L13 - Finite State Machines

Introduction – Making Change

Finite State Machines(FSMs) are a method to organize a system of inputs and outputs into a program that is easy to write, understand, and extend. For example, let's imagine you are writing code for a vending machine to output the correct amount of coins to dispense given the amount of change needed to dispense.

By this point in the programming labs, you should be pretty familiar with Java. To put this state machine together, we are going to need to know how to use switch statements, and create enums.

The easiest way to create a finite state machine is to create A diagram. If you just start programming without planning first you are going to rapidly create bugs in the system.

Let's try to describe the environment and goals that we need to achieve with our program.

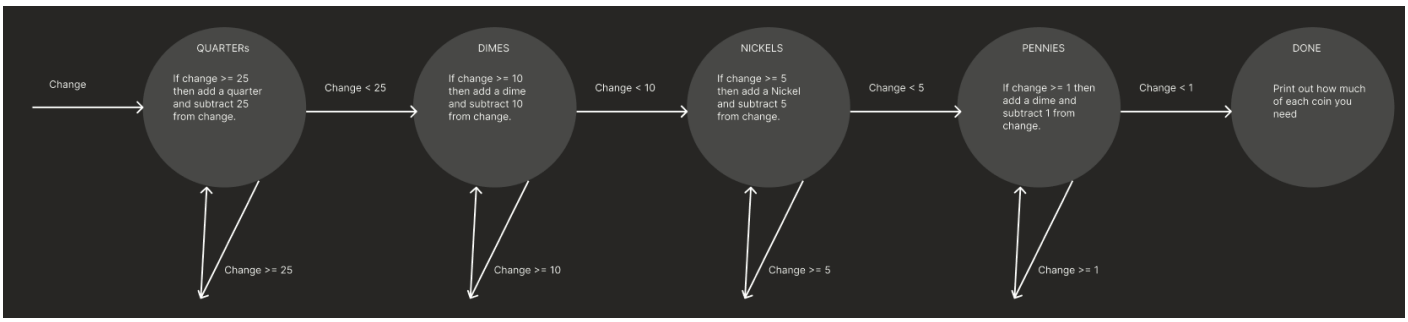
Inputs

- The amount of change we need to dispense which is an Integer between 0 and 99.

Output

- The number of Quarters, Dimes, Nickels, and Pennies that the machine needs to dispense.

So far writing this program is business as usual, next we are going to use a diagram to show us what states and transitions the FSM needs. These are essentially the way that the program will use the inputs to figure out the outputs.



This diagram represents everything you need to know about the state machine to implement it. A state machine is made us of several states that represent what the program is doing at a given moment. In this program, we have 5 possible states represented by the large circles. We have

Quarters, Dimes, Nickels, Pennies, and Done. You don't always need a done state, but here it makes it convenient to print out at the end. The arrows represent the flow of the program. We start with the QUARTERS state. Inside that circle, you can see what we need to do while in the QUARTERS state. We stay in the QUARTERS state as long as the remaining change we need to make is above 25.

Once the change is below 25 we Transition to the Dimes state, And we continue in this process until we are in the finished state.

That's enough background, its time for code.

PART A

First we need to set up the states. We represent the states as Enums to make it as readable as possible. Fill in the blanks below in a new Java Project

```
enum STATES {
    _____,
    ____,
    _____,
    _____
       }
```

Next we need to make the object that will store all the different kinds of change we are using.

```
class Change {
    int pennies;
    int nickels;
    int dimes;
    int quarters;

    public Change(){
        pennies = 0;
        nickels = 0;
        dimes = 0;
        quarters = 0;
    }

    public void addPenny(){
```

```
        pennies++;
    }

    public void addNickel(){
        nickels++;
    }
    public void addDime(){
        dimes++;
    }
    public void addQuarter(){
        quarters++;
    }

    @Override
    public String toString() {
        return "Change{" +
                "pennies=" + pennies +
                ", nickels=" + nickels +
                ", dimes=" + dimes +
                ", quarters=" + quarters +
                '}';
    }
}
```

that's the boilerplate out of the way and now we can work on the actual state machine.

```
public class Main {
    public static void main(String[] args) {


        makeChange(25);



    }


    public static Change makeChange(Integer changeToMake) {
        STATES state = STATES.QUARTERS;
        Change change = new Change();
        while (changeToMake > 0) {
            switch (state) {
                case ____:
```

```
            if (changeToMake >= ____) {
                change.____();
                changeToMake -= ____;
                state = STATES.____; // this isn't needed but it makes it clear we intent to stay in the state.
            }
            if (changeToMake < ____) {
                state = STATES.____;
            }

            break;
        case ____:
            if (changeToMake >= ____) {
                change.addDime();
                changeToMake -= ____;
                state = STATES.____; // this isn't needed but it makes it clear we intent to stay in the state.
            }
            if (changeToMake < 10) {
                state = STATES.____;
            }

            break;
        case ____:
            if (changeToMake >= ____) {
                change.____();
                changeToMake -= ____;
                state = STATES.____; // this isn't needed wbut it makes it clear we intent to stay in the state.
            }
            if (changeToMake < ____) {
                state = STATES.____;
            }
            break;
        case ____:
            if (changeToMake >= ____) {
                change.____();
                changeToMake -= ____;
                state = STATES.____; // this isn't needed but it makes it clear we intent to stay in the state.
            }

            break;
    }
```

```
        }

        System.out.println(change.toString());
        return change;
    }


}
```

After you've filled in those blanks with the proper states and values from the diagram you can work on extending the state machine

PART B

implement the same state machine but add single dollar bills to it, and make the diagram too.

Revision #1
Created 24 April 2025 22:14:34 by Brandon Duke
Updated 24 April 2025 22:16:19 by Brandon Duke