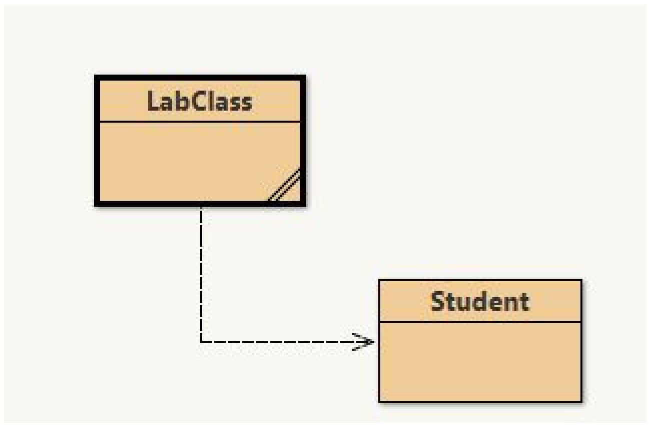
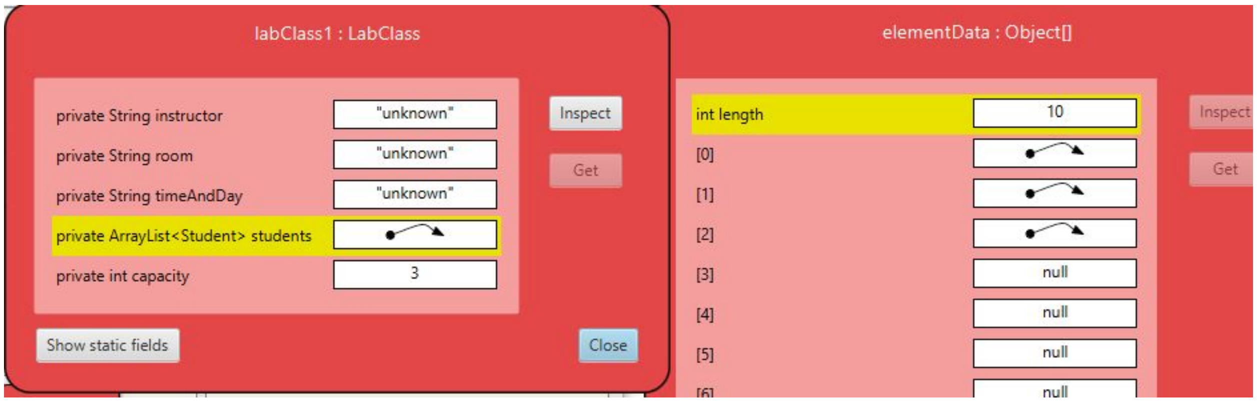
Chapter 3

3.1:class diagram:



Object diagram:



```
private String instructor              "unknown"
private String room                    "unknown"
private String timeAndDay              "unknown"
private ArrayList<Student> students
private int capacity                   3
```

```
int length    10
[0]
[1]
[2]
[3]           null
[4]           null
[5]           null
[6]           null
```

3.2: A class diagram can only change if the source code is modified

3.3: Object diagrams change when objects are created or modified during execution

3.4: private Instructor tutor;

3.5:

hours.increment();

hours.getValue()

  returned int 0

3.6:

minutes.increment();

minutes.getValue()

The program gives no feedback when the increment method is called

3.7: *code pad used*

3.8: Error: non-static method getValue() cannot be referenced from a static context

3.9: nd.setValue(int 5);

Error ‘.class’ expected
nd.setValue(5);
3.10: When the setValue method is called with an illegal value nothing happens. This is a good solution some user feedback would be prefered
3.11: Without the ‘=’ the setValue method will no longer accept 0
3.12: setValue would then accept values if only one of the conditions is met rather than both

3.13: False
True
False
False
True
3.14: (a && b || !a && !b)
3.15: (!a && b || a && !b)
3.16: (a = b)

3.17: The NumberDisplay will work with any limit but it is limited to two digits in the values that it will return

3.18: No, it won't matter because if one of the operands is a string then the end result will be a string
3.19: from 2.79 the results were
12cat and cat93 I was surprised at first but it makes sense if you read the expression from left to right. The string operand makes the end result a string but if the string is added in first rather then last the end result becomes a string before the numbers are added.

