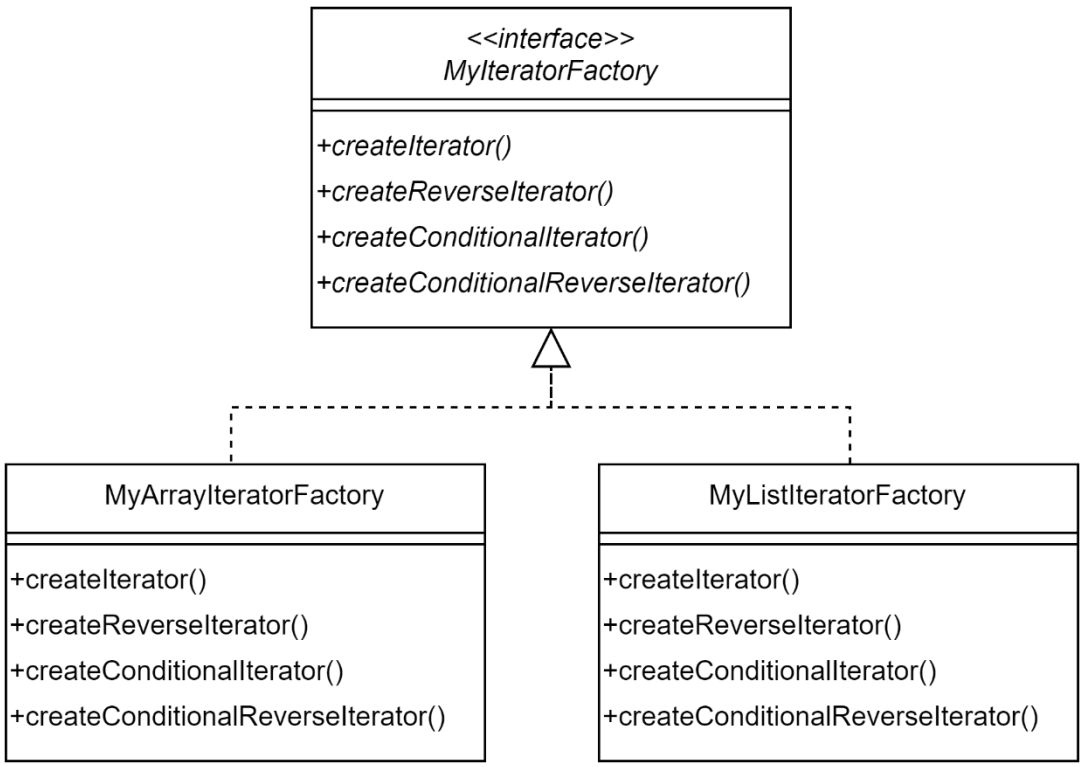
Abstract Factory 패턴 e1

목차

1. 개요


<<interface>>
MyIteratorFactory
+createIterator()
+createReverseIterator()
+createConditionalIterator()
+createConditionalReverseIterator()
MyArrayIteratorFactory
+createIterator()
+createReverseIterator()
+createConditionalIterator()
+createConditionalReverseIterator()
MyListIteratorFactory
+createIterator()
+createReverseIterator()
+createConditionalIterator()
+createConditionalReverseIterator()

2. MyObject

1) MyObject.java

```java
package abstract_factory.e1;

public interface MyObject {
    boolean equals(MyObject obj);
    int hashValue();
}
```

2) MyInt.java

```java
package abstract_factory.e1;

public class MyInt implements MyObject {
    int value;

    public MyInt(int value) {
        this.value = value;
    }

    @Override
    public boolean equals(MyObject obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
        return (value == ((MyInt)obj).value);
    }

    @Override
    public int hashValue() {
        return value;
    }

    @Override
    public String toString() {
        return String.format("MyInt(%d)", value);
    }
}
```

