Java RMI Washing Machine

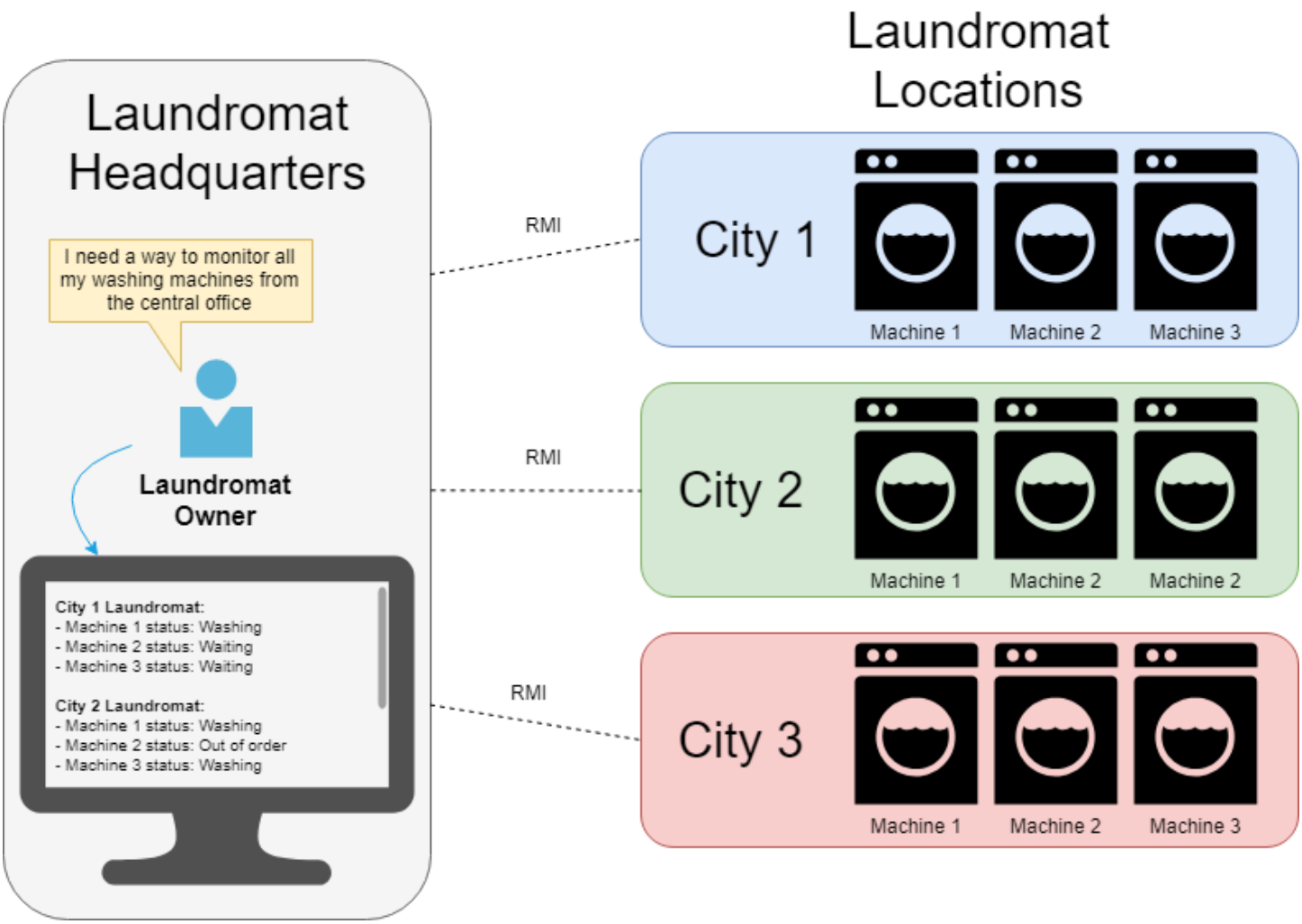
Skills Used

- Java
- Remote Method Invocation
- State Pattern
- Proxy Pattern
- Object Oriented Principles
- Open/Closed Principle

Description

Requirements

A fictional business owner has multiple laundromats and wants the ability to check the state of all washing machines from one central location. Using remote method invocation and the proxy pattern I was able to write a program for him that monitors all the machines he owns and displays each machine's current state.



State Pattern

The washing machine implemented the state pattern because it helps create code that is more easily maintained and closed for modification but open for extension. The washing machine can always have states added to it such as an "out of order state" that could easily be added. The only drawback to using this approach is that it can contain a lot of duplicate code between states which could be resolved by creating an abstract state class and using generic error responses for actions that are invalid.