3.20: modulo is similar to division but instead returns the remainder instead of the division rather than how many times a number could fit in another number
3.21: (8 % 3) = 2
3.22: (8 % 3) = 2; (-8 % 3) = -2; (8 % -3) = 2; (-8 % -3) = -2
(10 % 6) = 4; (-10 % 6) = -4; (10 % -6) = 4; (-10 % -6) = -4
Some rules from testing
If the first number is negative the result will be negative
If the second number in negative the result will be positive
If both numbers are positive the result will be positive
If both numbers are negative the result will be negative
Or the same sign as the left operand
3.23: 0, 1, 2, 3, 4

3.24: the all possible results of (n % m) will be all integers from 0 to (m-1 or one less then the right operand)
3.25: the increment method uses modulo to set the NumberDisplay to 0 if the value reaches the limit. This is happens because this expression uses the limit as the right operand and modulo will be the result of one less than the right operand. So whatever the limit is the expression will reset to zero once it counts all the way up.

3.26:
```
public void Increment()
    {
       if(value == limit){
          value = 0;
       }
       else{
          value ++;
       }
    }
```
The modulo expression is a simpler way of incrementing

3.27: The initial time the getTime method gets is 00:00.
This happens because of the code in the constructor that creates and stores 2 NumberDisplay objects so that the display objects are created at the same time as the ClockDisplay
3.28: // simulates the actual display
3.29: you would need to call the timeTick method 60 times to show 01:00 from a new clock object or you could use the setTime method and use 1 for hours and 0 for minutes.

3.30:
```
private rectangle window
      window = new Rectangle(int, int)
```

3.31: The second constructor creates 2 NumberDisplay just like the first one but the user must input the minutes and hour rather than getting a 00:00 display. After that instead of calling the updateDisplay(); method the second constructor calls the setTime method. The setTime method sets the variables to whatever the user inputs and then calls updateDisplay for the user to see.
3.32: Answered above^

3.33:
```
p1.print(fileA,true);
      p1.getStatus(3);
```

```
3.34: wall = new Square();
        window = new Square();
        roof = new Triangle();
        sun = new Circle();
        sun2 = new Circle();
        person1 = new Person();
```

The picture class creates 6 objects but only 4 different types Square, Triangle, Circle, and Person

```
3.35:   roof.changeSize(60, 180);
         roof.moveHorizontal(20);
         roof.moveVertical(-60);
         roof.makeVisible();
```

3.36: No, the picture class does not contain any internal method calls

3.37:

```
public void draw()
   {
      if(!drawn) {
         wall.moveHorizontal(-140);
         wall.moveVertical(20);
         wall.changeSize(120);
         wall.makeVisible();


         window.moveHorizontal(-120);
         window.moveVertical(40);
         window.changeSize(40);
         window.makeVisible();

         roof.changeSize(60, 180);
         roof.moveHorizontal(20);
         roof.moveVertical(-60);
         roof.makeVisible();

         sun.moveHorizontal(100);
         sun.moveVertical(-40);
         sun.changeSize(80);
         sun.makeVisible();

         sun2.changeColor("yellow");
         sun2.moveHorizontal(-175);
```

```
        sun2.moveVertical(-20);
        sun2.changeSize(90);

        setColor();
        drawn = true;
    }
}
```

3.38:

```
    public ClockDisplay()
    {
        hours = new NumberDisplay(13); //changed from 24 to 13
        minutes = new NumberDisplay(60);
        updateDisplay();
    }

    /**
     * Constructor for ClockDisplay objects. This constructor
     * creates a new clock set at the time specified by the
     * parameters.
     */
    public ClockDisplay(int hour, int minute)
    {
        hours = new NumberDisplay(13); //changed from 24 to 13
        minutes = new NumberDisplay(60);
        setTime(hour, minute);
    }


public void timeTick()
    {
        minutes.increment();
        if(minutes.getValue() == 0) {  // it just rolled over!
            hours.increment();
        }
        if(hours.getValue() == 0) { //added if statement to rollover hour from 12 to 1
            hours.increment();
        }
        updateDisplay();

    }
```

3.39:
```
private void updateDisplay()
{
  if(hours.getValue() > 12)
  hours.setValue(hours.getValue()-12);
   displayString = hours.getDisplayValue() + ":" + minutes.getDisplayValue();

}
```
This subtracts 12 whenever the hours field is greater than 12. This one seems like the easier option with less code and cleaner code