3) MyStr.java

```java
package abstract_factory.e1;

public class MyStr implements MyObject {
    private String value;

    public MyStr(String value) {
        this.value = value;
    }

    public MyStr(int value) {
        this.value = String.valueOf(value);
    }

    @Override
    public boolean equals(MyObject obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
        MyStr myString = (MyStr)obj;
        return (value == myString.value) ||
               (value != null && value.equals(myString.value));
    }

    @Override
    public int hashValue() {
        return value.hashCode();
    }

    @Override
    public String toString() {
        return String.format("MyStr(%s)", value);
    }
}
```

3. MyCollection

1) MyCollection.java

```java
package abstract_factory.e1;

public interface MyCollection {
    void add(MyObject value);
    boolean contains(MyObject value);
    MyIteratorFactory getIteratorFactory();
}
```

2) MyArray.java

```java
package abstract_factory.e1;

import java.util.Arrays;

public class MyArray implements MyCollection {
    MyObject[] data;
    int count;

    public MyArray() {
        this(8);
    }

    public MyArray(int size) {
        data = new MyObject[size];
        count = 0;
    }

    private void expand() {
        data = Arrays.copyOf(data, data.length * 2);
    }

    @Override
    public void add(MyObject value) {
        if (count == data.length) expand();
        data[count++] = value;
    }

    public MyObject get(int index) {
        return data[index];
    }

    public int getCount() {
        return count;
    }

    @Override
    public boolean contains(MyObject value) {
        for (int i = 0; i < count; ++i)
            if (data[i].equals(value)) return true;
        return false;
    }

    @Override
    public MyIteratorFactory getIteratorFactory() {
        return new MyArrayIteratorFactory(this);
    }
}
```

3) MyList.java

```java
package abstract_factory.e1;

public class MyList implements MyCollection {
    static class Node {
        MyObject data;
        Node prev, next;

        Node(MyObject data) {
            this.data = data;
        }
    }

    Node dummy;

    public MyList() {
        dummy = new Node(null);
        dummy.prev = dummy.next = dummy;
    }

    public void addHead(MyObject value) {
        Node node = new Node(value);
        node.next = dummy.next;
        node.prev = dummy;
        dummy.next.prev = node;
        dummy.next = node;
    }

    public void addTail(MyObject value) {
        Node node = new Node(value);
        node.next = dummy;
        node.prev = dummy.prev;
        dummy.prev.next = node;
        dummy.prev = node;
    }

    @Override
    public void add(MyObject value) {
        addTail(value);
    }

    @Override
    public boolean contains(MyObject value) {
        Node node = dummy.next;
        while (node != dummy) {
            if (node.data.equals(value)) return true;
            node = node.next;
        }
        return false;
    }

    @Override
    public MyIteratorFactory getIteratorFactory() {
        return new MyListIteratorFactory(this);
    }
}
```

4. MyIterator

1) MyIterator

```
1  package abstract_factory.e2;
2
3  public interface MyIterator {
4      MyObject getNext();
5      boolean isEnd();
6  }
```

2) MyArrayIterator.java

```java
package abstract_factory.e1;

class MyArrayIterator implements MyIterator {
    MyArray myArray;
    int current;

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

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

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

3) MyArrayReverseIterator.java

```java
package abstract_factory.e1;

class MyArrayReverseIterator implements MyIterator {
    MyArray myArray;
    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;
    }
}
```

4) MyListIterator.java

```java
package abstract_factory.e1;

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

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

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

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

5) MyLisReverseIterator.java

```java
package abstract_factory.e1;

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

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

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

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

6) MyConditionalIterator.java

```java
package abstract_factory.e1;

import java.util.function.Predicate;

class MyConditionalIterator implements MyIterator {
    private MyIterator iterator;
    private Predicate<MyObject> predicate;
    private MyObject value;

    public MyConditionalIterator(MyIterator iterator, Predicate<MyObject> predicate) {
        this.iterator = iterator;
        this.predicate = predicate;
        this.value = findNext();
    }

    private MyObject findNext() {
        while (!iterator.isEnd()) {
            MyObject value = iterator.getNext();
            if (predicate.test(value)) return value;
        }
        return null;
    }

