Lab 6. 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 More Exercises on Classes

1.1 Using JDK's BigInteger Class

Recall that primitive integer type byte, short, int and long represent 8-, 16-, 32-, and 64-bit signed integers, respectively. You cannot use them for integers bigger than 64 bits. Java API provides a class called BigInteger in a package called java.math. Study the API of the BigInteger class (Java API ⇒ From "Packages", choose "java.math" " From "classes", choose "BigInteger" " Study the constructors (choose "CONSTR") on how to construct a BigInteger instance, and the public methods available (choose "METHOD"). Look for methods for adding and multiplying two BigIntegers.

Write a program called TestBigInteger that:

1. adds "1111111111111111111111111111111111111111111111111111111111111111" to "2222222222222222222222222222222222222222222222222222" and prints the result.
2. multiplies the above two number and prints the result.

Hints

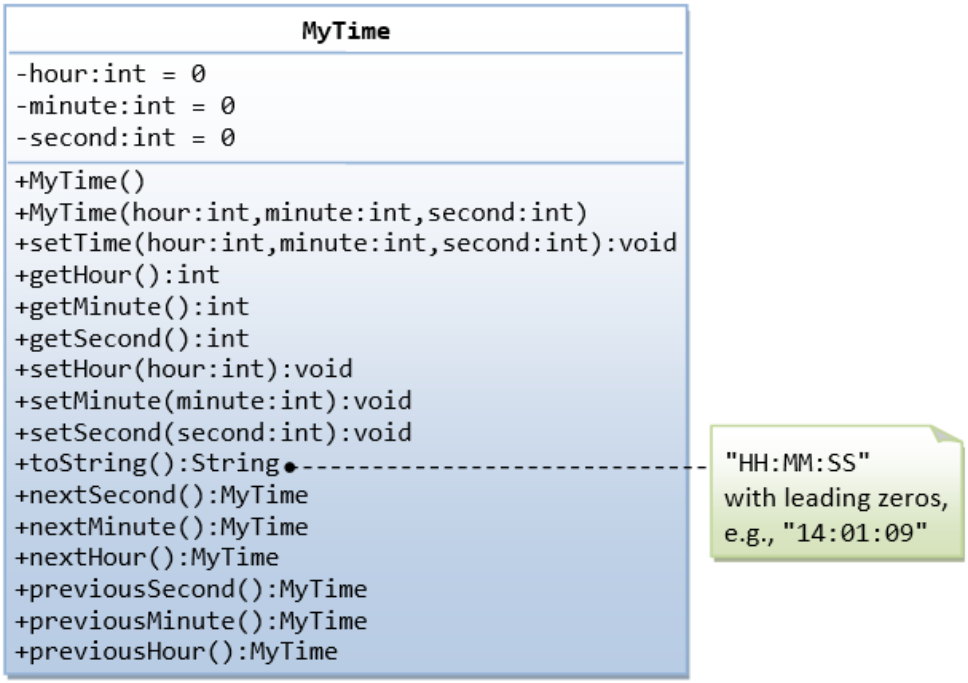
```
import java.math.BigInteger

public class TestBigInteger {
  public static void main(String[] args) {
    BigInteger i1 = new BigInteger(...);
    BigInteger i2 = new BigInteger(...);
    System.out.println(i1.add(i2));
    .......
  }
}
```

1.2 The MyTime Class

A class called MyTime, which models a time instance, is designed as shown in the class diagram.

It contains the following private instance variables:

- hour: between 0 to 23.
- minute: between 0 to 59.
- Second: between 0 to 59.



You are required to perform input validation.

It contains the following public methods:

- *setTime(int hour, int minute, int second)*: It shall check if the given hour, minute and second are valid before setting the instance variables.

  (Advanced: Otherwise, it shall throw an IllegalArgumentException with the message "Invalid hour, minute, or second!".)

- Setters *setHour(int hour)*, *setMinute(int minute)*, *setSecond(int second)*: It shall check if the parameters are valid, similar to the above.

- Getters *getHour()*, *getMinute()*, *getSecond()*.

- *toString()*: returns "HH:MM:SS".

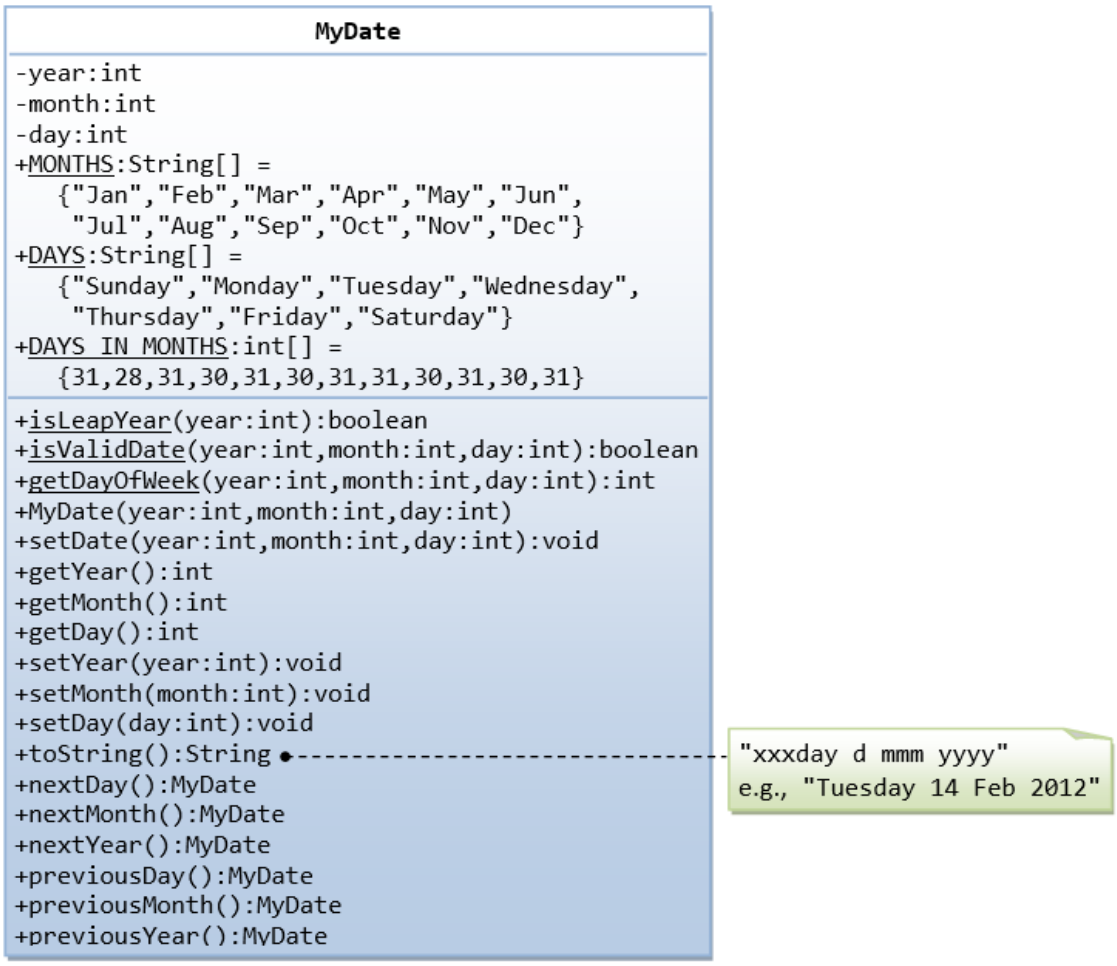
- *nextSecond()*: Update this instance to the next second and return this instance. Take note that the *nextSecond()* of 23 : 59 : 59 is 00 : 00 : 00.

- *nextMinute()*, *nextHour()*, *previousSecond()*, *previousMinute()*, *previousHour()*: similar to the above.

Write the code for the MyTime class. Also write a test driver (called TestMyTime) to test all the public methods defined in the MyTime class.

1.3 The MyDate Class

A class called MyDate, which models a date instance, is defined as shown in the class diagram.



The MyDate class contains the following private instance variables:

- *year* (*int*): Between 1 to 9999.
- *month* (*int*): Between 1 (Jan) to 12 (Dec).
- *day* (*int*): Between 1 to 28|29|30|31, where the last day depends on the month and whether it is a leap year for Feb (28|29).

It also contains the following public static final variables (drawn with underlined in the class diagram):

- MONTHS (*String*[ ]), DAYS (*String*[ ]), and DAY_IN_MONTHS (*int*[ ]): static variables, initialized as shown, which are used in the methods.

The MyDate class has the following public static methods (drawn with underlined in the class diagram):

- *isLeapYear(int year)*: returns true if the given year is a leap year. A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.
- *isValidDate(int year, int month, int day)*: returns true if the given year, month, and day constitute a valid date. Assume that year is between 1 and 9999, month is between 1 (Jan) to 12 (Dec) and day shall be between 1 and 28|29|30|31 depending on the month and whether it is a leap year on Feb.
- *getDayOfWeek(int year, int month, int day)*: returns the day of the week, where 0 for Sun, 1 for Mon, ..., 6 for Sat, for the given date. Assume that the date is valid. Read the earlier exercise on how to determine the day of the week (or Wiki "Determination of the day of the week").

The MyDate class has one constructor, which takes 3 parameters: *year*, *month* and *day*. It shall invoke *setDate()* method (to be described later) to set the instance variables.

The MyDate class has the following public methods:

- *setDate(int year, int month, int day)*: It shall invoke the static method isValidDate() to verify that the given *year*, *month* and *day* constitute a valid date.

  (Advanced: Otherwise, it shall throw an IllegalArgumentException with the message "Invalid year, month, or day!".)
- *setYear(int year)*: It shall verify that the given year is between 1 and 9999.

  (Advanced: Otherwise, it shall throw an IllegalArgumentException with the message "Invalid year!".)
- *setMonth(int month)*: It shall verify that the given month is between 1 and 12.

  (Advanced: Otherwise, it shall throw an IllegalArgumentException with the message "Invalid month!".)
- *setDay(int day)*: It shall verify that the given day is between 1 and *dayMax*, where *dayMax* depends on the month and whether it is a leap year for Feb.

  (Advanced: Otherwise, it shall throw an IllegalArgumentException with the message "Invalid month!".)
- *getYear()*, *getMonth()*, *getDay()*: return the value for the *year*, *month* and *day*, respectively.
- *toString()*: returns a date string in the format "xxxday d mmm yyyy", e.g., "Tuesday 14 Feb 2012".
- *nextDay()*: update this instance to the next day and return this instance. Take note that *nextDay()* for 31 Dec 2000 shall be 1 Jan 2001.
- *nextMonth()*: update this instance to the next month and return this instance. Take note that *nextMonth()* for 31 Oct 2012 shall be 30 Nov 2012.

- *nextYear()*: update this instance to the next year and return this instance. Take note that *nextYear()* for 29 Feb 2012 shall be 28 Feb 2013.

  (Advanced: throw an IllegalStateException with the message "Year out of range!" if year > 9999.)

- *previousDay()*, *previousMonth()*, *previousYear()*: similar to the above.

Write the code for the MyDate class.

Use the following test statements to test the MyDate class:

```
MyDate date1 = new MyDate(2012, 2, 28);
System.out.println(date1);              // Tuesday 28 Feb 2012
System.out.println(date1.nextDay());    // Wednesday 29 Feb 2012
System.out.println(date1.nextDay());    // Thursday 1 Mar 2012
System.out.println(date1.nextMonth()); // Sunday 1 Apr 2012
System.out.println(date1.nextYear());   // Monday 1 Apr 2013

MyDate date2 = new MyDate(2012, 1, 2);
System.out.println(date2);                  // Monday 2 Jan 2012
System.out.println(date2.previousDay());    // Sunday 1 Jan 2012
System.out.println(date2.previousDay());    // Saturday 31 Dec 2011
System.out.println(date2.previousMonth()); // Wednesday 30 Nov 2011
System.out.println(date2.previousYear());   // Tuesday 30 Nov 2010

MyDate date3 = new MyDate(2012, 2, 29);
System.out.println(date3.previousYear());   // Monday 28 Feb 2011

// MyDate date4 = new MyDate(2099, 11, 31); // Invalid year, month, or day!
// MyDate date5 = new MyDate(2011, 2, 29);  // Invalid year, month, or day!
```

Write a test program that tests the *nextDay()* in a loop, by printing the dates from 28 Dec 2011 to 2 Mar 2012.

