HSE 13PI Second course "State Pattern"

Created by Bakaev Nikita

Sorry for my English

Main file is `StatePattern.cpp` in folder `StatePattern`

Intent

- Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
- An object-oriented state machine
- wrapper + polymorphic wrappee + collaboration

Problem

A monolithic object's behavior is a function of its state, and it must change its behavior at run-time depending on that state. Or, an application is characterixed by large and numerous case statements that vector flow of control based on the state of the application.

Discussion

The State pattern is a solution to the problem of how to make behavior depend on state.

Define a "context" class to present a single interface to the outside world. Define a State abstract base class. Represent the different "states" of the state machine as derived classes of the State base class. Define state-specific behavior in the appropriate State derived classes. Maintain a pointer to the current "state" in the "context" class. To change the state of the state machine, change the current "state" pointer. The State pattern does not specify where the state transitions will be defined. The choices are two: the "context" object, or each individual State derived class. The advantage of the latter option is ease of adding new State derived classes. The disadvantage is each State derived class has knowledge of (coupling to) its siblings, which introduces dependencies between subclasses.

A table-driven approach to designing finite state machines does a good job of specifying state transitions, but it is difficult to add actions to accompany the state transitions. The pattern-based approach uses code (instead of data structures) to specify state transitions, but it does a good job of accomodating state transition actions.

Structure

The state machine's interface is encapsulated in the "wrapper" class. The wrappee hierarchy's interface mirrors the wrapper's interface with the exception of one additional parameter. The extra parameter allows wrappee derived classes to call back to the wrapper class as necessary. Complexity that would otherwise drag down the wrapper class is neatly compartmented and encapsulated in a polymorphic hierarchy to which the wrapper object delegates.

Check list

Identify an existing class, or create a new class, that will serve as the "state machine" from the client's perspective. That class is the "wrapper" class. Create a State base class that replicates the methods of the state machine interface. Each method takes one additional parameter: an instance of the wrapper class. The State base class specifies any useful "default" behavior. Create a State derived class for each domain state. These derived classes only override the methods they need to override. The wrapper class maintains a "current" State object. All client requests to the wrapper class are simply delegated to the current State object, and the wrapper object's this pointer is passed. The State methods change the "current" state in the wrapper object as appropriate.

How does it work?

We have class `Machine` which has field `current` which contains Class which implemented `IMachineActions` interfase.

And we have two implementation of the `IMachineActions`:

- **DefaultMachine** - You can't do anything with this car. You should choose another to continie

- **CoolCabriolet** - It's a *good* car. It's for you, bro.
- **OldRussianCar** - It's a *bad* car. It's for, bro.

How to use ???

Create with public constructor and `setCurrent` which get an object implementing `IMachineActions`

Depends of implementaion, you will have different max speed of the car, changes in the number of speed when you call the slow and fast.

```
Machine fsm = *new Machine();
fsm.setCurrent(new DefaultMachine());
```

IMachineActions

You should override following methods to creat and implementation of the interface:

```
virtual bool faster ()
virtual bool slower()
virtual string getName()
virtual int getMaxSpeed()
```

UML

That's all