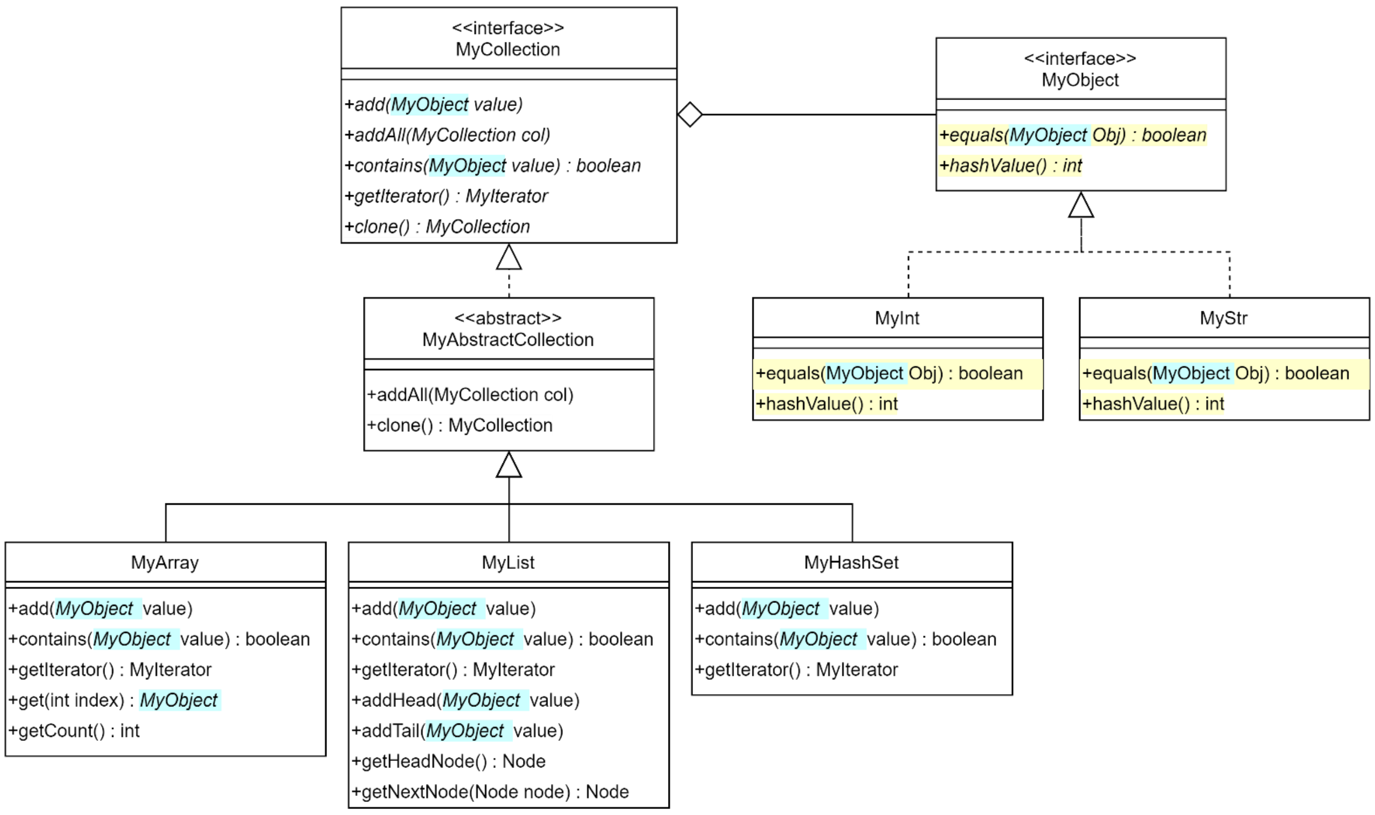
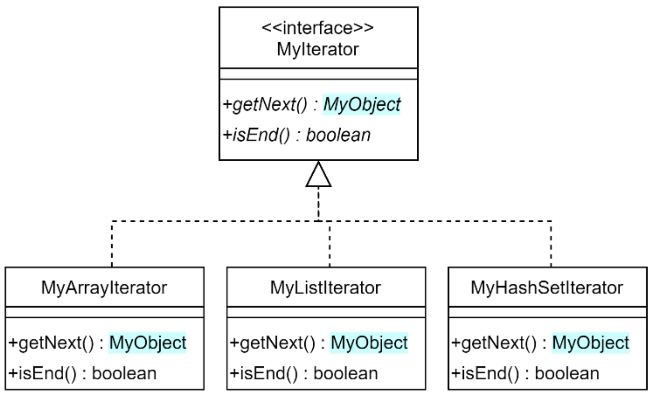
1) 개요



<<interface>>
MyCollection
+add(MyObject value)
+addAll(MyCollection col)
+contains(MyObject value) : boolean
+getIterator() : MyIterator
+clone() : MyCollection
<<interface>>
MyObject
+equals(MyObject Obj) : boolean
+hashValue() : int
<<abstract>>
MyAbstractCollection
+addAll(MyCollection col)
+clone() : MyCollection
MyInt
+equals(MyObject Obj) : boolean
+hashValue() : int
MyStr
+equals(MyObject Obj) : boolean
+hashValue() : int
MyArray
+add(MyObject value)
+contains(MyObject value) : boolean
+getIterator() : MyIterator
+get(int index) : MyObject
+getCount() : int
MyList
+add(MyObject value)
+contains(MyObject value) : boolean
+getIterator() : MyIterator
+addHead(MyObject value)
+addTail(MyObject value)
+getHeadNode() : Node
+getNextNode(Node node) : Node
MyHashSet
+add(MyObject value)
+contains(MyObject value) : boolean
+getIterator() : MyIterator



