Lab Report

Exercise 3: Rock around the Clock

Lab-Partner:	Pavel Tsvyatkov	Pavel.Tsvyatkov@Student.HTW-Berlin.de	
Lab-Partner:	Marie Lencer	Marie.Lencer@Student.HTW-Berlin.de	
Collaboration:	Kenneth Englisch	Kenneth.Englisch@Student.HTW-Berlin.de	
Date:	07.05.2019	Group:	1. Zug 1. Gruppe

Index

1. Lab

1.1 Exploring ClockDisplay

At first we took a look at ClockDisplay to find out how ClockDisplay works and how it uses NumberDisplay.

ClockDisplay has two fields of type NumberDisplay. One field is called hours and has a rollover limit of 24 while the limit of minutes is 60. We can create a clockDisplay which is either set to 00:00 or to any time we want, using two different constructors.

To make the clock tick we can use timeTick() which uses the method increment from the class NumberDisplay. Increment adds a 1 to the value and uses modulo limit. If the value reaches the limit the new value will be 0. If the new value of minute is 0 timeTick() will increment the hour. Similar to how the minute sets back to 0 when the limit is reached, hour will also set back to 0 using the limit of hour for modulo.

You can also use setTime which calls on setValue from NumberDisplay. setValue has a conditional that checks if the new value is equal or higher than 0 and also less than the limit. ClockDisplay also uses the updateDisplay() method. In this method ClockDisplay uses getDisplayValue from the class NumberDisplay. getDisplayValue also checks if the value is less than 10 and if it is, it adds a zero in front of the value. Otherwise it just returns the value as a string. So ClockDisplay has instances of NumberDisplay and calls on methods from NumberDisplay for these instances.

1.2 Changing the Clock to American Time

The ClockDisplay is set to a 24-hour format. To get the ClockDisplay to show the time in American time, am or pm, we have to change the method updateDisplay(). At first we thought it would be easiest to change the time to a 12-hour format in NumberDisplay, in getDisplayValue(). Because we thought it would be easier to subtract 12 from the value if its larger than 12, before the value becomes a String. This did not work very well because minutes also uses getDisplayValue() so 12 would not only be subtracted from the hour but also from the minutes. So we decided to only work in ClockDisplay. Therefore we took a look at getDisplayValue(). At first we tried to only use getValue() and setValue(). We implemented a conditional for

hours.getValue() > 12 and tried to change the value with hours.setValue(hours.getValue() - 12);. This did not work. So we asked Kenneth for advice and he suggested to use a local variable. So we created a local variable int displayHour and replaced hours.setValue() with displayHour = hours.getValue() - 12;. Because we used displayHour we also had to change displayString: displayString = displayHour + ":" + minutes.getDisplayValue() + americanTime;. americanTime is a local variable of type String. If the hour value is 12 or higher in the 24-hour clock, americanTime is set to “pm” otherwise it is set to “am”. In the else condition we just wrote: displayHour = hours.getValue();. Afterwards we still had a problem with 12:00 am and 12:00 pm. Because in the 12-hour clock there is no 0 o’clock so we added to the else condition another condition for hours.getValue() == 0 and changed the hour to 12: displayHour = 12;. We still had to get 12:00 in the 24-hour clock to be displayed as 12:00 pm. So we added an else if to the method for hours.getValue() == 12 and set displayHour = hours.getValue();. This did work but we came upon another problem. If the hour value consists of only one digit the 0 is not added in front of the String. So we had to add another if statement for this case and used another local variable of type String displayHourString. If the if statement becomes true displayHour is saved to displayHourString as a String with a 0 in front. DisplayString will then use displayHourString instead of displayHour.

```
    /**
 * Update the internal string that represents the display.
 */
private void updateDisplay()
{
    int displayHour;
    String displayHourString;
    String americanTime;
    if(hours.getValue() > 12){
        displayHour = hours.getValue() - 12;
        americanTime = "pm";
    }
    else if (hours.getValue() == 12){
        displayHour = hours.getValue();
        americanTime = "pm";
    }
    else {
        if(hours.getValue() == 0){
            displayHour = 12;
            americanTime = "am";
        }
        else {
            displayHour = hours.getValue();
            americanTime = "am";
        }
    }
    if(displayHour <10){
        displayHourString = "0" + displayHour;
         displayString = displayHourString + ":" +
                    minutes.getDisplayValue() + americanTime;
    }
    else {
        displayString = displayHour + ":" +
                    minutes.getDisplayValue() + americanTime;
    }
}
```

