Challenging new Java candidates

New candidates applying for the Java engineer position in Forescout are required to work on a small assignment consisting of three exercises. The candidate is required to provide a written analysis of the chosen solution and the code implementation where the exercise requires it. Applicants may choose to complete the non-mandatory bonus challenges to add value to their submission.

1 Implement a malware spreading model

Malware can spread quickly in a network. It often takes only one inattentive employee clicking on a link or one device not patched with the latest updates.

In this challenge, we will model the rate at which a theoretical malware spreads in a network. For this challenge we will assume that there is no limit to the number of devices in the network.

Of course, every malware is different, so we will make some basic assumptions and simplifications. Firstly, let's say that every device on the network infected by malware spreads to a new uninfected device once every 7 minutes. After this, it needs to find a new device to infect (which takes another 7 minutes).

Of course, this process isn't synchronized. So, one device might have been infected 1 minute ago, while another is already scanning for 5 minutes. So, you can model each infected device as a single number that represents the number of minutes until it infects a new device.

Furthermore, a newly infected device requires some setup and installation time before it's ready to infect other devices: two more minutes for its first cycle only.

So, suppose you have a device with an internal timer value of 3:

- After one minute, its internal timer would become 2.
- After another minute, its internal timer would become 1.
- After another minute, its internal timer would become 0.
- After another minute, its internal timer would reset to 6, and it would infect a new device with an internal timer of 8.
- After another minute, the first device would have an internal timer of 5, and the second device would have an internal timer of 7.

An infected device that infects a new device resets its timer to 6, not 7 (because 0 is included as a valid timer value). The new device starts with an internal timer of 8 and does not start counting down until the next minute.

With this model, we can determine how quickly things will go bad from any starting point by observing the network for a few minutes. We compiled a list of several hundred infected devices (your challenge input).

For example, suppose you were given the following list:

```
5,1,2,4,1
```

This list means that the first infected device has an internal timer of 5, the second infected device has an internal timer of 1, and so on until the fifth infected device, which also has an internal timer of 1. Simulating these devices over several minutes would proceed as follows:

```
Initial state:    5,1,2,4,1
After  1 minute:  4,0,1,3,0
After  2 minutes: 3,6,0,2,6,8,8
After  3 minutes: 2,5,6,1,5,7,7,8
After  4 minutes: 1,4,5,0,4,6,6,7
After  5 minutes: 0,3,4,6,3,5,5,6,8
After  6 minutes: 6,2,3,5,2,4,4,5,7,8
After  7 minutes: 5,1,2,4,1,3,3,4,6,7
After  8 minutes: 4,0,1,3,0,2,2,3,5,6
After  9 minutes: 3,6,0,2,6,1,1,2,4,5,8,8
After 10 minutes: 2,5,6,1,5,0,0,1,3,4,7,7,8
After 11 minutes: 1,4,5,0,4,6,6,0,2,3,6,6,7,8,8
After 12 minutes: 0,3,4,6,3,5,5,6,1,2,5,5,6,7,7,8,8
After 13 minutes: 6,2,3,5,2,4,4,5,0,1,4,4,5,6,6,7,7,8
After 14 minutes: 5,1,2,4,1,3,3,4,6,0,3,3,4,5,5,6,6,7,8
After 15 minutes: 4,0,1,3,0,2,2,3,5,6,2,2,3,4,4,5,5,6,7,8
After 16 minutes: 3,6,0,2,6,1,1,2,4,5,1,1,2,3,3,4,4,5,6,7,8,8
After 17 minutes: 2,5,6,1,5,0,0,1,3,4,0,0,1,2,2,3,3,4,5,6,7,7,8
After 18 minutes: 1,4,5,0,4,6,6,0,2,3,6,6,0,1,1,2,2,3,4,5,6,6,7,8,8,8,8
```

Each minute, a 0 becomes a 6 and adds a new 8 to the end of the list, while each other number decreases by 1 if it was present at the start of the minute.
In this example, after 18 minutes, there are a total of 27 infected devices. After 80 minutes, there would be a total of 5977.

Write an application to simulate this malware spreading and that answers the following questions:

1) How many infected devices would there be after 80 minutes?
2) How many devices are infected after 256 minutes?

We expect your application to calculate the answer to the above within a second. In addition, the code should either be annotated with comments explaining the approach or accompanied by a separate document containing those explanations.

Your challenge input:

```
1,1,3,1,3,2,1,3,1,1,3,1,1,2,1,3,1,1,3,5,1,1,1,3,1,2,1,1,1,1,4,4,1,2,1,2,1,1,1,5,3,
2,1,5,2,5,3,3,2,2,5,4,1,1,4,4,1,1,1,1,1,1,5,1,2,4,3,2,2,2,2,1,4,1,1,5,1,3,4,4,1,1,
3,3,5,5,3,1,3,3,3,1,4,2,2,1,3,4,1,4,3,3,2,3,1,1,1,5,3,1,4,2,2,3,1,3,1,2,3,3,1,4,2,
2,4,1,3,1,1,1,1,1,2,1,3,3,1,2,1,1,3,4,1,1,1,1,5,1,1,5,1,1,1,4,1,5,3,1,1,3,2,1,1,3,
1,1,1,5,4,3,3,5,1,3,4,3,3,1,4,4,1,2,1,1,2,1,1,1,2,1,1,1,1,1,5,1,1,2,1,5,2,1,1,2,3,
2,3,1,3,1,1,1,5,1,1,2,1,1,1,1,3,4,5,3,1,4,1,1,4,1,4,1,1,1,4,5,1,1,1,4,1,3,2,2,1,1,
2,3,1,4,3,5,1,5,1,1,4,5,5,1,1,3,3,1,1,1,1,5,5,3,3,2,4,1,1,1,1,1,5,1,1,2,5,5,4,2,4,
4,1,1,3,3,1,5,1,1,1,1,1
```

2 Extend an existing networking representation model

The *Networking* project in the attached package implements a very simplified model of an IPv4 network. When IPv6 later became available, that model could not hold IPv6 based hosts anymore. The model shall be extended to be able to manage IPv6 based hosts, with the following requirements.

- Propose a solution that will extend the existing functionalities to *Inet6Address* based hosts.
- Backward compatibility shall be preserved. Existing client code using this library must not break and it will keep working without changes. Tip: write some tests that proves it.
- The new model shall be able to represent three different classes each holding IPv4 only, IPv6 only or mixed IPv4/IPv6 based host networks. Code reuse is a must.
- Extend the test case to cover the new functionalities.
- Propose an implementation of *Network* that improves (lowers) the time complexity without changing the class contract.
- One of the classes in the project has a bug. Explain what this is about and provide a test case that proves it and how to fix it.
- Bonus challenge: explain why the *Network* class is not thread-safe, and how to make it so.

3 Design the solution for vulnerability forwarder

A java application is processing information triggering some vulnerabilities update. After an update, the vulnerabilities are locally persisted in a relational database. The problem is maintaining a copy of the vulnerabilities in a remote service.

- Each vulnerability has a unique ID.
- Some vulnerabilities are being updated at various frequencies.
- A vulnerability should not be sent if the remote service already has the latest version of it.
- The remote service cannot be queried to know which vulnerabilities are there and with which version.

Propose a solution that solves this problem. Write down any assumptions and gives some pros & cons of the solution.