Saayeh Siahmakoun

Engineering Notebook

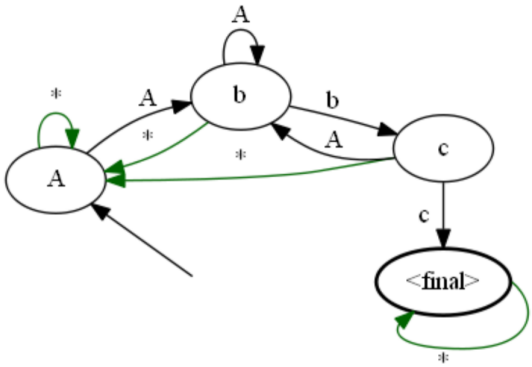
DAY 1: 10-01-2022

Problems addressed

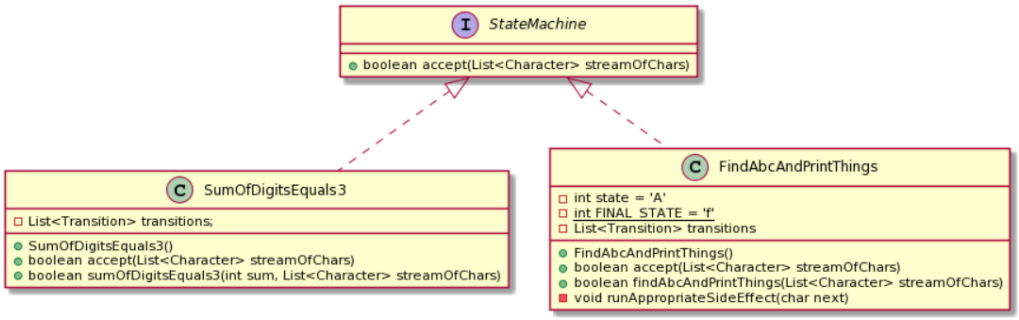
1. What is a *finite-state machine?*
2. What is the scope of this homework? (look at given source code)

Possible solutions explored

1. *Finite-State Machine*
    a. A mathematical model of computation that can be in EXACTLY one of a finite number of states at ANY given time
    b. Can change from one state to another in response to some inputs (*transition*: moving from one state to another)
    c. Defined by
        i. A list of states
        ii. An initial state
        iii. The inputs that trigger each transition



    d.
2. *Scope of the Homework (what code is given)*
    a. Object-oriented library for building & executing state machines
    b. **SumOfDigitsEquals3**: implements the state machine on the first page as-is
    c. **FindAbcAndPrintThings**: implements the above state machine. It also logs each state transition to a log file



    d.
    e. Each transition will have *at most* one side effect; different state transitions may have completely different side effects from each other

f. Goal: to assign side effects to the state of transitions existing state machines @ runtime w/o editing any existing code (flexibility)
g. Not allowed to use **extends** keyword (no inheritance)
h. Only *interfaces* and *composition*

Solution decided on

1. *Finite State Machines*: Key Concepts
    a. Final State = the last *state* that an input should reach
    b. Accepted Input = if the machine reaches the end of the input while **inside** a *final state*
    c. Rejected Input = if the machine reaches the end of the input while **outside** of a *final state*
2. *Scope of the Homework*: What to Focus On
    a. Modify design of classes to…
        i. **Favor composition over inheritance**
        ii. **Ensure flexibility** when assigning side effects to the state of transitions @ runtime
    b. Write an object-oriented library for building and executing state machines that can be used to *easily* construct new state machines without changing the library's code

---

Engineering Notebook

DAY 2: 11-01-2022

Problems addressed

1. What do libraries in Java generally look like? What are some key components?
    a. *Understanding libraries in general*

Possible solutions explored

1. Questions to ask when designing a library
    a. What problem will the library solve?
    b. What details do the users care about?
    c. What details would the users rather forget about?
        i. What elements and operations do users want other parts of the program to handle?
2. At some point, a user is going to have a problem and will need to know exactly what is going on inside the library
3. Give a set of classes its own namespace to avoid overlapping-name conflicts in the future
4. Inheritance provides a way to *customize* code without having to know all the details of the code you are customizing

Solution decided on

1. The library is trying to solve the client's desire to have a library that can be used by basic finite state machines without having to regenerate a bunch of code that might get lost and confusing along the way
2. The details the users care about…
    a. The input
    b. The final state (?)
3. The details the users would rather forget about…
    a. Implementation details of constructing a novel finite state machine
    b. The patterns followed by a given finite state machine
    c. The transition states & their side effects
4. Create clear error messages so that the user does not have to go into deep debug mode through the library
5. Create package and ensure all class files are nested within the package namespace
6. "... make sure it's clear what the user needs to provide to the library in order for it to do its work" https://www.baeldung.com/design-a-user-friendly-java-library