1.3 Another Way to Change to American Time

Another way to set the clock to american time is to actually change the clock from a 24-hour clock to a 12-hour clock and not just change the display. As this was supposed to be done in a second class, we copied the class ClockDisplay and called the new class ClockDisplayAT. We changed the constructor to set the limit for hour to 12. We also implemented a new field of type String called americanTime. In the constructor we set americanTime to “am”. For setValue we added a parameter String ampm and set americanTime to ampm.

```
/**
 * Set the time of the display to the specified hour and
 * minute.
 */
public void setTime(int hour, int minute, String ampm)
{
    hours.setValue(hour);
    minutes.setValue(minute);
    americanTime = ampm;
    updateDisplay();
}
```

Because setValue gained a parameter we also had to add a new parameter to the second constructor which lets you set a time.

```
/**
 * Constructor for ClockDisplayAT objects. This constructor
 * creates a new clock set at 00:00.
 */
public ClockDisplayAT()
{
    hours = new NumberDisplay(12);
    minutes = new NumberDisplay(60);
    americanTime = "am";
    updateDisplay();
}
```

```
/**
 * Constructor for ClockDisplayAT objects. This constructor
 * creates a new clock set at the time specified by the
 * parameters.
 */
public ClockDisplayAT(int hour, int minute, String ampm)
{
    hours = new NumberDisplay(12);
    minutes = new NumberDisplay(60);
    americanTime = ampm;
    setTime(hour, minute, ampm);
}
```

Because we now had a variable containing either am or pm we had to find a way to change am and pm with the rollover of the hours. So we added a conditional to timeTick() so that when the hours rolled over and depending on whether it was am or pm it would be set to the other.

```
/**
 * This method should get called once every minute - it makes
 * the clock display go one minute forward.
 */
public void timeTick()
{
    boolean rollover = false;
    minutes.increment();
    if(minutes.getValue() == 0) {  // it just rolled over!
        hours.increment();
        if (hours.getValue() == 0 && minutes.getValue() == 0){
            rollover = true;}
        if (rollover && americanTime == "am") {
            americanTime = "pm";
            rollover = false;}
        else if (rollover && americanTime == "pm") {
            americanTime = "am";
            rollover = false;}
    }
    updateDisplay();
}
```

For updateDisplay we added and worked with the local variable dispayHour again. We also added an if statement, that when hours.getValue() == 0 displayHour would be set to 12, else displayHour becomes hours.getValue(). Then we just exchanged hours.getDisplayValue() with displayHour and added americanTime to the end of the String. We also made sure that a 0 would be added in front of the displayHour if the value of hour is less than 10 and not 0.

```
/**
 * Update the internal string that represents the display.
 */
private void updateDisplay()
{
    int displayHour;
    if(hours.getValue() == 0){
        displayHour = 12;
    }
    else {
        displayHour = hours.getValue();
    }
    if (hours.getValue() < 10 && hours.getValue() != 0){
    displayString = "0" + displayHour + ":" +
                    minutes.getDisplayValue() + americanTime;
    }
    else{
    displayString = displayHour + ":" +
                    minutes.getDisplayValue() + americanTime;
    }
}
```

We tried it all out and it worked the way we wanted it to.

Comparing the two methods for implementing a 12-hour clock, we think that both ways have their pros and cons. With the first method it is probably easier to make the clock flexible in displaying the time either as a 12-hour clock or a 24-hour clock by just adding a second method that displays the time in the 24-hour clock again. On the other hand with our ClockDisplayAT we implemented a much neater way to display a 12-hour clock. We did have to change more methods but because the clock actually counts in 12-hours the updateDisplay() is structured more clearly. So we don't think that one way is better than the other, but that both ways have their advantages.

1.4 Programming an Alarm

For setting an alarm we worked in ClockDisplay again. First we added two new fields String alarmTime; and boolean alarmSet;. In the constructors alarmTime is set to an empty String and alarmSet is set to false. Afterwards we wrote a new method public void setAlarm. The method takes three parameters: int alarmHour, int alarmMinute, String alarmAT. In the body the parameters are concatenated and saved in alarmTime. alarmSet is also set to true. Here we came upon the same problem with single digit values for alarmHour and alarmMinute. We solved the problem by using two local variables of type String called alarmMinString and alarmHourString. We added two separate sets of if- and else-statements, one for alarmHour and one for alarmMinute. If the corresponding parameter is less than 10, "0" will be concatenated in front of the parameter and it will be saved in alarmMinString or alarmHourString. Otherwise just alarmMin/alarmHour will be saved as a String in alarmMinString/alarmHourString. Afterwards alarmHourString, alarmMinString and alarmAT are saved as one String to alarmTime. We also added a statement to print alarmTime to the terminal, so that we can remember the time we set the alarm to.

```
public void setAlarm (int alarmHour, int alarmMinute, String alarmAT){
    String alarmMinString;
    String alarmHourString;
    if(alarmHour <10){
        alarmHourString = "0" + alarmHour;
    }
    else{
        alarmHourString = "" + alarmHour;
    }
    if(alarmMinute <10){
        alarmMinString = "0" + alarmMinute;
    }
    else{
        alarmMinString = "" + alarmMinute;
    }
    alarmTime = alarmHourString + ":" + alarmMinString + alarmAT;
    alarmSet = true;
    System.out.println("The alarm is set to " + alarmTime);
}
```

To make the alarm go off we worked in timeTick(). There we added an if statement, so that when the alarm is set and alarmTime equals displayString, “Riiiiiiiing!” will be printed to the terminal.

```
/**
 * This method should get called once every minute - it makes
 * the clock display go one minute forward.
 */
public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) {  // it just rolled over!
        hours.increment();
    }
    updateDisplay();
    if(alarmSet && alarmTime.equals(displayString)){
        System.out.println("Riiiiiiiing!");
    }
}
```

We also added a setAlarmOff method. This sets alarmSet back to false and prints to the terminal: "The alarm is turned off.".

```
public void setAlarmOff(){
    alarmSet = false;
    System.out.println("The alarm is turned off.");
}
```

2. Reflection

Marie:

In this Lab I learned a lot about how two classes interact with one another and where it is better to implement changes to the methods to change the output. Furthermore I got to see how one outcome could be achieved through different methods.
I also learned a lot about testing out all the possibilities of what could go wrong with the changes and trying to come up with a solution for them. I do not think we found the nicest solution to the problems, but at least they work the way we want them to. Further experimenting would probably lead to better solutions.

Pavel:

This Lab exercise was different from the ones we had before, because we had to do things on our own and come up with our own solution. I learned a lot, mainly how classes interact with each other, how to properly set a conditional statement, so that I get the exact result that's needed. In this exercise I learned how to think more about the problem itself and what different steps we had to take, so that we can come up with a working solution. I learned that there could be more than one solution to a problem and different solutions have their advantages and disadvantages.

3. Appendix

3.1 Written Code

ClockDisplay:

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;    // simulates the actual display
    private String alarmTime;
    private boolean alarmSet;

public ClockDisplay()
{
    hours = new NumberDisplay(24);
    minutes = new NumberDisplay(60);
    alarmTime = "";
    alarmSet = false;
    updateDisplay();
}

public ClockDisplay(int hour, int minute)
{
    hours = new NumberDisplay(24);
    minutes = new NumberDisplay(60);
    setTime(hour, minute);
    alarmTime = "";
    alarmSet = false;
}

public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) {  // it just rolled over!
        hours.increment();
        }
    updateDisplay();
    if(alarmSet && alarmTime.equals(displayString)){
        System.out.println("Riiiiiiiing!");
    }
}
```

```
private void updateDisplay()
{
    int displayHour;
    String displayHourString;
    String americanTime;
    if(hours.getValue() > 12){
        displayHour = hours.getValue() - 12;
        americanTime = "pm";
    }
    else if (hours.getValue() == 12){
        displayHour = hours.getValue();
        americanTime = "pm";
    }
    else {
        if(hours.getValue() == 0){
            displayHour = 12;
            americanTime = "am";
        }
        else {
            displayHour = hours.getValue();
            americanTime = "am";
        }
    }
    if(displayHour <10){
        displayHourString = "0" + displayHour;
        displayString = displayHourString + ":" +
                    minutes.getDisplayValue() + americanTime;
    }
    else {
        displayString = displayHour + ":" +
                    minutes.getDisplayValue() + americanTime;
    }
}

public void setAlarm (int alarmHour, int alarmMinute, String alarmAT){
    String alarmMinString;
    String alarmHourString;
    if(alarmHour <10){
        alarmHourString = "0" + alarmHour;
    }
    else{
        alarmHourString = "" + alarmHour;
    }
```

```
    if(alarmMinute <10){
        alarmMinString = "0" + alarmMinute;
        }
    else{
        alarmMinString = "" + alarmMinute;
    }
    alarmTime = alarmHourString + ":" + alarmMinString + alarmAT;
    alarmSet = true;
    System.out.println("The alarm is set to " + alarmTime);
}

public void setAlarmOff(){
    alarmSet = false;
    System.out.println("The alarm is turned off.");
}
```

ClockDisplayAT:

```
public class ClockDisplayAT
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;    // simulates the actual display
    private String americanTime;

public ClockDisplayAT()
{
    hours = new NumberDisplay(12);
    minutes = new NumberDisplay(60);
    americanTime = "am";
    updateDisplay();
}

public ClockDisplayAT(int hour, int minute, String ampm)
{
    hours = new NumberDisplay(12);
    minutes = new NumberDisplay(60);
    americanTime = ampm;
    setTime(hour, minute, ampm);
}

