Lab 8. OOP - Collection Framework

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 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 Exercise on Lists

```java
package oop.collections.exercises;

import java.util.*;

public class Lists {

  /**
  * Function to insert an element into a list at the beginning
  */
  public static void insertFirst(ArrayList<Integer> list, int value) {
    /* TODO */
  }

  /**
  * Function to insert an element into a list at the end
  */
  public static void insertLast(ArrayList<Integer> list, int value) {
    /* TODO */
  }

  /**
  * Function to replace the 3rd element of a list with a given value
  */
  public static void replace(ArrayList<Integer> list, int value) {
    /* TODO */
  }

  /**
  * Function to remove the 3rd element from a list
  */
  public static void removeThird(ArrayList<Integer> list) {
    /* TODO */
  }

  /**
  * Function to remove the element "666" from a list
  */
  public static void removeEvil(ArrayList<Integer> list) {
    /* TODO */
  }

  /**
  * Function returning a ArrayList<Integer> containing
  * the first 10 square numbers (i.e., 1, 4, 9, 16, ...)
  */
  public static ArrayList<Integer> generateSquare() {
    /* TODO */
  }
```

```java

    /**
     * Function to verify if a list contains a certain value
     */
    public static boolean contains(ArrayList<Integer> list, int value) {
        /* TODO */
    }

    /**
     * Function to copy a list into another list (without using library functions)
     * Note well: the target list must be emptied before the copy
     */
    public static void copy(ArrayList<Integer> source, ArrayList<Integer>
        ↪ target) {
        /* TODO */
    }

    /**
     * Function to reverse the elements of a list
     */
    public static void reverse(ArrayList<Integer> list) {
        /* TODO */
    }

    /**
     * Function to reverse the elements of a list (without using library functions)
     */
    public static void reverseManual(ArrayList<Integer> list) {
        /* TODO */
    }

    /**
     * Function to insert the same element both at the
     * beginning and the end of the same LinkedList
     * Note well: you can use LinkedList specific methods
     */
    public static void insertBeginningEnd(LinkedList<Integer> list, int
        ↪ value) {
        /* TODO */
    }
}
```

2 Exercise on Sets

```java
package oop.collections.exercises;

import java.util.*;

public class Sets {
  /**
  * Function returning the intersection of two given sets
  * (without using library functions)
  */
  public static Set<Integer> intersectionManual(Set<Integer> first , Set
     ↪ <Integer> second) {
    /* TODO */
  }

  /**
  * Function returning the union of two given sets
  * (without using library functions)
  */
  public static Set<Integer> unionManual(Set<Integer> first , Set<
     ↪ Integer> second) {
    /* TODO */
  }

  /**
  * Function returning the intersection of two given sets (see retainAll())
  */
  public static Set<Integer> intersection(Set<Integer> first , Set<
     ↪ Integer> second) {
    /* TODO */
  }

  /**
  * Function returning the union of two given sets (see addAll())
  */
  public static Set<Integer> union(Set<Integer> first , Set<Integer>
     ↪ second) {
    /* TODO */
  }

  /**
  * Function to transform a set into a list without duplicates
  * Note well: collections can be created from another collection!
  */
  public static List<Integer> toList(Set<Integer> source) {
    /* TODO */
  }

  /**
```

```
45      * Function to remove duplicates from a list
        * Note well: collections can be created from another collection!
47      */
        public static List<Integer> removeDuplicates(List<Integer> source) {
49        /* TODO */
        }
51
        /**
53      * Function to remove duplicates from a list
        * without using the constructors trick seen above
55      */
        public static List<Integer> removeDuplicatesManual(List<Integer>
            ↪ source) {
57        /* TODO */
        }
59
        /**
61      * Function accepting a string s
        * returning the first recurring character
63      * For example firstRecurringCharacter("abaco") -> a.
        */
65      public static String firstRecurringCharacter(String s) {
          /* TODO */
67      }

69      /**
        * Function accepting a string s,
71      * and returning a set comprising all recurring characters.
        * For example allRecurringChars("mamma") -> [m, a].
73      */
        public static Set<Character> allRecurringChars(String s) {
75        /* TODO */
        }
77
        /**
79      * Function to transform a set into an array
        */
81      public static Integer[] toArray(Set<Integer> source) {
          /* TODO */
83      }

85      /**
        * Function to return the first item from a TreeSet
87      * Note well: use TreeSet specific methods
        */
89      public static int getFirst(TreeSet<Integer> source) {
          /* TODO */
91      }

93      /**
        * Function to return the last item from a TreeSet
95      * Note well: use TreeSet specific methods
```

```
    */
97  public static int getLast(TreeSet<Integer> source) {
      /* TODO */
99  }

101 /**
    * Function to get an element from a TreeSet
103 * which is strictly greater than a given element.
    * Note well: use TreeSet specific methods
105 */
    public static int getGreater(TreeSet<Integer> source, int value) {
107   /* TODO */
    }
109 }
```

