State Pattern

Shin-Jie Lee (李信杰)
Associate Professor
Computer and Network Center
Department of CSIE
National Cheng Kung University

Design Aspect of State

states of an object

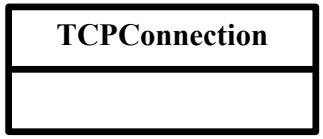
Outline

- A TCP Connection Requirements Statements
- Initial Design and Its Problems
- Design Process
- Refactored Design after Design Process
- Recurrent Problems
- Intent
- State Pattern Structure
- A Gumball Machine: Another Example

A TCP Connection

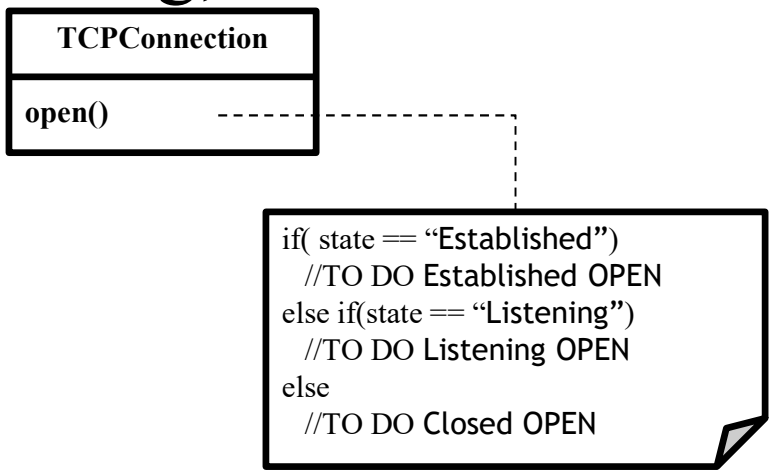
Requirements Statement$_1$

- A class TCPConnection that represents a network connection.