<<interface>>
MyIterator
+getNext() : MyObject
+isEnd() : boolean
MyArrayIterator
+getNext() : MyObject
+isEnd() : boolean
MyListIterator
+getNext() : MyObject
+isEnd() : boolean
MyHashSetIterator
+getNext() : MyObject
+isEnd() : boolean

2) MyObject.java

```java
package composite.e2;

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

3) MyInt.java

```java
package composite.e2;

public class MyInt implements MyObject {
    private 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);
    }
}
```

4) MyStr.java

```java
package composite.e2;

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);
    }
}
```

5) MyCollection.java

```java
package composite.e2;

public interface MyCollection {
    void add(MyObject value);
    void addAll(MyCollection col);
    boolean contains(MyObject value);
    MyIterator getIterator();
    MyCollection clone() throws CloneNotSupportedException;
}
```

6) MyAbstractCollection.java

```java
package composite.e2;

public abstract class MyAbstractCollection implements MyCollection {

    @Override
    public void addAll(MyCollection col) {
        MyIterator it = col.getIterator();
        while (!it.isEnd())
            add(it.getNext());
    }

    @Override
    public MyCollection clone() throws CloneNotSupportedException {
        MyCollection col = null;
        try {
            col = this.getClass().getDeclaredConstructor().newInstance();
        } catch (Exception e) {
            throw new CloneNotSupportedException();
        }
        col.addAll(this);
        return col;
    }
}
```

7) MyIterator.java

```java
package composite.e2;

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

8) MyArray.java

```java
package composite.e2;

import java.util.Arrays;

public class MyArray extends MyAbstractCollection {
    private MyObject[] data;
    private 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;
    }

    private class MyArrayIterator implements MyIterator {
        private int current;

        public MyArrayIterator() {
            current = 0;
        }

        @Override
        public MyObject getNext() {
            return data[current++];
        }

        @Override
        public boolean isEnd() {
            return current >= count;
        }
    }

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

}
```

9) MyList.java

```java
package composite.e2;

public class MyList extends MyAbstractCollection {
    private static class Node {
        private MyObject data;
        private Node prev, next;

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

    private 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;
    }

    private class MyListIterator implements MyIterator {
        private Node current;

        MyListIterator() {
            current = dummy.next;
        }

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

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

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

10) MyHashSet.java

```java
package composite.e2;

public class MyHashSet extends MyAbstractCollection {
    static final double A = 0.3758;
    MyObject[] a;
    int count, threshold;

    public MyHashSet() {
        this(32);
    }

    public MyHashSet(int size) {
        this.a = new MyObject[size];
        this.count = 0;
        this.threshold = (int) (this.a.length * 0.7);
    }

    private void expand() {
        int newSize = a.length * 2;
        MyHashSet newHashTable = new MyHashSet(newSize);
        for (int i = 0; i < a.length; ++i)
            if (a[i] != null) newHashTable.add(a[i]);
        this.a = newHashTable.a;
        this.threshold = newHashTable.threshold;
    }

    private int getStartIndex(MyObject value) { // 최초 저장할 위치 계산
        double fractionalPart = (value.hashValue() * A) % 1;
        return (int) (fractionalPart * this.a.length);
    }

    private static int getStepDistance(MyObject value) { // 충돌 발생한 경우 건너뛸 간격 계산
        final int[] STEPS = {3, 5, 7, 11, 13, 17, 19}; // 소수 크기 간격
        return STEPS[Math.abs(value.hashValue()) % STEPS.length];
    }

    @Override
    public void add(MyObject value) {
        int startIndex = getStartIndex(value);
        int step = getStepDistance(value);
        int collisionCount = 0;
        do {
            int index = (startIndex + collisionCount * step) % a.length;
            if (a[index] == null) {
                a[index] = value;
                this.count++;
                if (this.count >= this.threshold)
                    expand();
                return;
            } else if (a[index] == value)
                return;
            ++collisionCount;
        } while (collisionCount < a.length);
    }

    @Override
    public boolean contains(MyObject value) {
        int startIndex = getStartIndex(value);
        int step = getStepDistance(value);
        int collisionCount = 0;
        do {
            int index = (startIndex + collisionCount * step) % a.length;
            if (a[index] == null)
                return false;
            else if (a[index].equals(value))
                return true;
            ++collisionCount;
        } while (collisionCount < a.length);
```

```java
            return false;
        }

    private class MyHashSetIterator implements MyIterator {
        private int current;

        public MyHashSetIterator() {
            current = -1;
            next();
        }

        private void next() {
            ++current;
            while (current < a.length && a[current] == null)
                ++current;
        }

        @Override
        public MyObject getNext() {
            MyObject r = a[current];
            next();
            return r;
        }

        @Override
        public boolean isEnd() {
            return current >= a.length;
        }
    }

    @Override
    public MyIterator getIterator() {
        return new MyHashSetIterator();
    }
}
```

11) Example2.java

```java
package composite.e2;

public class Example2 {

    static void print(MyIterator it) {
        while (!it.isEnd())
            System.out.printf("%s ", it.getNext());
        System.out.println();
    }

    static void doSomething(MyCollection col, int count) {
        for (int i = 0; i < count; ++i)
            col.add(i % 2 == 0 ? new MyInt(i) : new MyStr(i));

        System.out.printf("%s %s ", col.contains(new MyInt(2)), !col.contains(new MyStr(2)));
        print(col.getIterator());
    }

    public static void main(String[] args) {
        doSomething(new MyArray(), 10);
        doSomething(new MyList(), 10);
        doSomething(new MyHashSet(), 10);
    }
}
```

출력

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