

Lab 10. OOP Exercises

Writing Good Programs

The only way to learn programming is program, program and program. Learning programming is like learning cycling, swimming or any other sports. You can't learn by watching or reading books. Start to program immediately. On the other hands, to improve your programming, you need to read many books and study how the masters program.

It is easy to write programs that work. It is much harder to write programs that not only work but also easy to maintain and understood by others – I call these good programs. In the real world, writing program is not meaningful. You have to write good programs, so that others can understand and maintain your programs.

Pay particular attention to:

1. Coding style:

- Read Java code convention: "Google Java Style Guide" or "Java Code Conventions - Oracle".
- Follow the Java Naming Conventions for variables, methods, and classes STRICTLY. Use CamelCase for names. Variable and method names begin with lowercase, while class names begin with uppercase. Use nouns for variables (e.g., radius) and class names (e.g., Circle). Use verbs for methods (e.g., getArea(), isEmpty()).
- **Use Meaningful Names:** Do not use names like a, b, c, d, x, x1, x2, and x1688 - they are meaningless. Avoid single-alphabet names like i, j, k. They are easy to type, but usually meaningless. Use single-alphabet names only when their meaning is clear, e.g., x, y, z for co-ordinates and i for array index. Use meaningful names like row and col (instead of x and y, i and j, x1 and x2), numStudents (not n), maxGrade, size (not n), and upperbound (not n again). Differentiate between singular and plural nouns (e.g., use books for an array of books, and book for each item).
- Use consistent indentation and coding style. Many IDEs (such as Eclipse / NetBeans) can re-format your source codes with a single click.

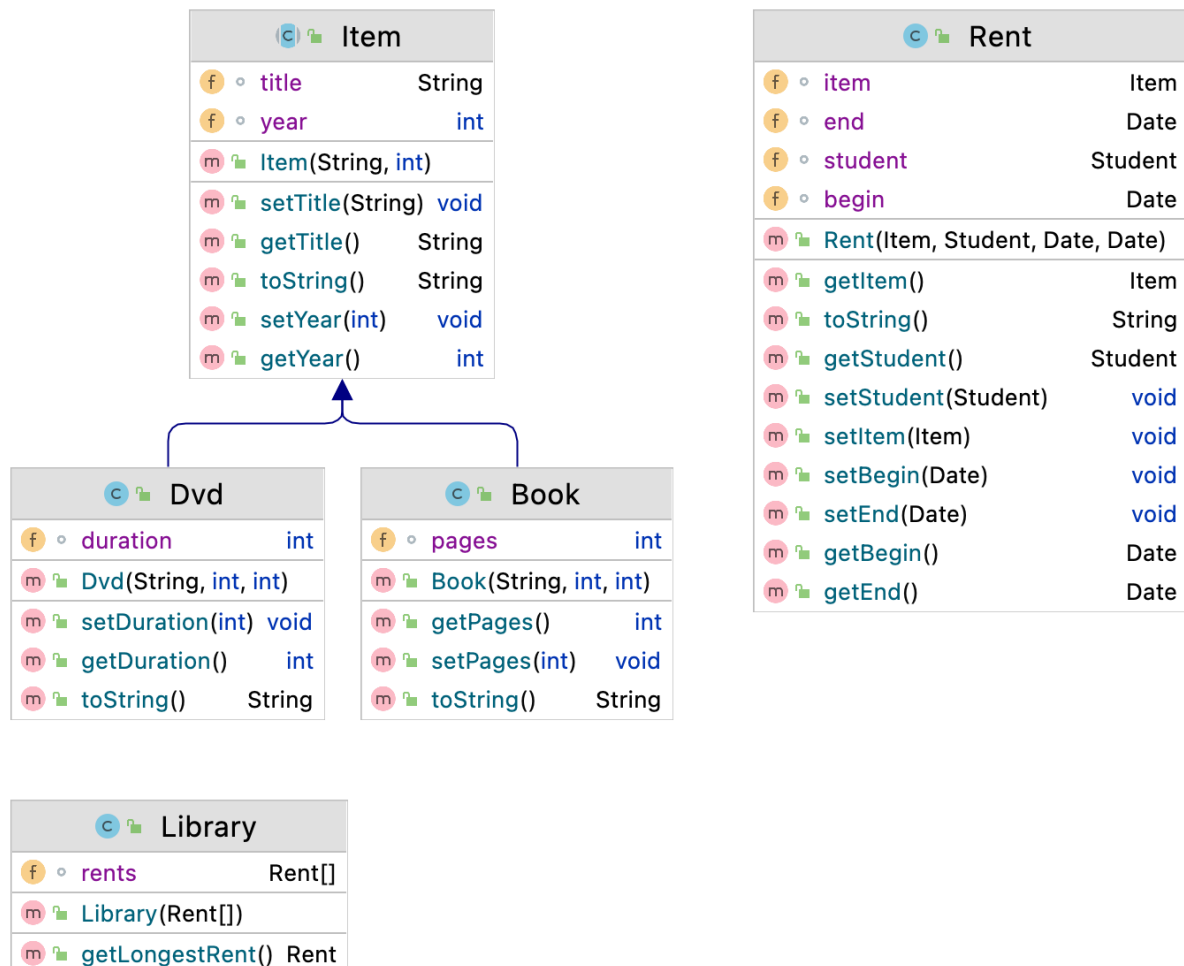
2. Program Documentation: Comment! Comment! and more Comment to explain your code to other people and to yourself three days later.