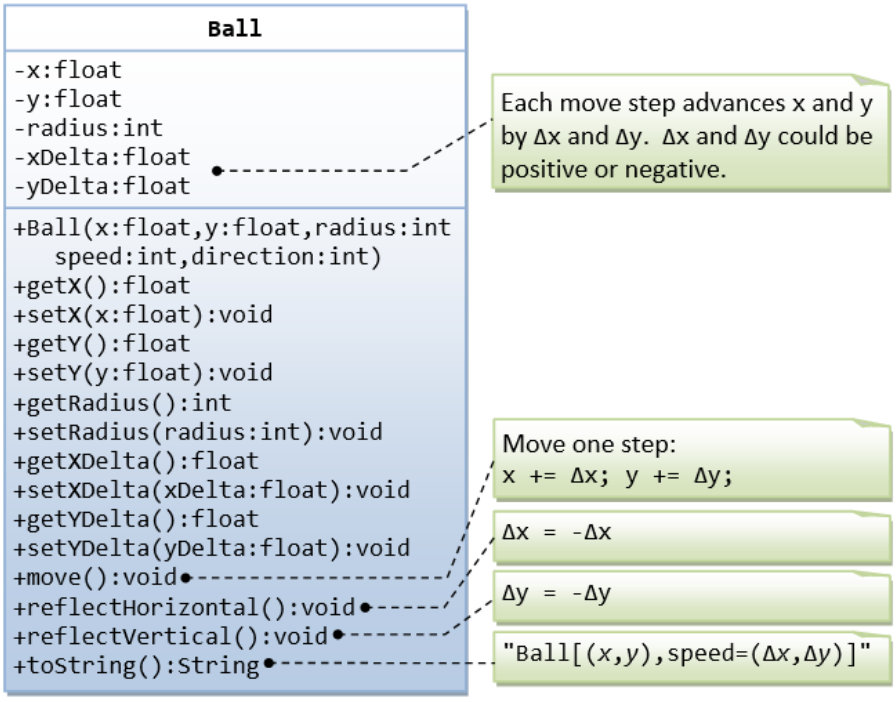
1.4 Bouncing Balls - Ball and Container Classes

A class called Ball is designed as shown in the class diagram.

The Ball class contains the following private instance variables:

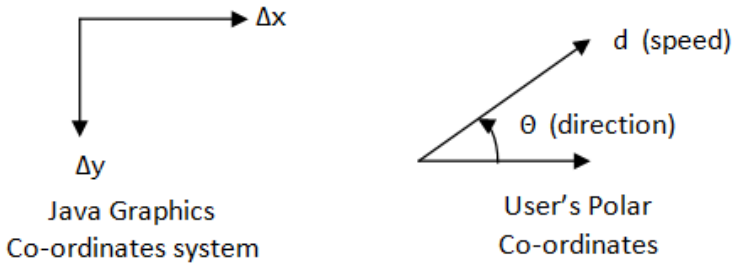
- x, y and *radius*, which represent the ball's center (x, y) co-ordinates and the radius, respectively.

- $xDelta$ (Δx) and $yDelta$ (Δy), which represent the displacement (movement) per step, in the x and y direction respectively.