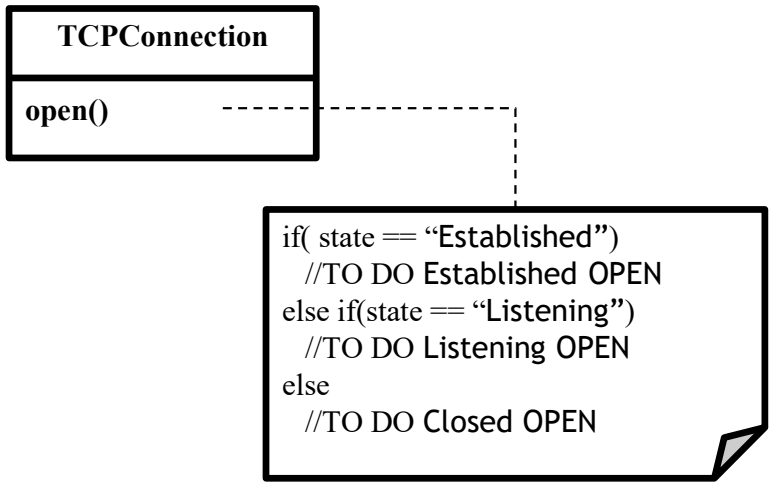


Requirements Statement$_2$

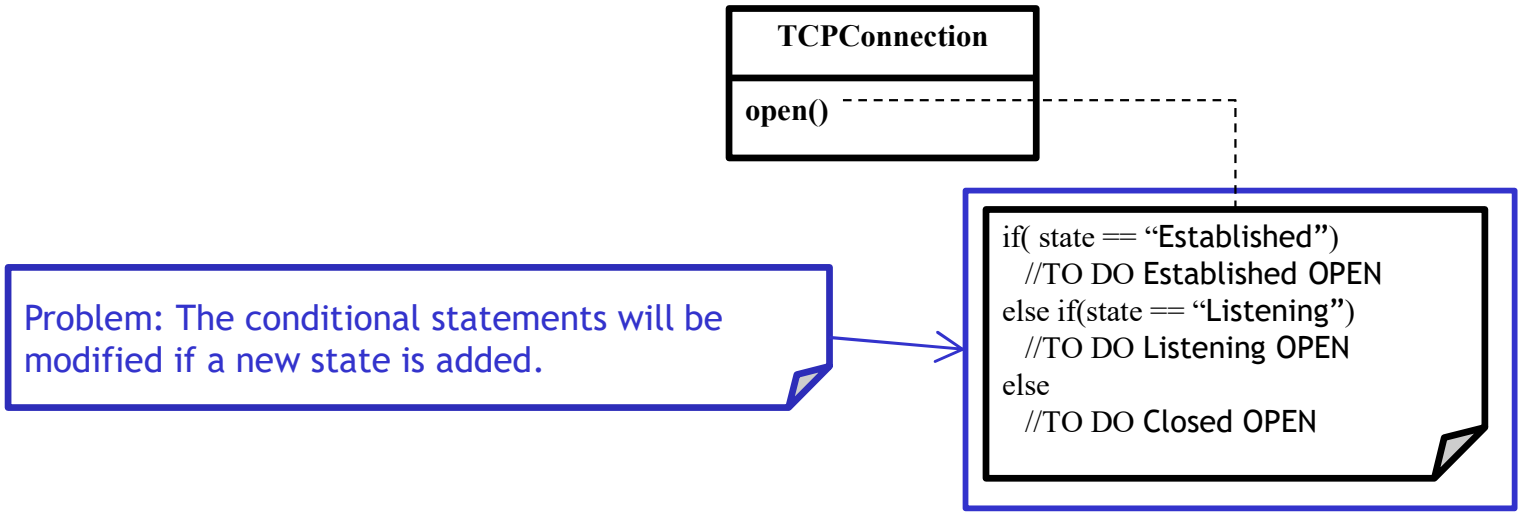
- When a TCPConnection object receives an Open request from other objects, it responds differently depending on its current state. A TCPConnection object can be in one of several different states: Established, Listening, and Closed.



