Lab 9. OOP and Collections Framework Exercises

Writing Good Programs

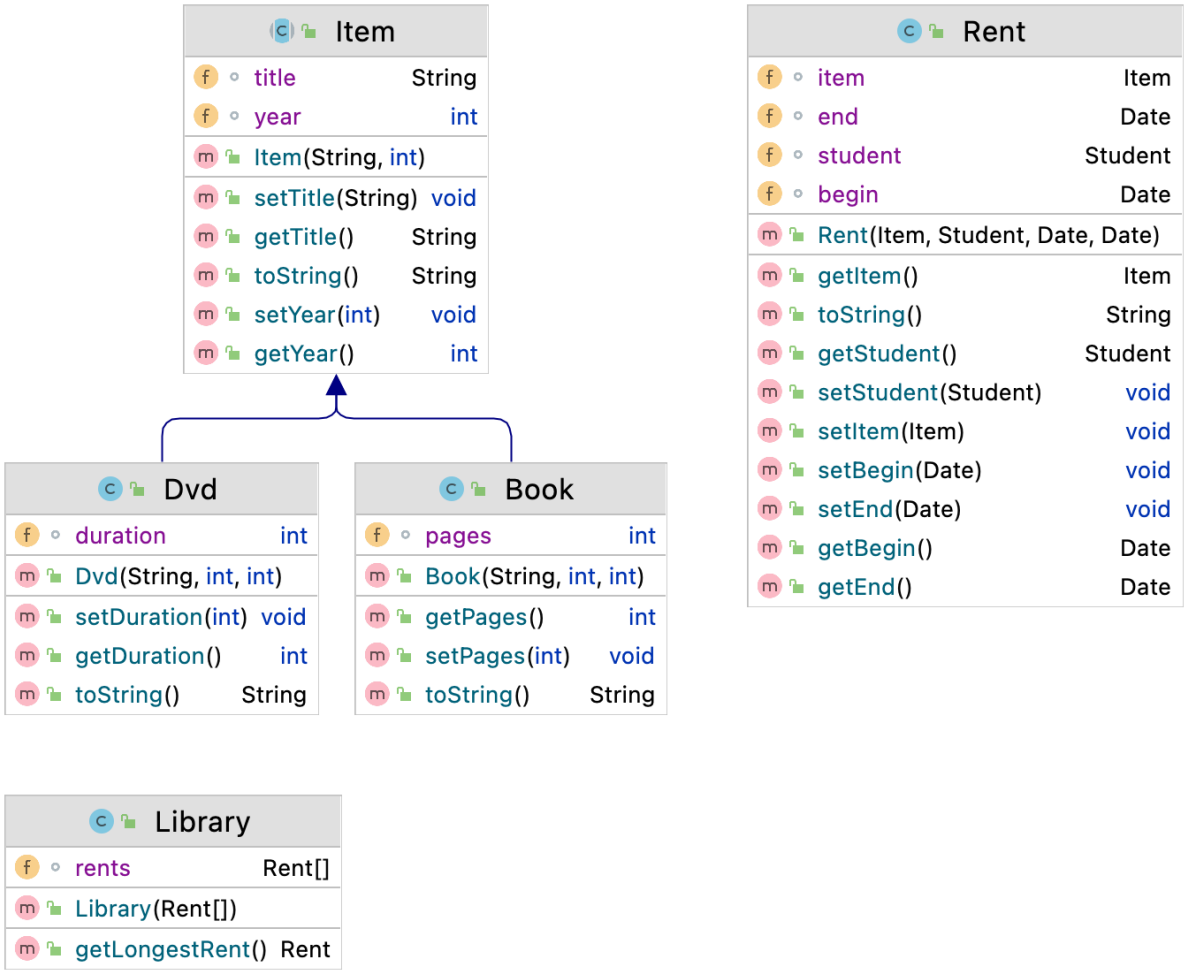
The only way to learn programming is program, program and program. Learning programming is like learning cycling, swimming or any other sports. You can't learn by watching or reading books. Start to program immediately. On the other hands, to improve your programming, you need to read many books and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




```java
package com.oop.library;

/**
 * Class generalizing Books and DVDs
 */
public abstract class Item {
  String title;
  int year;

  public Item(String title, int year) {
```

```
    /* TODO */
  }

  /* TODO */

  @Override
  public String toString() {
    return "Item[" + "title='" + title + '\''
      + ", year=" + year + ']';
  }
}
```

```
package com.oop.library;

/**
 * Class representing a DVD
 */
public class Dvd extends Item {
  int duration;

  public Dvd(String title, int year, int duration) {
    /* TODO */
  }

  /* TODO */

  @Override
  public String toString() {
    return "Dvd[" + "duration=" + duration
      + ", title='" + title + '\''
      + ", year=" + year + ']';
  }
}
```

```
package com.oop.library;

/**
 * Class representing a Book
 */
public class Book extends Item {
  int pages;

  public Book(String title, int year, int pages) {
```

```
    /* TODO */
  }

  /* TODO */

  @Override
  public String toString() {
    return "Book[" + "pages=" + pages
      + ", title='" + title + '\''
      + ", year=" + year + ']';
  }
}
```

```
package com.oop.library;

import java.util.Date;

/**
 * Rent implements a rent of an Item for a delimited time frame
 */
public class Rent {
  Item item;
  Student student;
  Date begin;
  Date end;

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

  /* TODO */

  @Override
  public String toString() {
    return "Rent[" + "item=" + item
      + ", student=" + student
      + ", begin=" + begin
      + ", end=" + end + ']';
  }
}
```

```
package com.oop.library;
```

```java
import java.util.Objects;

public class Student implements Comparable<Student> {
  String name;
  String lastname;
  String phone;
  double average;

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

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

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

  /* TODO */

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

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

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

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

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

  @Override
  public String toString() {
```

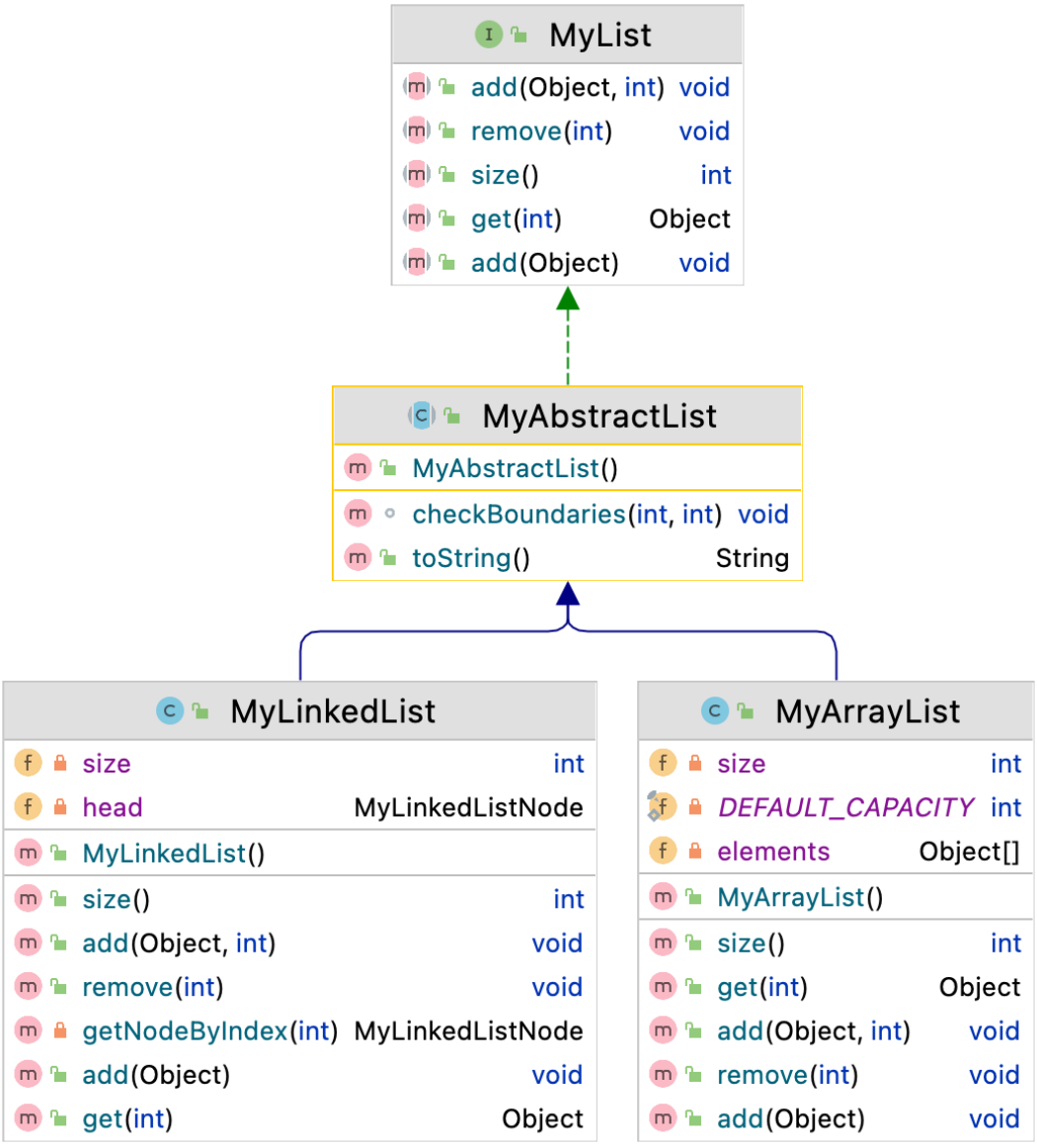
```
        return "Student[" +
55              "name='" + name + '\'' +
                ", lastname='" + lastname + '\'' +
57              ", phone='" + phone + '\'' +
                ", average=" + average +
59              ']';
    }
61 }
```

```
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");
17     Student s2 = new Student("0002", "Nick", "Drake");

19     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.




