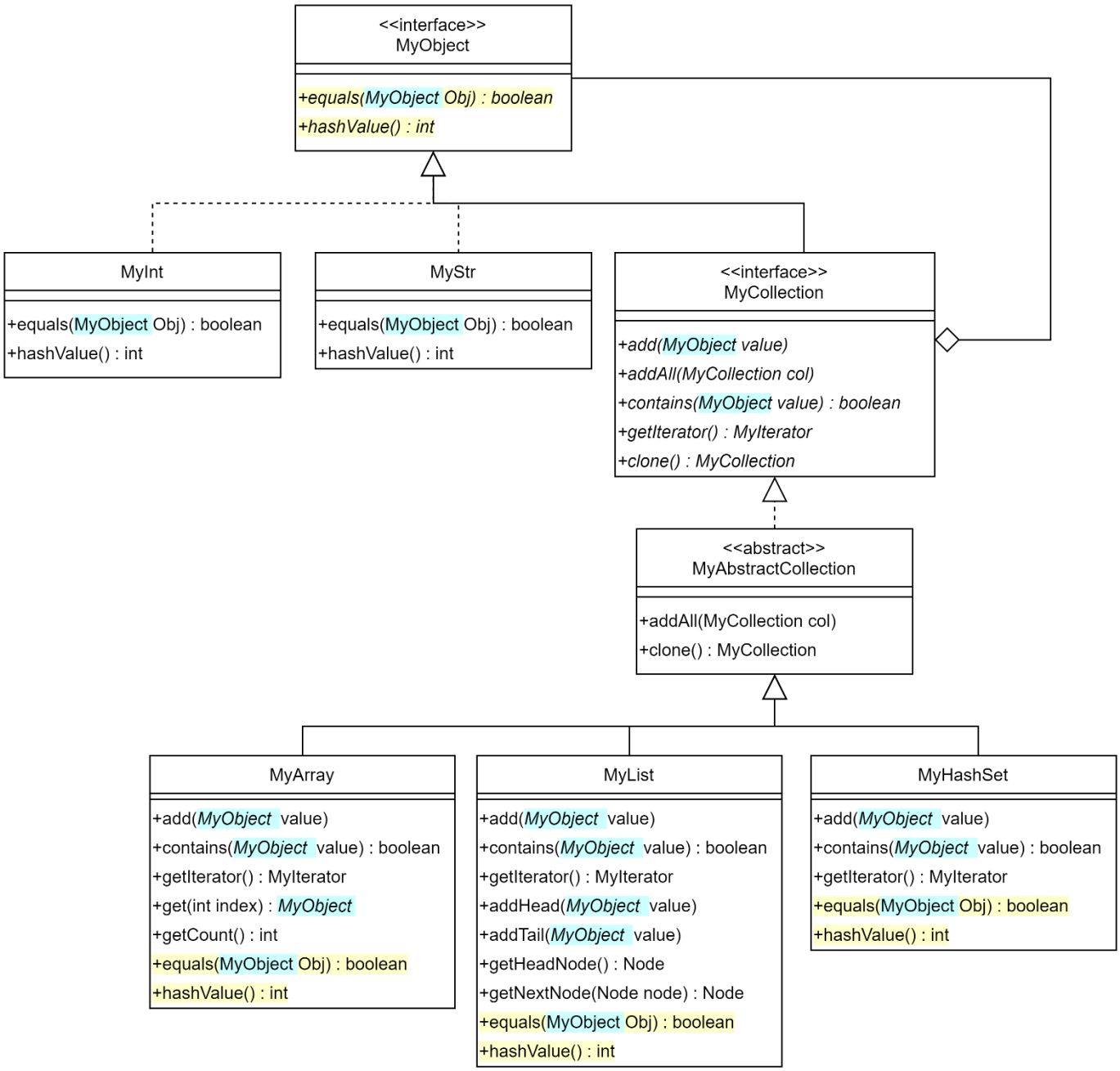
1) 개요



MyCollection extends MyObject

2) MyObject.java

```java
package composite.e3;

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

3) MyInt.java

```java
package composite.e3;

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.e3;

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

```
1  package composite.e3;
2
3  public interface MyCollection extends MyObject {
4      void add(MyObject value);
5      void addAll(MyCollection col);
6      boolean contains(MyObject value);
7      MyIterator getIterator();
8      MyCollection clone() throws CloneNotSupportedException;
9  }
```

