Exercise 3.1

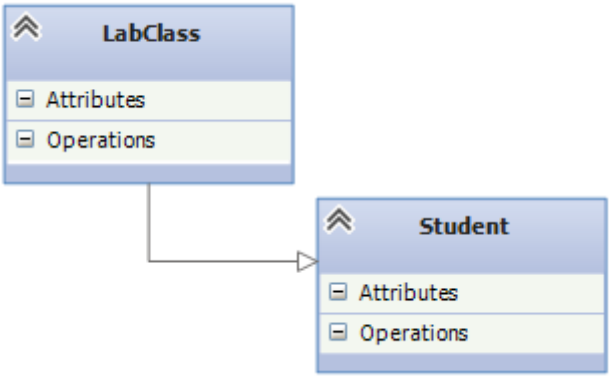

Figure 1: Class diagram

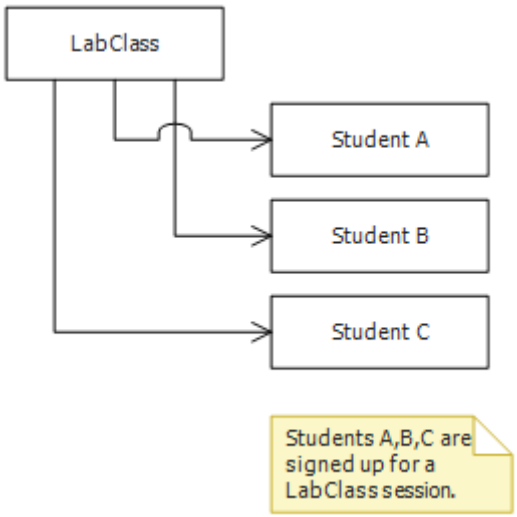

Figure 2: Object diagram

A class diagram illustrates the relationship between classes in general. As it can be seen in Figure 1, we can see that Student class is a part of a Lab class. An object diagram, in contrast to a class diagram, displays the way objects look like at a certain point in the objects' lifecycle. Figure 2 clearly shows that in that particular case, there are three Student objects that have signed up for a Lab Class session.

Exercise 3.2

A class diagram can change when the class is changed or the way the classes are connected. So, if we add a new field to the Lab Class or add a new class, *School* that will be a superset of Lab Class, the class diagram will change.

Exercise 3.3

An object diagram change anytime the objects change. So, if we add more students to a lab session, the object diagram will change too.

Exercise 3.9

The following are going to be true:

- `!false`
- `(34!=33) && !false`

Exercise 3.10

True for **a** and **b** when either both are *true* or both are *false*.

```
a && b || !a && !b
```

Exercise 3.11

True for **a** and **b** when either one is true but not both.

```
a ^ b
```

Exercise 3.12

```
! (!a || !b)
```

Exercise 3.21

The increment method can either be written as

```
int x = (value +1);
value = x - limit * (x/limit) ;
```

or, using an if statement

```
int x = value +1;

if(x < limit)
{
    value = x;
}
else
{
    value = 0;
}
```

∴ The modulo operator is better than either of these methods since it does not involve so many lines of code. We do not need to consider the formal definition of mod each time we want to use it, so instead we can focus on the complex logic.

Exercise 3.26

```
public Editor(string fileName, int x)
```

Exercise 3.27

```
Rectangle window;
Rectangle rectangle = new Rectangle(3,2);
window = rectangle;
```

Exercise 3.30

```
p1.print("helloworld.txt", true);
p1.getStatus(30);
```

Exercise 3.31

The **ClockDisplay** class

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;    // simulates the actual display
    private boolean pm = false;

    public ClockDisplay()
    {
        hours = new NumberDisplay(13,1);
        minutes = new NumberDisplay(60);
        updateDisplay();
    }


    public ClockDisplay(int hour, int minute)
    {
        hours = new NumberDisplay(13, 1);
        minutes = new NumberDisplay(60);
        setTime(hour-1, minute);
    }


    public void timeTick()
    {
        minutes.increment();
        if(minutes.getValue() == 0) {  // it just rolled over!
            hours.increment();

            if(hours.getValue() == 12)
            {
                pm = ! pm;
                hours.setValue(0);
            }
        }

        updateDisplay();
    }


    public void setTime(int hour, int minute)
    {
        hours.setValue(hour);
        minutes.setValue(minute);
        updateDisplay();
    }


    public String getTime()
```

```
    {
        return displayString;
    }


    private void updateDisplay()
    {
        displayString = hours.getDisplayValue()+  ":" +
                        minutes.getDisplayValue() + (pm  ? "pm" : "am");
    }
}
```

The **NumberDisplay** class