3.40:
```
public class Tree
{
   // instance variables - replace the example below with your own
   private Triangle leaves;
   private Square trunk;
   /**
    * Constructor for objects of class Tree
    */
   public Tree()
   {
      // initialise instance variables
      leaves = new Triangle();
      trunk = new Square();
   }
```
3.41:
```
public class Tree
{
   // instance variables - replace the example below with your own
   private Triangle leaves;
   private Square trunk;
   /**
    * Constructor for objects of class Tree
    */
   public Tree()
   {
      // initialise instance variables
      leaves = new Triangle();
      trunk = new Square();
   }
```

```
        /**
          *
          */
         public void setup()
         {
             // put your code here
             leaves.makeVisible();
             leaves.changeColor("green");
             leaves.moveVertical(-50);
             leaves.moveHorizontal(20);
             leaves.changeSize(135,75);

             trunk.makeVisible();
             trunk.changeColor("brown");
             trunk.moveVertical(102);
             trunk.moveHorizontal(-102);
             trunk.changeSize(40);
         }
       }
```

3.42: *Sent some mail*

3.43 //empty square	mailServ1: MailServer	//empty square
^ mailClie1: MailClient “Chris”	^ mailClie2: MailClient “Alex”	^ mailClie3: MailClient “Gary”

3.44; 3.45; 3.46; *Became a mail man*
3.47: The next line would be either 40 or 43 in this case i would guess 40 because we did make some mail for juan making the if(item == null) statement false
3.48: This time the next line would be 40 making the if(item == null) statement true so the user receives the “No new mail” message

3.49: The Step button in the debugger showed the code execution in the print() method. Then it showed from, to, and message one after another
3.50: *instance variables Initialized*
3.51: A MailClient object will call the sendMailItem method then a new MailItem object is created with the parameters from, to, message.

mailClie1: MailClient server[mailServ1] user[“sophie”]	mailServ1: MailServer items [mail]	Juan: MailClient Server[mailServ1] user[“Juan”]
	mail: To[“Juan”] Message [“Hello”]	

3.52: I didn't see anything that struck me as unusual stepping through creating and using the clock display with breakpoints and the debugger
3.53: *if-statement options with the debugger exhausted*
3.54:
Not all the code is here the bold parts are the modifications

```
public class MailItem
        {
          // The sender of the item.
          private String from;
          // The intended recipient.
          private String to;
          // The text of the message.
          private String message;
          // The subject for the message.
          private String subject;

        public MailItem(String from, String subject, String to, String message,)
                {
                    this.from = from;
                    this.subject = subject;
                    this.to = to;
                    this.message = message;
                }
public void print()
   {
      System.out.println("From: " + from);
     **System.out.println("Subject: " + subject);**
      System.out.println("To: " + to);
      System.out.println("Message: " + message);
   }
}
```
Then in the MailClient class

```
public void sendMailItem(String to, **String subject**, String message)
   {
      MailItem item = new MailItem(user, to, message, **subject**);
      server.post(item);
   }
```

3.55:

```
Screen screen1 = new Screen (1000, 800);
If (numberOfPixels() > 2000000)
{
        screen1.clear(true);
}
```
3.56:

1. Add a field for the NumberDisplay object for seconds in the ClockDisplay class
2. The two constructors need to create another NumberDisplay objects
3. The timeTick method needs to now increment seconds
4. The setTime method needs a new parameter for seconds and a way to set the seconds
5. The updateDisplay method needs to display the seconds with the other variables

3.57:

```
public class ClockDisplay
{
   private NumberDisplay hours;
   private NumberDisplay minutes;
   **private NumberDisplay seconds;**
   private String displayString;    // simulates the actual display

   public ClockDisplay()
```

```
{
   hours = new NumberDisplay(24);
   minutes = new NumberDisplay(60);
   seconds = new NumberDisplay(60);
   updateDisplay();
}

public ClockDisplay(int hour, int minute, int second)
{
   hours = new NumberDisplay(24);
   minutes = new NumberDisplay(60);
   seconds = new NumberDisplay(60);
   setTime(hour, minute, second);
}

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


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

public String getTime()
{
        return displayString;
    }

    private void updateDisplay()
    {
        displayString = hours.getDisplayValue() + ":" +
                        minutes.getDisplayValue() + ":" +
                        seconds.getDisplayValue();

    }
}
```

3.58: if you wanted to you could keep adding on more fractions of seconds in the same way we did by adding another if statement that check for a smaller fraction of time.

3.59: