }

    private MyObject findNext() {
        while (!(current == myList.dummy)) {
            MyObject value = current.data;
            current = current.prev;
            if (predicate.test(value)) return value;
        }
        this.end = true;
        return null;
    }

    @Override
    public MyObject getNext() {
        MyObject r = value;
        value = findNext();
        return r;
    }

    @Override
    public boolean isEnd() {
        return end;
    }
}
```

11) Example1.java

```
1  package decorator.i1;
2
3  public class Example1 {
4
5      static void print(MyIterator it) {
6          while (!it.isEnd())
7              System.out.printf("%s ", it.getNext());
8          System.out.println();
9      }
10
11     static void doSomething1(int count) {
12         MyArray myArray = new MyArray();
13         for (int i = 0; i < count; ++i)
14             myArray.add(i % 2 == 0 ? new MyInt(i) : new MyStr(i));
15
16         print(new MyArrayIterator(myArray));         // 가
17         print(new MyArrayReverseIterator(myArray)); // 나
18         print(new MyArrayConditionalIterator(myArray, (obj) -> obj instanceof MyInt));        // 다
19         print(new MyArrayConditionalReverseIterator(myArray, (obj) -> obj instanceof MyStr)); // 라
20     }
21
22     static void doSomething2(int count) {
23         MyList myList = new MyList();
24         for (int i = 0; i < count; ++i)
25             myList.add(i % 2 == 0 ? new MyInt(i) : new MyStr(i));
26
27         print(new MyListIterator(myList));         // 가
28         print(new MyListReverseIterator(myList));  // 나
29         print(new MyListConditionalIterator(myList, (obj) -> obj instanceof MyInt));        // 다
30         print(new MyListConditionalReverseIterator(myList, (obj) -> obj instanceof MyStr)); // 라
31     }
32
33     public static void main(String[] args) {
34         doSomething1(10);
35         doSomething2(10);
36     }
37 }
```

가) 모든 항목 순방향 출력
나) 모든 항목 역방향 출력
다) MyInt 항목만 순방향 출력
라) MyStr 항목만 역방향 출력

출력

```
MyInt(0) MyStr(1) MyInt(2) MyStr(3) MyInt(4) MyStr(5) MyInt(6) MyStr(7) MyInt(8) MyStr(9)
MyStr(9) MyInt(8) MyStr(7) MyInt(6) MyStr(5) MyInt(4) MyStr(3) MyInt(2) MyStr(1) MyInt(0)
MyInt(0) MyInt(2) MyInt(4) MyInt(6) MyInt(8)
MyStr(9) MyStr(7) MyStr(5) MyStr(3) MyStr(1)
MyInt(0) MyStr(1) MyInt(2) MyStr(3) MyInt(4) MyStr(5) MyInt(6) MyStr(7) MyInt(8) MyStr(9)
MyStr(9) MyInt(8) MyStr(7) MyInt(6) MyStr(5) MyInt(4) MyStr(3) MyInt(2) MyStr(1) MyInt(0)
MyInt(0) MyInt(2) MyInt(4) MyInt(6) MyInt(8)
MyStr(9) MyStr(7) MyStr(5) MyStr(3) MyStr(1)
```