```java
package com.oop.collections.mylist;

public interface MyList {
  void add(Object o);
```

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

```
package com.oop.collections.mylist;

public abstract class MyAbstractList implements MyList {
    void checkBoundaries(int index, int limit) {
        if (index < 0 || index > limit) {
            throw new ArrayIndexOutOfBoundsException();
        }
    }

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

```
package com.oop.collections.mylist;

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

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

    @Override
    public void add(Object o) {
        if (size >= elements.length - 1) {
```

```
      enlarge();
    }
    elements[size++] = o;
  }

  @Override
  public void add(Object o, int index) {
    /* TODO */
  }

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

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

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

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

```
package com.oop.collections.mylist;

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

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

  public MyLinkedListNode(Object payload, MyLinkedListNode next) {
```

```
    /* TODO */
  }

  public Object getPayload() {
    /* TODO */
  }

  public void setPayload(Object dataValue) {
    /* TODO */
  }

  public MyLinkedListNode getNext() {
    /* TODO */
  }

  public void setNext(MyLinkedListNode nextValue) {
    /* TODO */
  }
}
```

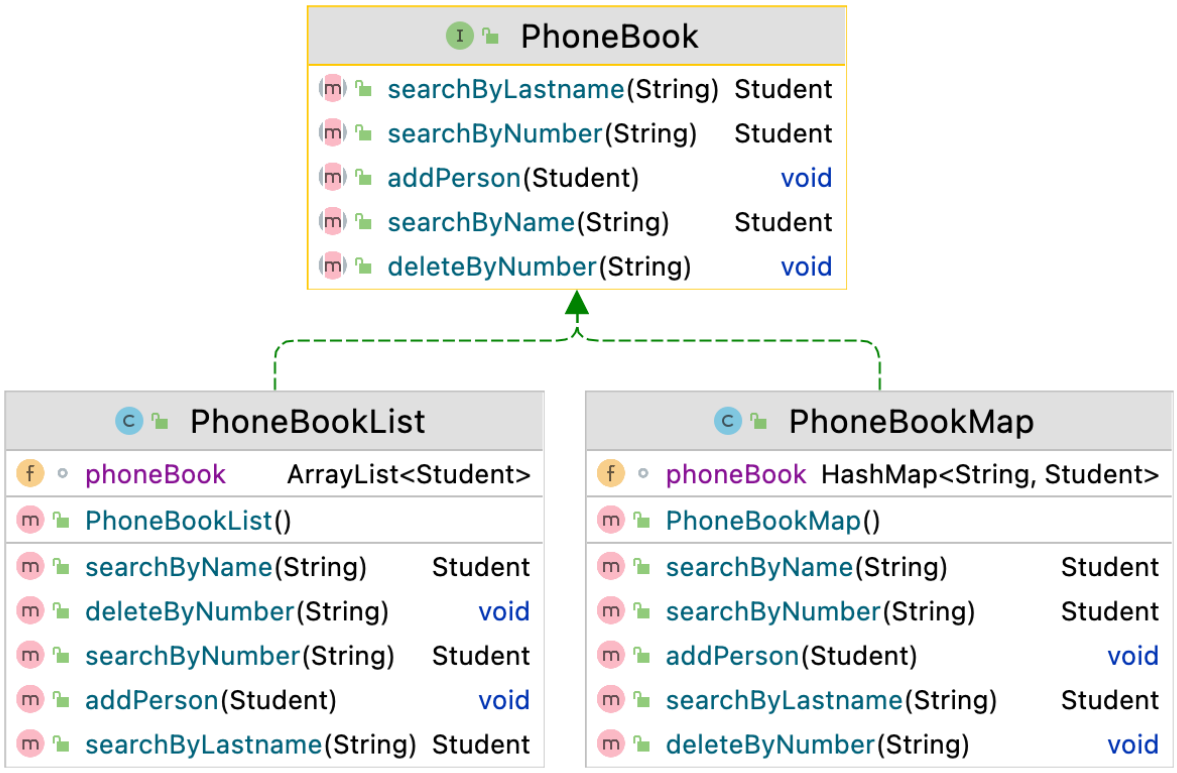
```
package com.oop.collections.mylist;

/**
 * Implementation of a simplified LinkedList class
 */
public class MyLinkedList extends MyAbstractList {
  MyLinkedListNode head;
  int size;

  public MyLinkedList() {
    head = null;
    size = 0;
  }

  private MyLinkedListNode getNodeByIndex(int index) {
    MyLinkedListNode current = head;
    for (int i = 0; i < index; i++) {
      current = current.getNext();
    }
    return current;
  }

  @Override
  public void add(Object o) {
    add(o, size);
  }

```

```
    @Override
29  public void add(Object o, int index) {
      checkBoundaries(index, size);
31    if (index == 0) {
        head = new MyLinkedListNode(o, head);
33    } else {
        MyLinkedListNode current = getNodeByIndex(index - 1);
35      current.setNext(new MyLinkedListNode(o, current.getNext()));
      }
37    size++;
    }
39
    @Override
41  public Object get(int index) {
      /* TODO */
43  }

45  @Override
    public void remove(int index) {
47    /* TODO */
    }
49
    @Override
51  public int size() {
      /* TODO */
53  }
}
```

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

2.2 PhoneBook

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




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

import java.util.Objects;

public class Student implements Comparable<Student> {
  /* TODO */

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

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

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

```
20    /* TODO */

22    @Override
      public int compareTo(Student s) {
24      /* TODO */
      }
26
      @Override
28    public int hashCode() {
        return Objects.hash(name, lastname, phone, average);
30    }

32    @Override
      public boolean equals(Object o) {
34      /* TODO */
      }
36
      @Override
38    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
     * @return The person found, null otherwise
     */
    Student searchByName(String name);