3. The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)).

1 Exercises on OOP

1.1 Library

Write code for an application designed as shown in the following class diagram.



```

1 package com.oop.library;
2
3 /**
4  * Class generalizing Books and DVDs
5  */
6 public abstract class Item {
7     String title;
8     int year;
9
10    public Item(String title, int year) {
11
12    }
13 }
  
```



```
11      /* TODO */  
12      }  
13  
14      /* TODO */  
15  
16      @Override  
17      public String toString() {  
18          return "Item[" + "title=" + title + '\'  
19              + ", year=" + year + ']'';  
20      }  
21  }
```



```
1  package com.oop.library;  
2  
3  /**  
4   * Class representing a DVD  
5   */  
6  public class Dvd extends Item {  
7      int duration;  
8  
9      public Dvd(String title, int year, int duration) {  
10         /* TODO */  
11     }  
12  
13     /* TODO */  
14  
15     @Override  
16     public String toString() {  
17         return "Dvd[" + "duration=" + duration  
18             + ", title=" + title + '\'  
19             + ", year=" + year + ']'';  
20     }  
21 }
```



```
1  package com.oop.library;  
2  
3  /**  
4   * Class representing a Book  
5   */  
6  public class Book extends Item {  
7      int pages;  
8  
9      public Book(String title, int year, int pages) {  
10         /* TODO */  
11     }  
12 }
```



```

13      /* TODO */

15      @Override
16      public String toString() {
17          return "Book[" + "pages=" + pages
18                  + ", title=" + title + '\',\n'
19                  + ", year=" + year + ']' + '\n';
20      }
21  }

```



```

1  package com.oop.library;

3  import java.util.Date;

5  /**
6   * Rent implements a rent of an Item for a delimited time frame
7   */
8  public class Rent {
9      Item item;
10     Student student;
11     Date begin;
12     Date end;

13     public Rent(Item item, Student student, Date begin, Date end) {
14         /* TODO */
15     }

16     /* TODO */

17     @Override
18     public String toString() {
19         return "Rent[" + "item=" + item
20                 + ", student=" + student
21                 + ", begin=" + begin
22                 + ", end=" + end + ']' + '\n';
23     }
24 }
25
26
27

```



```

1  package com.oop.library;

3  import java.util.Objects;

5  public class Student implements Comparable<Student> {
6      String name;

```



```

7      String lastname;
      String phone;
9      double average;

11     public Student(String name, String lastname, String phone) {
        /* TODO */
13     }

15     public Student(String name, String lastname, double average) {
        /* TODO */
17     }

19     public Student(String name, String lastname, String phone, double
        ↪ average) {
        /* TODO */
21     }

23     /* TODO */

25     @Override
    public int compareTo(Student s) {
27         /* TODO */
    }

29

31     @Override
    public boolean equals(Object o) {
        if (this == o) {
33             return true;
        }

35         if (o == null || getClass() != o.getClass()) {
37             return false;
        }

39         Student student = (Student) o;
41         return Double.compare(student.average, average) == 0
            && Objects.equals(name, student.name)
43             && Objects.equals(lastname, student.lastname)
            && Objects.equals(phone, student.phone);
45     }

47     @Override
    public int hashCode() {
49         return Objects.hash(name, lastname, phone, average);
    }

51

53     @Override
    public String toString() {
        return "Student[" +
55         "name=" + name + '\n' +
        ", lastname=" + lastname + '\n' +
57         ", phone=" + phone + '\n' +

