Lab 8. The Java Collections 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 hus.oop.collections.list;

import java.util.*;

public class Lists {

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

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

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

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

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

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

```java

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

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

    /**
     * Function to reverse the elements of a list (without using library functions)
     */
    public static void reverseManual(List<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 hus.oop.collections.set;

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 */
  }

  /**
```

```
48     * Function to remove duplicates from a list
       * Note well: collections can be created from another collection!
50     */
    public static List<Integer> removeDuplicates(List<Integer> source) {
52    /* TODO */
    }
54
    /**
56   * Function to remove duplicates from a list
       * without using the constructors trick seen above
58   */
    public static List<Integer> removeDuplicatesManual(List<Integer> source
       ↪ ) {
60    /* TODO */
    }
62
    /**
64   * Function accepting a string s
       * returning the first recurring character
66   * For example firstRecurringCharacter("abaco") -> a.
       */
68  public static String firstRecurringCharacter(String s) {
      /* TODO */
70  }

72  /**
     * Function accepting a string s,
74   * and returning a set comprising all recurring characters.
     * For example allRecurringChars("mamma") -> [m, a].
76   */
    public static Set<Character> allRecurringChars(String s) {
78    /* TODO */
    }
80
    /**
82   * Function to transform a set into an array
     */
84  public static Integer[] toArray(Set<Integer> source) {
      /* TODO */
86  }

88  /**
     * Function to return the first item from a TreeSet
90   * Note well: use TreeSet specific methods
     */
92  public static int getFirst(TreeSet<Integer> source) {
      /* TODO */
94  }

96  /**
     * Function to return the last item from a TreeSet
98   * Note well: use TreeSet specific methods
```

```
     */
100  public static int getLast(TreeSet<Integer> source) {
       /* TODO */
102  }

104  /**
       * Function to get an element from a TreeSet
106    * which is strictly greater than a given element.
       * Note well: use TreeSet specific methods
108    */
     public static int getGreater(TreeSet<Integer> source, int value) {
110    /* TODO */
     }
112 }
```

3 Exercise on Maps

```
     package hus.oop.collections.map;
2
     import java.util.Collection;
4    import java.util.HashMap;
     import java.util.Map;
6    import java.util.Set;

8    public class Maps {
       /**
10      * Function to return the number of key-value mappings of a map
        */
12     public static int count(Map<Integer, Integer> map) {
         /* TODO */
14     }

16     /**
        * Function to remove all mappings from a map
18      */
       public static void empty(Map<Integer, Integer> map) {
20       /* TODO */
       }
22
       /**
24      * Function to test if a map contains a mapping for the specified key
        */
26     public static boolean contains(Map<Integer, Integer> map, int key) {
         /* TODO */
28     }
```

```
30      /**
         * Function to test if a map contains a mapping for
32       * the specified key and if its value equals the specified value
         */
34      public static boolean containsKeyValue(Map<Integer, Integer> map,
                                               int key,
36                                             int value) {
          /* TODO */
38      }

40      /**
         * Function to return the key set of map
42       */
        public static Set<Integer> keySet(Map<Integer, Integer> map) {
44        /* TODO */
        }
46
        /**
48       * Function to return the values of a map
         */
50      public static Collection<Integer> values(Map<Integer, Integer> map) {
          /* TODO */
52      }

54      /**
         * Function, internally using a map, returning "black",
56       * "white", or "red" depending on int input value.
         * "black" = 0, "white" = 1, "red" = 2
58       */
        public static String getColor(int value) {
60        /* TODO */
        }
62    }
```

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
 */

package hus.oop.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
package hus.oop.comparable;

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));

        Collections.sort(list);

        System.out.println("Movies after sorting : ");
        for (Movie movie : list) {
          System.out.println(movie.getName() + " " +
                             movie.getRating() + " " +
                             movie.getYear());
        }
    }
}
```

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.

```
/**
 * A Java program to demonstrate Comparator interface
 */

package hus.oop.comparator;

import java.io.*;
import java.util.*;

/**
 * A class 'Movie' that implements Comparable
 */
class Movie implements Comparable<Movie> {
```

```
14    private String name;
      private double rating;
16    private int year;

18    // Used to sort movies by year
      public int compareTo(Movie m) {
20      /* TODO */
      }
22
      // Constructor
24    public Movie(String name, double rating, int year) {
        /* TODO */
26    }

28    // Getter methods for accessing private data
      public double getRating() {
30      /* TODO */
      }
32
      public String getName() {
34      /* TODO */
      }
36
      public int getYear() {
38      /* TODO */
      }
40  }
```

```
  package hus.oop.comparator;
2
  /**
4  * Class to compare Movies by name
   */
6 class NameCompare implements Comparator<Movie> {
    public int compare(Movie left, Movie right) {
8     /* TODO */
    }
10 }
```

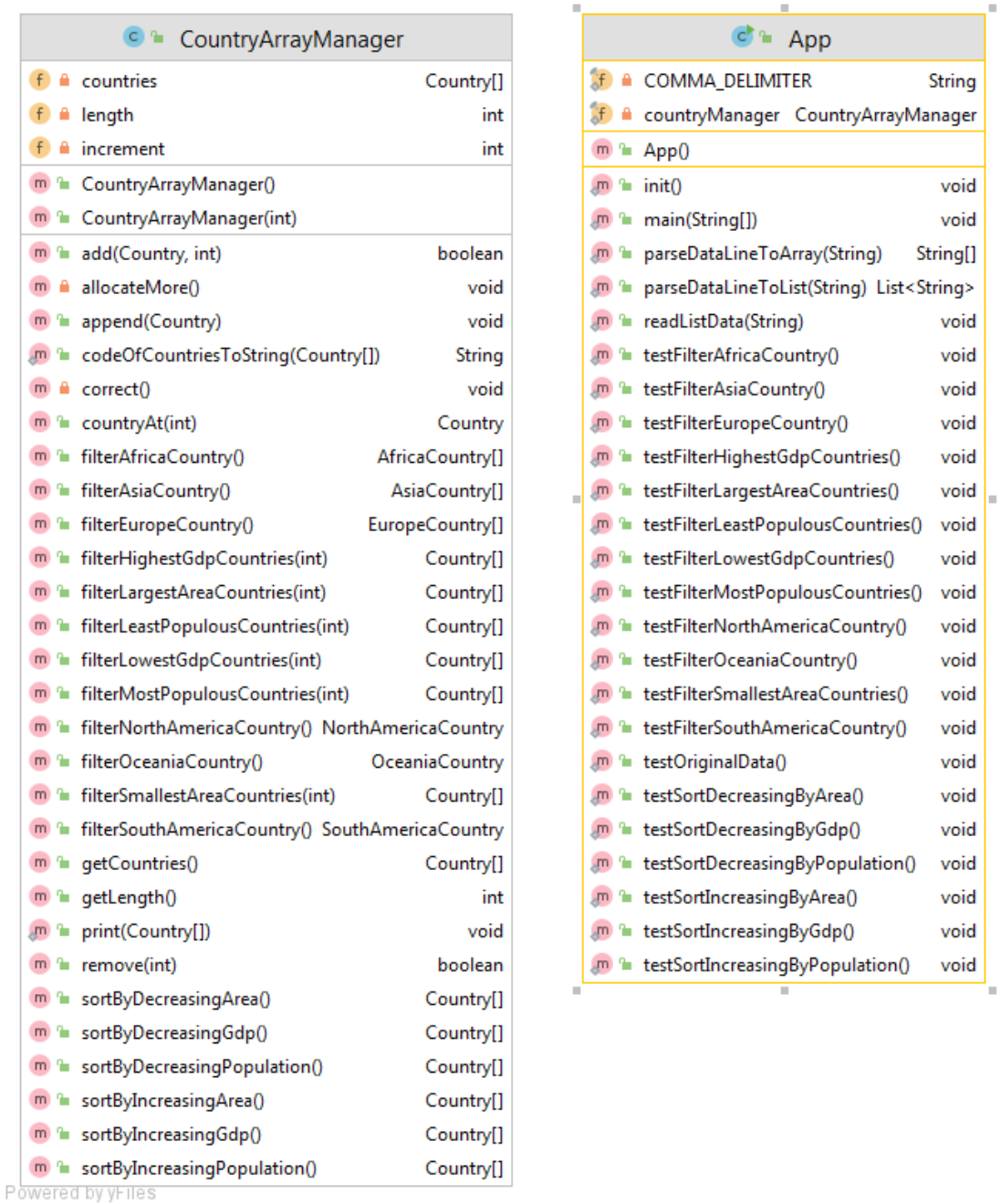
```
  package hus.oop.comparator;
2
  /**
4  * Class to compare Movies by ratings
   */
6 class RatingCompare implements Comparator<Movie> {
```

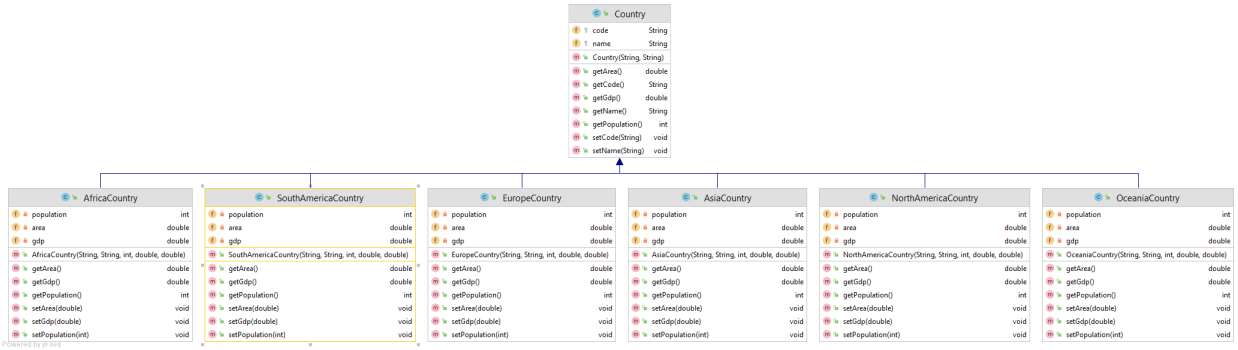
```
  public int compare(Movie left, Movie right) {
    /* TODO */
  }
}
```