3 Exercise on Maps

```
1   package oop.collections.exercises;

3   import java.util.Collection;
    import java.util.HashMap;
5   import java.util.Map;
    import java.util.Set;
7
    public class Maps {
9     /**
      * Function to return the number of key-value mappings of a map
11    */
      public static int count(Map<Integer, Integer> map) {
13      /* TODO */
      }
15
      /**
17    * Function to remove all mappings from a map
      */
19    public static void empty(Map<Integer, Integer> map) {
        /* TODO */
21    }

23    /**
      * Function to test if a map contains a mapping for the specified key
25    */
      public static boolean contains(Map<Integer, Integer> map, int key) {
27      /* TODO */
      }
```

```java

    /**
     * Function to test if a map contains a mapping for
     * the specified key and if its value equals the
     * specified value
     */
    public static boolean containsKeyValue(Map<Integer, Integer> map, int key, int value) {
      /* TODO */
    }

    /**
     * Function to return the key set of map
     */
    public static Set<Integer> keySet(Map<Integer, Integer> map) {
      /* TODO */
    }

    /**
     * Function to return the values of a map
     */
    public static Collection<Integer> values(Map<Integer, Integer> map) {
      /* TODO */
    }

    /**
     * Function, internally using a map, returning "black",
     * "white", or "red" depending on int input value.
     * "black" = 0, "white" = 1, "red" = 2
     */
    public static String getColor(int value) {
      /* TODO */
    }
}
```

4 Exercise on Comparable vs Comparator

4.1 Comparable

A comparable object is capable of comparing itself with another object. The class itself must implements the java.lang.Comparable interface to compare its instances.

Consider a Movie class that has members like, rating, name, year. Suppose we wish to sort a list of Movies based on year of release. We can implement the Comparable interface with the Movie class, and we override the method compareTo() of Comparable interface.

```java
/**
* A Java program to demonstrate use of Comparable
*/
import java.io.*;
import java.util.*;

/**
* A class 'Movie' that implements Comparable
*/
class Movie implements Comparable<Movie> {
  private String name;
  private double rating;
  private int year;

  // Used to sort movies by year
  public int compareTo(Movie movie) {
    /* TODO */
  }

  // Constructor
  public Movie(String name, double rating, int year) {
    /* TODO */
  }

  // Getter methods for accessing private data
  public double getRating() {
    /* TODO */
  }

  public String getName() {
    /* TODO */
  }

  public int getYear() {
    /* TODO */
  }
}
```

```java
/**
* Comparable driver test class
*/
class ComparableTest {
  public static void main(String[] args) {
    List<Movie> list = new ArrayList<>();
    list.add(new Movie("Force Awakens", 8.3, 2015));
    list.add(new Movie("Star Wars", 8.7, 1977));
    list.add(new Movie("Empire Strikes Back", 8.8, 1980));
    list.add(new Movie("Return of the Jedi", 8.4, 1983));
```

```
11
        Collections.sort(list);
13
        System.out.println("Movies after sorting : ");
15      for (Movie movie: list) {
          System.out.println(movie.getName() + " " +
17        movie.getRating() + " " +
          movie.getYear());
19      }
      }
21  }
```

4.2 Comparator

Unlike Comparable, Comparator is external to the element type we are comparing. It's a separate class. We create multiple separate classes (that implement Comparator) to compare by different members. Collections class has a second sort() method and it takes Comparator. The sort() method invokes the compare() to sort objects.

To compare movies by Rating, we need to do 3 things:

1. Create a class that implements Comparator (and thus the compare() method that does the work previously done by compareTo()).
2. Make an instance of the Comparator class.
3. Call the overloaded sort() method, giving it both the list and the instance of the class that implements Comparator.

```
1   /**
    * A Java program to demonstrate Comparator interface
3   */
    import java.io.*;
5   import java.util.*;

7   /**
    * A class 'Movie' that implements Comparable
9   */
    class Movie implements Comparable<Movie> {
11    private String name;
      private double rating;
13    private int year;

15    // Used to sort movies by year
```

```java
    public int compareTo(Movie m) {
      /* TODO */
    }

    // Constructor
    public Movie(String name, double rating, int year) {
      /* TODO */
    }

    // Getter methods for accessing private data
    public double getRating() {
      /* TODO */
    }

    public String getName() {
      /* TODO */
    }

    public int getYear() {
      /* TODO */
    }
}
```

```java
/**
* Class to compare Movies by name
*/
class NameCompare implements Comparator<Movie> {
  public int compare(Movie left, Movie right) {
    /* TODO */
  }
}
```

```java
/**
* Class to compare Movies by ratings
*/
class RatingCompare implements Comparator<Movie> {
  public int compare(Movie left, Movie right) {
    /* TODO */
  }
}
```

```
/**
* Comparator driver test class
*/
class ComparatorTest {
  public static void main(String[] args) {
    List<Movie> list = new ArrayList<>();
    list.add(new Movie("Force Awakens", 8.3, 2015));
    list.add(new Movie("Star Wars", 8.7, 1977));
    list.add(new Movie("Empire Strikes Back", 8.8, 1980));
    list.add(new Movie("Return of the Jedi", 8.4, 1983));

    // Sort by rating : (1) Create an object of ratingCompare
    //                  (2) Call Collections.sort
    //                  (3) Print Sorted list
    System.out.println("Sorted by rating");
    RatingCompare ratingCompare = new RatingCompare();
    Collections.sort(list, ratingCompare);
    for (Movie movie: list) {
      System.out.println(movie.getRating() + " " +
                         movie.getName() + " " +
                         movie.getYear());
    }

    // Call overloaded sort method with RatingCompare
    // (Same three steps as above)
    System.out.println("\nSorted by name");
    NameCompare nameCompare = new NameCompare();
    Collections.sort(list, nameCompare);
    for (Movie movie: list) {
      System.out.println(movie.getName() + " " +
                         movie.getRating() + " " +
                         movie.getYear());
    }

    // Uses Comparable to sort by year
    System.out.println("\nSorted by year");
    Collections.sort(list);
    for (Movie movie: list) {
      System.out.println(movie.getYear() + " " +
                         movie.getRating() + " " +
                         movie.getName() + " ");
    }
  }
}
```