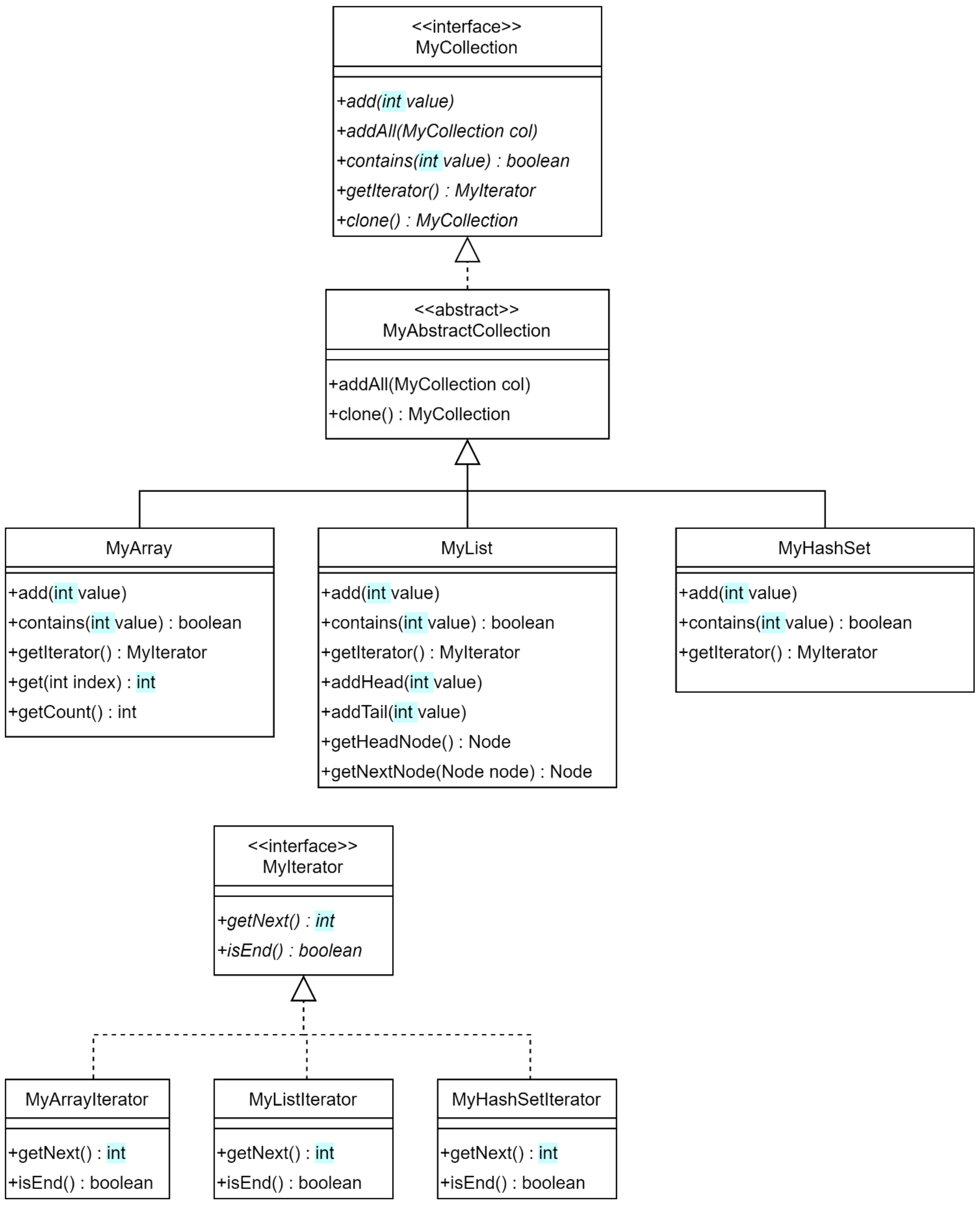
## 1) 개요



<<interface>>
MyCollection
+add(int value)
+addAll(MyCollection col)
+contains(int value) : boolean
+getIterator() : MyIterator
+clone() : MyCollection
<<abstract>>
MyAbstractCollection
+addAll(MyCollection col)
+clone() : MyCollection
MyArray
+add(int value)
+contains(int value) : boolean
+getIterator() : MyIterator
+get(int index) : int
+getCount() : int
MyList
+add(int value)
+contains(int value) : boolean
+getIterator() : MyIterator
+addHead(int value)
+addTail(int value)
+getHeadNode() : Node
+getNextNode(Node node) : Node
MyHashSet
+add(int value)
+contains(int value) : boolean
+getIterator() : MyIterator
<<interface>>
MyIterator
+getNext() : int
+isEnd() : boolean
MyArrayIterator
+getNext() : int
+isEnd() : boolean
MyListIterator
+getNext() : int
+isEnd() : boolean
MyHashSetIterator
+getNext() : int
+isEnd() : boolean

## 2) MyCollection.java

```java
package composite.e1;

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

## 3) MyAbstractCollection.java

```java
package composite.e1;

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;
    }
}
```

## 4) MyIterator.java

```java
package composite.e1;

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

## 5) MyArray.java

```java
package composite.e1;

import java.util.Arrays;

public class MyArray extends MyAbstractCollection {
    private int[] data;
    private int count;

    public MyArray() {
        this(8);
    }

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

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

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

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

    public int getCount() {
        return count;
    }

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

    private class MyArrayIterator implements MyIterator {
        private int current;

        public MyArrayIterator() {
            current = 0;
        }

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

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

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

}
```

## 6) MyList.java

```java
package composite.e1;

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

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

    private Node dummy;

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

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

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

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

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

    private class MyListIterator implements MyIterator {
        private Node current;

        MyListIterator() {
            current = dummy.next;
        }

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

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

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

## 7) MyHashSet.java

```java
package composite.e1;

import java.util.Arrays;

public class MyHashSet extends MyAbstractCollection {

    // 빈 칸을 표시하는 상수로 사용. 따라서 이 값을 사용할 수 없다
    static final int EMPTY = Integer.MIN_VALUE;

    static final double A = 0.3758;
    int[] a;
    int count, threshold;

    public MyHashSet() {
        this(32);
    }

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

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

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

    private static int getStepDistance(int 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) % STEPS.length];
    }

    @Override
    public void add(int value) {
        int startIndex = getStartIndex(value);
        int step = getStepDistance(value);
        int collisionCount = 0;
        do {
            int index = (startIndex + collisionCount * step) % a.length;
            if (a[index] == EMPTY) {
                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(int value) {
        int startIndex = getStartIndex(value);
        int step = getStepDistance(value);
        int collisionCount = 0;
        do {
            int index = (startIndex + collisionCount * step) % a.length;
            if (a[index] == EMPTY)
                return false;
            else if (a[index] == value)
                return true;
            ++collisionCount;
        } while (collisionCount < a.length);
        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] == EMPTY)
                ++current;
        }

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

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

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

## 8) Example1.java

```java
package composite.e1;

public class Example1 {

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

    static void doSomething(MyCollection col, int count) {
        for (int i = 0; i < count; ++i)
            col.add(i);
        System.out.printf("%s %s ", col.contains(3), !col.contains(count));
        print(col.getIterator());
    }

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

출력

```
true true 0 1 2 3 4 5 6 7 8 9
true true 0 1 2 3 4 5 6 7 8 9
true true 0 3 8 6 1 4 9 7 2 5
```