```
package hus.oop.comparator;

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() + " ");
    }
  }
}
```

4.3 Country Manager

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






```java
package hus.oop.countryarraymanager;

public abstract class Country {
    protected String code;
    protected String name;

    public Country(String code, String name) {
        this.code = code;
        this.name = name;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public abstract int getPopulation();

    public double getArea();

    public double getGdp();
}
```

```java
package hus.oop.countryarraymanager;

public class AfricaCountry extends Country {
  private int population;
  private double area;
  private double gdp;

  public AfricaCountry(String code,
                       String name,
                       int population,
                       double area,
                       double gdp) {
    super(code, name);
    this.population = population;
    this.area = area;
    this.gdp = gdp;
  }

  public int getPopulation() {
    return population;
  }

  public void setPopulation(int population) {
    this.population = population;
  }

  public double getArea() {
    return area;
  }

  public void setArea(double area) {
    this.area = area;
  }

  public double getGdp() {
    return gdp;
  }

  public void setGdp(double gdp) {
    this.gdp = gdp;
  }
}
```

```java
package hus.oop.countryarraymanager;

public class AsiaCountry extends Country {
  private int population;
  private double area;
```

```
6   private double gdp;

8   public AsiaCountry(String code,
                       String name,
10                     int population,
                       double area,
12                     double gdp) {
      super(code, name);
14    this.population = population;
      this.area = area;
16    this.gdp = gdp;
    }
18
    ...
20 }
```

```
package hus.oop.countryarraymanager;
2
public class EuropeCountry extends Country {
4   private int population;
    private double area;
6   private double gdp;

8   public EuropeCountry(String code,
                         String name,
10                       int population,
                         double area,
12                       double gdp) {
      super(code, name);
14    this.population = population;
      this.area = area;
16    this.gdp = gdp;
    }
18
    ...
20 }
```

```
package hus.oop.countryarraymanager;
2
import java.util.Arrays;
4
public class CountryArrayManager {
6   private Country[] countries;
    private int length;
8
```

```
  public CountryArrayManager() {
    countries = new Country[1];
    this.length = 0;
  }

  public CountryArrayManager(int maxLength) {
    countries = new Country[maxLength];
    this.length = 0;
  }

  public int getLength() {
    return this.length;
  }

  public Country[] getCountries() {
    return this.countries;
  }

  private void correct() {
    int nullFirstIndex = 0;
    for (int i = 0; i < this.countries.length; i++) {
      if (this.countries[i] == null) {
        nullFirstIndex = i;
        break;
      }
    }

    if (nullFirstIndex > 0) {
      this.length = nullFirstIndex;
      for (int i = nullFirstIndex; i < this.countries.length; i++) {
        this.countries[i] = null;
      }
    }
  }

  private void allocateMore() {
    Country[] newArray = new Country[2 * this.countries.length];
    System.arraycopy(this.countries, 0, newArray, 0, this.countries.
        ↪ length);
    this.countries = newArray;
  }

  public void append(Country country) {
    if (this.length >= this.countries.length) {
      allocateMore();
    }

    this.countries[this.length] = country;
    this.length++;
  }

  public boolean add(Country country, int index) {
```

```
60      if ((index < 0) || (index > this.countries.length)) {
          return false;
62      }