    /**
     * Search a person within the PhoneBook by lastname
     *
     * @param lastname The lastname to be searched
     * @return The person found, null otherwise
     */
    Student searchByLastname(String lastname);

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

    /**
     * Delete a person from the PhoneBook
     *
     * @param phone The phone number to be searched.
     */
    void deleteByNumber(String phone);
}
```

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

import java.util.ArrayList;

/**
 * A PhoneBook implementation internally using ArrayList. Slow!
 *
 */
public class PhoneBookList implements PhoneBook {
  ArrayList<Student> phoneBook;

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

  @Override
  public void addPerson(Student p) {
    /* TODO */
```

```
    }

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

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

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

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

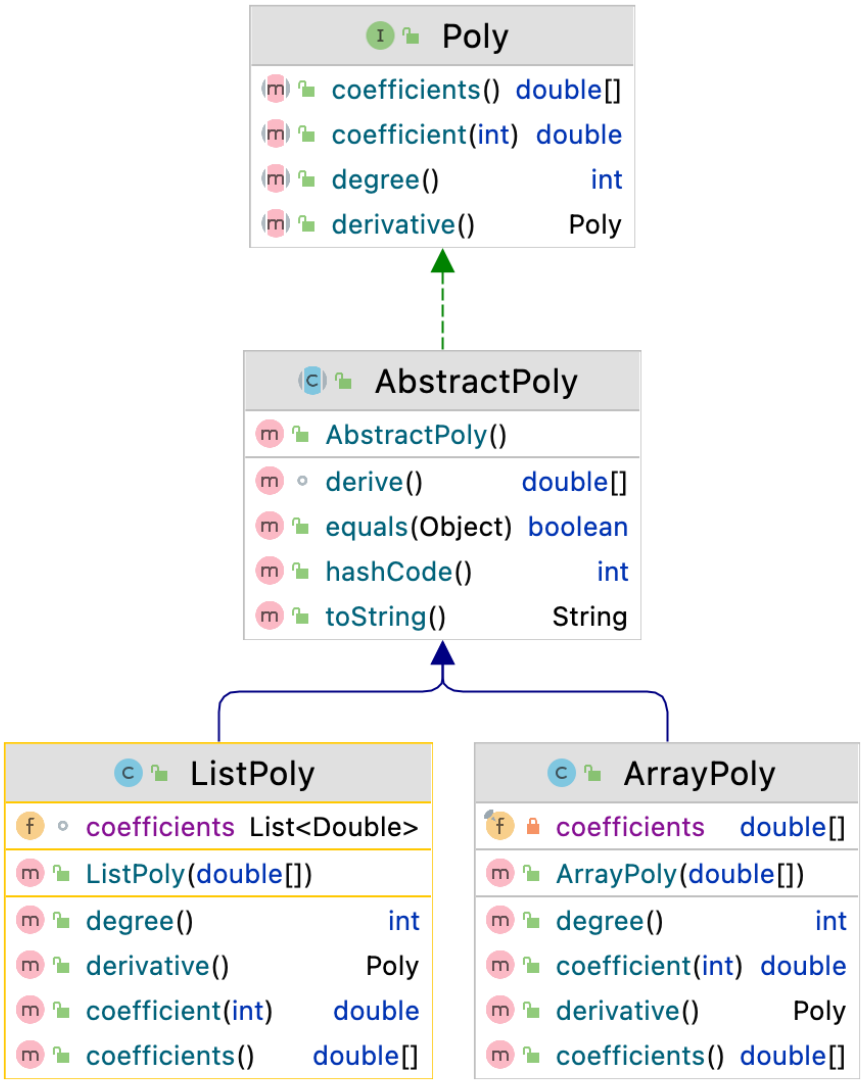
```
package com.oop.collections.phonebook;

import java.util.HashMap;