```



```

59         ", average=" + average +
        ']' ,';
61     }

```



```

1  package com.oop.library;

3  import java.text.ParseException;
   import java.text.SimpleDateFormat;
5  import java.util.Locale;

7  /**
   * Implement the classes described in UML diagram.
9  * Test them with the following main.
   */
11 public class TestApp {
    public static void main(String[] args) throws ParseException {
13         Item i1 = new Book("Soffocare", 2002, 170);
        Item i2 = new Dvd("Moon", 2011, 130);

15         Student s1 = new Student("0001", "Darrell", "Abbott");
        Student s2 = new Student("0002", "Nick", "Drake");

17         SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy", Locale.
            ↪ ENGLISH);
        Rent[] rents = new Rent[5];
21         rents[0] = new Rent(i1, s1, sdf.parse("15/06/2020"), sdf.parse("
            ↪ 15/07/2020"));
        rents[1] = new Rent(i1, s2, sdf.parse("10/07/2020"), sdf.parse("
            ↪ 20/07/2020"));
23         rents[2] = new Rent(i1, s1, sdf.parse("25/08/2020"), sdf.parse("
            ↪ 14/11/2020"));
        rents[3] = new Rent(i2, s2, sdf.parse("10/07/2020"), sdf.parse("
            ↪ 20/07/2020"));
25         rents[4] = new Rent(i2, s1, sdf.parse("25/08/2020"), sdf.parse("
            ↪ 28/08/2020"));

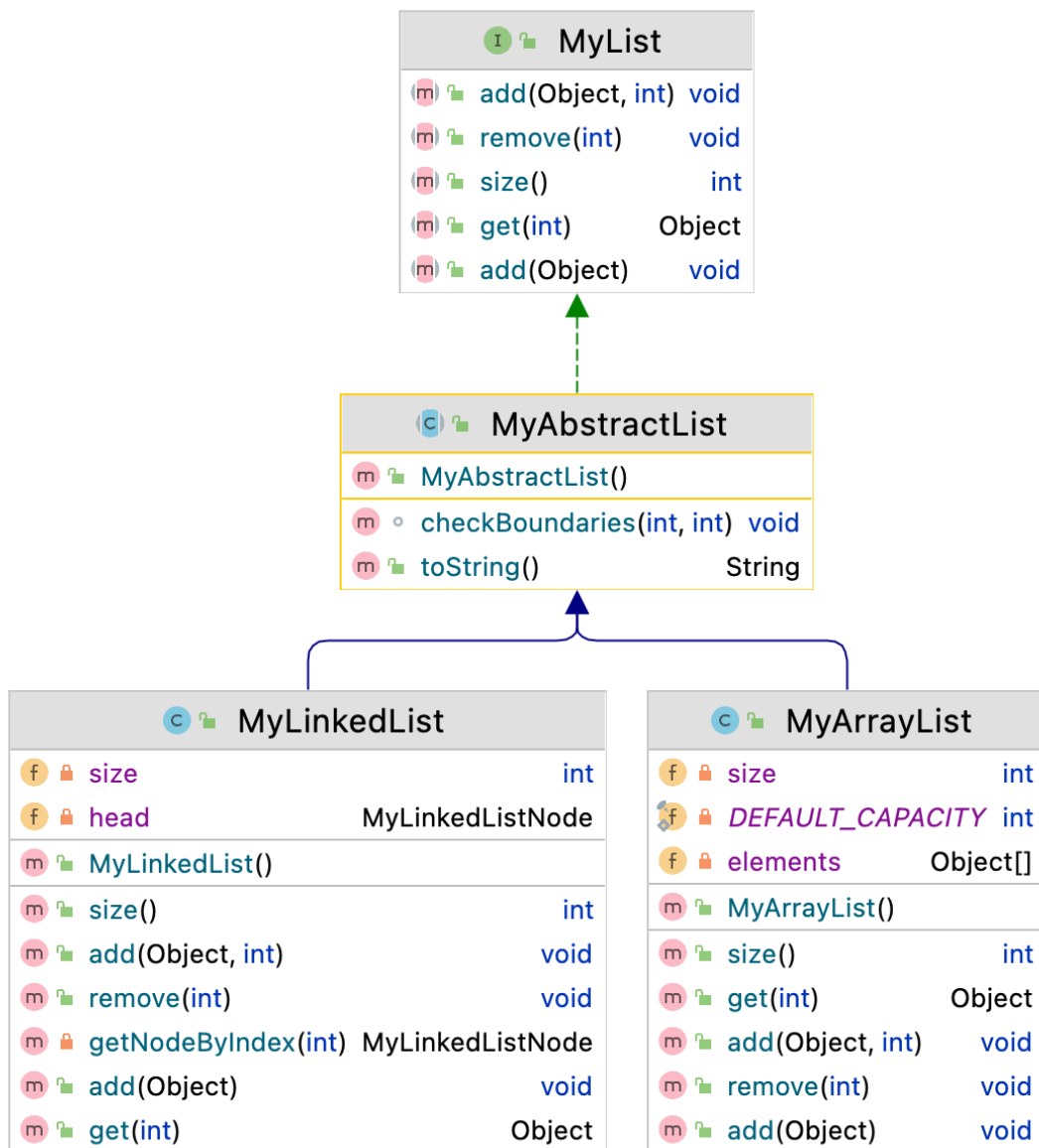
27         Library l = new Library(rents);
        System.out.println(l.getLongestRent());
29     }
}

```

2 Exercises on Collections

2.1 MyList

Write code for an application designed as shown in the following class diagram.



```

1 package com.oop.collections.mylis;
2
3 public interface MyList {
4     void add(Object o);
5 }
  
```



```
void add(Object o, int index);
6 void remove(int index);
Object get(int index);
8 int size();
}
```



```
1 package com.oop.collections.mylis;

3 public abstract class MyAbstractList implements MyList {

5     void checkBoundaries(int index, int limit) {
        if (index < 0 || index > limit) {
7         throw new ArrayIndexOutOfBoundsException();
        }
9     }

11    @Override
    public String toString() {
13        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < size(); i++) {
15            sb.append(String.format("%s", get(i).toString()));
        }
17        return sb.toString();
    }
19 }
```



```
1 package com.oop.collections.mylis;

3 /**
 * Implementation of a simplified ArrayList
5 */
public class MyArrayList extends MyAbstractList {
7     static final int INITIAL_SIZE = 16;
    Object[] elements;
9     int size;

11    public MyArrayList() {
        elements = new Object[INITIAL_SIZE];
13        size = 0;
    }

15    @Override
    public void add(Object o) {
17        if (size >= elements.length - 1) {
19            enlarge();
        }
    }
}
```




```

21     }
    elements[size++] = o;
    }
23
    @Override
25     public void add(Object o, int index) {
        /* TODO */
27     }

    @Override
29     public Object get(int index) {
31         checkBoundaries(index, size - 1);
        return elements[index];
33     }

    @Override
35     public void remove(int index) {
37         /* TODO */
    }
39

    @Override
41     public int size() {
        /* TODO */
43     }

45     void enlarge() {
        Object[] tmp = new Object[elements.length * 2];
47         System.arraycopy(elements, 0, tmp, 0, elements.length);
        elements = tmp;
49     }
}

