7.1) The busiest time of the day is hour 18.

7.2) Person[] people

7.3) boolean[] vacant

7.4)

```
private int[] hourCounts;
hourCounts = new int[24];
hourCounts[hour]++;
```

```
for(int hour = 0; hour < hourCounts.length; hour++) {
    System.out.println(hour + ": " + hourCounts[hour]);
}
```

The pair of brackets is always used with the variable, which indicates that it is an array.

7.5) The part where it says ‘ []int counts’ is incorrect, as the brackets have to go after the int.

Along with that, the second line incorrectly declares the array, as the 5000 cannot go within those set of brackets.

7.6)

double[] readings = new double[60];

String[] urls = new String[90];

TicketMachine[] machines = new TicketMachine[5];

7.7)

There are 20 new string objects created.

7.8)

The following array is double prices = new double(50);

The problem with this is that the ending uses parentheses instead of brackets.

7.9)

The minor change gives us the following error as shown:

```
java.lang.ArrayIndexOutOfBoundsException:
Index 24 out of bounds for length 24
```

This means that the index 24 is out of bounds for the array.

7.10)

This is the equivalent of the for loop in the while format:

```
public void printHourlyCounts()
{
    System.out.println("Hr: Count");
    /*
    for(int hour = 0; hour < hourCounts.length; hour++) {
        System.out.println(hour + ": " + hourCounts[hour]);
    }
    */
    int hour = 0;
    while(hour < hourCounts.length){
        System.out.println(hour + ": " + hourCounts[hour]);
        hour++;
    }
}
```

7.11)

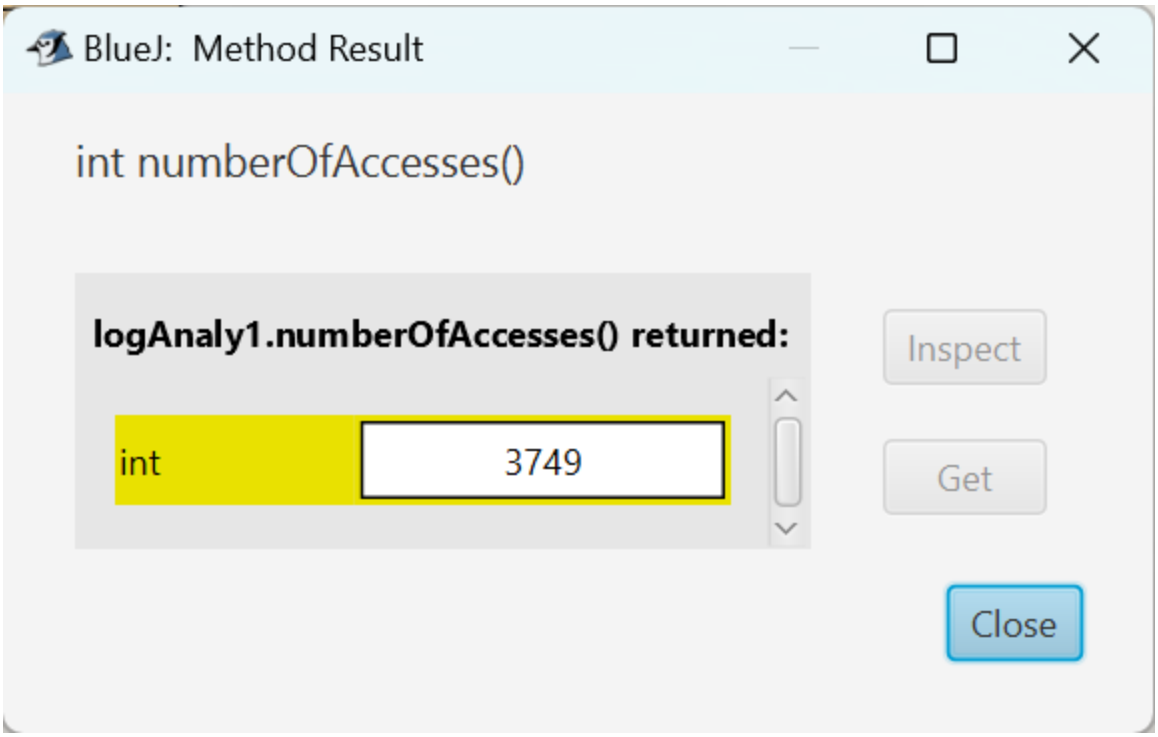
```
public void printGreater(double marks, double mean)
{
        for(int index = 0; index <= marks.length; index++){
                if(marks[index] > mean){
                        System.out.println(marks[index]);
}
}
}
```

7.12)

7.13)

```
public int numberOfAccesses(){
    int total = 0;
    for(int i = 0; i < hourCounts.length; i++){
        total = total + hourCounts[i];
    }
    return total;
}
```

7.14)



7.15)

```
public int busiestHour(){
    int busy = 0;
    for(int i = 0; i < hourCounts.length; i++){
        if(busy < hourCounts[i]){
            busy = hourCounts[i];
        }
    }
    return busy;
}
```

7.16)

```
public int quietestHour(){
    int quiet = hourCounts[0];
    for(int i = 0; i < hourCounts.length; i++){
        if(quiet > hourCounts[i]){
            quiet = hourCounts[i];
        }
    }
    return quiet;
}
```

7.17) The hour with the highest count is returned by the busiest hour, as it goes through the array and compares each one to each other to figure out which one is the largest member of the array.

