Laboratory 4: A Better Notebook
Informatik 1, Group 1, Tuesday

Pavel Tsvyatkov, Lena Zellin

May 20, 2019

Contents

1 Lab plan

This lab exercise had us thinking quite a lot. After we found each other and sat down to began our work, we agreed that we would take our time and talk a lot about what we had to do and how we could do it. We then downloaded the Notebook file from moodle and opened it in BlueJ.

2 A Better Notebook

This project has a single class called Notebook that we were asked to add methods too and modify our solutions, as well as adding a class, that we had prepared in our pre lab work. We will go over all the following tasks and questions, given to us by Prof. Weber-Wulff[1], in the following sections of our report.

2.1 1. Using the CodePad, test that your prelab work in P1 and P2 is correct. What test cases did you have to create?

To start we both compared our pre lab work and told the other group member how and why we came to our solution. After that we started typing our code into the CodePad. However, we had had a few problems in the beginning, since this was the first time we put loops into the CodePad and we struggled with a few error messages, until we realized we had to put everything in a single line for it to work without any problems. For P1 both our solutions worked and they were very similar. Pavel wrote: for(int a = 10; a <= 95; a++){if(a%5==0){System.out.println(a);}}.
Lena wrote: for(int i = 10; i <= 95; i = i+5){System.out.println(i);}. These loops print the multiples of five starting with 10 and ending with 95.
Of course, Pavels for loop is better in the way that the if condition checks if the number is actually a multiple of five, while Lenas for loop uses the fact that 10 is divisible by 5 and adding 5 will always result in a number divisible by 5.
Next we tested our solutions for P2. We both had pretty much the same idea:

```
public void sumBetween(int a, int b){
 int sum = 0;

 while(a < b){
  sum = sum + a;
  a++;
 }
 System.out.println(sum);
}
```

This code is pretty much a mash up from both our solutions that works like intended. The only difference to Lenas solution is, that she wrote while(a!=b), which also works for this task, and Pavel wrote while(a<=b), which works, but unfortunately also adds b to sum, which is not what the task wanted. However, when omitting the equal sign it works as wanted.
The test cases we created by passing a and b were: a is smaller(including negative numbers) than b, which works, as that is the condition that needs to be met. B is smaller than a, which also works in the sense of that it prints out sum, which is zero in that case.

2.2 2. Adapt the notebook project, which is similar to the music library we worked on in class, to list all notes. Create a notebook and add some notes, checking that this method works as intended.

For this task we thought the best solution would be to use an Iterator. We started by importing java.util.Iterator for this to work. Since we wrote a similar method for our pre lab exercise P3, we used that to write the following method:

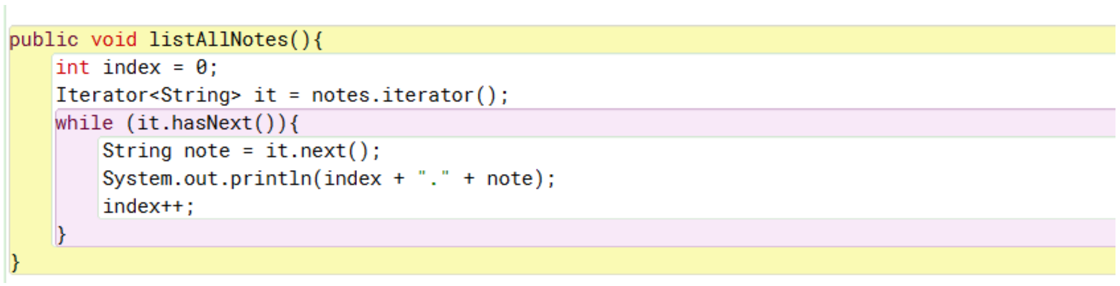
```
public void listAllNotes(){

 Iterator<String> it = notes.iterator();
 while (it.hasNext()){
  String note = it.next();
  System.out.println(note);
 }
}
```

We created an Iterator of type String that we called it for convenience and 'it' iterates over every note, which are Strings, in our Arraylist notes. Then we wrote a while loop that calls the Iterator method .hasNext, which checks for 'next' items in our iteration or basically if there is a next item that our while loop body gets performed on. The 'next' item in our Iterator gets saved as a String called note and then printed out. This way every item gets printed to the terminal. After we finished writing our method we added a few notes to out notebook and called our method. All of our notes were printed to the terminal, so we knew our method worked.

2.3 3. Modify your list method so that it prints a number in front of each note that corresponds to its index in the ArrayList.

This task made us think for a bit. We talked about how to add a number in front of every note and which way we could implement. We settled on initializing a new variable that we named int index that we assigned the value 0 to. Then we added it to our System.out.println and added 1 to it, every time a note in our ArrayList was printed.

```
 public void listAllNotes(){

  int index = 0;
  Iterator<String> it = notes.iterator();
   while (it.hasNext()){
    String note = it.next();
    System.out.println(index + "." + note);
    index++;
   }
 }
```



```
public void listAllNotes(){
    int index = 0;
    Iterator<String> it = notes.iterator();
    while (it.hasNext()){
        String note = it.next();
        System.out.println(index + "." + note);
        index++;
    }
}
```

To test if our method works we added a few notes in our object notebook and then called the method listAllNotes. Our notes had the correct number printed in front of them. Just to test it out, we implemented this method with indexOf instead of a local variable, this didn't work however, notes which are identical are listed with the same index.

2.4 4. Implement removeNote. You may have to look at the API for java.util.ArrayList to find out how to do this.

After searching for ArrayList and remove in the API, we read that the method remove takes an int called index as a parameter and removes whatever item is found at said index in the ArrayList.[2] We thought about it for a moment and wrote down the following method:

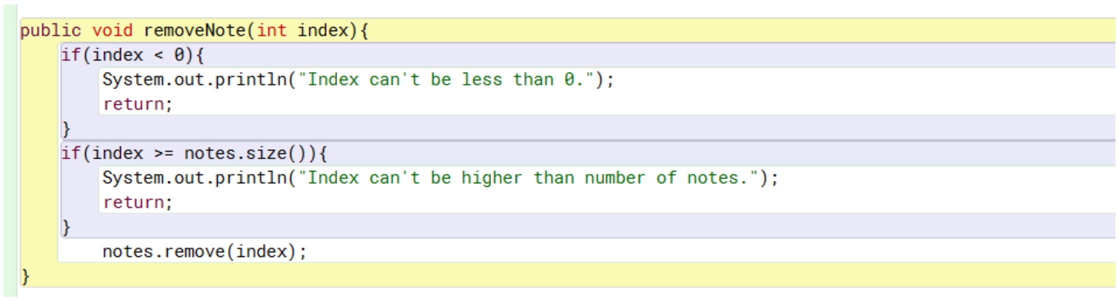
```
 public void removeNote(int index){

  notes.remove(index);
 }
```

We pass an integer as a parameter that should correspond to an index of an existing note. Then the remove method gets called on our ArrayList notes, for the note to the corresponding index. We weren't sure if the method would work as intended like that, but it compiled without a problem. Then we added a few notes to our notebook and printed them to the terminal. Then we called our removeNote method and typed in the index of a note. After that we printed our notes again and saw, that the one we called removeNote on was gone.

2.5 5. Modify removeNote to print out an error message if the note number entered was not valid.

This task made us think for a while. We talked about ways to check if a method could be removed or not but we just couldn't get anything to work. We then downloaded Prof. Weber-Wulffs Music Organizer from moodle and took a look at the method indexValid in the MusicOrganizer class. We tried adapting it so it would work with our removeNote method and it did. However this left us a little unsatisfied as we wanted to at least have a solution of our own. Later, as we thought about it again we realized that we could instead implement if statements in our method that would check for invalid input and print the error messages accordingly. This is what we wrote:

```
public void removeNote(int index){

  if(index < 0){
   System.out.println("Index can't be less than 0.");
   return;
  }
  if(index >= notes.size()){
   System.out.println("Index can't be higher than number of notes.");
   return;
  }
  notes.remove(index);
}
```



```
public void removeNote(int index){
    if(index < 0){
        System.out.println("Index can't be less than 0.");
        return;
    }
    if(index >= notes.size()){
        System.out.println("Index can't be higher than number of notes.");
        return;
    }
        notes.remove(index);
}
```

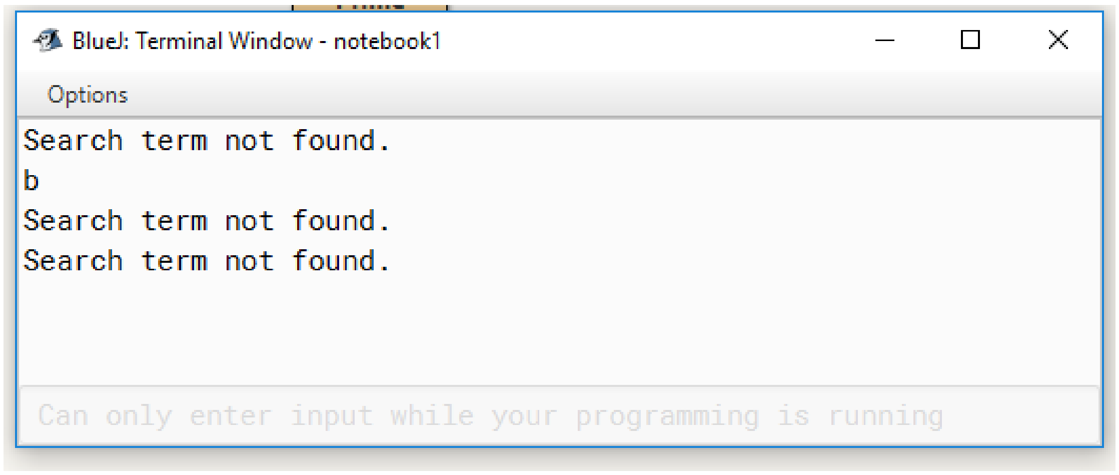
The first if statement is for checking, if the input is a negative number, which is invalid and an error message gets printed. The next if statement checks, if the input is higher or equal to the size of notes. This also results in an error message. If the input is neither too small nor to high it's valid and thus the note at the corresponding index gets removed. We, again, added notes to our Notebook and tried if everything still worked as before with Prof. Weber-Wulffs method. And it did!

2.6 6. Try and implement the search method that looks for a particular note. You need to add code to print out either the note that was found or the string "Search term not found.". Be sure to test your method with terms that are in the list, and ones which are not.

This task had us thinking back and forth. We just weren't sure how to implement a search method. At first we tried using a for-each loop, but what we wrote didn't work the way as intended. Unfortunately our time in the lab was almost up at that point so we read the next task in an attempt to go on working and get a little more done before the lesson would end. There we read that we should change our loop to a while loop for searching a note so we tried writing a method with the help of task 7 and implemented a while loop.

```
public void searchNote(String noteName){

  int index = 0;
  while(index < notes.size()){
   String note = notes.get(index);
   if(note.contains(noteName)){
    System.out.println(note);
   }else{
    System.out.println("Search term not found.");
   }
   index ++;
  }
}
```

However we didn't test our code immediately, as our lab time was up so unfortunately we only found out that it wasn't working as intended at home. Our method prints the notes that contain the String passed in the parameter, but it also prints out "Search term not found." for every note in our ArrayList that doesn't contain the String.


BlueJ: Terminal Window - notebook1
Options
Can only enter input while your programming is running

```
Search term not found.
b
Search term not found.
Search term not found.
```

We only wanted the message to be printed when no note contains the String we are looking for. We sat down and thought about a way to find the solution and tried to get it working by using a boolean variable, that 'remembers' if a note had been printed or not. Only if there was no note that contains the String we are looking for, the error message gets printed.

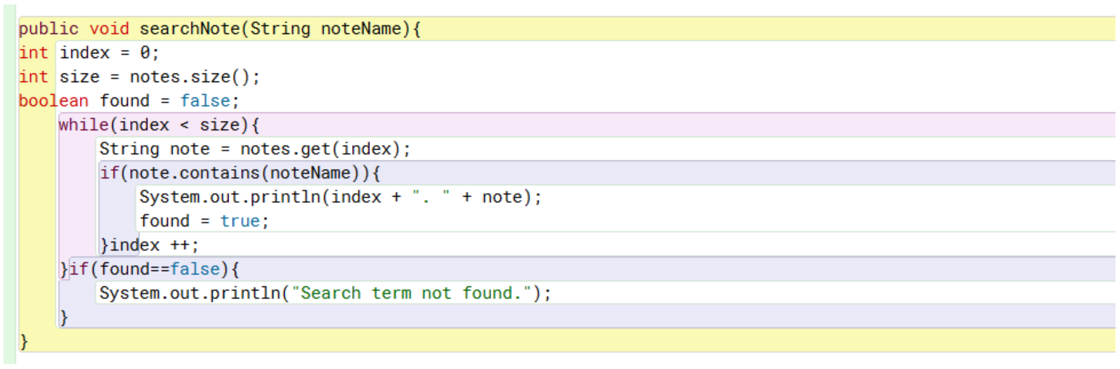
```
public void searchNote(String noteName){

  int index = 0;
  boolean found = false;
  while(index < notes.size()){
   String note = notes.get(index);
   if(note.contains(noteName)){
    System.out.println(index + ". " + notes.size());
    found = true;
   }index ++;
  }if(found==false){
   System.out.println("Search term not found.");
   }
}
```

Here, we initialized two local variables, int index that is used for going over every note in our notebook and as stated above a boolean. The while loop still checks if our index is smaller than the size of notes and if it is, the note at that index gets saved into a String called note. An if statement then checks if the note contains the String we searched for and if it does it gets printed out with the corresponding index at the beginning. When this happens, our boolean becomes true, because we now have one note that contains our search String. After that the index gets increased by one and the whole procedure starts again until we reach the last note. The last if statement is for checking if after looking at every note there was not a single one containing our search String, which would mean our boolean is still false. If that is the case a message gets printed out stating ”Search term not found.”. With this method, everything worked like we wanted it to and not only the first note containing our search String gets printed out but every note containing it.

2.7 7. How did you implement the search method? If you used a for-each loop or an iterator, first rewrite it to use a while loop. Within a single execution of the search method, the notes collection might be asked repeatedly how many notes it is currently storing. Does the value returned by size vary from one check to the next? Rewrite your search method so that the size is not checked unnecessarily, perhaps using a local variable. Check that your version gives the same results! What cases will you have to test?

Since we already kind of did task 7 as our solution to task 6, all we had to do was create a local variable instead of checking notes.size every time, since we didn’t do that from the very start.

```
public void searchNote(String noteName){

 int index = 0;
 int size = notes.size();
 boolean found = false;
 while(index < size){
  String note = notes.get(index);
  if(note.contains(noteName)){
   System.out.println(index + ". " + note);
   found = true;
  }index ++;
 }if(found==false){
  System.out.println("Search term not found.");
  }
}
```



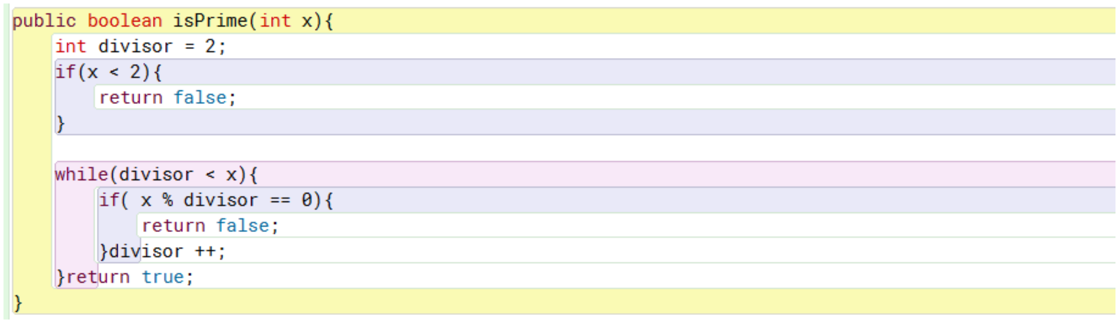
```
public void searchNote(String noteName){
int index = 0;
int size = notes.size();
boolean found = false;
    while(index < size){
        String note = notes.get(index);
        if(note.contains(noteName)){
            System.out.println(index + ". " + note);
            found = true;
        }index ++;
    }if(found==false){
        System.out.println("Search term not found.");
    }
}
```

We saved the length of our ArrayList notes in a variable called size. After adding notes to our notebook and calling seachNote, we saw that our method worked like it did before.

2.8 8. Make a new class and put the method isPrime you defined in your prelab 4 into it. In this class, create a collection primes and fill it with the prime numbers between 1 and 1000. Oops, this won't work directly, because collections don't take ints. You can use something called autoboxing to solve this. Look up Integer in the API and use this. How many prime numbers are there between 1 and 1000?

For this task we decided to use this method that we wrote for our pre lab to check if a number is prime or not.

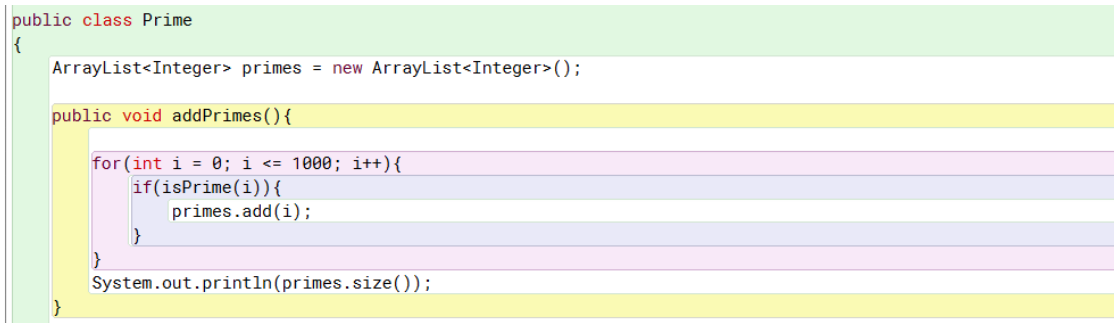
```
public boolean isPrime(int x){

  int divisor = 2;
  if(x < 2){
   return false;
  }
  while(divisor < x){
   if( x % divisor == 0){
    return false;
   }divisor ++;
  }return true;
 }
```



```
public boolean isPrime(int x){
    int divisor = 2;
    if(x < 2){
        return false;
    }

    while(divisor < x){
        if( x % divisor == 0){
            return false;
        }divisor ++;
    }return true;
}
```

This method takes an integer as a parameter and then checks if its smaller than the local variable divisor which is set to 2. Since 2 is the smallest prime number everything smaller is not a prime number. We then have a while loop that checks if divisor is smaller than x, the number we put in as a parameter, and then an if statement that calculates if x modulo divisor is 0. If a number is divisible by anything else than itself and 1 it's not a prime number, so in that case we return false and add one to divisor and do the calculation all over again until divisor is x. Since x would be divisible by x but nothing else we return true. After that we looked up autoboxing. Autoboxing means automatically converting a primitive type like int or float into Integer or Float, for example.[3] For this to work we need an ArrayList of type Integer and not int so we can call the method add of ArrayLists to store the prime numbers from 1 to 1000 in it. We wrote this method:

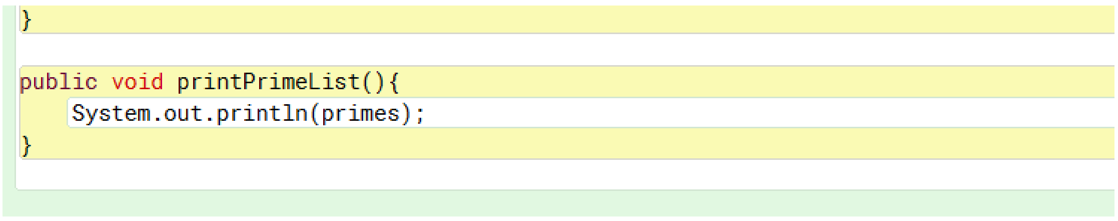
```
ArrayList<Integer> primes = new ArrayList<Integer>();

public void addPrimes(){

 for(int i = 0; i <= 1000; i++){
  if(isPrime(i)){
   primes.add(i);
  }
 }
 System.out.println(primes.size());
}
```



```
public class Prime
{
    ArrayList<Integer> primes = new ArrayList<Integer>();

    public void addPrimes(){

        for(int i = 0; i <= 1000; i++){
            if(isPrime(i)){
                primes.add(i);
            }
        }
        System.out.println(primes.size());
    }
```

First we had to import java.util.ArrayList for creating ArrayLists. We then declared an ArrayList of Integers, as int wasn't an option, named it primes and initialized it with an empty ArrayList. Next we wrote a method called addPrimes, that adds the prime numbers from 1 to 1000 into our ArrayList. It does this, by using a for loop to count from 1 up to 1000 (including 1000) and then calls our method isPrime on every number to check, if it is a prime number. If the number is prime it's added with the .add method for ArrayLists to ours. When calling this method the amount of numbers stored in our ArrayList also gets printed to the terminal, so we know immediately how many primes they are from 1 to 1000. The number is 168. Just out of curiosity we implemented another method that prints the ArrayList, to see what numbers between 1 and 1000 are stored in it.

```
public void printPrimeList(){

 System.out.println(primes);
}
```



```
}

public void printPrimeList(){
    System.out.println(primes);
}
```

With this we finished our work on this class and our lab exercise.

3 After the lab

Since we weren't finished when our lab time was up, we emailed the notes and the BlueJ file we worked on to each other so we both could continue working at home, as we both wanted to finish the exercises and practice more by working on them. After we both had solutions that worked out, we mailed each other about the lab report and pieced it together until we were happy with it. All in all we spend the time we had at the lab and around 2-3 hours for the rest of the code and the report all together.

4 What I learned

4.1 Pavel

After working on this exercise I learned more about loops and how to structure them well. I learned how to write a loop, which checks if a given number is prime or not. I worked with array lists and searched in the Java API about some of the methods that are available. I learned how to implement a search method that looks for something specific. It was important to write the conditional statement correctly, so that I get the exact result that I need and also test it with different cases, to make sure it really does what it needs to. I also learned how to use autoboxing, I had to look it up in the Java API to understand it well.

4.2 Lena

This exercise taught me about autoboxing, before that I didn't even know that this was a thing you could do. I also learned more about the remove method and feel a little more confident in using it and how to search for Strings in ArrayLists. This is something that still troubles me a little even after finding a solution that works, it took me more time than I wanted it to, so in a way this exercise taught me what I need to practice more. I also feel like I understand how loops work a little more, especially after our first attempt in task 6.

5 Appendix

5.1 Citation

References

[1] HTW Berlin, Prof.Dr. Weber-Wulff, `https://people.f4.htw-berlin.de/~weberwu/info1/Labs/Lab4.shtml`, last accessed: 19.05.2019

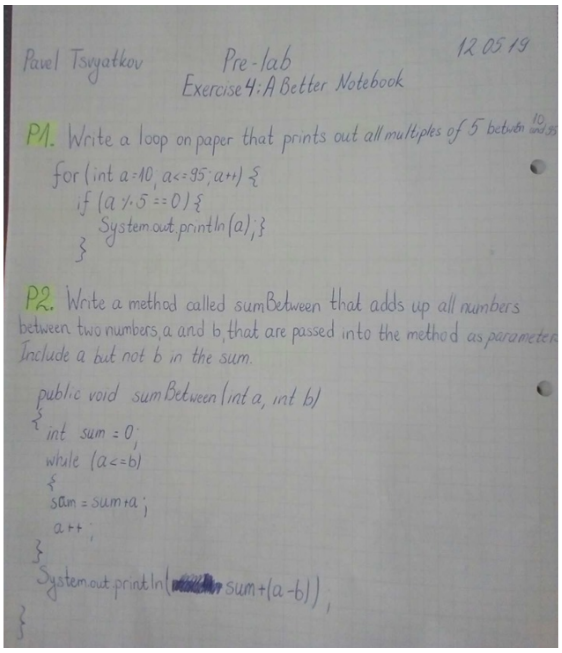
[2] Oracle, `https://docs.oracle.com/javase/9/docs/api/java/util/ArrayList.html#remove-int-` last accessed: 18.05.2019

[3] Oracle, `https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html` last accessed 18.05.2019

5.2 Pre lab

To document our pre lab work we decided to insert pictures of our solutions.

5.2.1 Pavel



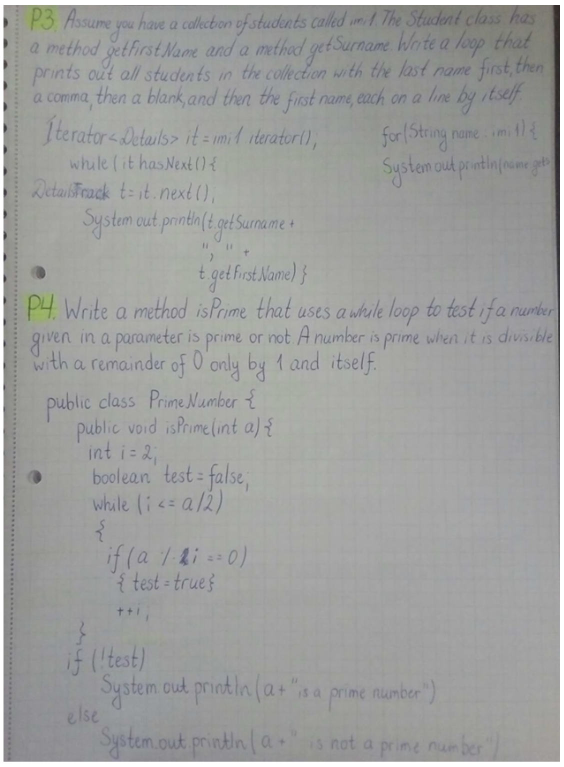
Pavel Tsvyatkov   Pre-lab   12.05.19
Exercise 4: A Better Notebook

P1. Write a loop on paper that prints out all multiples of 5 betwen 10 and 95

```
for (int a=10; a<=95; a++) {
    if (a % 5 == 0) {
      System.out.println(a); }
    }
```

P2. Write a method called sumBetween that adds up all numbers between two numbers, a and b, that are passed into the method as parameters. Include a but not b in the sum.

```
public void sumBetween (int a, int b)
{
 int sum = 0;
 while (a<=b)
 {
  sum = sum+a;
  a++;
 }
 System.out.println( sum+(a-b));
}
```



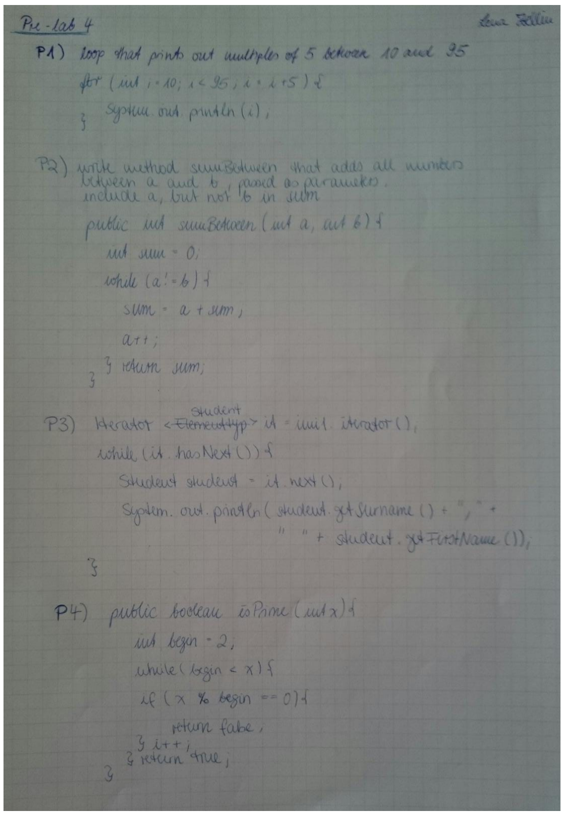
P3. Assume you have a collection of students called imi1. The Student class has a method getFirstName and a method getSurname. Write a loop that prints out all students in the collection with the last name first, then a comma, then a blank, and then the first name, each on a line by itself.

```
Iterator<Details> it = imi1.iterator();              for(String name : imi1) {
    while (it.hasNext() {                             System.out.println(name.gets
Details~~Track~~ t = it.next();
       System.out.println(t.getSurname +
                          ", " +
                          t.getFirstName) }
```

P4. Write a method isPrime that uses a while loop to test if a number given in a parameter is prime or not. A number is prime when it is divisible with a remainder of 0 only by 1 and itself.

```
public class PrimeNumber {
   public void isPrime(int a) {
    int i = 2;
    boolean test = false;
    while (i <= a/2)
    {
      if (a % ~~2~~i == 0)
       { test = true }
       ++i;
    }
  if (!test)
     System.out.println(a+"is a prime number")
  else
     System.out.println(a+" is not a prime number")
```

5.2.2 Lena



Pre-lab 4 [illegible]

P1) loop that prints out multiples of 5 between 10 and 95

```
for (int i = 10; i < 95; i = i +5) {
    System.out.println(i);
}
```

P2) write method sumBetween that adds all numbers between a and b, passed as parameters. include a, but not b in sum

```
public int sumBetween (int a, int b) {
    int sum = 0;
    while (a != b) {
        sum = a + sum;
        a++;
    } return sum;
}
```

P3)

```
Iterator <~~Elementtyp~~ Student> it = unit1.iterator();
while (it.hasNext()) {
    Student student = it.next();
    System.out.println( student.getSurname () + ", " +
                        " " + student.getFirstName ());
}
```

P4)

```
public boolean isPrime (int x) {
    int begin = 2;
    while (begin < x) {
    if (x % begin == 0) {
        return false;
    } i++;
    } return true;
}
```

5.3 Our Code

5.3.1 Notebook

```
import java.util.ArrayList;
import java.util.Iterator;
/**
* A class to maintain an arbitrarily long list of notes.
* Notes are numbered for external reference by a human user.
* In this version, note numbers start at 0.
*
* @author David J. Barnes and Michael Kolling.
* @version 2008.03.30
*/

 public class Notebook{

// Storage for an arbitrary number of notes.

private ArrayList<String> notes;

/**
* Perform any initialization that is required for the
* notebook.
*/

 public Notebook(){

  notes = new ArrayList<String>();
 }

/**
* Store a new note into the notebook.
* @param note The note to be stored.
*/

 public void storeNote(String note){

  notes.add(note);
 }

/**
* @return The number of notes currently in the notebook.
*/
 public int numberOfNotes(){

  return notes.size();
 }

/**
* Show a note.
* @param noteNumber The number of the note to be shown.
*/
 public void showNote(int noteNumber){

  if(noteNumber < 0) {
 // This is not a valid note number, so do nothing.
  }
   else if(noteNumber < numberOfNotes()) {
```

```
 // This is a valid note number, so we can print it.
    System.out.println(notes.get(noteNumber));
   }
   else {
   // This is not a valid note number, so do nothing.
  }
 }

 public void listAllNotes(){

  int index = 0;
  Iterator<String> it = notes.iterator();
  while (it.hasNext()){
   String note = it.next();
   System.out.println(index + "." + note);
   index++;
  }
 }

 public void removeNote(int index){

  if(index < 0){
   System.out.println("Index can't be less than 0.");
   return;
 }
  if(index >= notes.size()){
   System.out.println("Index can't be higher than number of notes.");
   return;
  }
  notes.remove(index);
 }

 public void searchNote(String noteName){

  int index = 0;
  int size = notes.size();
  boolean found = false;
  while(index < size){
   String note = notes.get(index);
    if(note.contains(noteName)){
     System.out.println(index + ". " + note);
     found = true;
    }index ++;
  }if(found==false){
    System.out.println("Search term not found.");
   }
 }
}
```

5.3.2 Prime

```
import java.util.ArrayList;

/**
* Write a description of class Prime here.
*
* @author (your name)
* @version (a version number or a date)
*/
public class Prime{

ArrayList<Integer> primes = new ArrayList<Integer>();

 public void addPrimes(){

  for(int i = 0; i <= 1000; i++){
   if(isPrime(i)){
    primes.add(i);
   }
  }
  System.out.println(primes.size());
 }

 public boolean isPrime(int x){

  int divisor = 2;
  if(x < 2){
   return false;
 }
  while(divisor < x){
   if( x % divisor == 0){
    return false;
   }divisor ++;
  }return true;
 }

 public void printPrimeList(){

 System.out.println(primes);
 }

}
```