```
public class NumberDisplay
{
    private int limit;
    private int value;
    private int startingIndex;

    public NumberDisplay(int rollOverLimit)
    {
        limit = rollOverLimit;
        value = 0;
        startingIndex = 0;
    }

    public NumberDisplay(int rollOverLimit, int _startingIndex)
    {
        limit = rollOverLimit;
        value = 0;
        startingIndex = _startingIndex;
    }


    public int getValue()
    {
        return value;
    }

    public String getDisplayValue()
    {
        int temp = value + startingIndex;
        if(value < 10) {
            return "0" + temp;
        }
        else {
            return "" + temp;
        }
    }


    public void setValue(int replacementValue)
    {
        if((replacementValue >= 0) && (replacementValue < limit)) {
            value = replacementValue;
        }
    }


    public void increment()
    {
        value = (value + 1) % limit;
    }
}
```

Exercise 3.32

In *Exercise 3.31* and *Exercise 3.32*, two ways of constructing a 12-hour based clocks are presented. To answer the question which is better, we must define what is to be valued more in both situations. I think it all depends on the time we want to put into it. It took longer for me to construct the clock in *Exercise 3.31* in comparison to *Exercise 3.32*. Personally, I would go with *Exercise 3.32*, since, in terms of mathematics, much easier. A 12 hours based clock requires modulo arithmetic with some minor changes, as we do not allow zero. In a 24 hour based clock, maths is easier since zero is included.

The **ClockDisplay** class.

```
1  public class ClockDisplay
2  {
3      private NumberDisplay hours;
4      private NumberDisplay minutes;
5      private String displayString;    // simulates the actual display
6
7      public ClockDisplay()
8      {
9          hours = new NumberDisplay(24);
10         minutes = new NumberDisplay(60);
11         updateDisplay();
12     }
13
14     public ClockDisplay(int hour, int minute)
15     {
16         hours = new NumberDisplay(24);
17         minutes = new NumberDisplay(60);
18         setTime(hour, minute);
19     }
20
21
22     public void timeTick()
23     {
24         minutes.increment();
25         if(minutes.getValue() == 0) {  // it just rolled over!
26             hours.increment();
27         }
28         updateDisplay();
29     }
30
31
32     public void setTime(int hour, int minute)
33     {
34         hours.setValue(hour);
35         minutes.setValue(minute);
36         updateDisplay();
37     }
38
39     public String getTime()
40     {
41         return displayString;
42     }
43
44     private void updateDisplay()
45     {
46         int temp = hours.getValue ();
47
48         hours.setValue(temp % 12 +1);
49
50         displayString = hours.getDisplayValue() + ":" +
51                         minutes.getDisplayValue() + (temp/12 > 0 ? "pm" : "am");
52
53        hours.setValue(temp);
54     }
55 }
```

The **NumberDisplay** class.