---

Engineering Notebook

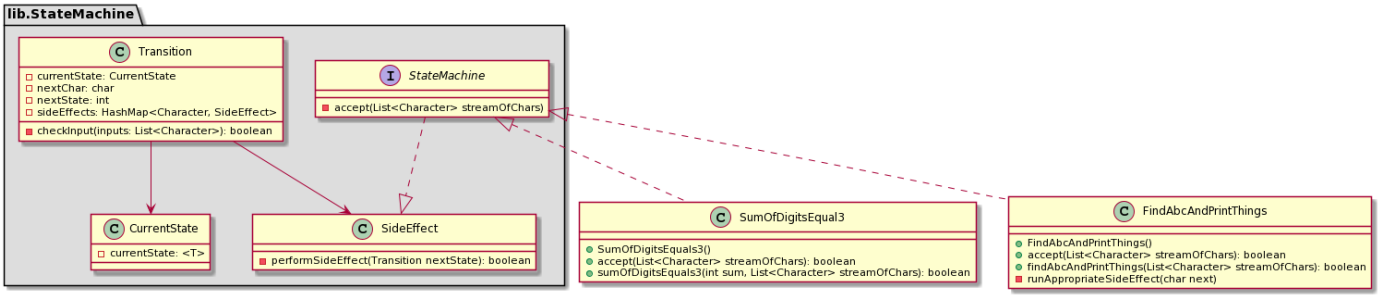
DAY 3: 13-01-2022

Problems addressed

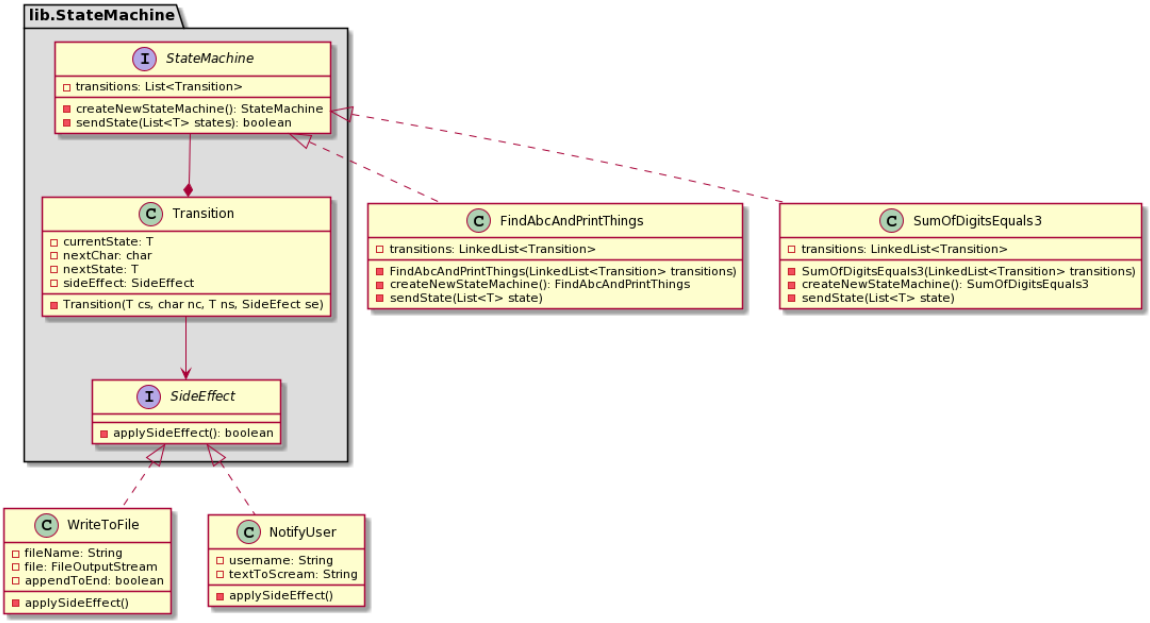
1. What does each *State Machine* need to provide to the library for it to be built properly (what is unique to each *State Machine*?)
2. Redesign UML diagram: **Strategy Pattern!**
3. Begin coding to find *holes* in proposed design

Possible solutions explored

1. Provided information from *State Machine* to *Library*
    a. List of transition states
    b. List of side effects at each transition state
    c. Final state
2. UML Diagrams
    a. #1



    b. #2



Solution decided on

Engineering Notebook

1. Delegate *SideEffect* to interface
2. Design choose: UML #2