State Machine Exercise

Background

State machines are a very useful and simple computational model for processing input streams.

Since much of their functionality is common to many different instances of state machines, it is more efficient to have a common framework for creating different state machines.

In this exercise you will be expected to create such a framework, and then to use it to solve a specific problem.

Exercise

We will now detail the different parts of the exercise and the requirements for each one.
In all parts, feel free to choose the programming language you prefer.
It's best if you use either Java or Python, but we can accommodate other languages as well.

There are 3 parts to the exercise:

- Framework: where we design and implement the framework
- Toy Problem: in which we use the framework to solve a simple use-case
- Persistence: where we design and implement a solution for saving and loading a State Machine from disk

Framework

Design and code a framework for implementing state machines.

During the design phase, try and think about the following considerations:

- Power: can the framework support many use-cases? How flexible is it?
- Usefulness: will the framework save time when programming?
- Simplicity: how simple is the framework itself? How simple is it to use it?
- Abstraction: will the framework help separate concerns in a way that makes the entire code clearer?

Tip: feel free to read ahead to make sure the framework can (easily) solve the requirements of the toy problem

Toy Problem

Now that we have a framework, let's use it to solve a simple problem.

Setting

Let's say we work for a large candy manufacturer. In one production line they create lollipops in one of 2 flavors:

- Strawberry
- Lemon

The production line is supposed to alternately output one Strawberry-flavored lollipop, then one Lemon-flavored one, then another one Strawberry-flavored lollipop, and so on and so forth.
It should look something like this:

Now, in some cases, due to some fluke there can be a case where two lollipops of the same flavor appear one after the other, and that's fine.

However, if **three** lollipops of the same flavor appear in a row, that likely means there is some serious problem, and we need to call an Engineer to review the machines.

Your Task

Build a State Machine which will:

- Take some form of input
    - Input source is entirely your choice: it can be from the keyboard, from file, or something else; it doesn't matter or affect the exercise
    - The input should be some representation of the series of flavors: whether you use strings or numbers or something else is immaterial
- Detect when the Production line output **three** consecutive lollipops of the **same flavor**
    - It should do so **every time** this happens
    - However, once it recognized an error in the production line, it should not treat additional consecutive output of the same flavor as another error
        - For instance, in the following sequence it will only detect **two** (2) errors

    - The first error is after the 3rd consecutive Strawberry lollipop (5th in the sequence)
    - The second error is after the 3rd consecutive Lemon lollipop (10th in the sequence)
- Output an appropriate error message to the console (or any other output format of your choice) when each error is detected

Persistence

Some State Machines are expected to run for prolonged periods of time - potentially even indefinitely.

However, it is common that the system needs to be shut down for maintenance or, say, crashes due to an internal error. This means that we need some way to store the current state of the system on disk and then to restore the system to a previously-saved state during system load.

Your task is to implement this feature and demonstrate its use on the toy problem above.

Guidelines

- Please send the exercise back to omeshulam@ebay.com and rwilson3@ebay.com at least 24 hours before the technical interview
- Come to the interview ready to discuss the different design decisions you've made and how they might have changed if you made different assumptions about the problem