**Chapter 2**

Ex. 3.1 – A class diagram for the project LabClass would have arrows drawm from the LabClass to the Student class. This is because the information in a object is dependent on student objects.

Ex 3.2 - Class diagrams will change as we write, remove, or change code that relates classes.

Ex 3.3 – Object diagrams can change as it runs. The object diagram will always reflect the present state of the program.

Ex. 3.4 – public class Instructor

{

Private Tutor;

}

Ex. 3.5 – Done

Ex. 3.6 – Every time we click increment we should check its value once the value reverts we increment the hour once. This process will repeat as long as we want to run our clock.

Ex. 3.7 – Done

Ex. 3.8 – The code pad displays “Error: cannot find symbol - variable getValue”.

Ex. 3.9 – To get the command to work we must remove “int” from inside the parenthesis

Ex. 3.10 – If an “illegal value” is entered into the “setValue” method it keeps the previous value. Though this keeps things from getting out of hand however the user may not know that the value they tried to pass. In short, displaying a warning message when such an error occurs would be best.

Ex. 3.11 – This change to the code makes it so you couldn’t change the replacement value to 0 or the limit. In short you couldn’t reset the value with this method.

Ex. 3.12 –

Ex. 3.13– Of the items listed only “(34 !=33) && ! false” is true

Ex. 3.14– if (a= true && b= true) {

Return true;

}

Else if (a = false && b = false){

Return false;

}

Else {

Return true;

}

Ex. 3.15– if( a = true && b =true){

Return false;

}

Else if( a = true && b =true){

Return false:

}

Else{

Return true;

}

Ex. 3.16 – if( a == b) {

Return true;

}

Else {

Return false;

}

Ex. 3.17 – The method getDisplayValue will work most of the time. with the way the code is written its assumption that the numbers will always be positive is always true. If we create an object with a display limit of 800 it will function just as it should and when it resets to 0 it will continue to work.

Ex. 3.18 – There seems to be no functional between writing “return value + “”;” and “return “” + value;”

Ex. 3.19– the first piece (9 + 3 + “cat”) will come out to 12cat. This is because it reads left to right, first it adds two ints then places that result in a string with cat. The second, “cat” + 3 + 9, will come out to cat39. This is because we first at 3 to a string and then it is followed by a 9 so it becomes cat39.

Ex. 3.20– The modulo operator (%) is also known as the remainder operator. We don’t often think of remainders. But they have some uses, some examples are determining if a umber is even or odd with an if and else statement if the remainder is 0 then we know it is even everything else is odd. Apparently, it can also be used with arrays. It is useful here because can “wrap” the last filled element around to the first filled element. This can help us avoid empty element if they aren’t needed.

Ex. 3.21– The result of 8 % 3 would be 2.

Ex. 3.22– If we input negative numbers and us modulo the remainder comes back as negative.

Ex. 3.23– if we have n % 5 the only possible answers are 0, 1, 2, 3, or 4.

Ex. 3.24– all possible answers to n % m are 0=< anwers < m.

Ex. 3.25– The increment method works by starting at zero and adding one to value every time it is used. By having %limit follow we are finding the remainder every time and until we reach the limit the remainder will be the value. Once we hit the limit the remainder is 0 and the value roles over.

Ex. 3.26– I prefer using modulo as it is shorter and simpler. So I would choose it over an if and else statements.

Ex. 3.27– The clock starts at zero because that is the default for objects of the NumberDisplay class. When we create an instance of clock display we also create two private number display objects, with limit 24 the other 60.

Ex. 3.28– Done

Ex. 3.29– To get our clock to display to read 1:00 we would have to use method timeTick 60 time or we could simply use the method “setTime” to manually change it to that time.

Ex. 3.30– public house (Rectangle window)

{

Window = newRectangle (5 ,3)

}

Ex. 3.31– The second constructor for ClockDisplay is almost identical to first exepct that is has us set a time for the clock to start at so instead of being forced to start at 00:00 we could have it start wherever we wanted on our 24-hour clock.