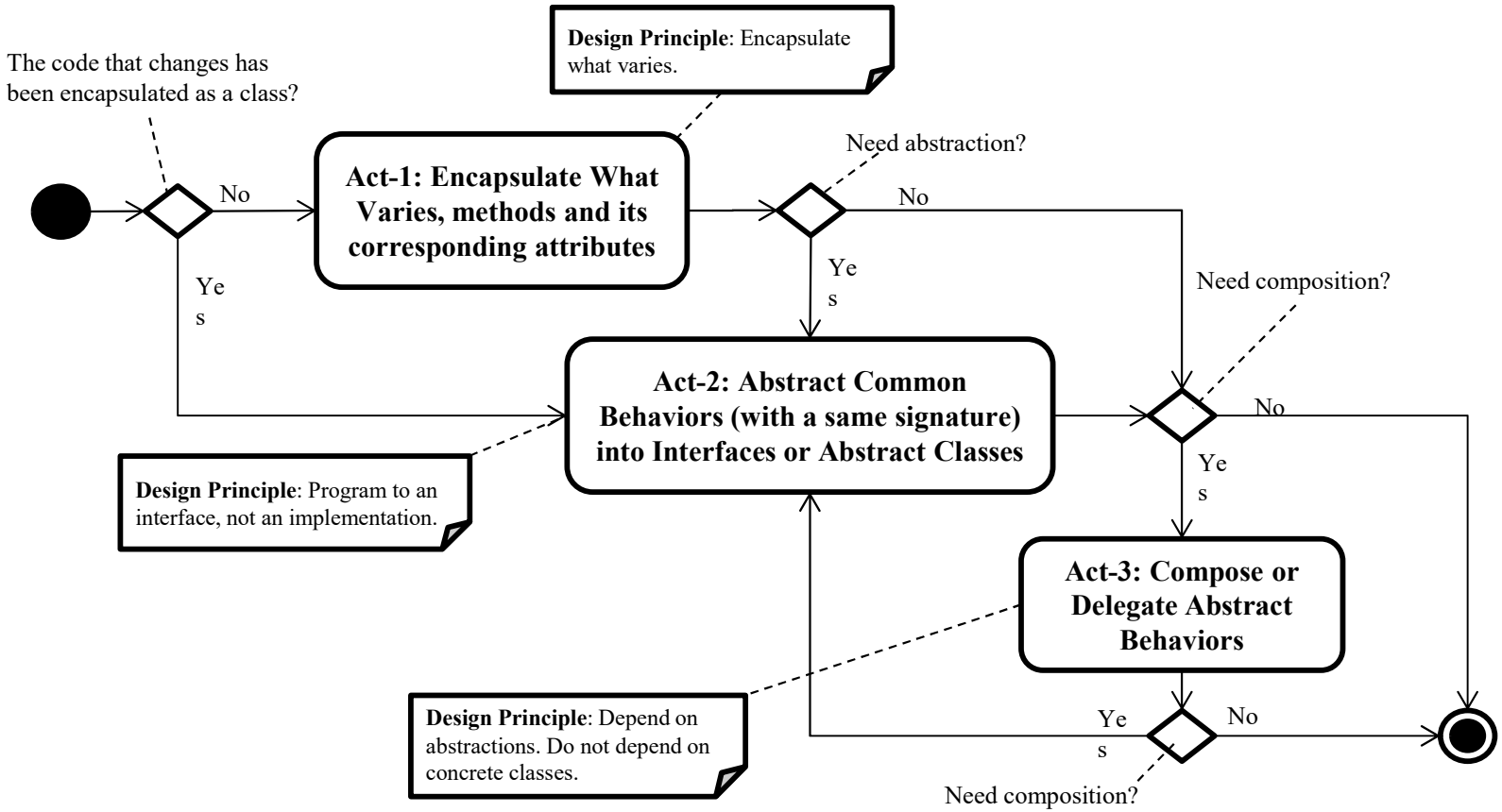
Initial Design - Class Diagram



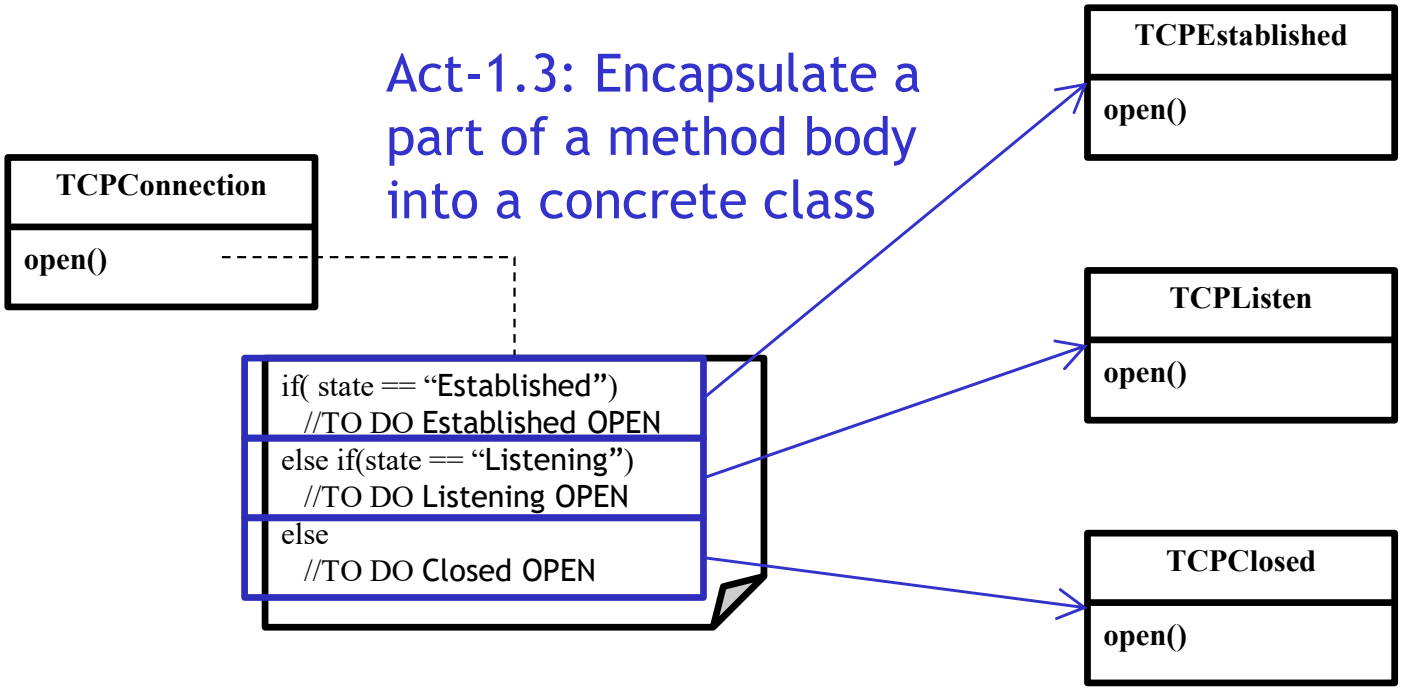
Problems with Initial Design



Design Process for Change