```
1
2  /**
3   * The NumberDisplay class represents a digital number display that can hold
4   * values from zero to a given limit. The limit can be specified when creating
5   * the display. The values range from zero (inclusive) to limit-1. If used,
6   * for example, for the seconds on a digital clock, the limit would be 60,
7   * resulting in display values from 0 to 59. When incremented, the display
8   * automatically rolls over to zero when reaching the limit.
9   *
10  * @author Michael KÃ¶lling and David J. Barnes
11  * @version 2011.07.31
12  */
13 public class NumberDisplay
14 {
15     private int limit;
16     private int value;
17
18     /**
19      * Constructor for objects of class NumberDisplay.
20      * Set the limit at which the display rolls over.
21      */
22     public NumberDisplay(int rollOverLimit)
23     {
24         limit = rollOverLimit;
25         value = 0;
26     }
27
28     /**
29      * Return the current value.
30      */
31     public int getValue()
32     {
33         return value;
34     }
35
36     /**
37      * Return the display value (that is, the current value as a two-digit
38      * String. If the value is less than ten, it will be padded with a leading
39      * zero).
40      */
41     public String getDisplayValue()
42     {
43         if(value < 10) {
44             return "0" + value;
45         }
46         else {
47             return "" + value;
48         }
49     }
50
51     /**
52      * Set the value of the display to the new specified value. If the new
53      * value is less than zero or over the limit, do nothing.
54      */
55     public void setValue(int replacementValue)
56     {
57         if((replacementValue >= 0) && (replacementValue < limit)) {
58             value = replacementValue;
59         }
60     }
61
62     /**
63      * Increment the display value by one, rolling over to zero if the
64      * limit is reached.
65      */
66     public void increment()
67     {
68         value = (value + 1) % limit;
69     }
70 }
```

Exercise 3.34

The object diagram of the *Mail Server* and three *Mail Clients*.

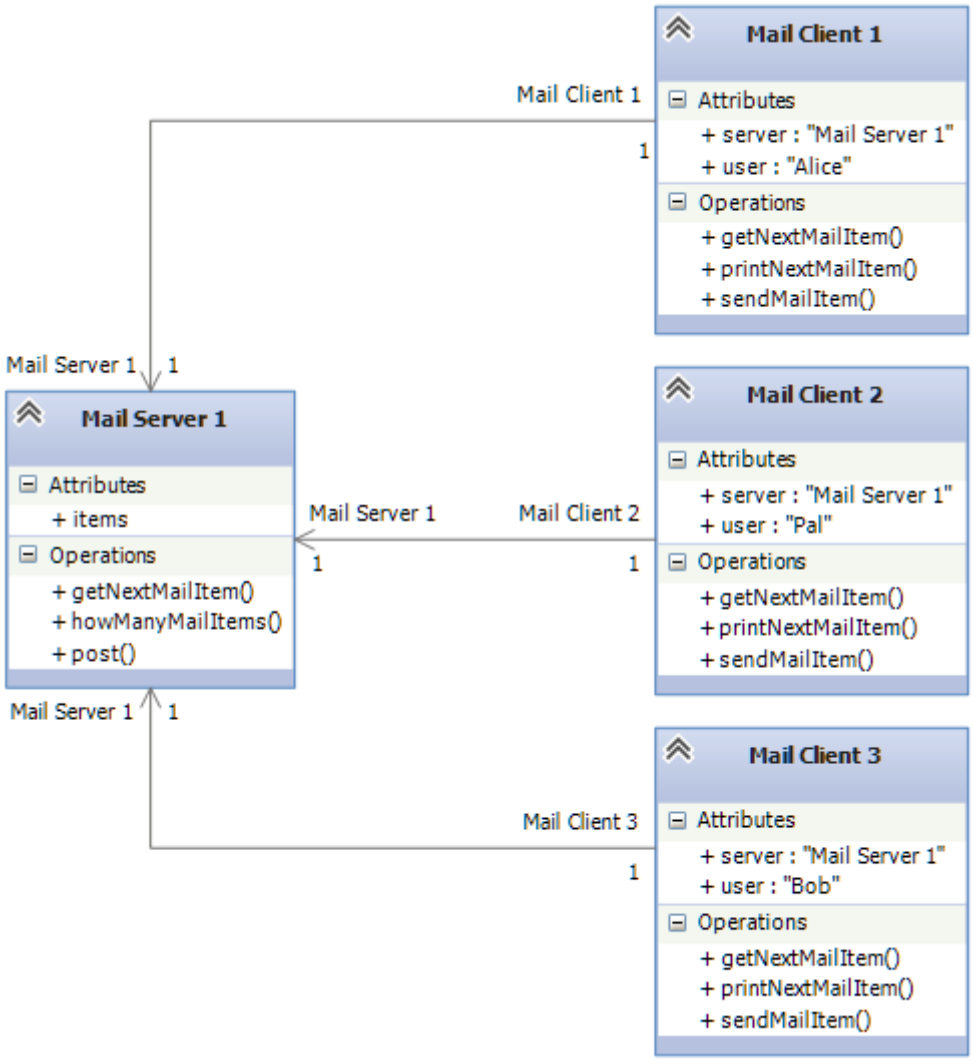

Figure 3: The arrow from Mail Clients is drawn because each Mail Client is connected to a certain server. The field "server" in each Mail Client takes care of the server.