The Ball class contains the following public methods:

- A constructor which accepts x, y, *radius*, *speed*, and direction as arguments. For user friendliness, user specifies speed (in pixels per step) and direction (in degrees in the range of (-180°, 180°]). For the internal operations, the speed and direction are to be converted to $(\Delta x, \Delta y)$ in the internal representation. Note that the y-axis of the Java graphics coordinate system is inverted, i.e., the origin $(0, 0)$ is located at the top-left corner.



$$\Delta x = d\cos(\theta)$$
$$\Delta y = d\sin(\theta)$$

- Getter and setter for all the instance variables.
- A method move() which move the ball by one step.

$$x \mathrel{+}= \Delta x$$
$$y \mathrel{+}= \Delta y$$

- *reflectHorizontal()* which reflects the ball horizontally (i.e., hitting a vertical wall)

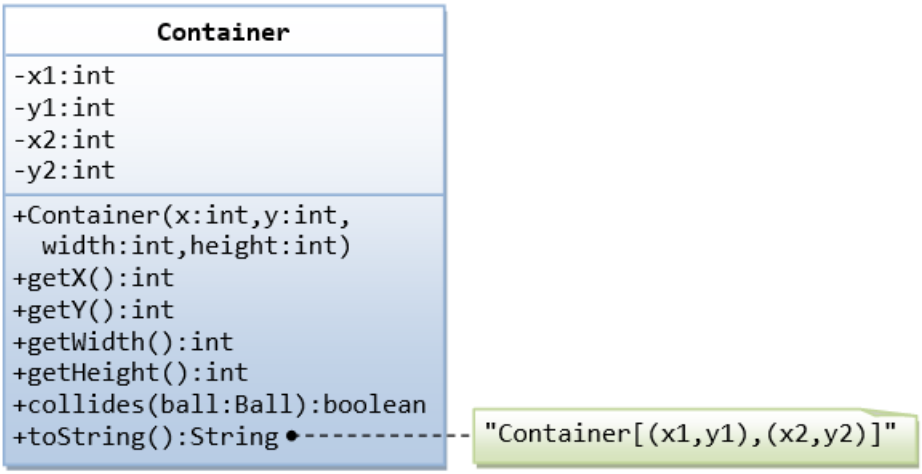
$$\begin{aligned} \Delta x &= -\Delta x \\ \Delta y &\text{ no changes} \end{aligned}$$

- *reflectVertical()* (the ball hits a horizontal wall).

$$\begin{aligned} \Delta x &\text{ no changes} \\ \Delta y &= -\Delta y \end{aligned}$$

- *toString()* which prints the message "Ball at (x, y) of velocity $(\Delta x, \Delta y)$".

Write the Ball class. Also write a test program to test all the methods defined in the class.



A class called Container, which represents the enclosing box for the ball, is designed as shown in the class diagram. It contains:

- Instance variables $(x1, y1)$ and $(x2, y2)$ which denote the top-left and bottom-right corners of the rectangular box.
- A constructor which accepts (x, y) of the top-left corner, width and height as argument, and converts them into the internal representation (i.e., $x2 = x1 + \ width\ - 1$). Width and height is used in the argument for safer operation (there is no need to check the validity of $x2 > x1$, etc.).
- A *toString()* method that returns "Container at $(x1, y1)$ to $(x2, y2)$".
- A boolean method called *collidesWith(Ball)*, which check if the given Ball is outside the bounds of the container box. If so, it invokes the Ball's *reflectHorizontal()* and/or *reflectVertical()* to change the movement direction of the ball, and returns true.

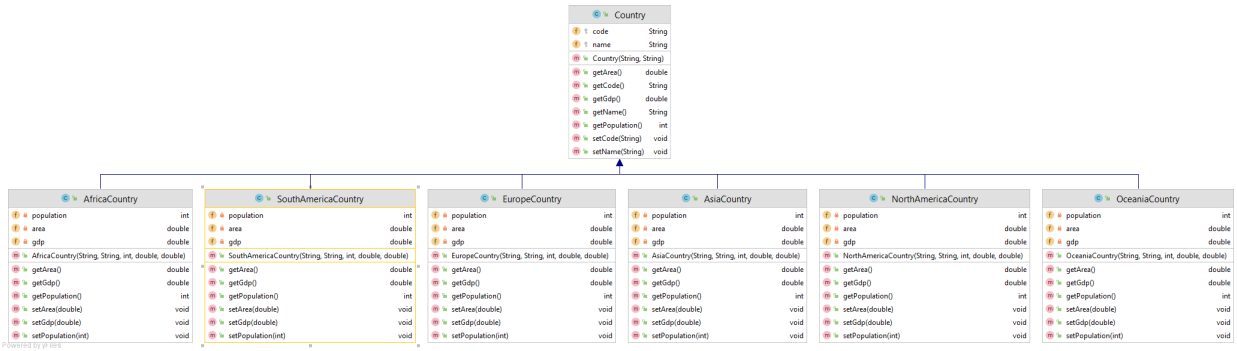
```java
public boolean collidesWith(Ball ball) {
  if ((ball.getX() - ball.getRadius() <= this.x1) ||
      (ball.getX() - ball.getRadius() >= this.x2)) {
    ball.reflectHorizontal();
    return true;
  }
  ......
}
```

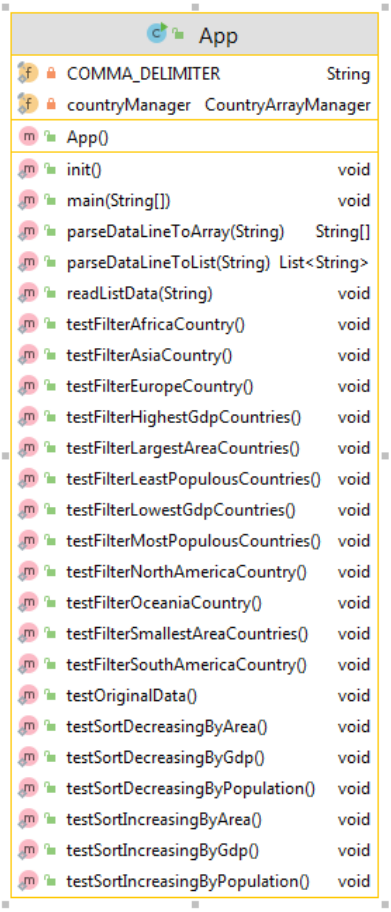
Use the following statements to test your program:

```java
Ball ball = new Ball(50, 50, 5, 10, 30);
Container box = new Container(0, 0, 100, 100);
for (int step = 0; step < 100; ++step) {
  ball.move();
  box.collidesWith(ball);
  System.out.println(ball); // manual check the position of the ball
}
```

1.5 Country Manager

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






```java
package com.countryarraymanager;

public 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;
  }
```

```
15
    public void setCode(String code) {
17    this.code = code;
    }
19
    public String getName() {
21    return name;
    }
23
    public void setName(String name) {
25    this.name = name;
    }
27
    public int getPopulation() {
29    return 0;
    }
31
    public double getArea() {
33    return 0.0;
    }
35
    public double getGdp() {
37    return 0.0;
    }
39 }
```

```
1  package com.countryarraymanager;

3  public class AfricaCountry extends Country {
     private int population;
5    private double area;
     private double gdp;
7
     public AfricaCountry(String code,
9                         String name,
                          int population,
11                        double area,
                          double gdp) {
13     super(code, name);
       this.population = population;
15     this.area = area;
       this.gdp = gdp;
17   }

19   public int getPopulation() {
       return population;
21   }
```

```
23      public void setPopulation(int population) {
          this.population = population;
25      }

27      public double getArea() {
          return area;
29      }

31      public void setArea(double area) {
          this.area = area;
33      }

35      public double getGdp() {
          return gdp;
37      }

39      public void setGdp(double gdp) {
          this.gdp = gdp;
41      }
    }
```

```
    package com.countryarraymanager;
2
    public class AsiaCountry extends Country {
4     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 com.countryarraymanager;
2
    public class EuropeCountry extends Country {
```

```
    private int population;
    private double area;
    private double gdp;

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

    ...
}
```

```
package com.countryarraymanager;

import java.util.Arrays;

public class CountryArrayManager {
  private Country[] countries;
  private int length;
  private int increment;

  public CountryArrayManager() {
    this.increment = 10;
    countries = new Country[this.increment];
    this.length = 0;
  }

  public CountryArrayManager(int maxLength) {
    this.increment = 10;
    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;
32    for (int i = 0; i < this.countries.length; i++) {
        if (this.countries[i] == null) {
34        nullFirstIndex = i;
          break;
36      }
      }
38
      if (nullFirstIndex > 0) {
40      this.length = nullFirstIndex;
        for (int i = nullFirstIndex; i < this.countries.length; i++) {
42        this.countries[i] = null;
        }
44    }
  }
46
  private void allocateMore() {
48  Country[] newArray = new Country[this.countries.length + this.
      ↪ increment];
    System.arraycopy(this.countries, 0, newArray, 0, this.countries.
      ↪ length);
50  this.countries = newArray;
  }
52
  public void append(Country country) {
54  if (this.length >= this.countries.length) {
      allocateMore();
56  }

58  this.countries[this.length] = country;
    this.length++;
60 }

62 public boolean add(Country country, int index) {
    if ((index < 0) || (index > this.countries.length)) {
64    return false;
    }
66
    if (this.length >= this.countries.length) {
68    allocateMore();
    }
70
    for (int i = this.length; i > index; i--) {
72    this.countries[i] = this.countries[i-1];
    }
74
    this.countries[index] = country;
76  this.length++;
    return true;
78 }

80 public boolean remove(int index) {
```