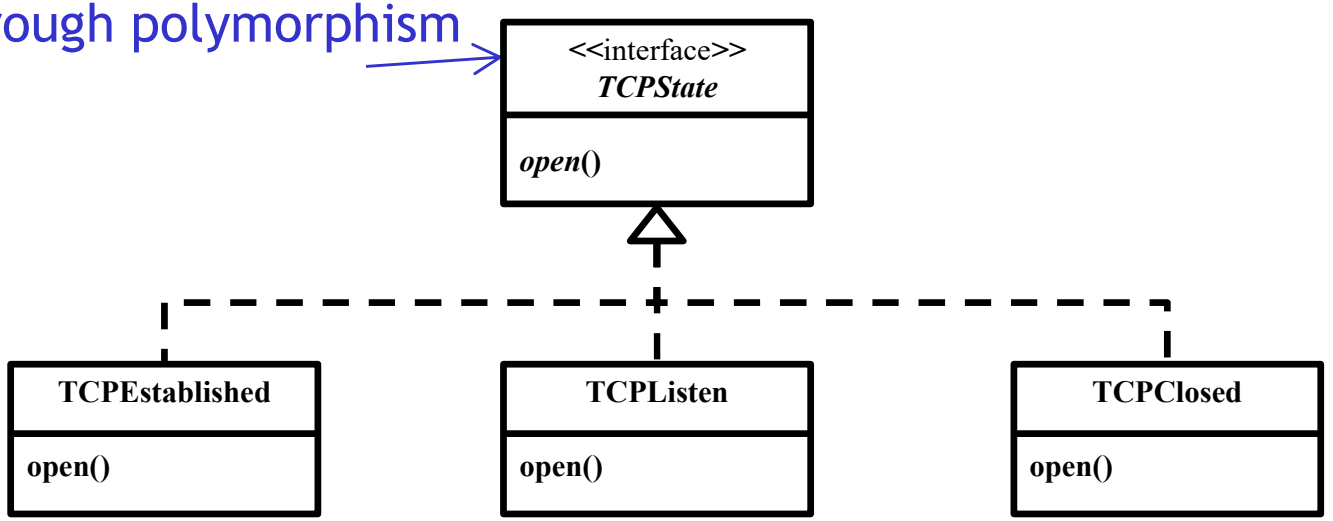


Act-1: Encapsulate What Varies



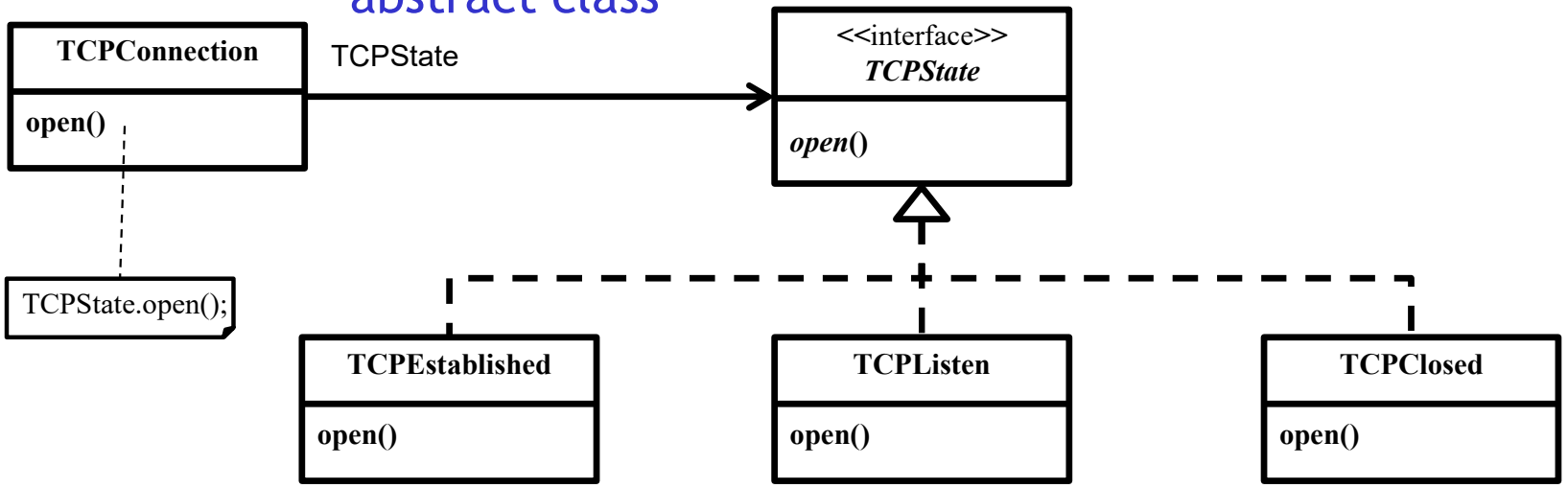
Act-2: Abstract Common Behaviors

Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism

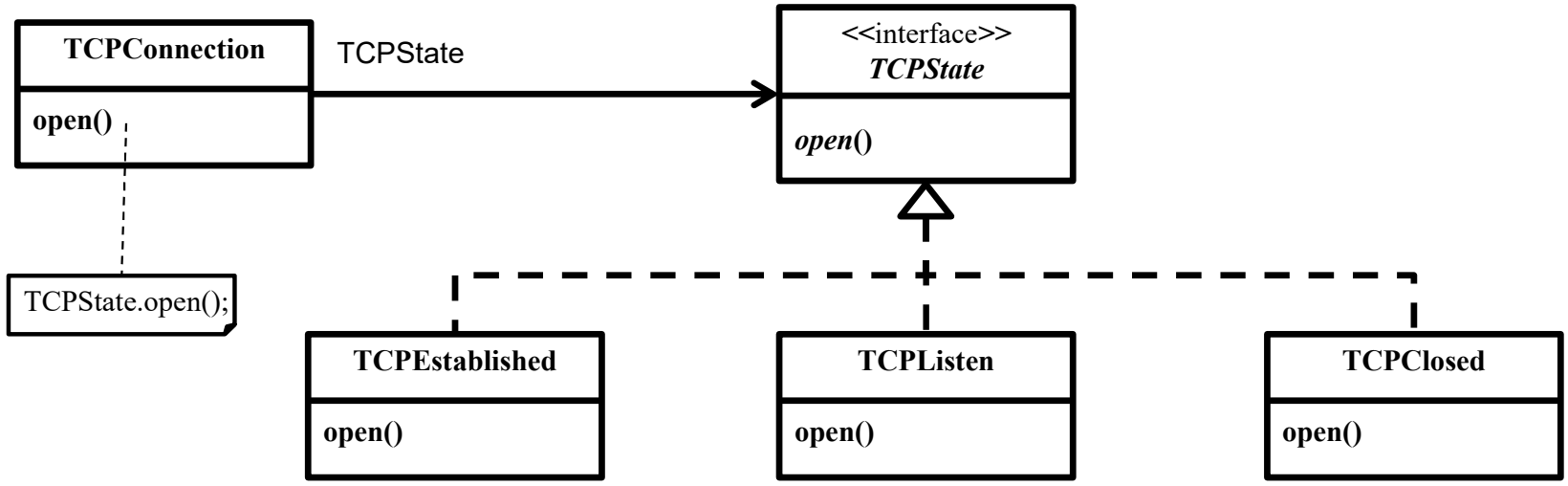


Act-3: Compose Abstract Behaviors

Act-3.3: Delegate behavior to an interface or an abstract class



Refactored Design after Design Process



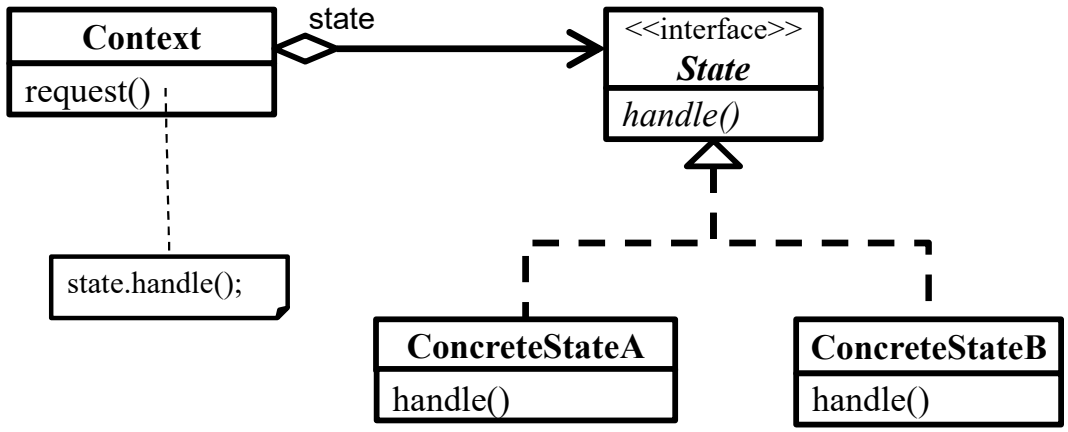
Recurrent Problem

- An object's behavior depends on its state, and it must change its behavior at run-time depending on that state.

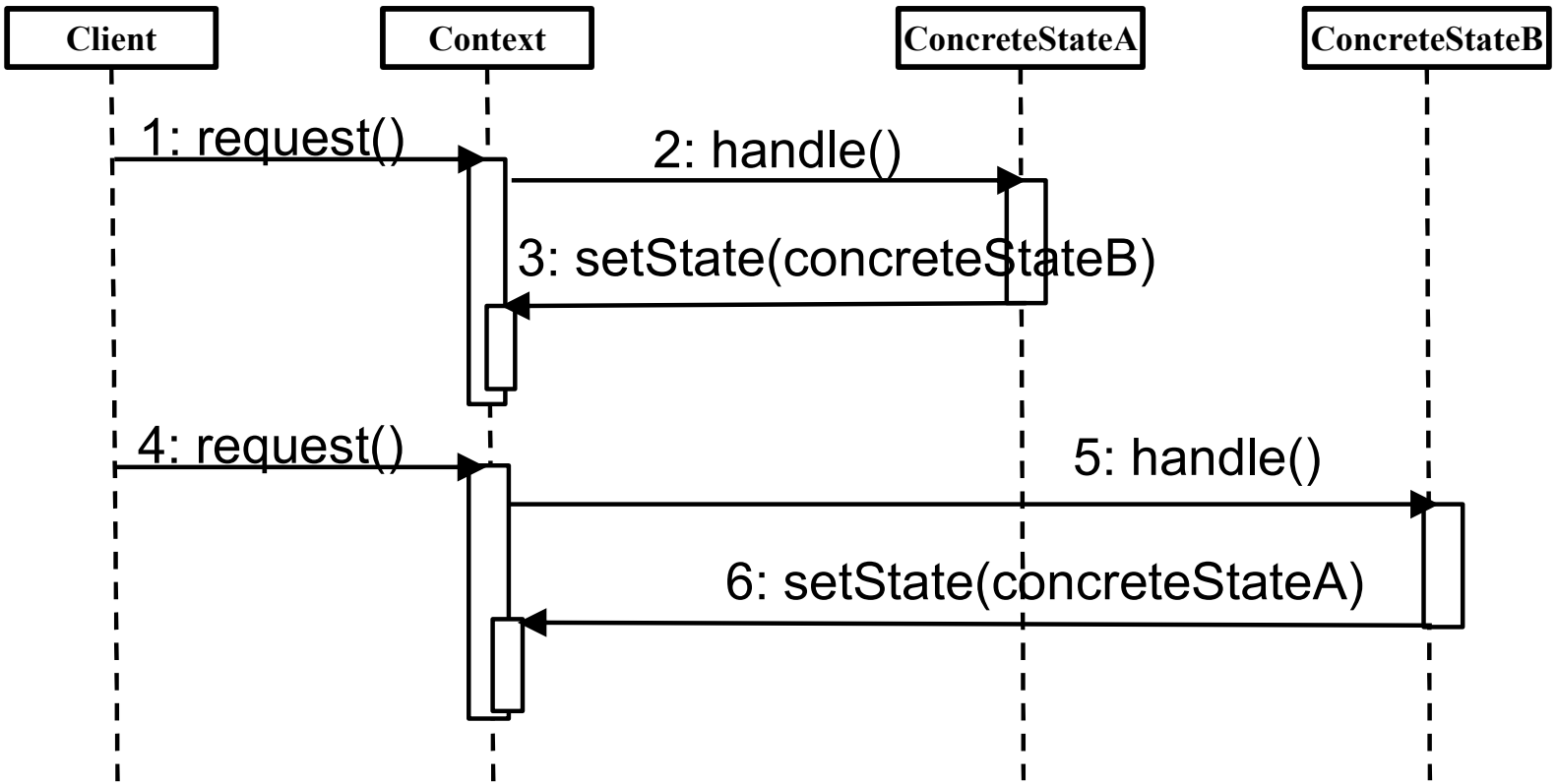
State Pattern

- Intent
  - Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

State Pattern Structure$_1$



State Pattern Structure$_2$



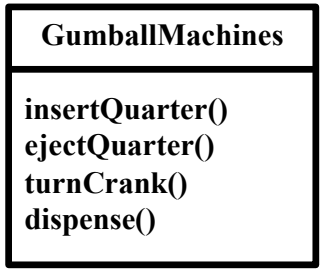
State Pattern Structure$_3$

	Instantiation	Use	Termination
Context	Don't Care	Don't Care	Don't Care
State	X	Context uses State to invoke ConcreteStates' method through polymorphism.	X
ConcreteStateA	Don't Care	Context invokes ConcreteStateA's method through polymorphism.	Don't Care
ConcreteStateB	Don't Care	Context invokes ConcreteStateB's method through polymorphism.	Don't Care