```



```

package com.oop.collections.mylist;

2
/**
4  * Implementation of a single node composing the linked list
  */
6  public class MyLinkedListNode {
    Object payload;
8    MyLinkedListNode next;

10     public MyLinkedListNode(Object payload) {
        /* TODO */
12     }

14     public MyLinkedListNode(Object payload, MyLinkedListNode next) {
        /* TODO */
16     }
}

```



```
18     public Object getPayload() {
19         /* TODO */
20     }
21
22     public void setPayload(Object dataValue) {
23         /* TODO */
24     }
25
26     public MyLinkedListNode getNext() {
27         /* TODO */
28     }
29
30     public void setNext(MyLinkedListNode nextValue) {
31         /* TODO */
32     }
33 }
```



```
1  package com.oop.collections.mylis;
2
3  /**
4   * Implementation of a simplified LinkedList class
5   */
6  public class MyLinkedList extends MyAbstractList {
7      MyLinkedListNode head;
8      int size;
9
10     public MyLinkedList() {
11         head = null;
12         size = 0;
13     }
14
15     private MyLinkedListNode getNodeByIndex(int index) {
16         MyLinkedListNode current = head;
17         for (int i = 0; i < index; i++) {
18             current = current.getNext();
19         }
20         return current;
21     }
22
23     @Override
24     public void add(Object o) {
25         add(o, size);
26     }
27
28     @Override
29     public void add(Object o, int index) {
30         checkBoundaries(index, size);
31     }
32 }
```



```

31     if (index == 0) {
32         head = new MyLinkedListNode(o, head);
33     } else {
34         MyLinkedListNode current = getNodeByIndex(index - 1);
35         current.setNext(new MyLinkedListNode(o, current.getNext()));
36     }
37     size++;
38 }
39
40 @Override
41 public Object get(int index) {
42     /* TODO */
43 }
44
45 @Override
46 public void remove(int index) {
47     /* TODO */
48 }
49
50 @Override
51 public int size() {
52     /* TODO */
53 }
54 }

```



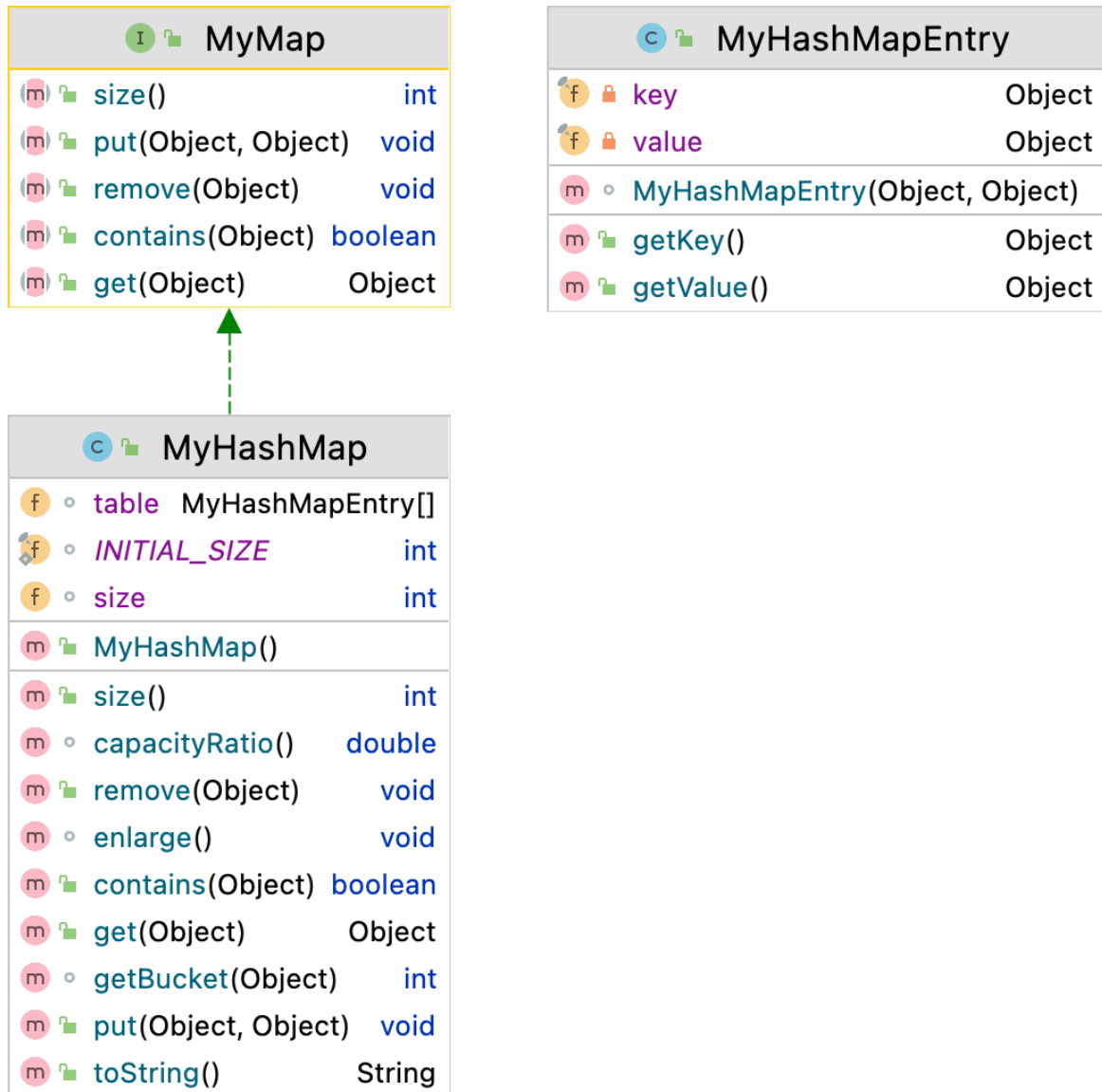
```

1 package com.oop.collections.mylist;
2
3 /**
4  * Provide an implementation of the MyList interface
5  * test it with the main() function reported below
6  */
7
8 public class TestApp {
9     public static void main(String[] args) {
10         MyList l = new MyArrayList();
11         l.add("a", 0);
12         l.add("b");
13         l.add("c", 0);
14         l.add("c", 3);
15         l.remove(3);
16         System.out.println(l.size());
17         System.out.println(l);
18     }
19 }