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

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

    this.countries[this.length - 1] = null;
    this.length--;
    return true;
  }

  public Country countryAt(int index) {
    if ((index < 0) || (index >= this.length)) {
      return null;
    }

    return this.countries[index];
  }

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

    /* TODO: sort newArray */

    return newArray;
  }

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

    /* TODO: sort newArray */

    return newArray;
  }

  /**
  * Sort the countries in order of increasing area
  * using bubble sort algorithm.
```

```
     * @return array of increasing area countries.
134  */
     public Country[] sortByIncreasingArea() {
136    Country[] newArray = new Country[this.length];
       System.arraycopy(this.countries, 0, newArray, 0, this.length);
138
       /* TODO: sort newArray */
140
       return newArray;
142  }

144  /**
     * Sort the countries in order of decreasing area
146  *  using bubble sort algorithm.
     * @return array of increasing area countries.
148  */
     public Country[] sortByDecreasingArea() {
150    Country[] newArray = new Country[this.length];
       System.arraycopy(this.countries, 0, newArray, 0, this.length);
152
       /* TODO: sort newArray */
154
       return newArray;
156  }

158  /**
     * Sort the countries in order of increasing GDP
160  *  using insertion sort algorithm.
     * @return array of increasing GDP countries.
162  */
     public Country[] sortByIncreasingGdp() {
164    Country[] newArray = new Country[this.length];
       System.arraycopy(this.countries, 0, newArray, 0, this.length);
166
       /* TODO: sort newArray */
168
       return newArray;
170  }

172  /**
     * Sort the countries in order of increasing GDP
174  * using insertion sort algorithm.
     * @return array of increasing insertion countries.
176  */
     public Country[] sortByDecreasingGdp() {
178    Country[] newArray = new Country[this.length];
       System.arraycopy(this.countries, 0, newArray, 0, this.length);
180
       /* TODO: sort newArray */
182
       return newArray;
184  }
```

```
186     public AfricaCountry[] filterAfricaCountry() {
          /* TODO */
188     }

190     public AsiaCountry[] filterAsiaCountry() {
          /* TODO */
192     }

194     public EuropeCountry[] filterEuropeCountry() {
          /* TODO */
196     }

198     public NorthAmericaCountry filterNorthAmericaCountry() {
          /* TODO */
200     }

202     public OceaniaCountry filterOceaniaCountry() {
          /* TODO */
204     }

206     public SouthAmericaCountry filterSouthAmericaCountry() {
          /* TODO */
208     }

210     public Country[] filterMostPopulousCountries(int howMany) {
          /* TODO */
212     }

214     public Country[] filterLeastPopulousCountries(int howMany) {
          return null;
216     }

218     public Country[] filterLargestAreaCountries(int howMany) {
          /* TODO */
220     }