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

    public PhoneBookMap() {
      /* TODO */
    }

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

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

    public Student searchByLastname(String lastname) {
```

```
    /* TODO */
  }

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

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

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

/**
 * The PhoneBook Interface defines the functionalities of a
 *  basic phone book.
 *
 * Provide two different implementations of the PhoneBook interface
 *  working with the use case below. The first, (a) internally uses
 *  an Arraylist, the second (b) internally uses an HashMap.
 */
public class TestApp {
  public static void main(String[] args) {
    PhoneBook pb = new PhoneBookMap();

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

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

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

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

2.3 Polynomials

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




```java
package com.oop.collections.polynomials;

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

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

```
     *
     * @return the degree of the polynomial
     */
    int degree();

    /**
     * Returns an new polynomial which is the derivative of the current
     *  object. The call invoked on object (1), returns object (2)
     * (1) c0 + c_1 * x + ... + c_n * x^n
     * (2) c1 + 2c_2 * x + ... + nc_n * x^(n-1)
     *
     * @return Returns an new polynomial which is the derivative of the
     * current object
     */
    Poly derivative();

    /**
     * Returns the coefficient of the monomial with the specified degree
     *
     * @param degree The degree to be queried (get the coefficient)
     * @return The coefficient of the monomial with the specified degree
     */
    double coefficient(int degree);

    /**
     * Returns a double[] containing all the coefficients
     *
     * @return A double[] containing all the coefficients
     */
    double[] coefficients();
}
```

```
package com.oop.collections.polynomials;

import java.util.Objects;

/**
 * An abstract class providing an implementation for shared parts of
     ↪ ArrayPoly
 *  and ListPoly
 */
public abstract class AbstractPoly implements Poly {

    double[] derive() {
      /* TODO */
    }

```

```
    @Override
16  public boolean equals(Object o) {
      /* TODO */
18  }

20  @Override
    public int hashCode() {
22    return Objects.hashCode(coefficients());
    }
24
    @Override
26  public String toString() {
      /* TODO */
28  }
}
```

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

3  /**
    * Class representing a polynomial with coefficients stored as on
5   * array of doubles
    */
7  public class ArrayPoly extends AbstractPoly {
    private final double[] coefficients;
9
    public ArrayPoly(double[] coefficients) {
11    /* TODO */
    }
13
    @Override
15  public int degree() {
      /* TODO */
17  }

19  @Override
    public Poly derivative() {
21    /* TODO */
    }
23
    @Override
25  public double coefficient(int degree) {
      /* TODO */
27  }

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

```
    }
}
```

```java
package com.oop.collections.polynomials;

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

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

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

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

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

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

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

```java
package oop.collections.exercises.polynomials;

/**
 * Develop two classes, namely ArrayPoly and ListPoly,
```

```
 *  for managing polynomials.
 * More specifically, the two classes must exhibit the same
 *  functionalities (they both implement the Poly Interface) but
 *  using different mechanisms internally.
 * <p>
 * Given a generic polynomial, c0 + c_1 * x + ... + c_n * x^n
 * <p>
 * ArrayPoly stores a polynomial using double[] (c0..c_n).
 * <p>
 * ListPoly, stores a polynomial using List<Double> (c0..c_n).
 * <p>
 * Both classes must work with the main method provided below.
 */
public class TestApp {
  public static void main(String[] args) {

    Poly ap = new ArrayPoly(new double[]{1, 3, 4, 8});
    Poly lp = new ListPoly(new double[]{1, 3, 4, 8});

    System.out.println("ap = " + ap);
    System.out.println("lp = " + lp);

    if (ap.equals(lp) && lp.equals(ap))
    System.out.println("ap == lp");
    else
    System.out.println("ap != lp");

    ap = ap.derivative();
    System.out.println("ap' = " + ap.toString());

    ap = ap.derivative();
    System.out.println("ap'' = " + ap.toString());

    lp = lp.derivative();
    System.out.println("lp' = " + lp.toString());

    lp = lp.derivative();
    System.out.println("lp'' = " + lp.toString());
  }
}
```