```

2.2 MyMap

Write code for an application designed as shown in the following class diagram.



```

1 package com.oop.collections.mymap;
2
3 public interface MyMap {
4     Object get(Object key);
5     void put(Object key, Object value);
6     void remove(Object key);
  
```



```
boolean contains(Object key);
8  int size();
}
```



```
1  package com.oop.collections.mymap;

3  public class MyHashMapEntry {
    private final Object key;
5    private final Object value;

7    MyHashMapEntry(Object key, Object value) {
        this.key = key;
9        this.value = value;
    }

11   public Object getKey() {
12       return key;
13   }

15   public Object getValue() {
16       return value;
17   }
19 }
```



```
1  package com.oop.collections.mymap;

3  /**
   * Implementation of a simplified HashMap
5  */
   public class MyHashMap implements MyMap {
7       final static int INITIAL_SIZE = 8;
       MyHashMapEntry[] table;
9       int size = 0;

11      public MyHashMap() {
12          table = new MyHashMapEntry[INITIAL_SIZE];
13      }

15      public Object get(Object key) {
16          int bucket = getBucket(key);
17          if (table[bucket] != null) {
18              return table[bucket].getValue();
19          }
20          return -1;
21      }
}
```



```

23     public void put(Object key, Object value) {
24         if (capacityRatio() > 0.6) {
25             enlarge();
26         }
27         int bucket = getBucket(key);
28         if (table[bucket] == null) {
29             table[bucket] = new MyHashMapEntry(key, value);
30             size++;
31         }
32     }
33
34     @Override
35     public void remove(Object key) {
36         /* TODO */
37     }
38
39     @Override
40     public boolean contains(Object o) {
41         /* TODO */
42     }
43
44     @Override
45     public int size() {
46         return size;
47     }
48
49     @Override
50     public String toString() {
51         StringBuilder sb = new StringBuilder();
52         for (int i = 0; i < table.length; i++) {
53             if (table[i] == null) {
54                 sb.append(String.format("[bucket %d] -> null\n", i));
55             } else {
56                 sb.append(String.format("[bucket %d] -> (%s, %s)\n", i, table[i]
57                     ↪ .getKey(), table[i].getValue()));
58             }
59         }
60         return sb.toString();
61     }
62
63     int getBucket(Object key) {
64         int bucket = (Math.abs(key.hashCode()) % table.length);
65         while (table[bucket] != null && table[bucket].getKey() != key) {
66             bucket = (bucket + 1) % table.length;
67         }
68         return bucket;
69     }
70
71     double capacityRatio() {
72         return size / (double)table.length;
73     }