64      if (this.length >= this.countries.length) {
          allocateMore();
66      }

68      for (int i = this.length; i > index; i--) {
          this.countries[i] = this.countries[i-1];
70      }

72      this.countries[index] = country;
        this.length++;
74      return true;
    }
76
    public boolean remove(int index) {
78      if ((index < 0) || (index >= countries.length)) {
          return false;
80      }

82      for (int i = index; i < length - 1; i++) {
          this.countries[i] = this.countries[i + 1];
84      }

86      this.countries[this.length - 1] = null;
        this.length--;
88      return true;
    }
90
    public Country countryAt(int index) {
92      if ((index < 0) || (index >= this.length)) {
          return null;
94      }

96      return this.countries[index];
    }
98
    /**
100  * Sort the countries in order of increasing population
     *  using selection sort algorithm.
102  * @return array of increasing population countries.
     */
104 public Country[] sortByIncreasingPopulation() {
      Country[] newArray = new Country[this.length];
106   System.arraycopy(this.countries, 0, newArray, 0, this.length);

108   /* TODO: sort newArray */

110   return newArray;
    }
```

```
112
    /**
114  * Sort the countries in order of decreasing population
     *  using selection sort algorithm.
116  * @return array of decreasing population countries.
     */
118 public Country[] sortByDecreasingPopulation() {
      Country[] newArray = new Country[this.length];
120   System.arraycopy(this.countries, 0, newArray, 0, this.length);

122   /* TODO: sort newArray */

124   return newArray;
    }
126
    /**
128  * Sort the countries in order of increasing area
     *  using bubble sort algorithm.
130  * @return array of increasing area countries.
     */
132 public Country[] sortByIncreasingArea() {
      Country[] newArray = new Country[this.length];
134   System.arraycopy(this.countries, 0, newArray, 0, this.length);

136   /* TODO: sort newArray */

138   return newArray;
    }
140
    /**
142  * Sort the countries in order of decreasing area
     *  using bubble sort algorithm.
144  * @return array of increasing area countries.
     */
146 public Country[] sortByDecreasingArea() {
      Country[] newArray = new Country[this.length];
148   System.arraycopy(this.countries, 0, newArray, 0, this.length);

150   /* TODO: sort newArray */

152   return newArray;
    }
154
    /**
156  * Sort the countries in order of increasing GDP
     *  using insertion sort algorithm.
158  * @return array of increasing GDP countries.
     */
160 public Country[] sortByIncreasingGdp() {
      Country[] newArray = new Country[this.length];
162   System.arraycopy(this.countries, 0, newArray, 0, this.length);
```

```
164     /* TODO: sort newArray */
166     return newArray;
    }
168
    /**
170  * Sort the countries in order of increasing GDP
     * using insertion sort algorithm.
172  * @return array of increasing insertion countries.
     */
174 public Country[] sortByDecreasingGdp() {
      Country[] newArray = new Country[this.length];
176   System.arraycopy(this.countries, 0, newArray, 0, this.length);
178   /* TODO: sort newArray */
180   return newArray;
    }
182
    public AfricaCountry[] filterAfricaCountry() {
184   /* TODO */
    }
186
    public AsiaCountry[] filterAsiaCountry() {
188   /* TODO */
    }
190
    public EuropeCountry[] filterEuropeCountry() {
192   /* TODO */
    }
194
    public NorthAmericaCountry filterNorthAmericaCountry() {
196   /* TODO */
    }
198
    public OceaniaCountry filterOceaniaCountry() {
200   /* TODO */
    }
202
    public SouthAmericaCountry filterSouthAmericaCountry() {
204   /* TODO */
    }
206
    public Country[] filterMostPopulousCountries(int howMany) {
208   /* TODO */
    }
210
    public Country[] filterLeastPopulousCountries(int howMany) {
212   return null;
    }
214
    public Country[] filterLargestAreaCountries(int howMany) {
```

```
216     /* TODO */
    }
218
    public Country[] filterSmallestAreaCountries(int howMany) {
220   return null;
    }
222
    public Country[] filterHighestGdpCountries(int howMany) {
224   /* TODO */
    }
226
    public Country[] filterLowestGdpCountries(int howMany) {
228   /* TODO */
    }
230
    public static String codeOfCountriesToString(Country[] countries) {
232   StringBuilder codeOfCountries = new StringBuilder();
      codeOfCountries.append("[");
234   for (int i = 0; i < countries.length; i++) {
        Country country = countries[i];
236     if (country != null) {
          codeOfCountries.append(country.getCode())
238       .append(" ");
        }
240   }
      return codeOfCountries.toString().trim() + "]";
242 }