6) MyAbstractCollection.java

```
1  package composite.e3;
2
3  public abstract class MyAbstractCollection implements MyCollection {
4
5      @Override
6      public void addAll(MyCollection col) {
7          MyIterator it = col.getIterator();
8          while (!it.isEnd())
9              add(it.getNext());
10     }
11
12     @Override
13     public MyCollection clone() throws CloneNotSupportedException {
14         MyCollection col = null;
15         try {
16             col = this.getClass().getDeclaredConstructor().newInstance();
17         } catch (Exception e) {
18             throw new CloneNotSupportedException();
19         }
20         col.addAll(this);
21         return col;
22     }
23 }
```

7) MyIterator.java

```
1  package composite.e3;
2
3  public interface MyIterator {
4      MyObject getNext();
5      boolean isEnd();
6  }
```

8) MyArray.java

```java
package composite.e3;

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();
    }

    @Override
    public boolean equals(MyObject obj) {
        if (this == obj) return true;
```

```
69          if (obj == null) return false;
70          if (getClass() != obj.getClass()) return false;
71          MyArray myArray = (MyArray)obj;
72          if (count != myArray.count) return false;
73          for (int i = 0; i < count; ++i) {
74              if (data[i] == null && myArray.get(i) != null) return false;
75              if (data[i] != null && data[i].equals(myArray.get(i)) == false) return false;
76          }
77          return true;
78      }
79
80      @Override
81      public int hashValue() {
82          int result = 0;
83          for (int i = 0; i < count; ++i)
84              if (data[i] != null)
85                  result = (31 * result + data[i].hashValue()) & 0xFFFFFF;
86          return result;
87      }
88
89      @Override
90      public String toString() {
91          StringBuilder builder = new StringBuilder();
92          builder.append("MyArray( ");
93          for (int i = 0; i < count; ++i)
94              builder.append(data[i]).append(' ');
95          builder.append(')');
96          return builder.toString();
97      }
98  }
99
```

9) MyList.java

```
1   package composite.e3;
2
3   public class MyList extends MyAbstractCollection {
4       private static class Node {
5           private MyObject data;
6           private Node prev, next;
7
8           Node(MyObject data) {
9               this.data = data;
10          }
11      }
12
13      private Node dummy;
14
15      public MyList() {
16          dummy = new Node(null);
17          dummy.prev = dummy.next = dummy;
18      }
19
20      public void addHead(MyObject value) {
21          Node node = new Node(value);
22          node.next = dummy.next;
23          node.prev = dummy;
24          dummy.next.prev = node;
25          dummy.next = node;
26      }
27
28      public void addTail(MyObject value) {
29          Node node = new Node(value);
30          node.next = dummy;
31          node.prev = dummy.prev;
32          dummy.prev.next = node;
33          dummy.prev = node;
34      }
35
36      @Override
37      public void add(MyObject value) {
38          addTail(value);
39      }
40
41      @Override
42      public boolean contains(MyObject value) {
43          Node node = dummy.next;
44          while (node != dummy) {
45              if (node.data.equals(value)) return true;
46              node = node.next;
47          }
48          return false;
49      }
50
51      private class MyListIterator implements MyIterator {
52          private Node current;
53
54          MyListIterator() {
55              current = dummy.next;
56          }
57
58          @Override
59          public MyObject getNext() {
60              MyObject r = current.data;
61              current = current.next;
62              return r;
63          }
64
65          @Override
66          public boolean isEnd() {
67              return current == dummy;
68          }
```

```
69      }
70
71      @Override
72      public MyIterator getIterator() {
73          return new MyListIterator();
74      }
75
76      @Override
77      public boolean equals(MyObject obj) {
78          if (this == obj) return true;
79          if (obj == null) return false;
80          if (getClass() != obj.getClass()) return false;
81          MyList myList = (MyList)obj;
82          Node node1 = dummy.next, node2 = myList.dummy.next;
83          while (node1 != dummy && node2 != dummy) {
84              if (node1.data == null && node2.data != null) return false;
85              if (node1.data != null && node1.data.equals(node2.data) == false) return false;
86               node1 = node1.next;
87               node2 = node2.next;
88          }
89          if (node1 != dummy || node2 != myList.dummy) return false;
90          return true;
91      }
92
93      @Override
94      public int hashValue() {
95          int result = 0;
96          Node node = dummy.next;
97          while (node != dummy) {
98              if (node.data != null)
99                  result = (31 * result + node.data.hashValue()) & 0xFFFFFF;
100             node = node.next;
101         }
102         return result;
103     }
104
105     @Override
106     public String toString() {
107         StringBuilder builder = new StringBuilder();
108         builder.append("MyList( ");
109         Node node = dummy.next;
110         while (node != dummy) {
111             builder.append(node.data).append(' ');
112             node = node.next;
113         }
114         builder.append(')');
115         return builder.toString();
116     }
117 }
```