7.18)

```
logAnaly1.printHourlyCounts();
Hr: Count
0: 149
1: 149
2: 148
3: 109
4: 92
5: 177
6: 185
7: 142
8: 91
9: 85
10: 227
11: 142
12: 114
13: 164
14: 227
15: 185
16: 167
17: 198
18: 237
19: 172
20: 142
21: 113
22: 168
23: 166
```

```
logAnaly1.twoHourBusiest()
    returned int 435
```

7.19)

```
public int getYear(){
        return dataValues[YEAR];
}
public int getMonth(){
        return dataValues[MONTH];
}
public int getDay(){
        return dataValues[DAY];
}
public void totalCount(){
```

```
        int total = 0;


        for(int = 0; i < hourCOunts.length; i++){
                total += hourCounts[i];
}
        System.out.println("Count for whole day is: " + total);
}
```

7.20) n/a

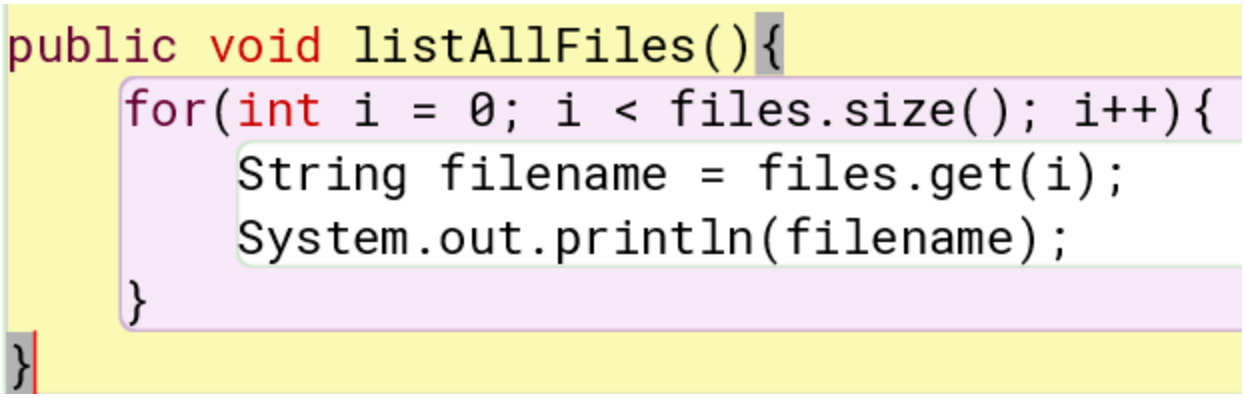
7.21)

Read through LabClass to reinforce concepts.

7.22)

Although it would be better to have a flexible sized array, so that you can input as many students as you want per class, it is not possible in java to have an array which changes it's size. Therefore, the more reasonable option is a fixed-length array.

7.23)



```
public void listAllFiles(){
    for(int i = 0; i < files.size(); i++){
        String filename = files.get(i);
        System.out.println(filename);
    }
}
```

7.24)

```
AutomatonController automato1 = new AutomatonController();
                                *
automato1.run(2);
                               ***
                              * * *
```

7.25)

```
AutomatonController automato1 = new AutomatonController();
                                *
automato1.run(2);
                               ***
                              * * *
automato1.reset();
                                *
automato1.run(3);
                               ***
                              * * *
```

Yes, the same steps show up.

7.26)

In java there are two versions of the fill method which take the parameter type int[]. The purpose of the fill methods is to initialize or reset elements in an array. In the reset method of the class, the fill(int[] a, int val) method is used.

7.27)

If it is dependent on the number that user inputs, then there would be star at first over whatever number inputted.

7.28)

```
public void update()
{
    // Build the new state in a separate array.
    int[] nextState = new int[state.length];
    // Naively update the state of each cell
    // based on the state of its two neighbors.
    for(int i = 0; i < state.length; i++) {
        int center;
        int left = (i==0) ? 0 : state[i-1];
        int right = (i+1 < state.length) ? state[i+1] : 0;
        center = state[i];
        nextState[i] = (left + center + right) % 2;
    }
    state = nextState;
}
```

7.29)

We may have made a different array so that any change we make doesn't affect the original array, but rather the secondary array.

7.30) n/a

7.31)

```java
public void update(){
    // Build the new state in a spearate array.
    int[] nextState = new int[state.length];
    // Updates state of each cell
    int left = 0;
    int center = state[0];
    for(int i = 0; i < state.length; i++){
        int right = i + 1 < state.length ? state[i+1] : 0;
        nextState[i] = (left + center + right) % 2;
        left = center;
        center = right;
    }
    state = nextState;
}
```

7.32)

```
public int calculateNextState(int left, int center, int right){
        return ((left + center + right) % 2);
}
```

```
nextState[i] = calculateNextState(left, center, right);
```

7.33)

Examples -> (right * left * center) % 2;

((right * left * center) * (right * left * center)) % 2;

There are an undefined amount of possibilities since you can go and add as many of those conditions as you want, which is unlimited.

7.34)

```
public void update(){
    // Build the new state in a spearate array.
    int[] nextState = new int[state.length];
    // Updates state of each cell
    int left = 0;
    int center = state[0];
    for(int i = 0; i < state.length; i++){
        int right = state[i + 1];
        nextState[i] = calculateNextState(left,center,right);
        left = center;
        center = right;
    }
    state = nextState;
}
```

7.35)

Experimented with different initialization pattens of the lookup table in automaton-v4.

7.36)

```
public Automaton(int numberOfCells, int wolframCode)
{
    this.numberOfCells = numberOfCells;
    // Allow an extra element to avoid 'fencepost' errors.
    state = new int[numberOfCells + 1];
    stateTable = new int[8];
    for(int i = 0; i < 8; i++){
        stateTable[i] = (wolframCode >> i) & 1;
    }
    // Seed the automaton with a single 'on' cell.
    state[numberOfCells / 2] = 1;
}
```