```



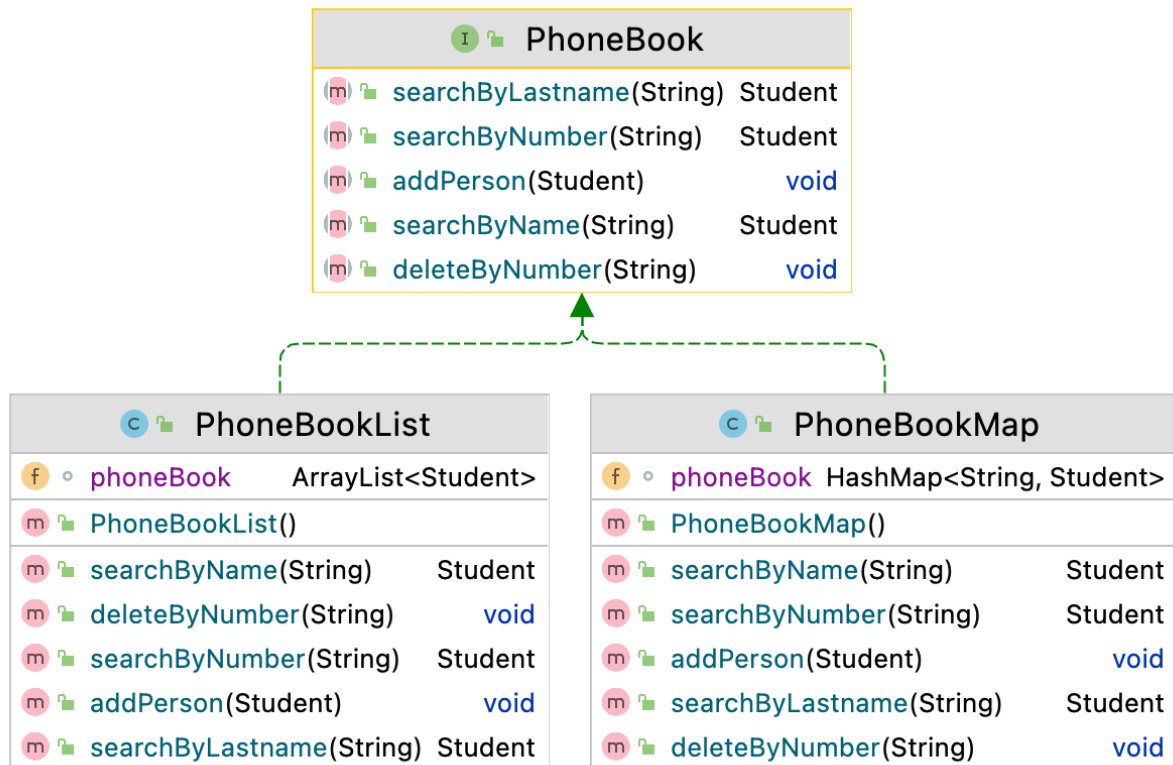
```
73     void enlarge() {  
75         /* TODO */  
77     }
```



```
1  package com.oop.collections.mymap;  
  
3  /**  
   * Provide an implementation of the MyMap interface  
   * test it with the main() function reported below  
   */  
7  public class TestApp {  
    public static void main(String[] args) {  
9      MyMap m = new MyHashMap();  
      m.put(11, "Nicola");  
11     m.put(11, "Nicola");  
      m.put(21, "Marzia");  
13     m.put(33, "Agata");  
      m.remove(21);  
15     System.out.println(m.contains(11));  
      System.out.println(m.contains(77));  
17     System.out.println(m.size());  
      System.out.println(m);  
19     }  
}
```

2.3 PhoneBook

Write code for an application designed as shown in the following class diagram.



```

1 package com.oop.collections.phonebook;
2
3 import java.util.Objects;
4
5 public class Student implements Comparable<Student> {
6     /* TODO */
7
8     public Student(String name, String lastname, String phone) {
9         /* TODO */
10    }
11
12    public Student(String name, String lastname, double average) {
13        /* TODO */
14    }
15
16    public Student(String name, String lastname, String phone, double
17        ↪ average) {
18        /* TODO */
19    }
20 }
  
```




```

20      /* TODO */

22      @Override
      public int compareTo(Student s) {
24          /* TODO */
      }

26

28      @Override
      public int hashCode() {
          return Objects.hash(name, lastname, phone, average);
30      }

32      @Override
      public boolean equals(Object o) {
34          /* TODO */
      }

36

38      @Override
      public String toString() {
          return "Student[" +
40              "name=" + name + '\ ' +
              ", lastname=" + lastname + '\ ' +
42              ", phone=" + phone + '\ ' +
              ", average=" + average +
44              ']' ;
      }
46  }

```



```

      package com.oop.collections.phonebook;

2
      /**
4       * Interface representing a generic PhoneBook Implementing classes
      * must provide methods for inserting, deleting and searching persons
6       * within the PhoneBook
      */
8       public interface PhoneBook {
          /**
10          * Add a person to the PhoneBook
          *
12          * @param p The person to be added to the PhoneBook
          */
14          void addPerson(Student p);

16          /**
          * Search a person within the PhoneBook by name
18          *
          * @param name The name to be searched

```



```

20     * @return The person found, null otherwise
21     */
22     Student searchByName(String name);

24     /**
25     * Search a person within the PhoneBook by lastname
26     *
27     * @param lastname The lastname to be searched
28     * @return The person found, null otherwise
29     */
30     Student searchByLastname(String lastname);

32     /**
33     * Search a person within the PhoneBook by number
34     *
35     * @param phone The phone to be searched
36     * @return The person found, null otherwise
37     */
38     Student searchByNumber(String phone);

40     /**
41     * Delete a person from the PhoneBook
42     *
43     * @param phone The phone number to be searched.
44     */
45     void deleteByNumber(String phone);
46 }