public void timeTick()
{
    boolean rollover = false;
    minutes.increment();
    if(minutes.getValue() == 0) {  // it just rolled over!
        hours.increment();
        if (hours.getValue() == 0 && minutes.getValue() == 0){
            rollover = true;}
```

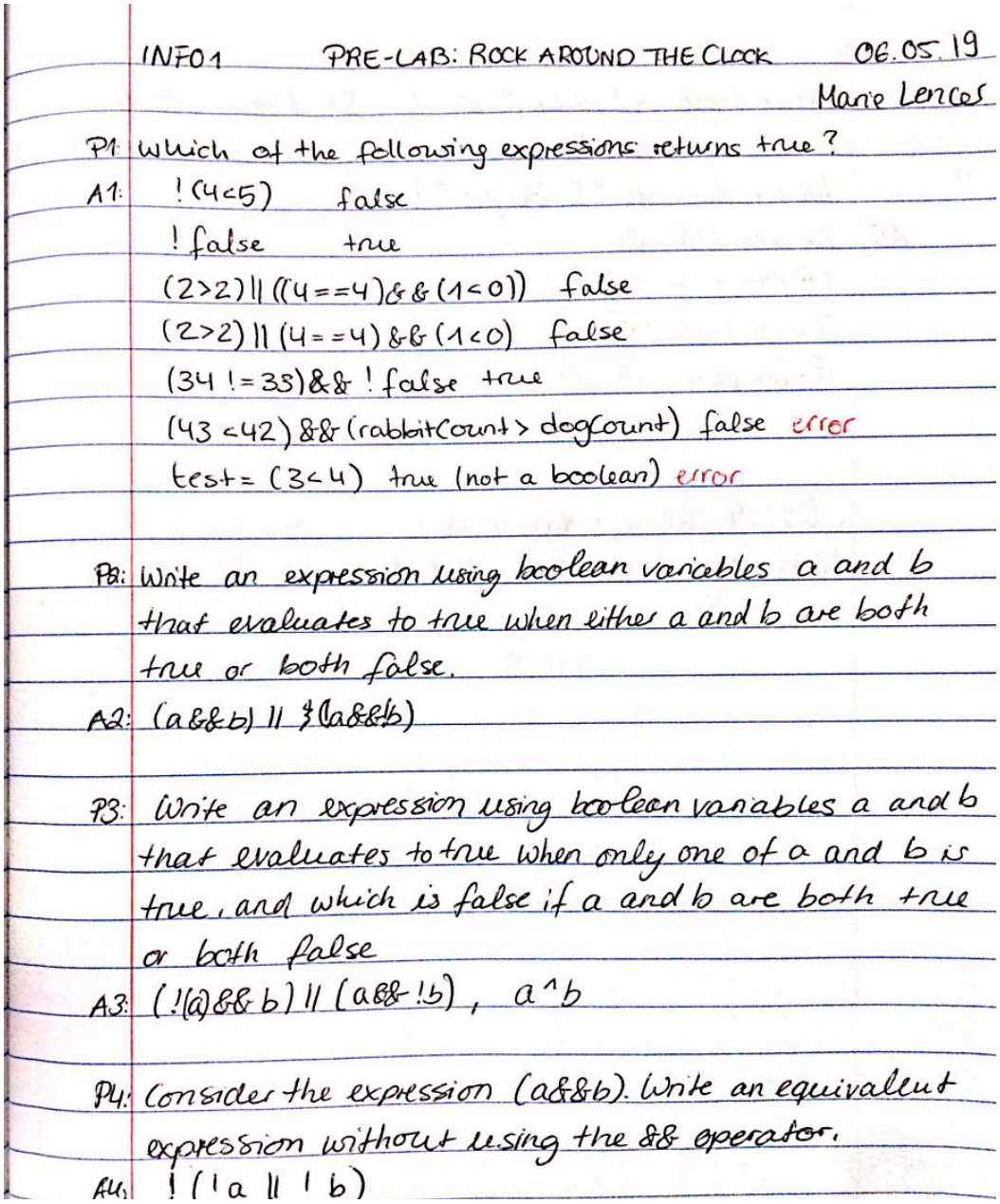
```
        if (rollover && americanTime == "am") {
            americanTime = "pm";
            rollover = false;}
        else if (rollover && americanTime == "pm") {
            americanTime = "am";
            rollover = false;}
    }
    updateDisplay();
}

