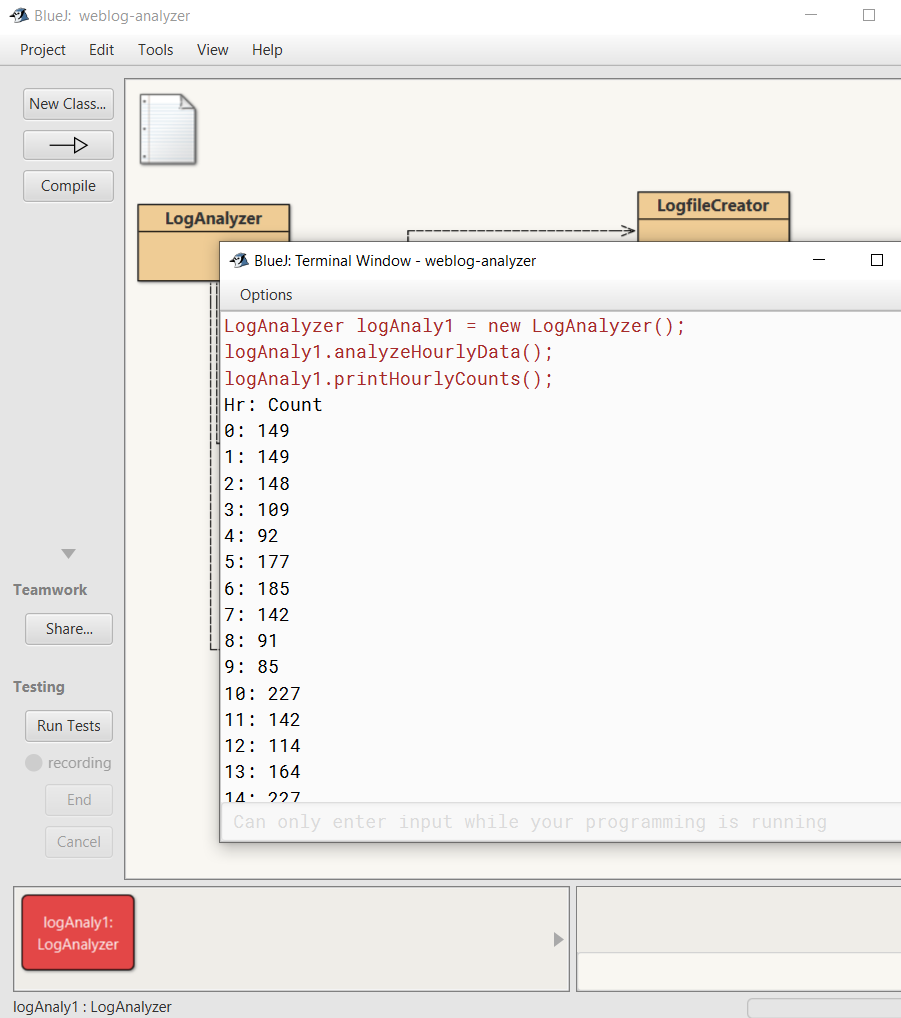
7.1



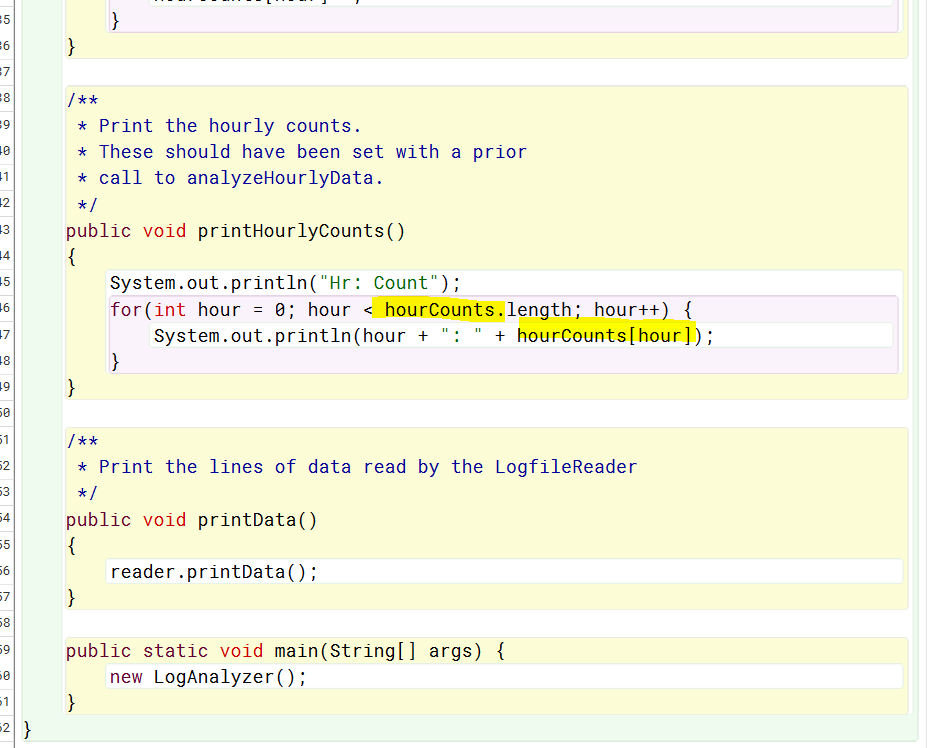
7.2

Person[] people;

7.3

boolean[] vacant;

7.4



7.5

Int[] counts;

Boolean[] occupied;

7.6

Double[] readings = new double[60];

String[] urls = new String[90];

TicketMachine[] machine = new TicketMachine[5];

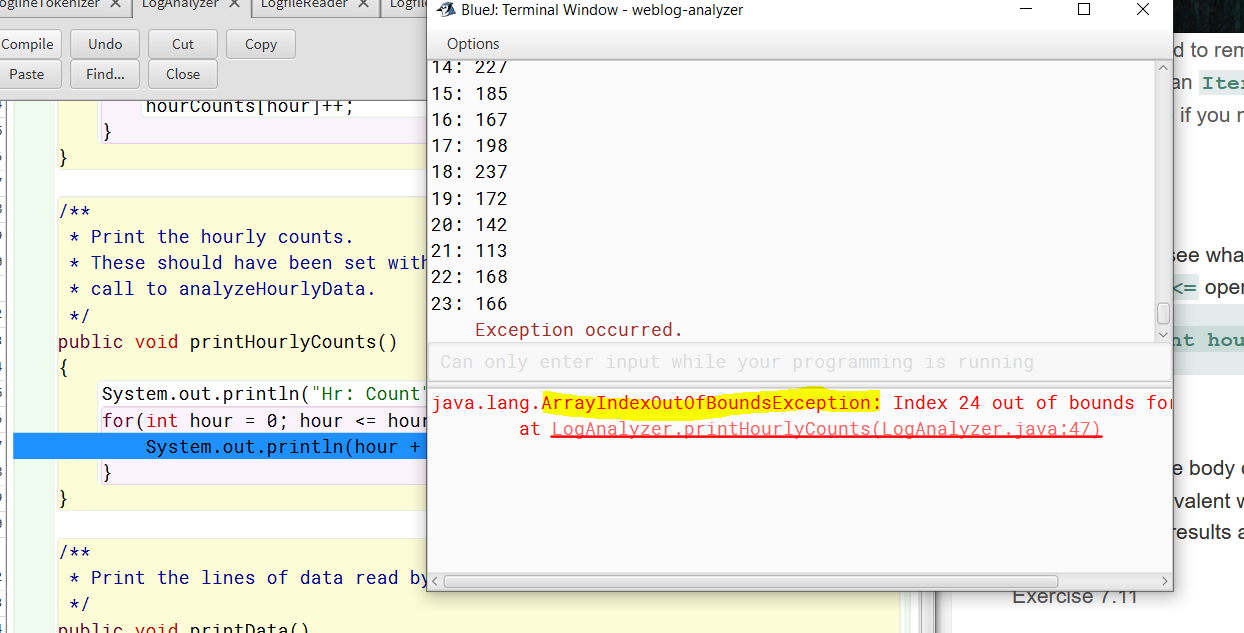
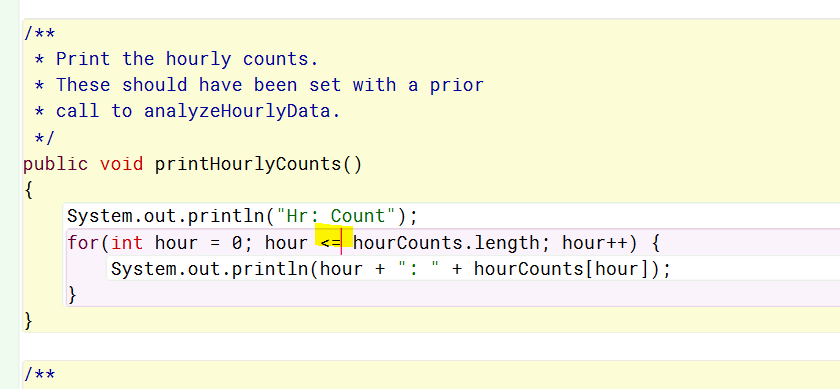
7.7

None what is created is one array object that can hold up to 20 String values.

