Assignment 2

PersonData and CustomerData Classes	50 pts.
Shapes	50 pts.

TOTAL: 100 pts.

General Requirements

- *All files you are turning in must be signed. Please add your name to the source code in comments. Unsigned work automatically loses **5 pts**.*
- *Add comments to the source code you are writing:*
  - *Describe the purpose of every variable*
  - *Explain the algorithm you are using for solution*
  - *Add proper comments for all methods. Include @param, @return, and @throws tags*
- *Uncommented work loses up to **25% of the grade.***
- *Archive the entire project using ZIP or RAR utility. Turn it in into the digital drop box.*

PersonData and CustomerData Classes

PersonData Class

Design a class named PersonData with the following fields:

- `lastName`
- `firstName`
- `address`
- `phone`

All fields are of type String.

- Write appropriate accessor and mutator functions for all the fields.
- Write a non-argument constructor that sets all fields to empty strings.
- Write a constructor that takes all the data about customer as parameters (4 parameters) and initializes fields.
- Override toString() and equals() methods for the class
- Override clone() method. You can consider all of the fields to be of a primitive type, although strings are not. Because stings are immutable, default clone() will work just fine for them. Add declaration that the class `implements Cloneable` interface, otherwise your code will not compile.

CustomerData Class

Design a class named CustomerData, which is derived from the PersonData class. The CustomerData class should have the following fields:

- `customerNumber`
- `mailingList`
- `transactions`
- The customerNumber field will be used to hold a unique integer for each customer. The mailingList field should be a boolean. It will be set to true if the customer wishes to be on a mailing list, or false if the customer does not wish to be on a mailing list. `Transactions` represents the list of all transactions done with the customer for the last 5 years. Transactions are stored as an ArrayList of long integers.
- Write appropriate accessor and mutator methods for these fields.
    - Notice that you want to create a deep copy when writing accessor for `transactions` field.
    - When writing mutator for ArrayList field, create a deep copy of the argument. If a null was passed as parameter `IllegalArgumentException` must be generated
- Write a non-argument constructor that sets all variables (including inherited ones) to empty strings and `transactions` to an empty ArrayList (but not null!)
- Write a constructor that takes all the data about customer as parameters (name, address, etc., total of 7 parameters) and initializes member variables. When working with transaction field, don’t forget to create a deep copy of the object that is given to you as an argument.
- Write `void addTransaction(long t)` method that adds a transaction to the list. When adding a transaction, argument validation must be performed:
    - t must be no more than 15 digits long
    - t must be positive
    - when any of the above errors occur – throw `IllegalArgumentException`
- Override `toString()`
    - toString() must list all the transactions
- Write `boolean equals(Object other)` methods for the class.
    - When writing equals() make sure to compare object data not references.
- Override `clone()` method to create deep copy of the object. Use clone() method examples we worked on. When adding clone() method make sure to add `implements Cloneable` statement to the class header. Otherwise the override will not compile.

To illustrate the concept of polymorphism and late binding:
In main() declare and allocate an array of base type PersonData for 3 elements. Populate the array with 2 objects of type CustomerData, and 1 object of type PersonData. Next use overridden toString() method to print out all the object data into the screen of all 3 objects. Observe how late binding works and the correct toString() method implementation is being used with different types of objects.

Test <u>all the classes and methods</u> you wrote in main. Place the main() in a file separate from all the classes named CustomersDemo.java.

Name your project `Customers.java`

Shapes

First, define interface Relatable (borrowed from the text book code)

```
/**
   Relatable interface
*/

public interface Relatable
{
   boolean equals(BasicShape s);
   boolean isGreater(BasicShape s);
   boolean isLess(BasicShape s);
}
```

Next, define a pure abstract base class called `BasicShape` that implements `Relatable` interface.
Class must have the following:
<u>Protected</u> field:

- `color`, an array of 3 integers that are representing Red, Green, and Blue components in the color.

Public methods:

- `int[] getColor()` accessor - returns an array of 3 integers. A deep copy of the field must be returned.
- `void setColor (int[]) throws IllegalArgumantException` - mutator method for color field. The array must be validated and exception must be thrown in case the values passed are not valid
  - the array must be exactly 3 elements long
  - all values in the array must be in the range 0 – 255 as they represent colors.
- `double calcArea()`, abstract method.

- Implement all interface methods. Shapes must be compared by comparing the areas of shapes. Although calcArea() is defined as abstract, it can be used without limitations to implement the calculations needed to compare shapes.

Next, define a class named `Circle`. It should be derived from the `BasicShape` class. It should have the following:

Private fields:

- `centerX`, integer to hold x coordinate of the circle's center.
- `centerY`, integer to hold y coordinate of the circle's center.
- `radius`, integer to hold the radius of the circle.

Public methods:

- Non-argument constructor that sets all variables to 1 and color to black
- Constructor – accepts values for centerX, centerY, radius, and color as an array of 3 integers. Constructor throws `IllegalArgumentException` when
  - radius is not a positive number or
  - the array must is not exactly 3 elements long or
  - not all values in the array must be in the range 0 – 255
- Accessor and mutator methods for all non-inherited fields.
  - Mutator for radius field must throw `IllegalArgumentException` when an argument passed to method is not positive
- `calcArea` – calculates and returns the area of the circle (area = Math.PI*radius*radius)
- Override of `toString()`.
- Override clone() method. Use clone() method examples we worked on. When adding clone() method make sure to add `implements Cloneable` statement to the class header. Otherwise the override will not compile.

Don't forget to create deep copies whenever working with field of type Color.

Next, define a class named `Rectangle`. It should be derived from the `BasicShape` class. It should have the following fields and methods:

Private fields:

- `cornerX` integer to hold x coordinate of the rectangle's left upper corner.
- `cornerY` integer to hold y coordinate of the rectangle's left upper corner.
- `width`, integer to hold the width of the rectangle.
- `length`, integer to hold the length of the rectangle.

Public methods:

- Non-argument constructor that sets all variables to 1 and color to black.
- Constructor – accepts values for length, width, and color and sets fields.
  - Throws `IllegalArgumentException` when width or length are not positive numbers or color is null

- Accessor and mutator methods for all non-inherited fields.
    - Mutators for length and width fields must throw `IllegalArgumentException` when an argument passed to method is not positive
- `calcArea` – calculates and returns the area of the rectangle (area = length*width)
- Override of `toString()` that represents all the fields as strings
- Override clone() method

To illustrate the concept of polymorphism:

- In a file next to main() create a method `public static void orderShapes (BasicShape one, BasicShape two, BasicShape three)` that prints out three shapes ordered by the size of the area from smallest to largest. Use methods of Relatable interface to compare areas of the shapes.
- Test the method in main() by creating a couple of Circles and a Rectangle object and passing all 3 into the orderShapes() method.

Test all the classes and their methods you wrote in main. Place the main() in a file separate from all the classes, name the file **ShapesDemo.java.**