10) MyHashSet.java

```java
package composite.e3;

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);
```

```
69          return false;
70      }
71
72      private class MyHashSetIterator implements MyIterator {
73          private int current;
74
75          public MyHashSetIterator() {
76              current = -1;
77              next();
78          }
79
80          private void next() {
81              ++current;
82              while (current < a.length && a[current] == null)
83                  ++current;
84          }
85
86          @Override
87          public MyObject getNext() {
88              MyObject r = a[current];
89              next();
90              return r;
91          }
92
93          @Override
94          public boolean isEnd() {
95              return current >= a.length;
96          }
97      }
98
99      @Override
100     public MyIterator getIterator() {
100         return new MyHashSetIterator();
101     }
102
103     @Override
104     public boolean equals(MyObject obj) {
105         if (this == obj) return true;
106         if (obj == null) return false;
107         if (getClass() != obj.getClass()) return false;
108         MyHashSet mySet = (MyHashSet)obj;
109         if (count != mySet.count) return false;
110         MyIterator it = getIterator();
111         while (!it.isEnd())
112             if (mySet.contains(it.getNext()) == false) return false;
113         return true;
114     }
115
116     @Override
117     public int hashValue() {
118         int result = 0;
119         MyIterator it = getIterator();
120         while (!it.isEnd()) {
121             MyObject value = it.getNext();
122             if (value != null)
123                 result = (31 * result + value.hashValue()) & 0xFFFFFF;
124         }
125         return result;
126     }
127
128     @Override
129     public String toString() {
130         StringBuilder builder = new StringBuilder();
131         builder.append("MyHashSet( ");
132         MyIterator it = getIterator();
133         while (!it.isEnd())
134             builder.append(it.getNext()).append(' ');
135         builder.append(')');
136         return builder.toString();
```

```
137         }
138 }
```

11) Example3.java

```
1  package composite.e3;
2
3  public class Example3 {
4
5      static void addData(MyCollection col, int count) {
6          for (int i = 0; i < count; ++i)
7              col.add( i % 2 == 0 ? new MyInt(i) : new MyStr(i) );
8      }
9
10     static MyArray createCompositeArray() {
11         MyArray a1 = new MyArray(), a2 = new MyArray(), a3 = new MyArray();
12         addData(a1, 3);
13         addData(a2, 3);
14         addData(a3, 3);
15         a2.add(a1);
16         a3.add(a2);
17         return a3;
18     }
19
20     static MyList createCompositeList() {
21         MyList a1 = new MyList(), a2 = new MyList(), a3 = new MyList();
22         addData(a1, 3);
23         addData(a2, 3);
24         addData(a3, 3);
25         a2.add(a1);
26         a3.add(a2);
27         return a3;
28     }
29
30     static MyHashSet createCompositeHashSet() {
31         MyHashSet a1 = new MyHashSet(), a2 = new MyHashSet(), a3 = new MyHashSet();
32         addData(a1, 3);
33         addData(a2, 3);
34         addData(a3, 3);
35         a2.add(a1);
36         a3.add(a2);
37         return a3;
38     }
39
40     static void testArray() {
41         MyArray a1 = createCompositeArray();
42         MyArray a2 = createCompositeArray();
43
44         System.out.println(a1.equals(a2));
45         System.out.println(a1);
46     }
47
48
49     static void testList() {
50         MyList b1 = createCompositeList();
51         MyList b2 = createCompositeList();
52
53         System.out.println(b1.equals(b2));
54         System.out.println(b1);
55     }
56
57     static void testHashset() {
58         MyHashSet a1 = createCompositeHashSet();
59         MyHashSet a2 = createCompositeHashSet();
60
61         System.out.println(a1.equals(a2));
62         System.out.println(a1);
63     }
64
65     public static void main(String[] args) {
66         testArray();
67         testList();
68         testHashset();
```

```
69      }
70
71  }
```

출력

```
true
MyArray( MyInt(0) MyStr(1) MyInt(2) MyArray( MyInt(0) MyStr(1) MyInt(2) MyArray( MyInt(0) MyStr(1) MyInt(2) ) ) )
true
MyList( MyInt(0) MyStr(1) MyInt(2) MyList( MyInt(0) MyStr(1) MyInt(2) MyList( MyInt(0) MyStr(1) MyInt(2) ) ) )
true
MyHashSet( MyInt(0) MyHashSet( MyInt(0) MyStr(1) MyHashSet( MyInt(0) MyStr(1) MyInt(2) ) MyInt(2) ) MyStr(1) MyInt(2) )
```