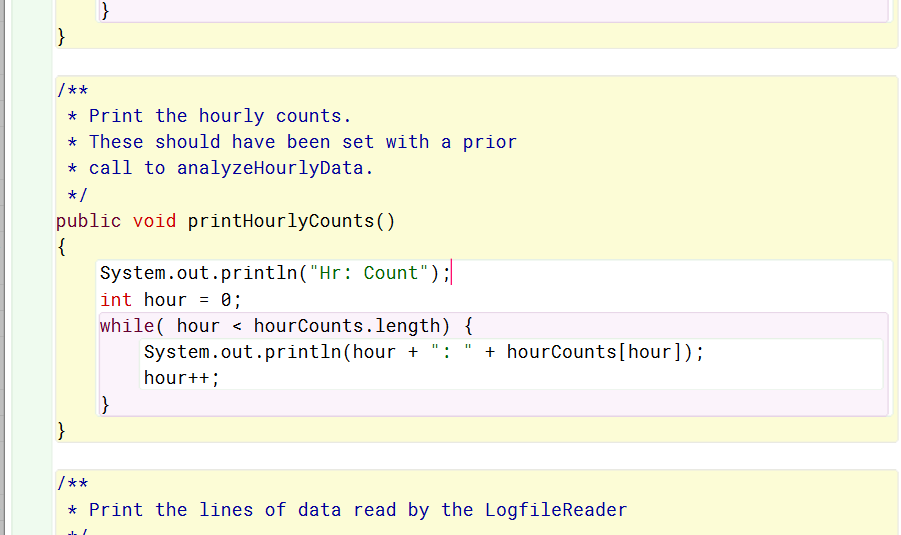
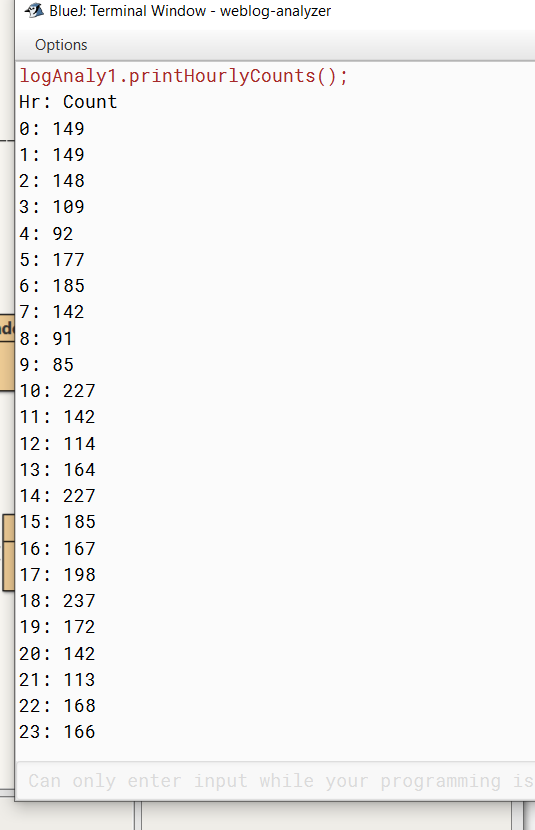
7.8

Double[] prices = new double[50]; vs not “... = new double(50)”;

7.9



7.10

7.11

Public void printGreater(double marks double mean)

{

For(int index = 0; index < marks.length; index++){

If(marks[index] > mean) {

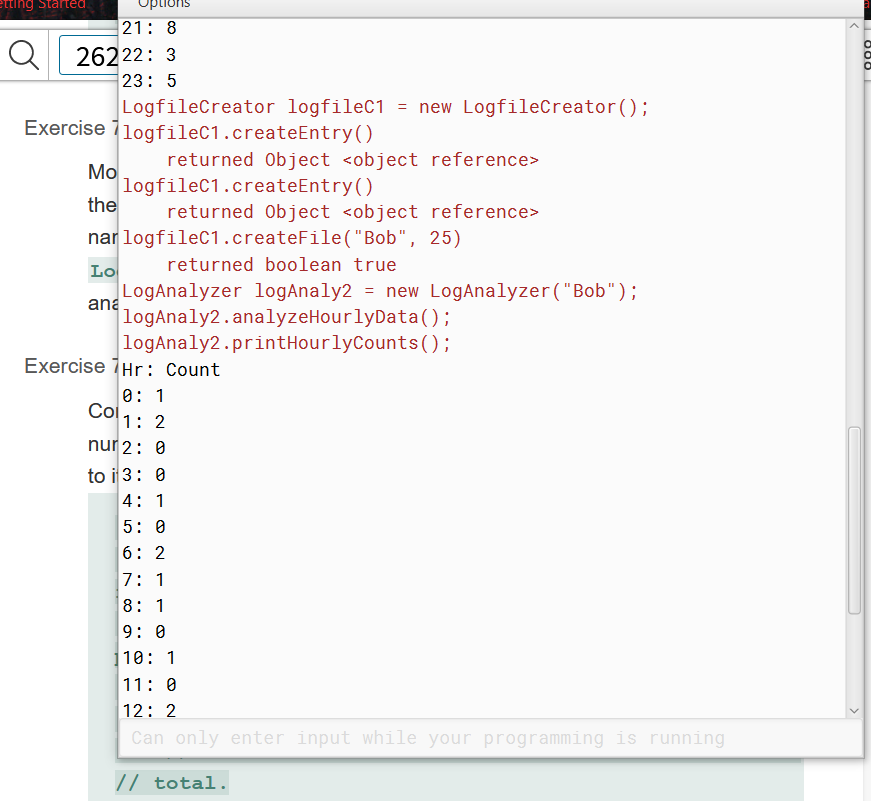
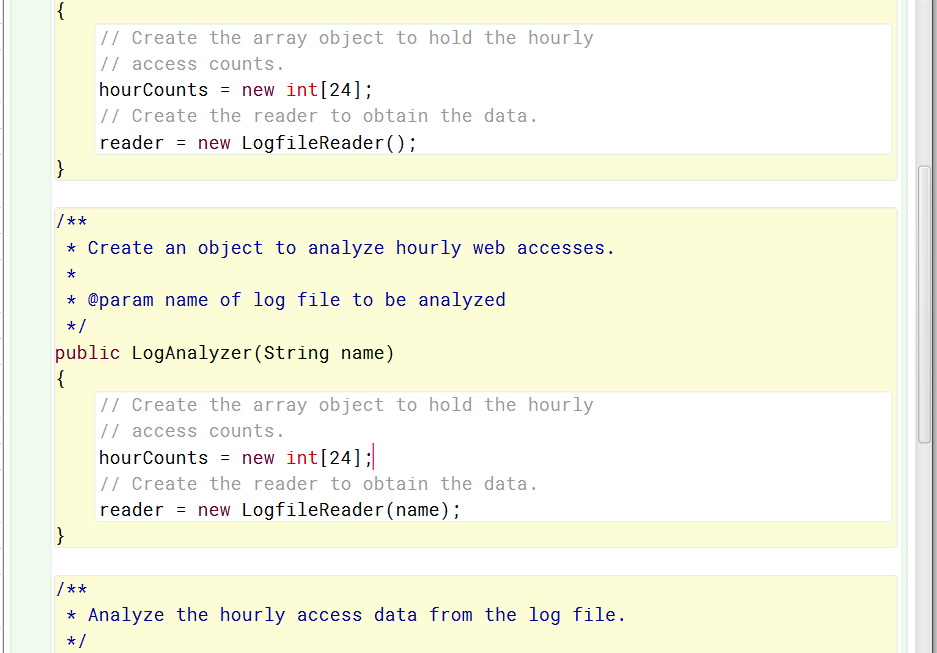
System.out.println(marks[index]);

}

}

}

7.12



7.13

Public int numberOfAccesses()

{

Int total = 0;

For(int I = 0; I < hourCounts.length; I++){

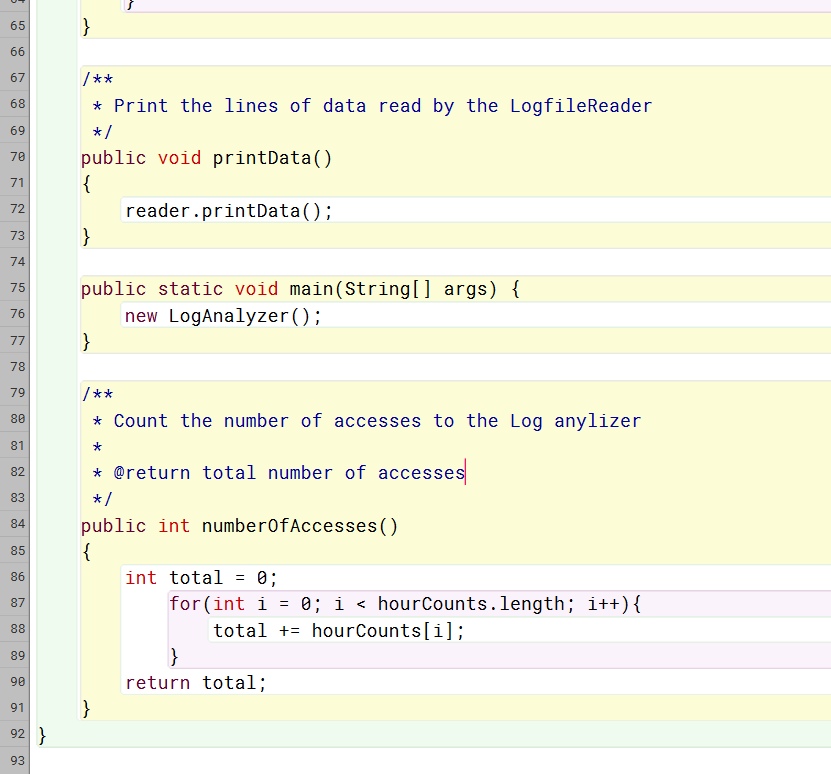
Total += hourCounts[i];

}

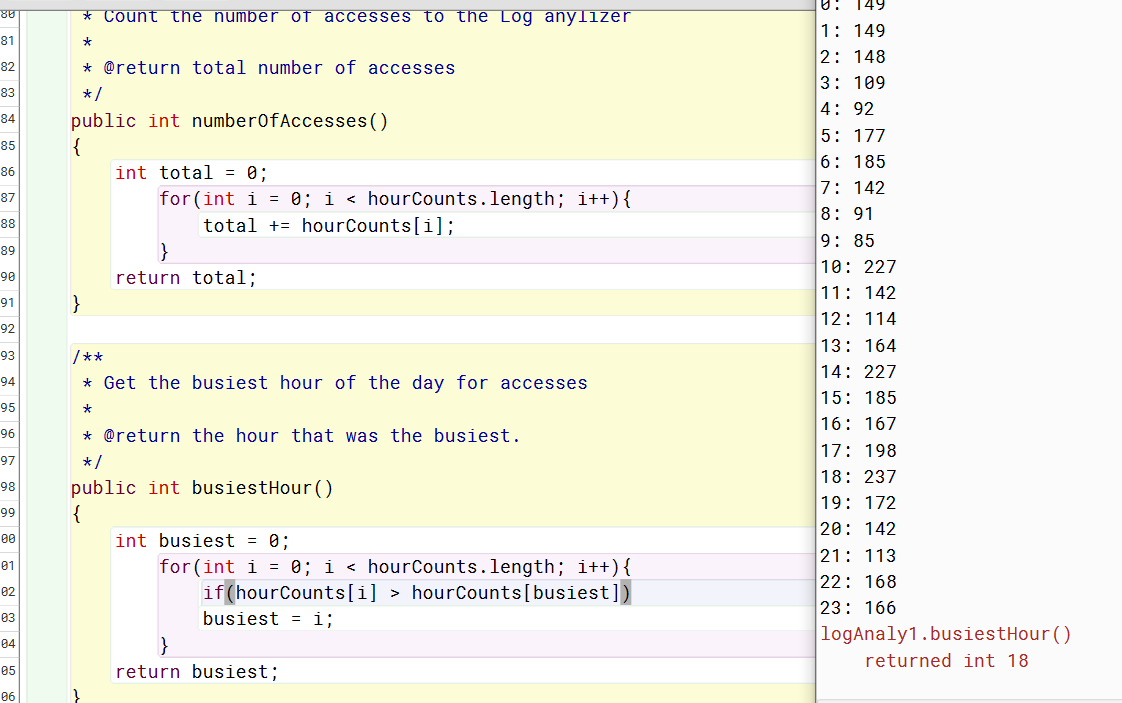
Return total;

}

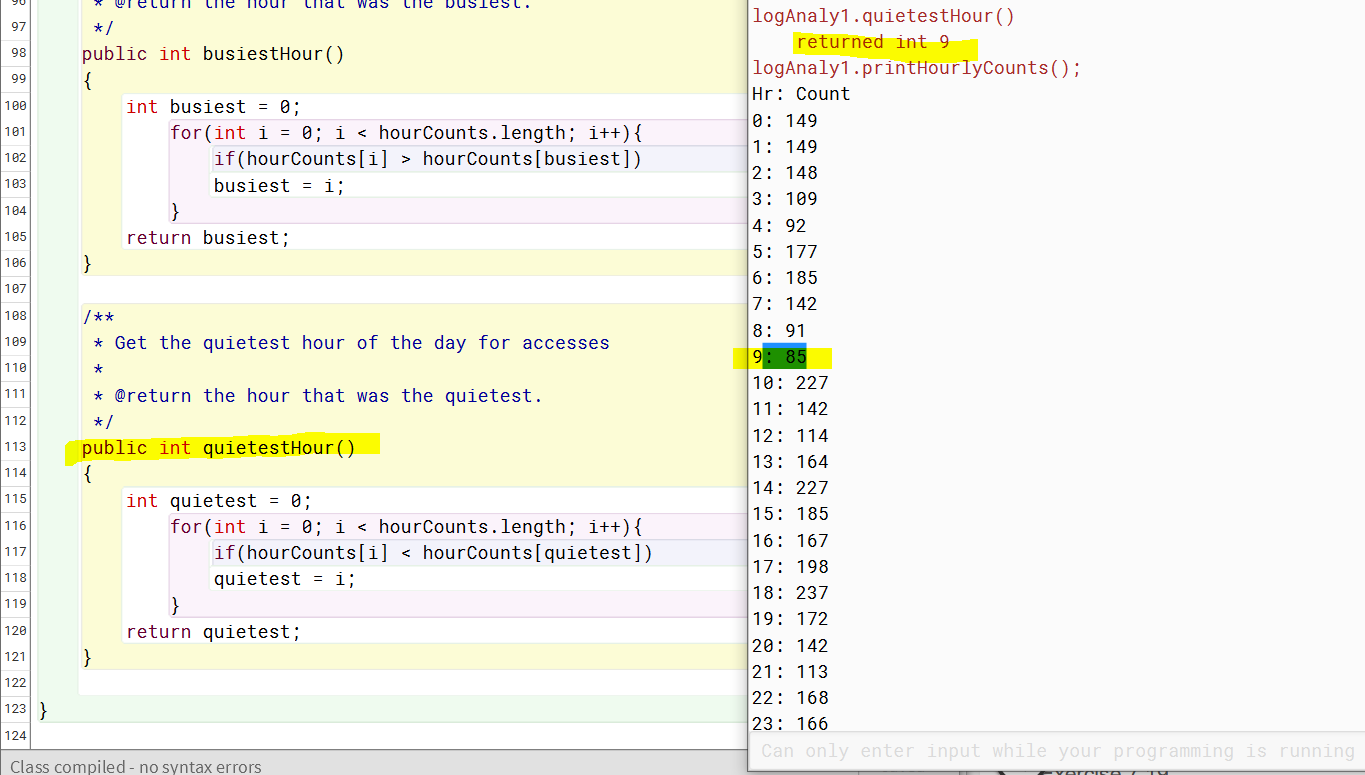
7.14



7.15



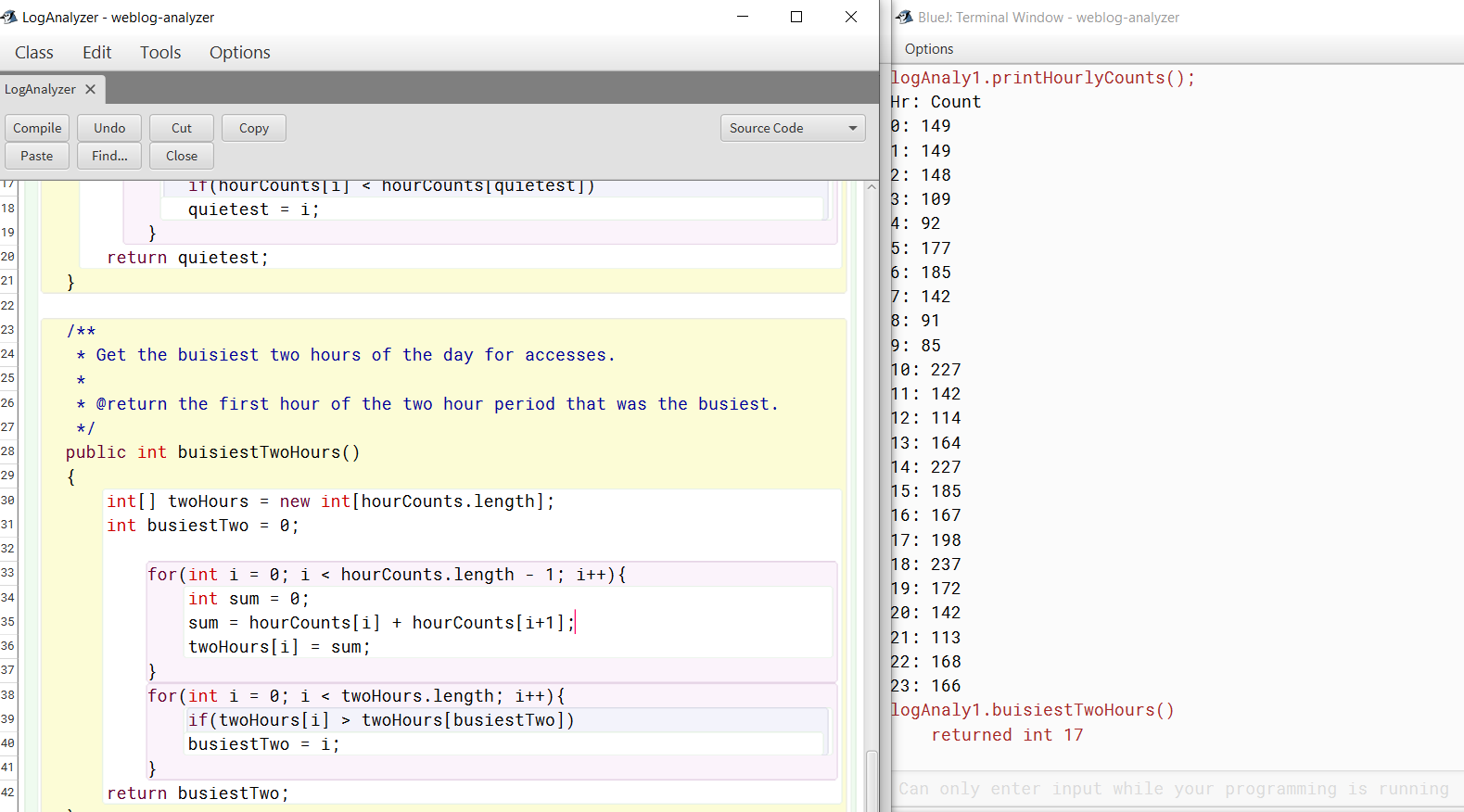
7.16



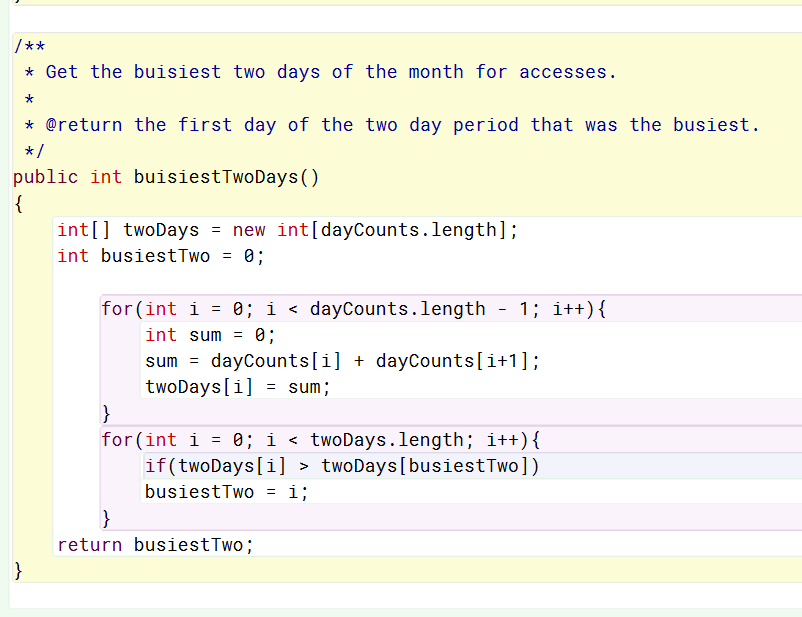
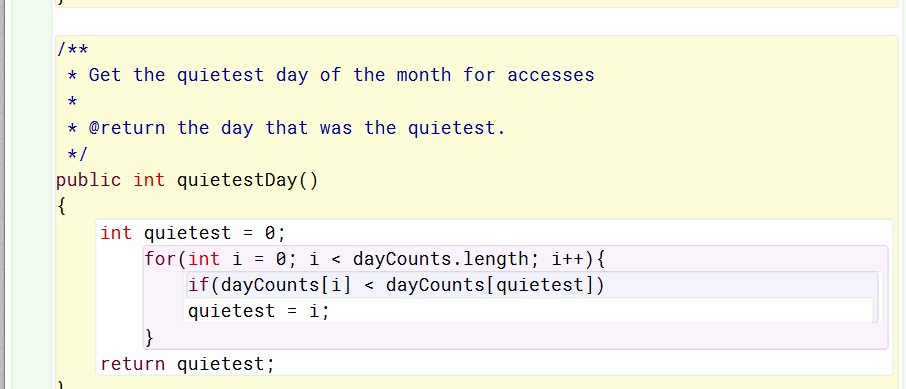
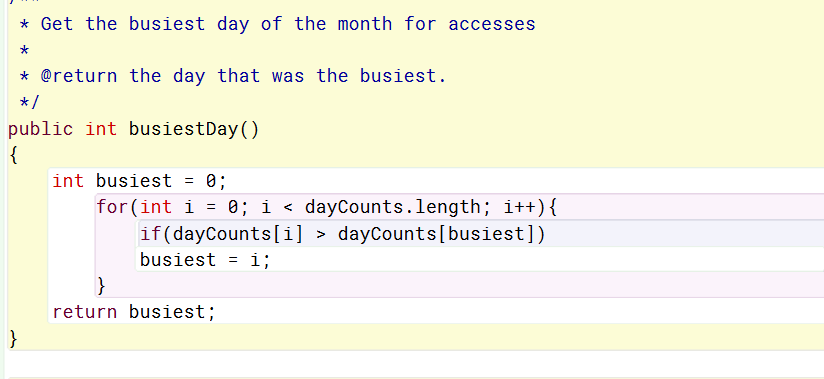
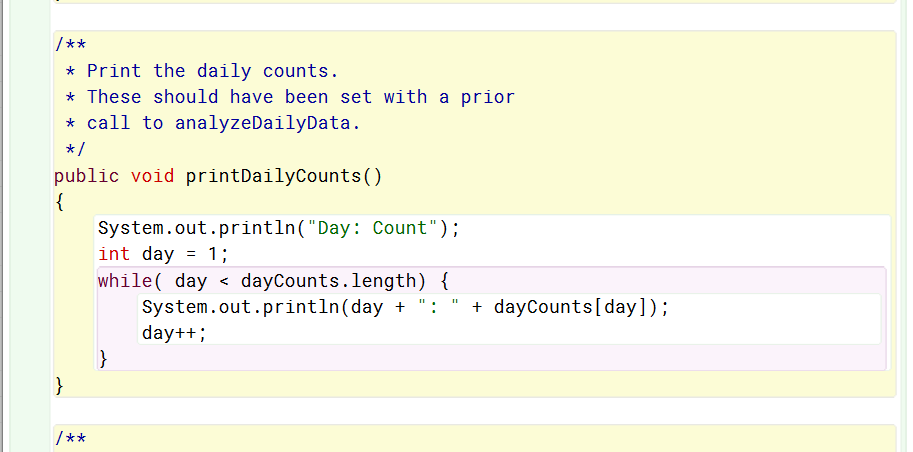
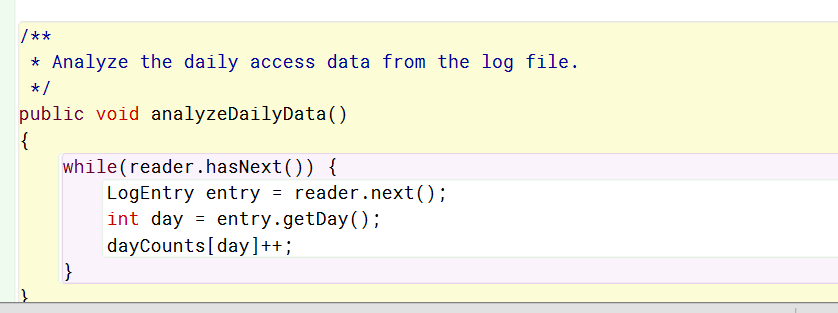
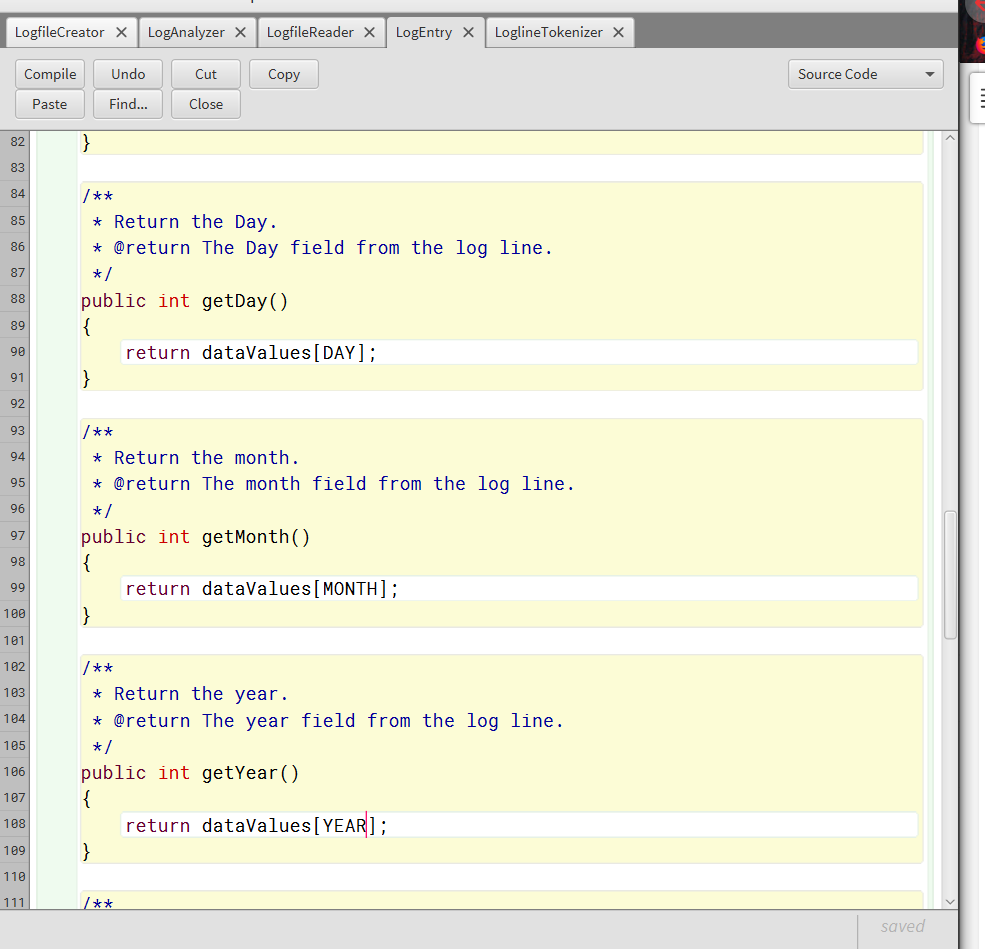
7.17

If more than one hour has the biggest count of the day, then my method will return the first biggest one to be found. Unless I changed a ‘>’ to a ‘>=’ then it would return the last one found that had the biggest count. I have been trying to think if it would also be helpful to return both in this instance, or all the hourCounts that are the biggest. This could be done using an ArrayList.

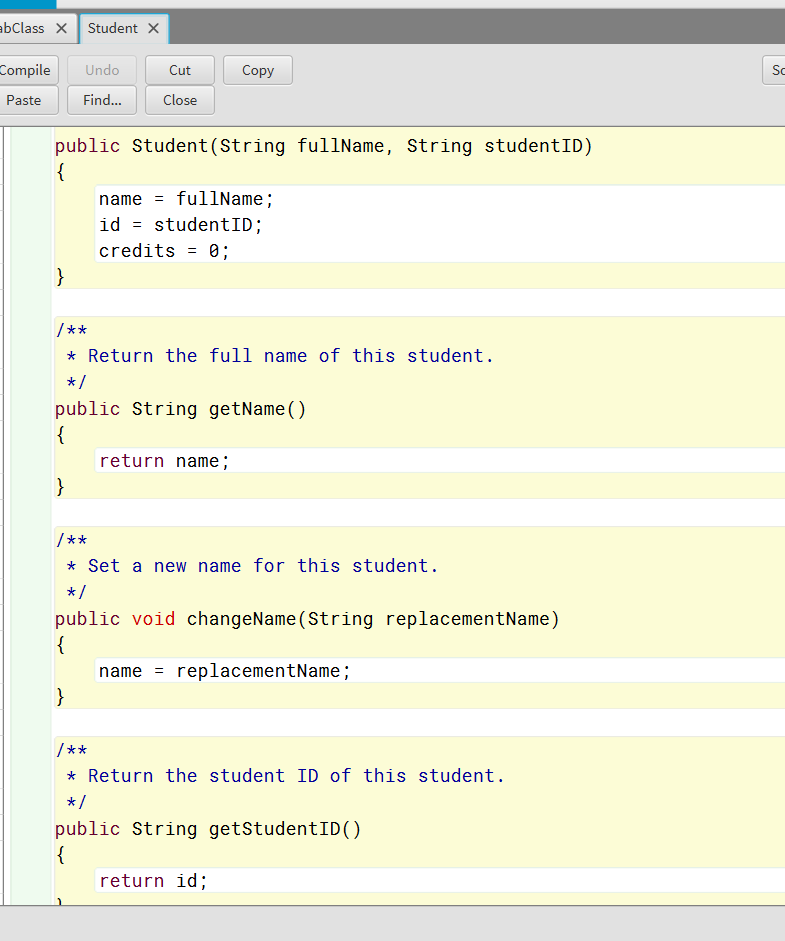
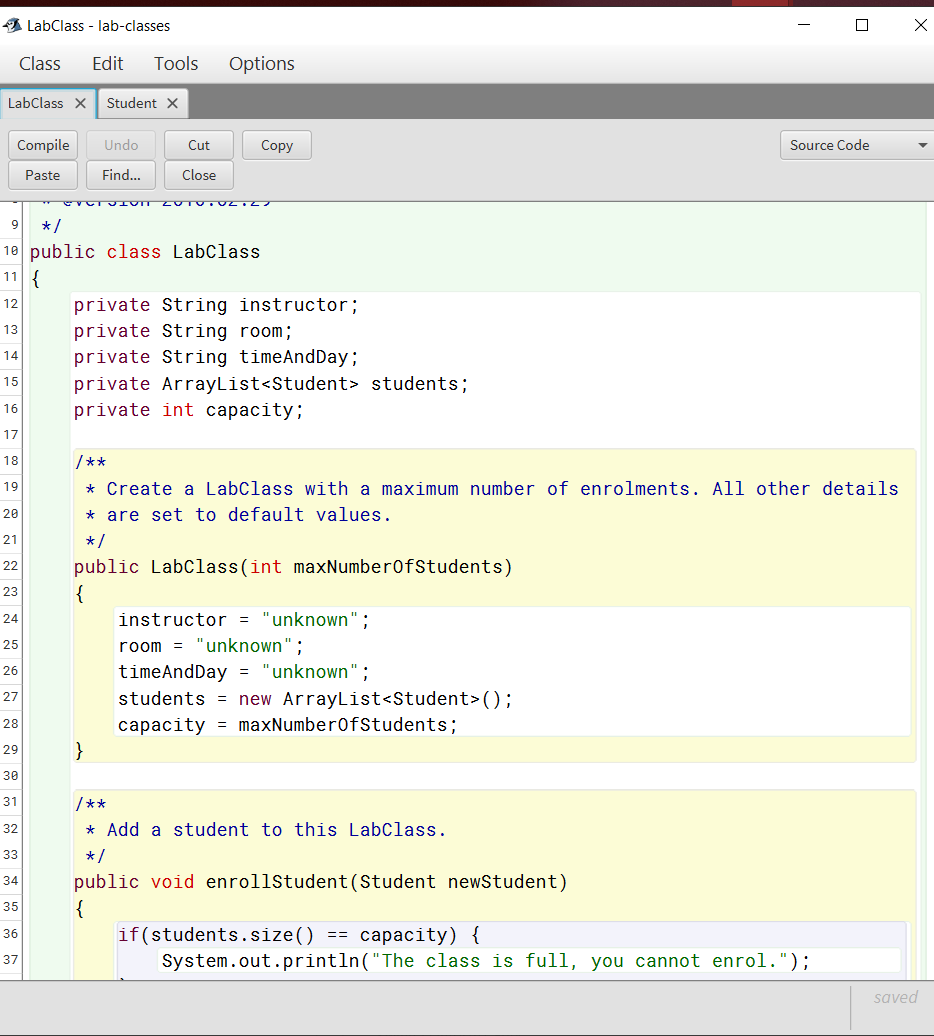
7.18



7.19



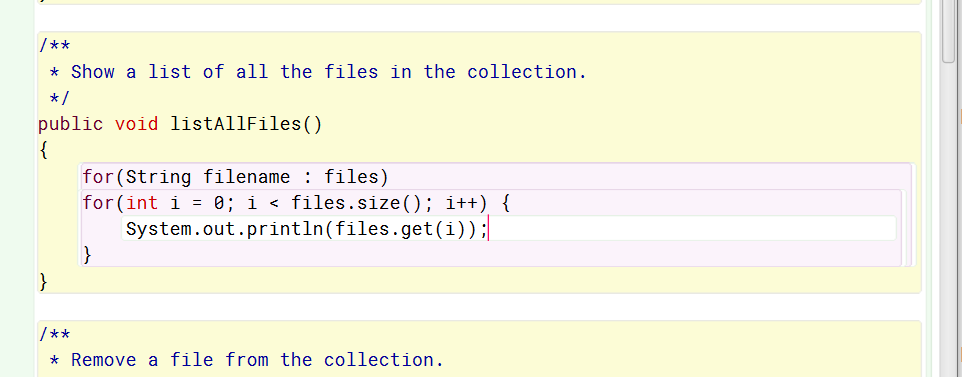
7.21



7.22

I could see using either a flexible array list, like we did use for this, or even a fixed size array. Since the capacity does determine the size of the list that we need we could have used a fixed size array instead. However, since we may not always fill all the spots it makes sense to use one that can shrink to fit the actual size of the number of students attending. So, there are not empty spots left at the end.

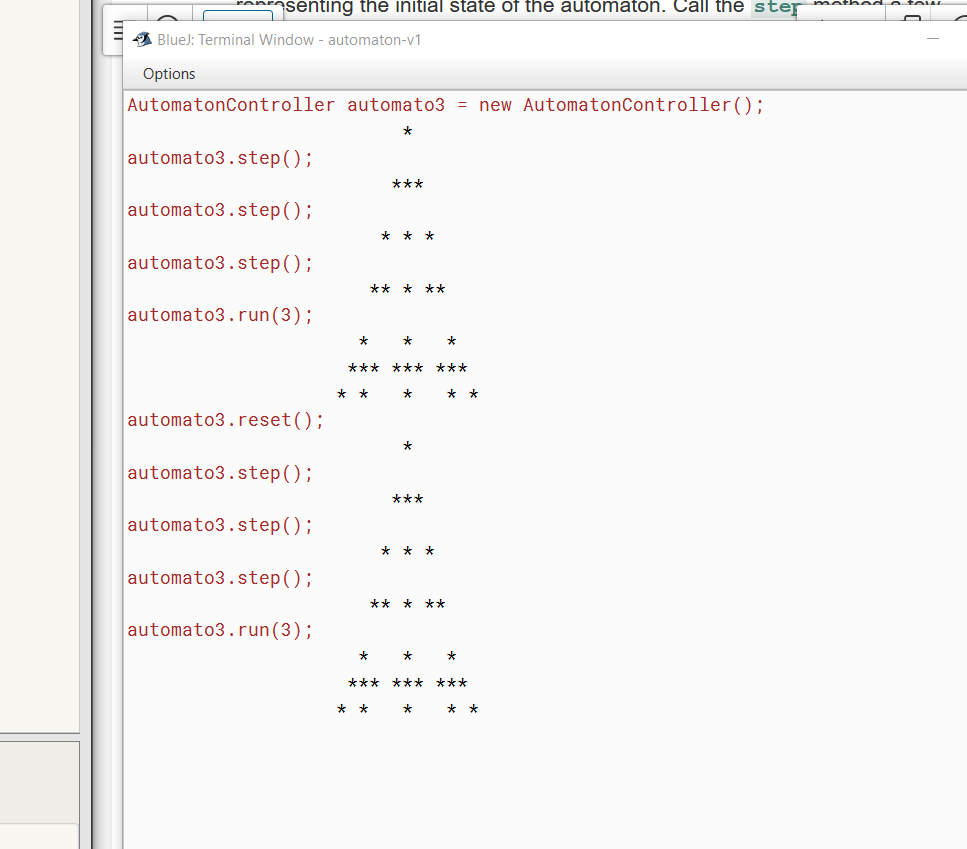
7.23



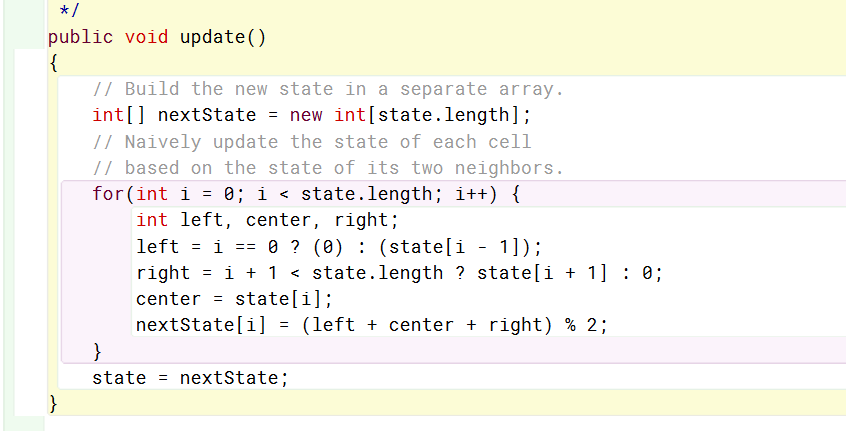
7.24



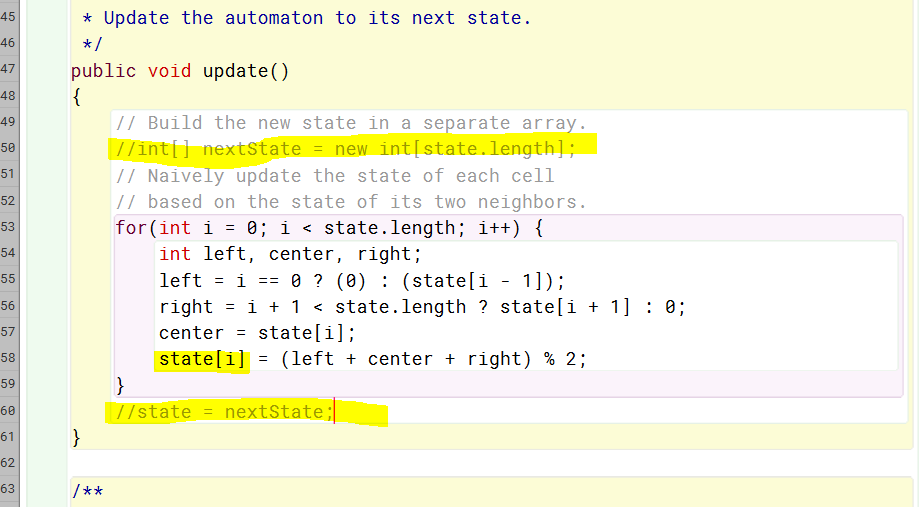
7.25

Yes,

7.28

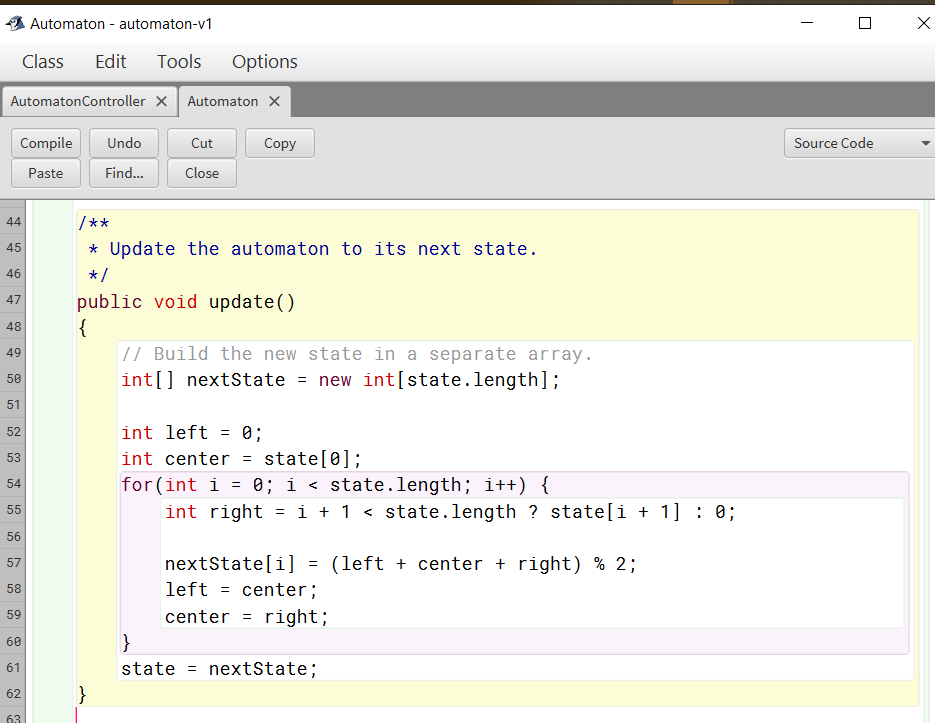


7.29

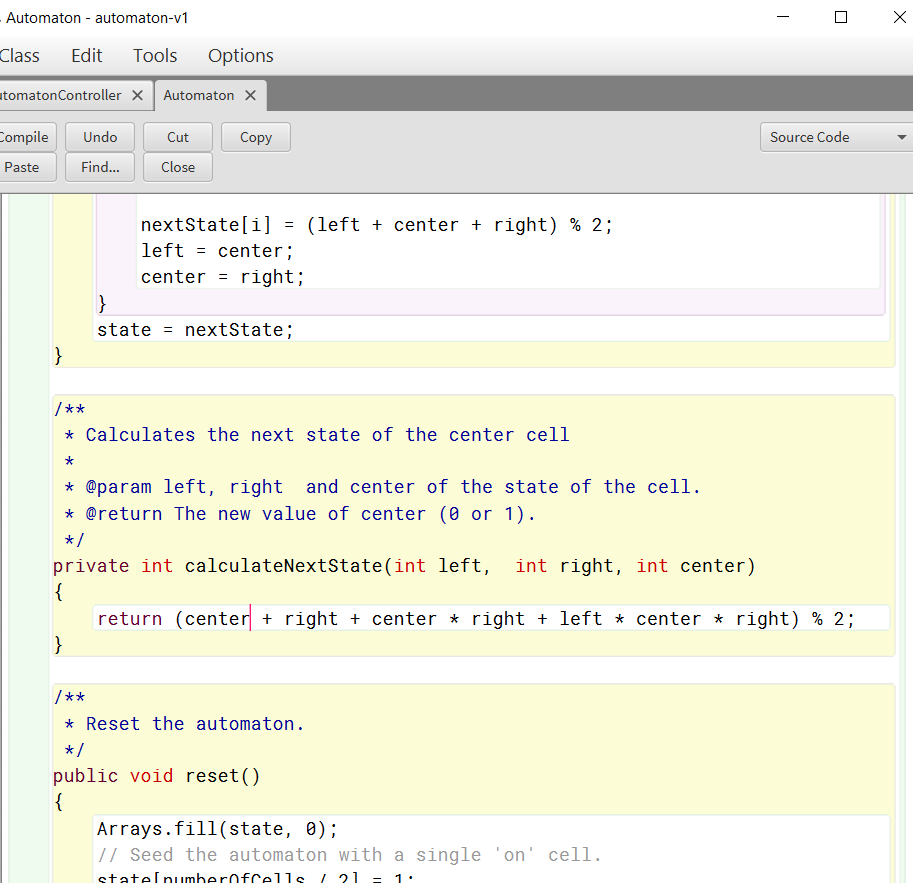


As you can see above yes, the behavior of the Automator is differed when we access the state array directly in this loop body of the update method. I think this is because the state of the left neighbor is changed before we get to I = 1 I = 0 has been changed already.

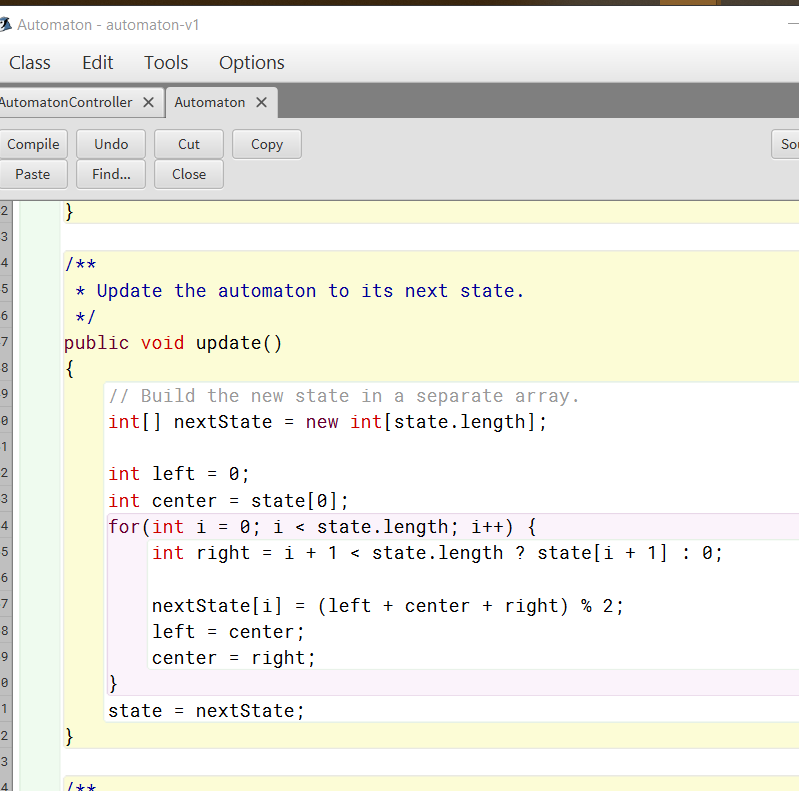
7.31



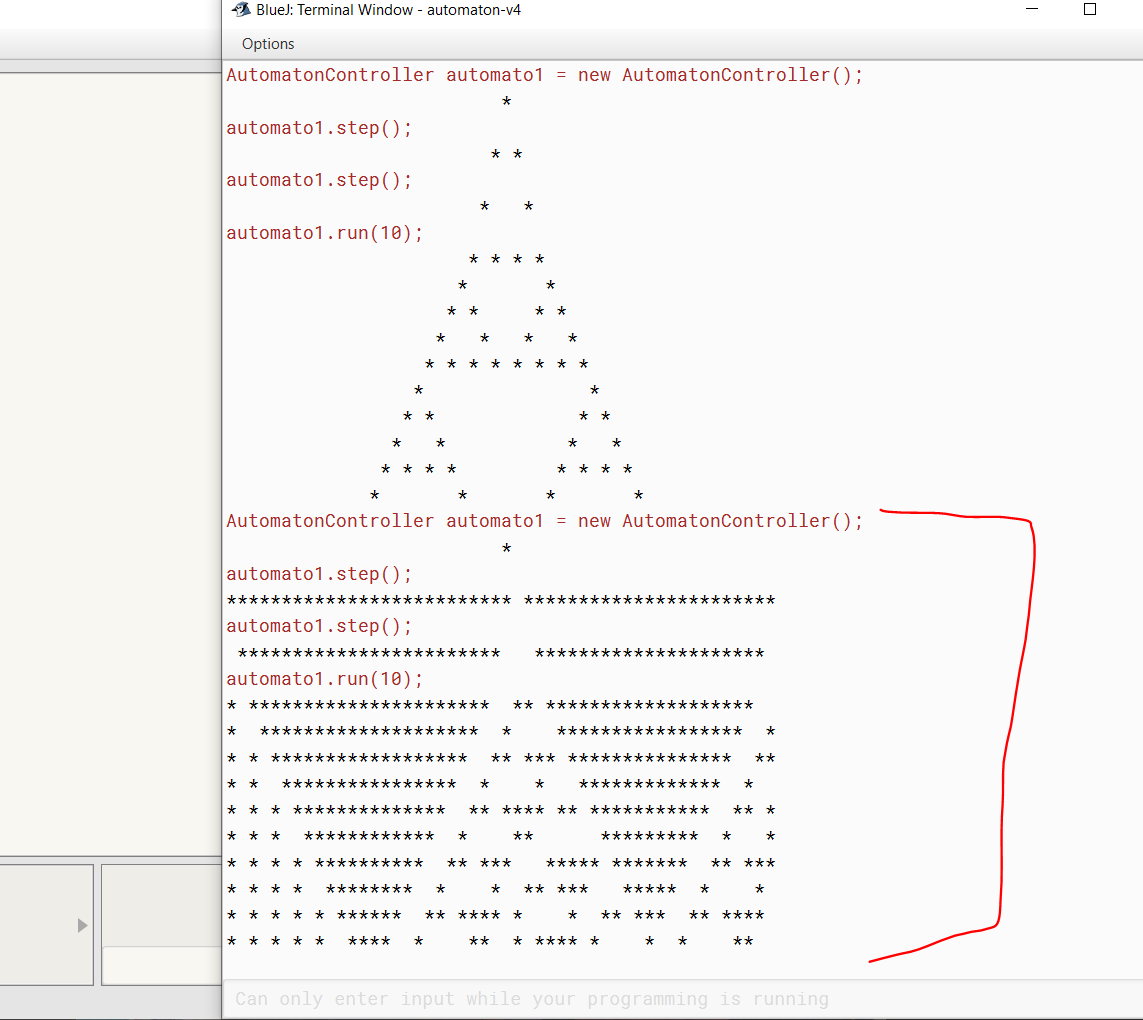
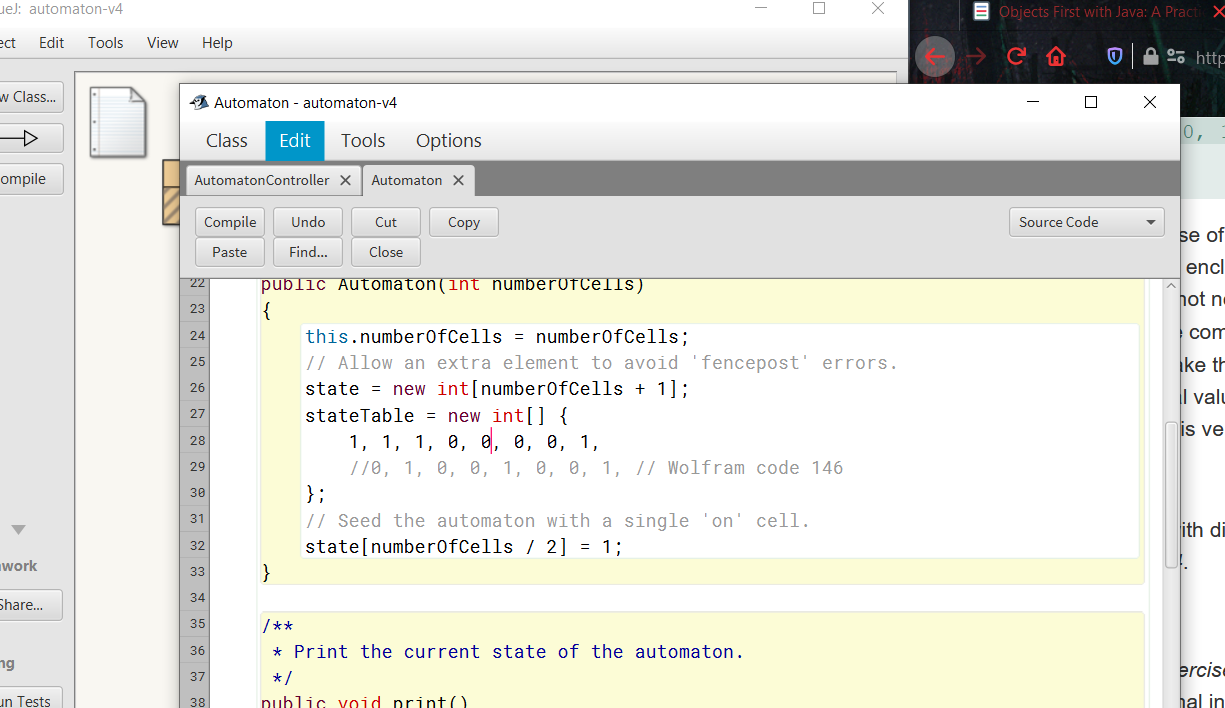
7.32/33

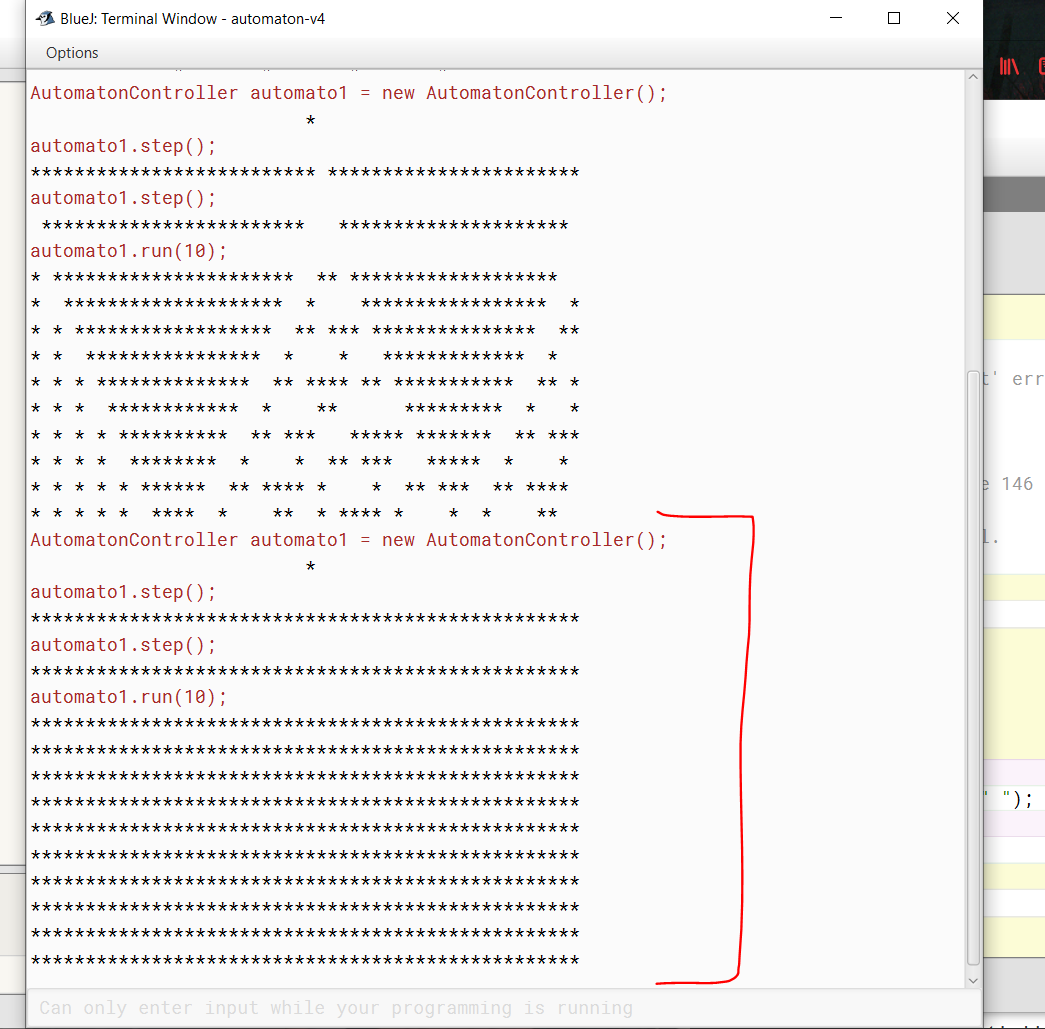
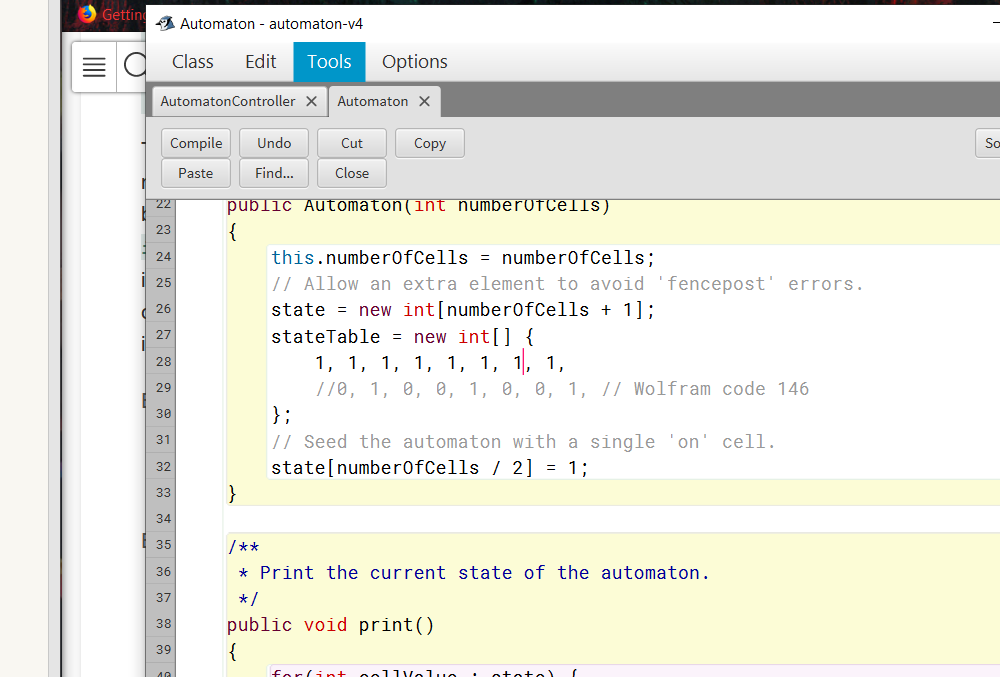


7.34



7.35





7.42

AsList, binarySearch, fill and sort are all methods of the java.util.Arrays class. The asList method returns a fixed size list from the array specified in its parameters. BinarySearch will search a specified array for a specified value. The array must be sorted first. This is where the sort method comes in. With the sort method one can sort a specified array of objects according to the order induced by the specified comparator. The array and the comparator are given as parameters. That leaves fill which is a method that can be used to fill an array with a specified value. Parameters are the array and the value to be filled.

7.43

int[] arr = {13, 7, 6, 45, 21, 9, 101, 102};

Arrays.sort(arr);

int ar[] = {2, 2, 1, 8, 3, 2, 2, 4, 2};

Arrays.fill(ar, 8);

int arr[] = {10,20,30,40,50};

int key = 30;

int last=arr.length-1;

binarySearch(arr,0,last,key);

7.44

Int[][] original;

Int[][] copy;

for (int I : original){

copy[0][i] = original[0][i];

copy[1][i] = original[1][i];

}

7.45