222     public Country[] filterSmallestAreaCountries(int howMany) {
          return null;
224     }

226     public Country[] filterHighestGdpCountries(int howMany) {
          /* TODO */
228     }

230     public Country[] filterLowestGdpCountries(int howMany) {
          /* TODO */
232     }

234     public static String codeOfCountriesToString(Country[] countries) {
          StringBuilder codeOfCountries = new StringBuilder();
236       codeOfCountries.append("[");
```

```
      for (int i = 0; i < countries.length; i++) {
        Country country = countries[i];
        if (country != null) {
          codeOfCountries.append(country.getCode())
          .append(" ");
        }
      }
      return codeOfCountries.toString().trim() + "]";
    }

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

```
  package com.countryarraymanager;

  import java.io.BufferedReader;
  import java.io.FileReader;
  import java.io.IOException;
  import java.util.List;
  import java.util.ArrayList;

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

    public static void main(String[] args) {
      init();

      /* TODO: write code to test program */
    }

    public static void readListData(String filePath) {
      BufferedReader dataReader = null;
      try {
        dataReader = new BufferedReader(new FileReader(filePath));

        // Read file in java line by line.
```

```
        String line;
        while ((line = dataReader.readLine()) != null) {
          List<String> dataList = parseDataLineToList(line);

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

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

          /*
          * TODO: create Country and append countries into
          *  CountryArrayManager here.
          */
        }
      } catch (IOException e) {
        e.printStackTrace();
      } finally {
        try {
          if (dataReader != null) {
            dataReader.close();
          }
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    }

    public static List<String> parseDataLineToList(String dataLine) {
      List<String> result = new ArrayList<>();
      if (dataLine != null) {
        String[] splitData = dataLine.split(COMMA_DELIMITER);
        for (int i = 0; i < splitData.length; i++) {
          result.add(splitData[i]);
        }
      }

      return result;
    }

    public static String[] parseDataLineToArray(String dataLine) {
      if (dataLine == null) {
        return null;
      }

      return dataLine.split(COMMA_DELIMITER);
    }

    public static void init() {
      String filePath = "data/countries.csv";
```

```
        readListData(filePath);
78    }

80    public static void testOriginalData() {
        String codesString = CountryArrayManager.codeOfCountriesToString(
            ↪ countryManager.getCountries());
82      System.out.print(codesString);
      }
84
      public static void testSortIncreasingByPopulation() {
86      Country[] countries = countryManager.sortByIncreasingPopulation();
        String codesString = CountryArrayManager.codeOfCountriesToString(
            ↪ countries);
88      System.out.print(codesString);
      }
90
      public static void testSortDecreasingByPopulation() {
92      /* TODO */
      }
94
      public static void testSortIncreasingByArea() {
96      /* TODO */
      }
98
      public static void testSortDecreasingByArea() {
100     /* TODO */
      }
102
      public static void testSortIncreasingByGdp() {
104     /* TODO */
      }
106
      public static void testSortDecreasingByGdp() {
108     /* TODO */
      }
110
      public static void testFilterAfricaCountry() {
112     /* TODO */
      }
114
      public static void testFilterAsiaCountry() {
116     /* TODO */
      }
118
      public static void testFilterEuropeCountry() {
120     /* TODO */
      }
122
      public static void testFilterNorthAmericaCountry() {
124     /* TODO */
      }
126
```

```
    public static void testFilterOceaniaCountry() {
128   /* TODO */
    }
130
    public static void testFilterSouthAmericaCountry() {
132   /* TODO */
    }
134
    public static void testFilterMostPopulousCountries() {
136   /* TODO */
    }
138
    public static void testFilterLeastPopulousCountries() {
140   /* TODO */
    }
142
    public static void testFilterLargestAreaCountries() {
144   /* TODO */
    }
146
    public static void testFilterSmallestAreaCountries() {
148   /* TODO */
    }
150
    public static void testFilterHighestGdpCountries() {
152   /* TODO */
    }
154
    public static void testFilterLowestGdpCountries() {
156   /* TODO */
    }
158 }
```