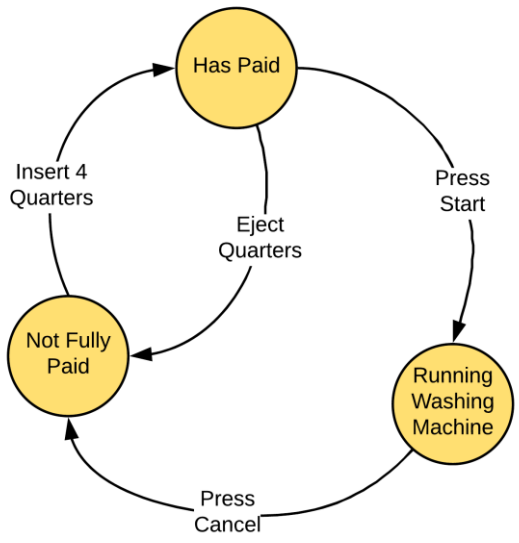
Proxy Pattern

Using a remote proxy pattern, we can monitor the washing machines that are in place at multiple locations. I gave the monitor a reference to a proxy version of a washing machine and the proxy pretends it's the real object but it's actually communicating over the network to the real object. I also added the ability for the washing machine to be able to accept requests over the network.

Remote Method Invocation

I implemented the remote monitoring with Java RMI which allowed me to invoke methods on a remote washing machine object. I make each remote washing machine available to the monitor by binding it to the registry and the monitoring program can then fetch the remote washing machine from the registry and invoke its methods.

State Diagram



Not Fully Paid:

- Eject quarters: eject any quarters currently in machine
- Insert quarter: add $0.25 to total
- Press start: do nothing
- Press cancel: do nothing

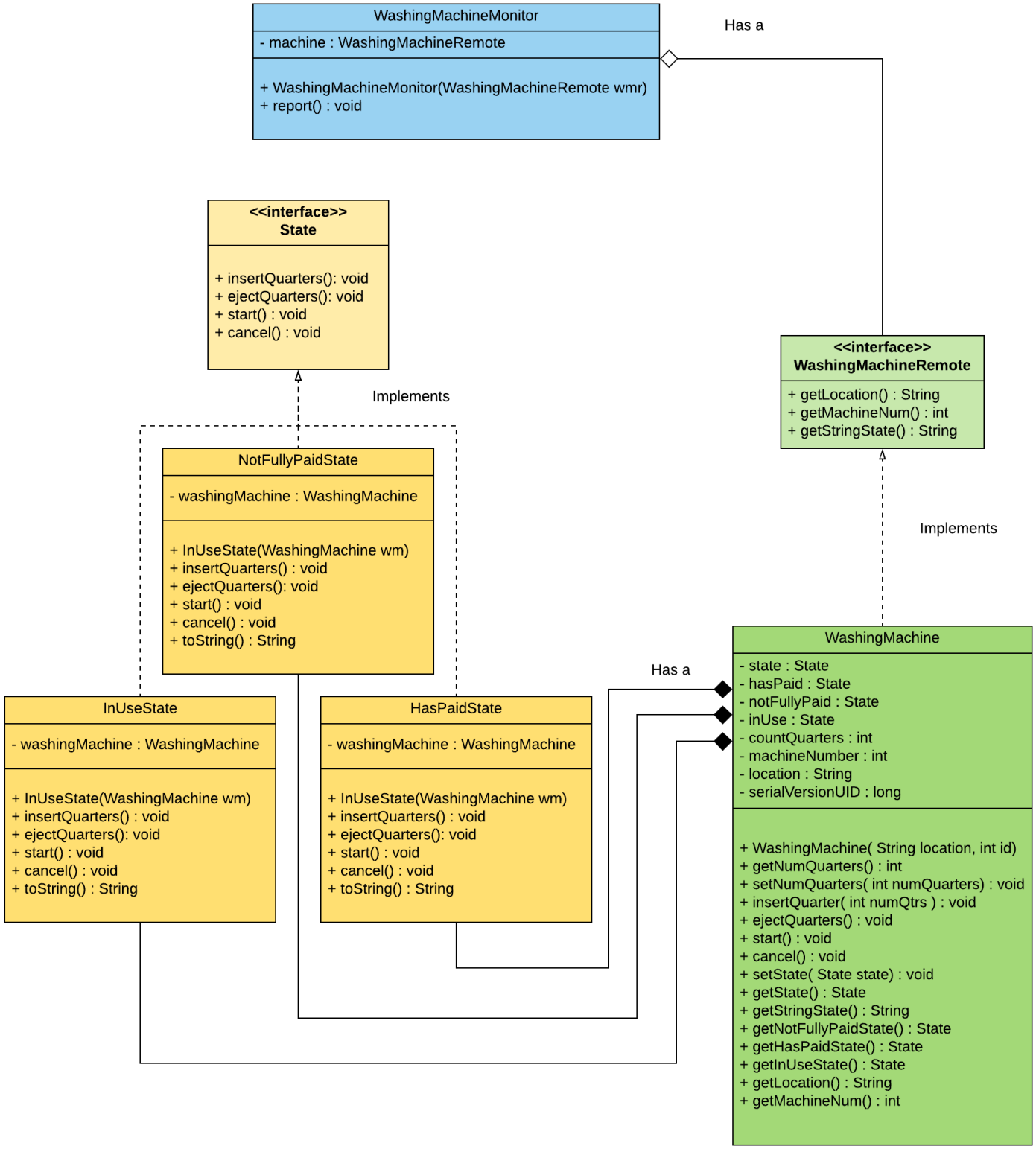
Has Paid:

- Eject quarters: eject four quarters
- Insert quarter: eject the inserted quarter
- Press Start: start the washing machine
- Press Cancel: do nothing

Running:

- Eject quarters: do nothing
- Insert quarter: eject the inserted quarter
- Press start: do nothing
- Press cancel: stop the machine

CLASS DIAGRAM



CODE SNIPPETS

This code would be in the laundromat itself. A new washing machine object is created and stored along with its machine number. On line 22 the machine is added to the rmi registry so it can be accessed remotely.

```
18          try {
19              machineNum = Integer.parseInt(args[1]);
20
21              washingMachine = new WashingMachine(args[0],machineNum);
22              Naming.rebind("//"+args[0]+"/washingmachine",washingMachine);
23          }catch(Exception e) {
24              e.printStackTrace();
25          }
```

This is the State interface which declares the signature that every washing machine state will inherit from. It has four possible actions that will need to be implemented in the derived class. Using the state pattern we are able to encapsulate what varies and have every state implement its own actions. This makes it easy to add a new state without needing to modify existing ones.

```
3 public interface State {
4     public void insertQuarters();
5     public void ejectQuarters();
6     public void start();
7     public void cancel();
8 }
```

The code below is just a portion of the “not fully paid” state that the washing machine could potentially be in. The method insertQuarters( ) handles what would happen if a person were to insert a quarter into the machine while it is in this state. They need 4 quarters to start a wash and once the fourth quarter is inserted the machine will change from its current state the “has paid” state.

```
3  public class NotFullyPaidState implements State {
4      WashingMachine washingMachine;
5
6      //Constructor
7      public NotFullyPaidState(WashingMachine wm) {
10
11     public void insertQuarters() {
12         if(washingMachine.getNumQuarters() < 4) {
13             System.out.println("You have inserted " + washingMachine.getNumQuarters() + " quarters, "
14                     + (4 - washingMachine.getNumQuarters()) + " more needed.");
15         }
16         else if(washingMachine.getNumQuarters() == 4) {
17             System.out.println("Machine is ready to be started.");
18             washingMachine.setState(washingMachine.getHasPaidState());
19         }else if(washingMachine.getNumQuarters() > 4) {
20             System.out.println("You have already inserted $1.00, no more payment is needed.");
21             System.out.println("Returning "+ (washingMachine.getNumQuarters() -4) + " extra quarters");
22             washingMachine.setNumQuarters(4);
23             System.out.println("Machine is ready to be started.");
24             washingMachine.setState(washingMachine.getHasPaidState());
25         }
26     }
```

The two loops below belong to the monitoring application that remotely accesses all the washing machines spread across various locations. On line 23 you can see that we get a reference to each remote machine from the registry using the lookup( ) method. The second loop will print out a report of all the remote machines with their location, machine number, and state they are currently in.

```
21          for (int i=0;i < location.length; i++) {
22              try {
23                  WashingMachineRemote machine = (WashingMachineRemote) Naming.lookup(location[i]);
24                  monitor[i] = new WashingMachineMonitor(machine);
25              } catch (Exception e) {
26                  e.printStackTrace();
27              }
28          }
29
30          for(int i=0; i < monitor.length; i++) {
31              monitor[i].report();
32          }
33      }
```

SCREEN SHOTS

```
You have inserted 2 quarters, 2 more needed.

Please select an option:
1. Insert a quarter (cost $1.00)
2. Eject quarters
3. Start Machine
4. Cancel Wash
1
YOUR CHOICE: 1
You have inserted 3 quarters, 1 more needed.

Please select an option:
1. Insert a quarter (cost $1.00)
2. Eject quarters
3. Start Machine
4. Cancel Wash
2
YOUR CHOICE: 2
Returning 3 quarters
```

This is a screen shot of machine number 2 that is running in Moroni. The user inserted 3 quarters and then ejected them which returned all three quarters.

Along with this machine there are also two others running. One in Manti and one in Ephraim each in different states as shown below when we run the monitoring application.

This is the remote monitoring application that allows us to see the states of washing machines in other locations. I only created three machines for this example, but it would be able remotely monitor any washing machines that are created.