Ex. 3.32– The second constructor doesn’t have to update the display because we set the time before it is constructed whereas the first has it’s time set during construction so the information has to be assigned during that process.

Ex. 3.33– The two possible method calls are

Public void print()

{

String filename = “File”;

If(“File”= doubleSided){

return System.out.println(“File is double-sided”);

}

Else {

return System.out.println(“File is not double-sided”);

}

}

And

Public int getStatus(int delay)

{

Delay = glitch;

Return delay;

}

Ex. 3.34– The house project creates two rectangles, a triangle, and a circle. All of these shapes are their own class and used by the picture class to generate the image.

Ex. 3.35– it the following methods for all objects created, moveHorizontal, moveVertical, makeVisible, and changeSize. All of these methods were defined the shape classes but are called using dot notation in the picture class.

Ex. 3.36 – All methods in the picture class call external methods.

Ex. 3.37 – Making it internal is stupid and illogical as it increases complexity of the class Picture.

Ex. 3.38 – done and saved

Ex. 3.39 – with what we have the best way to make a twelve hour clock was to leave the internal workings of it alone and simply changing what we see. This not only saved time but makes it so we only add some conditional statements to the “updateDisplay” method. If we were to actually make a 12-hour clock we would have to rewrite portions of both classes. In short it is easier to alter what we already have into what we want than go through the long process of essentially starting from scratch.

Ex. 3.40 – Done and Saved

Ex. 3.41 – Done and Saved

Ex. 3.42 –Done

Ex. 3.43 – the object diagram would be an object mail server with arrows from it pointing to all users.

Ex. 3.44 –done

Ex. 3.45 – done

Ex. 3.46 – done

Ex. 3.47 – When I click the step button the program will test the if statements conditions but it isn’t until the next step that it will do something we can see because it must read the next line to know what to do next.

Ex. 3.48 – As we step through the method it is only the lines above the current on that have been read and completed.

Ex. 3.49 – When using the step into it goes into the else statement whereas simply clicking step will cause the to go to the end of the if/else statements. I believe this is because step button treats if and else statements as one thing where as step in goes line by line through the statements.

Ex. 3.50 – Done

Ex. 3.51 – The classes MailClient and MailItem interact a lot in the mail system project. All the methods for MailClient include at least a call to the MailItem class. The sendMailItem method will create a MailItem object which is then sent to the server and can be viewed by the recipient. To run the print mail an if statement checks to see if a MailItem exists and will either diplay the message ore read no mail.

Ex. 3.52 – To me the ClockDisplay behaves pretty as much as I expected it would. All methods required both classes and that every time we used a method it would update the clock at the end of every method.

Ex. 3.53 – Done

Ex. 3.54 – Done and saved.

Ex. 3.55 –

public class Screen

{

// fields for screen

private int xRes;

private int yRes;

// constructor for string

public Screen( int xRes, int yRes){

this.xRes = xRes;

this.yRes = yRes;

}

// gives number of pixels

public int numberOfPixels(){

return xRes\*yres

}

// Clears screen if enough pixels present.

public void clear (Boolean invert){

int pixels = xRes\*yRes

if(invert)

{

if(pixels >= 2000000)

pixels = 0;

}

else System.out.println("Not enough pixels to clear screen.");

}

}

Ex. 3.56 – In order to create a clock that would display seconds as well as minutes and hours another number display must be added and int will be almost identical to the minute display as 60 seconds in a minute and 60 minutes in an hour. To get it to work I will create another method that is secondsTick and rename timeTick to minuteTick. The secondsTick method will be identical to the minuteTick except for it will increase minutes and when it roles over it will increment minuts by one instead of hours.

Ex. 3.57 – Done and saved.

Ex. 3.58 – The design of clock display could be increases so that it shows down to the hundredth of a second however it wouldn’t be very pretty but with enough checks to make sure everything increments when it is supposed to. It could be altered enough to support such a display.

Ex. 3.59 – I guarantee it is possible however I would have no idea where to start with it.