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

private void updateDisplay()
{
    int displayHour;
    if(hours.getValue() == 0){
       displayHour = 12;
    }
    else {
        displayHour = hours.getValue();
    }
    if (hours.getValue() < 10 && hours.getValue() != 0){
    displayString = "0" + displayHour + ":" +
                minutes.getDisplayValue() + americanTime;
    }
    else{
    displayString = displayHour + ":" +
                minutes.getDisplayValue() + americanTime;
    }
}
```

3.2 Pre-Lab

- Marie



INFO1 PRE-LAB: ROCK AROUND THE CLOCK 06.05.19

Marie Lences

P1: Which of the following expressions returns true?

A1: !(4<5) false

!false true

(2>2)||((4==4)&&(1<0)) false

(2>2)||(4==4)&&(1<0) false

(34!=35)&&!false true

(43<42)&&(rabbitCount>dogCount) false error

test=(3<4) true (not a boolean) error

P2: Write an expression using boolean variables a and b that evaluates to true when either a and b are both true or both false.

A2: (a&&b)||!(a&&b)

P3: Write an expression using boolean variables a and b that evaluates to true when only one of a and b is true, and which is false if a and b are both true or both false

A3: (!(a)&&b)||(a&&!b), a^b

P4: Consider the expression (a&&b). Write an equivalent expression without using the && operator.

A4: !(!a||!b)

P5: What time is "12 am" on the 24-hour clock? What time is "12 pm"? What time is "3 am"? What time is "5.30 pm"?

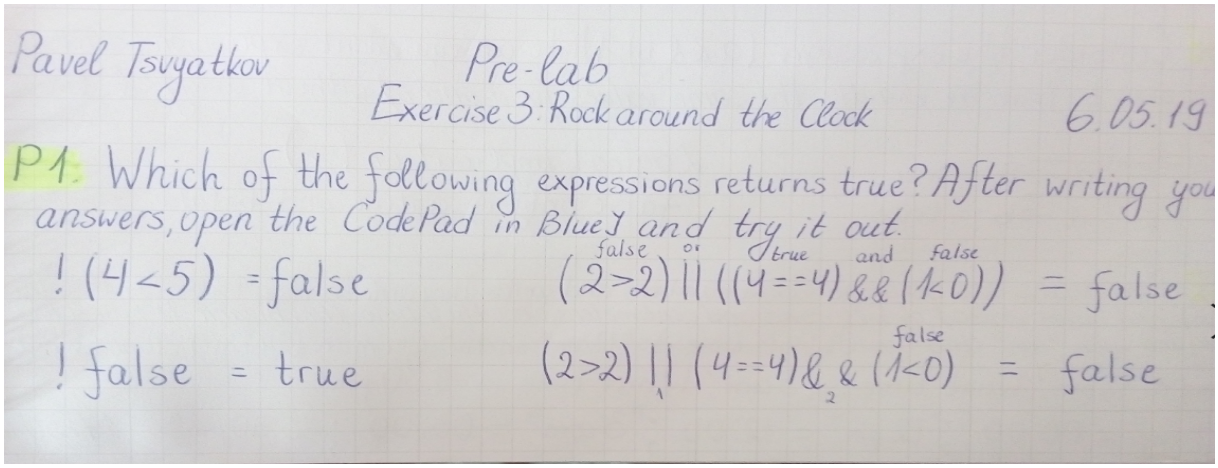
A5: 12 am =00:00

12 pm = 12:00

3 am = 03:00

5:30 pm = 17:30

- Pavel



Pavel Tsvyatkov — Pre-lab — 6.05.19

Exercise 3: Rock around the Clock