244 public static void print(Country[] countries) {
      StringBuilder countriesString = new StringBuilder();
246   countriesString.append("[");
      for (int i = 0; i < countries.length; i++) {
248     Country country = countries[i];
        if (country != null) {
250       countriesString.append(country.toString()).append("\n");
        }
252   }
      System.out.print(countriesString.toString().trim() + "]");
254 }
}
```

```
1 package hus.oop.countryarraymanager;

3 import java.io.BufferedReader;
  import java.io.FileReader;
5 import java.io.IOException;
  import java.util.List;
7 import java.util.ArrayList;
```

```
9  public class App {
     private static final String COMMA_DELIMITER = ",";
11   private static final CountryArrayManager countryManager = new
       ↪ CountryArrayManager();

13   public static void main(String[] args) {
       init();
15
       /* TODO: write code to test program */
17   }

19   public static void readListData(String filePath) {
       BufferedReader dataReader = null;
21     try {
         dataReader = new BufferedReader(new FileReader(filePath));
23
         // Read file in java line by line.
25       String line;
         while ((line = dataReader.readLine()) != null) {
27         List<String> dataList = parseDataLineToList(line);

29         if (dataList.get(0).equals("code")) {
             continue;
31         }

33         if (dataList.size() != 6) {
             continue;
35         }

37         /*
            * TODO: create Country and append countries into
39          *  CountryArrayManager here.
            */
41       }
       } catch (IOException e) {
43       e.printStackTrace();
       } finally {
45       try {
           if (dataReader != null) {
47           dataReader.close();
           }
49       } catch (IOException e) {
           e.printStackTrace();
51       }
       }
53   }

55   public static List<String> parseDataLineToList(String dataLine) {
       List<String> result = new ArrayList<>();
57     if (dataLine != null) {
         String[] splitData = dataLine.split(COMMA_DELIMITER);
```

```
59        for (int i = 0; i < splitData.length; i++) {
            result.add(splitData[i]);
61        }
      }
63
      return result;
65    }

67    public static String[] parseDataLineToArray(String dataLine) {
      if (dataLine == null) {
69      return null;
      }
71
      return dataLine.split(COMMA_DELIMITER);
73    }

75    public static void init() {
      String filePath = "data/countries.csv";
77    readListData(filePath);
    }
79
    public static void testOriginalData() {
81    String codesString = CountryArrayManager.codeOfCountriesToString(
        ↪ countryManager.getCountries());
      System.out.print(codesString);
83    }

85    public static void testSortIncreasingByPopulation() {
      Country[] countries = countryManager.sortByIncreasingPopulation();
87    String codesString = CountryArrayManager.codeOfCountriesToString(
        ↪ countries);
      System.out.print(codesString);
89    }

91    public static void testSortDecreasingByPopulation() {
      /* TODO */
93    }

95    public static void testSortIncreasingByArea() {
      /* TODO */
97    }

99    public static void testSortDecreasingByArea() {
      /* TODO */
101   }

103   public static void testSortIncreasingByGdp() {
      /* TODO */
105   }

107   public static void testSortDecreasingByGdp() {
      /* TODO */
```

```
109     }
111     public static void testFilterAfricaCountry() {
            /* TODO */
113     }
115     public static void testFilterAsiaCountry() {
            /* TODO */
117     }
119     public static void testFilterEuropeCountry() {
            /* TODO */
121     }
123     public static void testFilterNorthAmericaCountry() {
            /* TODO */
125     }
127     public static void testFilterOceaniaCountry() {
            /* TODO */
129     }
131     public static void testFilterSouthAmericaCountry() {
            /* TODO */
133     }
135     public static void testFilterMostPopulousCountries() {
            /* TODO */
137     }
139     public static void testFilterLeastPopulousCountries() {
            /* TODO */
141     }
143     public static void testFilterLargestAreaCountries() {
            /* TODO */
145     }
147     public static void testFilterSmallestAreaCountries() {
            /* TODO */
149     }
151     public static void testFilterHighestGdpCountries() {
            /* TODO */
153     }
155     public static void testFilterLowestGdpCountries() {
            /* TODO */
157     }
    }
```