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

    @Override
    public boolean isEnd() {
        return value == null;
    }
}
```

5. MyIteratorFactory.java

1) MyIteratorFactory.java

```java
package abstract_factory.e1;

import java.util.function.Predicate;

public interface MyIteratorFactory {
    MyIterator getIterator();
    MyIterator getReverseIterator();
    MyIterator getConditionalIterator(Predicate<MyObject> predicate);
    MyIterator getConditionalReverseIterator(Predicate<MyObject> predicate);
}
```

2) MyArrayIteratorFactory.java

```java
package abstract_factory.e1;

import java.util.function.Predicate;

public class MyArrayIteratorFactory implements MyIteratorFactory {
    MyArray myArray;

    public MyArrayIteratorFactory(MyArray myArray) {
        this.myArray = myArray;
    }

    @Override
    public MyIterator getIterator() {
        return new MyArrayIterator(myArray);
    }

    @Override
    public MyIterator getReverseIterator() {
        return new MyArrayReverseIterator(myArray);
    }

    @Override
    public MyIterator getConditionalIterator(Predicate<MyObject> predicate) {
        return new MyConditionalIterator(new MyArrayIterator(myArray), predicate);
    }

    @Override
    public MyIterator getConditionalReverseIterator(Predicate<MyObject> predicate) {
        return new MyConditionalIterator(new MyArrayReverseIterator(myArray), predicate);
    }

}
```

3) MyListIteratorFactory.java

```java
package abstract_factory.e1;

import java.util.function.Predicate;

public class MyListIteratorFactory implements MyIteratorFactory {
    MyList myList;

    public MyListIteratorFactory(MyList myList) {
        this.myList = myList;
    }

    @Override
    public MyIterator getIterator() {
        return new MyListIterator(myList);
    }

    @Override
    public MyIterator getReverseIterator() {
        return new MyListReverseIterator(myList);
    }

    @Override
    public MyIterator getConditionalIterator(Predicate<MyObject> predicate) {
        return new MyConditionalIterator(new MyListIterator(myList), predicate);
    }

    @Override
    public MyIterator getConditionalReverseIterator(Predicate<MyObject> predicate) {
        return new MyConditionalIterator(new MyListReverseIterator(myList), predicate);
    }
}
```

6. Client

1) Example1.java

```
1  package abstract_factory.e1;
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 doSomething(MyCollection col, int count) {
12         for (int i = 0; i < count; ++i)
13             col.add(i % 2 == 0 ? new MyInt(i) : new MyStr(i));
14
15         MyIteratorFactory factory = col.getIteratorFactory();
16         print(factory.getIterator());
17         print(factory.getReverseIterator());
18         print(factory.getConditionalIterator((obj) -> obj instanceof MyInt));
19         print(factory.getConditionalReverseIterator((obj) -> obj instanceof MyStr));
20     }
21
22     public static void main(String[] args) {
23         doSomething(new MyArray(), 10);
24         doSomething(new MyList(), 10);
25     }
26 }
```

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)
```

2) 이전 예제 코드와 비교

```
1  package decorator.i2;
2
3  public class Example2 {
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 doSomething(MyCollection col, int count) {
12         for (int i = 0; i < count; ++i)
13             col.add(i % 2 == 0 ? new MyInt(i) : new MyStr(i));
14
15         print(col.getIterator());
16         print(col.getReverseIterator());
17         print(new MyConditionalIterator(col.getIterator(), (obj) -> obj instanceof MyInt));
18         print(new MyConditionalIterator(col.getReverseIterator(), (obj) -> obj instanceof MyStr));
19     }
20
21     public static void main(String[] args) {
22         doSomething(new MyArray(), 10);
23         doSomething(new MyList(), 10);
24     }
25 }
```

MyIteratorFactory를 구현하기 전