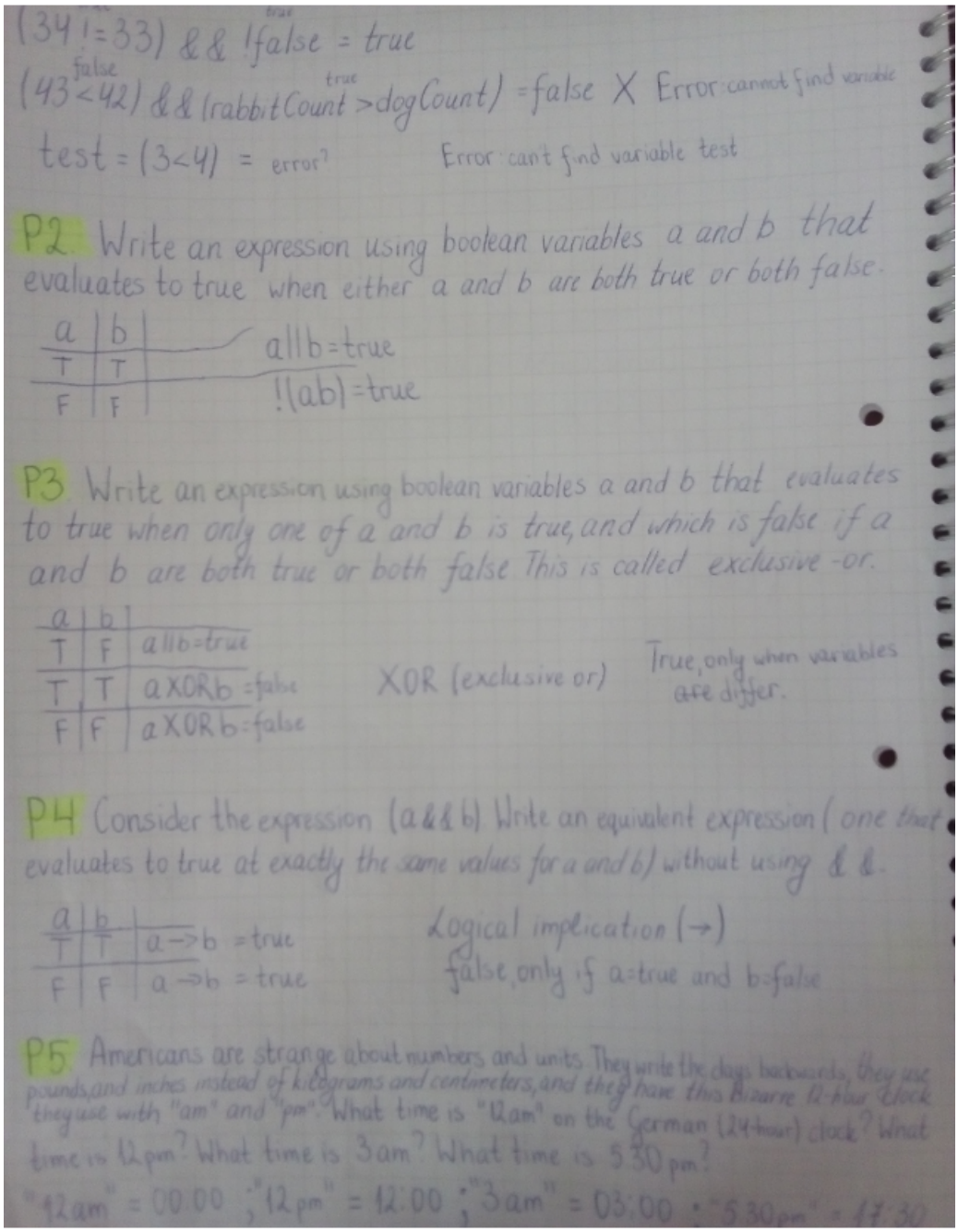
P1: Which of the following expressions returns true? After writing your answers, open the CodePad in BlueJ and try it out.

!(4<5) = false

! false = true

(2>2) || ((4==4) && (1<0)) = false (false or true and false)

(2>2) || (4==4) && (1<0) = false (false, true, false)



(34 != 33) && !false = true

(43 < 42) && (rabbitCount > dogCount) = false X Error: cannot find variable

test = (3<4) = error? Error: can't find variable test

P2: Write an expression using boolean variables a and b that evaluates to true when either a and b are both true or both false.

a	b	
T	T	a||b = true
F	F	!(a|b) = true

P3: Write an expression using boolean variables a and b that evaluates to true when only one of a and b is true, and which is false if a and b are both true or both false. This is called exclusive-or.

a	b	
T	F	a||b = true
T	T	aXORb = false
F	F	aXORb = false

XOR (exclusive or) — True, only when variables are differ.

P4: Consider the expression (a && b). Write an equivalent expression (one that evaluates to true at exactly the same values for a and b) without using &&.

a	b	
T	T	a->b = true
F	F	a->b = true

Logical implication (→) false, only if a=true and b=false

P5: Americans are strange about numbers and units. They write the days backwards, they use pounds and inches instead of kilograms and centimeters, and they have this bizarre 12-hour clock they use with "am" and "pm". What time is "12am" on the German (24-hour) clock? What time is 12pm? What time is 3am? What time is 5:30 pm?

"12am" = 00:00 ; "12pm" = 12:00 ; "3am" = 03:00 ; "5:30pm" = 17:30