Objectives

- Vending Machine Exercise
  - Implementing against unit tests
  - Illustrate the State Pattern

Uses of State Machines[1]

- State Machines can model simple mechanistic systems such as vending machines and turnstiles
- Vending machine actions are insert quarters, selecting items and states are change entered, number of items remaining etc.

[1] image:

https://commons.wikimedia.org/wiki/File:Nichirei_food_vending_machine_02_in_Japan.jpg

The State Pattern

- Create a state machine in an object oriented fashion
- Classes Involved:
  - Context
  - State Interface
  - Concrete State Classes

State

"Allow an object to alter its behavior when its internal state changes. The object will appear to change its class." *GoF*

State Pattern Summary

- The state pattern allows an object to have an internal state that changes its behavior
- Each state is represented by a class (will increase the number of classes in your design)
- The class diagrams for State and Strategy are the same
  - *Strategy*: alternative to subclassing
  - *State*: prevent a lot of conditional statements from appearing in your main class

Clone the Starter Code

- **In this exercise we will implement a simulation of a basic vending machine**
- We have set up a public repository containing the starter code and unit tests for the state pattern exercise[2]
- You can **clone** this project and import to Eclipse (Import → maven → existing maven projects)
- Can be in either Java EE or Java SE perspective (upper right corner)

[2] `https://github.com/marks1024/vending-exercise-361`

Requirements: Vending Machine Exercise

- `VendingMachine` is the main context class
- There are 3 states: *idle*, *entering coins*, and *paid*
- There are 3 actions: *insert coin*, *refund*, *vend*
- The vending machine should always begin in the idle state with a balance of 0
- The vending machine should only accept coins with value 50 or 100. Any other amount should result raise an `IllegalArgumentException`
- `insertCoin()` causes the machine to enter the *entering coins* state

Requirements (cont.)

- The vending machine should enter the *paid* state when a balance of 200 or greater accrues (the vended item costs 200)
- To vend an item call the `vend()` method
  - The value returned by `vend()` is equal to the surplus balance
- Both `vend()` and `refund()` should return the machine to the idle state with a balance of 0
- The balance should accumulate until either `vend()` or `refund()` is called
- The complete expected behavior is documented in the JUnit tests
- The classes also contain information in javadoc comments

Task: Vending Machine Exercise

- Note that the project will have errors (due to missing classes) when you first import it
- Implement the 3 concrete state classes (`IdleState`, `EnteringCoinsState`, `PaidState`) and complete the methods labeled `// TODO`
- All of the unit tests (9 of them) in the project should pass
- Submit your zipped project folder to the moodle
- Your solution should also follow the state pattern as described in lecture
- **Don't edit the unit tests**