```



```

package com.oop.collections.phonebook;

2
import java.util.ArrayList;

4
/**
6  * A PhoneBook implementation internally using ArrayList. Slow!
7  *
8  */
9  public class PhoneBookList implements PhoneBook {
10     ArrayList<Student> phoneBook;

12     public PhoneBookList() {
13         phoneBook = new ArrayList<>();
14     }

16     @Override
17     public void addPerson(Student p) {
18         /* TODO */
19     }
20 }

```



```
22  @Override
    public Student searchByName(String name) {
    24      /* TODO */
    }

    26  @Override
    public Student searchByLastname(String lastname) {
    28      /* TODO */
    }

    30  @Override
    32  public Student searchByNumber(String phone) {
    34      /* TODO */
    }

    36  @Override
    38  public void deleteByNumber(String phone) {
    40      /* TODO */
    }
```



```
package com.oop.collections.phonebook;

2  import java.util.HashMap;

4  /**
6   * A PhoneBook implementation internally using HashMap
   */
8  public class PhoneBookMap implements PhoneBook {
    HashMap<String, Student> phoneBook;

    10  public PhoneBookMap() {
    12      /* TODO */
    }

    14  // We use the phone number as key because it is unique
    16  public void addPerson(Student s) {
    18      /* TODO */
    }

    20  public Student searchByName(String name) {
    22      /* TODO */
    }

    24  public Student searchByLastname(String lastname) {
    26      /* TODO */
    }
```



```
28 public Student searchByNumber(String phone) {
    /* TODO */
30 }

32 public void deleteByNumber(String phone) {
    /* TODO */
34 }
}
```



```
1 package com.oop.collections.phonebook;

3 /**
 * The PhoneBook Interface defines the functionalities of a
5 * basic phone book.
 *
7 * Provide two different implementations of the PhoneBook interface
 * working with the use case below. The first, (a) internally uses
9 * an ArrayList, the second (b) internally uses an HashMap.
 */

11 public class TestApp {
13     public static void main(String[] args) {
        PhoneBook pb = new PhoneBookMap();

15         // Comment this line for switching implementation
17         // PhoneBook pb = new PhoneBookArray();
        pb.addPerson(new Student("Nicola", "Bicocchi", "34567"));
19         pb.addPerson(new Student("Marco", "Rizzo", "45243"));
        pb.addPerson(new Student("Luisa", "Poppi", "24564"));

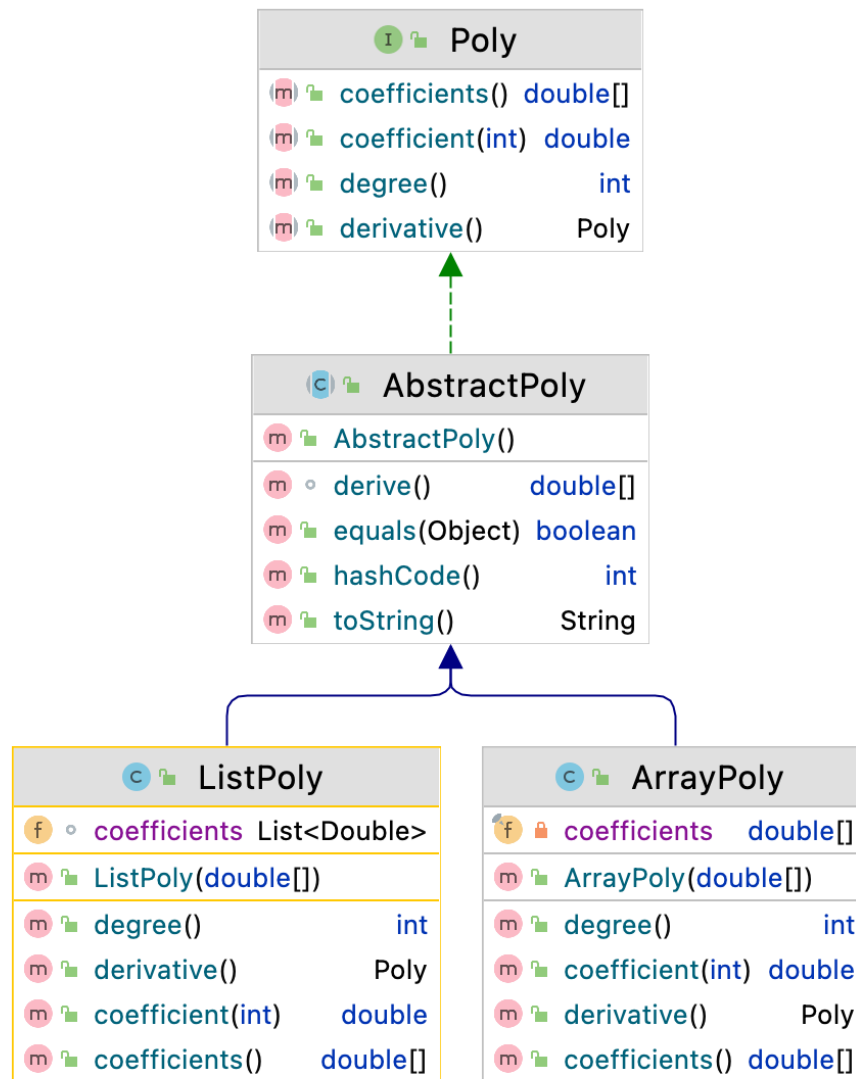
21         System.out.println(pb.searchByName("Marco"));
23         System.out.println(pb.searchByLastname("Poppi"));

25         // Do not exist!
        System.out.println(pb.searchByNumber("1111"));

27         // Delete an element!
29         pb.deleteByNumber("24564");
        System.out.println(pb.searchByLastname("Poppi"));
31     }
}
```

2.4 Polynomials

Write code for an application designed as shown in the following class diagram.



```

package com.oop.collections.polynomials;

/**
 * Interface representing a polynomial with arbitrary grade
 */
public interface Poly {

    /**
     * Returns the degree of the polynomial
     */

```



```

10  *
11  * @return the degree of the polynomial
12  */
13  int degree();
14
15  /**
16  * Returns an new polynomial which is the derivative of the current
17  * object. The call invoked on object (1), returns object (2)
18  * (1)  $c_0 + c_1 * x + \dots + c_n * x^n$ 
19  * (2)  $c_1 + 2c_2 * x + \dots + nc_n * x^{(n-1)}$ 
20  *
21  * @return Returns an new polynomial which is the derivative of the
22  * current object
23  */
24  Poly derivative();
25
26  /**
27  * Returns the coefficient of the monomial with the specified degree
28  *
29  * @param degree The degree to be queried (get the coefficient)
30  * @return The coefficient of the monomial with the specified degree
31  */
32  double coefficient(int degree);
33
34  /**
35  * Returns a double[] containing all the coefficients
36  *
37  * @return A double[] containing all the coefficients
38  */
39  double[] coefficients();
40  }

```



```

package com.oop.collections.polynomials;

2  import java.util.Objects;

4
5  /**
6  * An abstract class providing an implementation for shared parts of
7  *   ↪ ArrayPoly
8  *   and ListPoly
9  */
10 public abstract class AbstractPoly implements Poly {
11
12     double[] derive() {
13         /* TODO */
14     }
15
16     @Override

```



```
16 public boolean equals(Object o) {
17     /* TODO */
18 }
19
20 @Override
21 public int hashCode() {
22     return Objects.hashCode(coefficients());
23 }
24
25 @Override
26 public String toString() {
27     /* TODO */
28 }
29 }
```



```
1 package com.oop.collections.polynomials;
2
3 /**
4  * Class representing a polynomial with coefficients stored as on
5  * array of doubles
6  */
7 public class ArrayPoly extends AbstractPoly {
8     private final double[] coefficients;
9
10    public ArrayPoly(double[] coefficients) {
11        /* TODO */
12    }
13
14    @Override
15    public int degree() {
16        /* TODO */
17    }
18
19    @Override
20    public Poly derivative() {
21        /* TODO */
22    }
23
24    @Override
25    public double coefficient(int degree) {
26        /* TODO */
27    }
28
29    @Override
30    public double[] coefficients() {
31        /* TODO */
32    }
33 }
```



```

1  package com.oop.collections.polynomials;

3  import java.util.ArrayList;
   import java.util.List;

5

7  /**
   * Class representing a polynomial with coefficients stored as a list
   */
9  public class ListPoly extends AbstractPoly {
   List<Double> coefficients;

11

13  public ListPoly(double[] coeffs) {
   /* TODO */
   }

15

17  @Override
   public int degree() {
   /* TODO */
   }

19

21  @Override
   public Poly derivative() {
   /* TODO */
   }

23

25

27  @Override
   public double coefficient(int degree) {
   /* TODO */
   }

29

31  @Override
   public double[] coefficients() {
   /* TODO */
   }

33  }

35  }

```



```

1  package oop.collections.exercises.polynomials;

3  /**
   * Develop two classes, namely ArrayPoly and ListPoly,
5  * for managing polynomials.
   * More specifically, the two classes must exhibit the same
7  * functionalities (they both implement the Poly Interface) but
   * using different mechanisms internally.
9  * <p>
   * Given a generic polynomial,  $c_0 + c_1 * x + \dots + c_n * x^n$ 
11 * <p>
   * ArrayPoly stores a polynomial using double[] (c0..c_n).

```




```
13  * <p>
14  * ListPoly, stores a polynomial using List<Double> (c0..c_n).
15  * <p>
16  * Both classes must work with the main method provided below.
17  */
18  public class TestApp {
19      public static void main(String[] args) {
20
21          Poly ap = new ArrayPoly(new double[]{1, 3, 4, 8});
22          Poly lp = new ListPoly(new double[]{1, 3, 4, 8});
23
24          System.out.println("ap = " + ap);
25          System.out.println("lp = " + lp);
26
27          if (ap.equals(lp) && lp.equals(ap))
28              System.out.println("ap == lp");
29          else
30              System.out.println("ap != lp");
31
32          ap = ap.derivative();
33          System.out.println("ap' = " + ap.toString());
34
35          ap = ap.derivative();
36          System.out.println("ap'' = " + ap.toString());
37
38          lp = lp.derivative();
39          System.out.println("lp' = " + lp.toString());
40
41          lp = lp.derivative();
42          System.out.println("lp'' = " + lp.toString());
43      }
44  }
```