A Gumball Machine

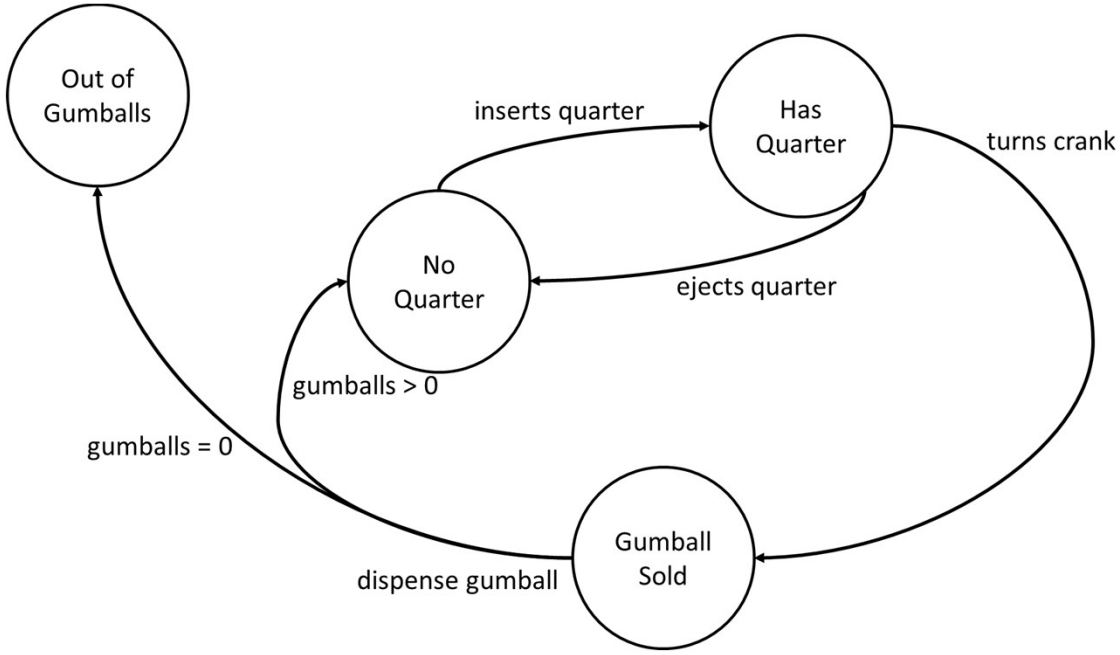
Requirements Statement$_1$

- A GumballMachine has four actions: Insert Quarter, Eject Quarter, Turn Crank, and Dispense.



Requirements Statement$_2$

- There are four states in the GumballMachine: No Quarter, Has Quarter, Out of Gumballs and Gumball Sold. As the following state diagram.



Requirements Statement$_3$

```
if (state == HAS_QUARTER)
 "Turning twice doesn't get you another gumball!“
else if (state == NO_QUARTER)
 "You turned but there's no quarter“
else if (state == SOLD_OUT)
 "You turned, but there are no gumballs“
else if (state == SOLD)
   "You turned...“
   state = SOLD;
   dispense();
```

```
if (state == HAS_QUARTER)
 “No gumball dispensed”
else if (state == NO_QUARTER)
 "You need to pay first“
else if (state == SOLD_OUT)
 "No gumball dispensed“
else if (state == SOLD)
   "A gumball comes rolling out the slot“
   count = count - 1;
   …..
```

GumballMachines

insertQuarter()
ejectQuarter()
turnCrank()
dispense()

```
if (state == HAS_QUARTER)
 " You can't insert another quarter"
else if (state == NO_QUARTER)
   state = HAS_QUARTER;
   "You inserted a quarter"
else if (state == SOLD_OUT)
   "You can't insert a quarter, the machine is sold out“
else if (state == SOLD)
   "Please wait, we're already giving you a gumball"
```

```
if (state == HAS_QUARTER)
  "Quarter returned“
   state = NO_QUARTER;
else if (state == NO_QUARTER)
   "You haven't inserted a quarter"
else if (state == SOLD_OUT)
 "Sorry, you already turned the crank"
else if (state == SOLD)
 "You can't eject, you haven't inserted a quarter yet"
```

Initial Design - Class Diagram

```
if (state == HAS_QUARTER)
 "Turning twice doesn't get you another gumball!“
else if (state == NO_QUARTER)
 "You turned but there's no quarter“
else if (state == SOLD_OUT)
 "You turned, but there are no gumballs“
else if (state == SOLD)
   "You turned...“
   state = SOLD;
   dispense();
```

```
if (state == HAS_QUARTER)
 “No gumball dispensed”
else if (state == NO_QUARTER)
 "You need to pay first“
else if (state == SOLD_OUT)
 "No gumball dispensed“
else if (state == SOLD)
   "A gumball comes rolling out the slot“
   count = count - 1;
   …..
```

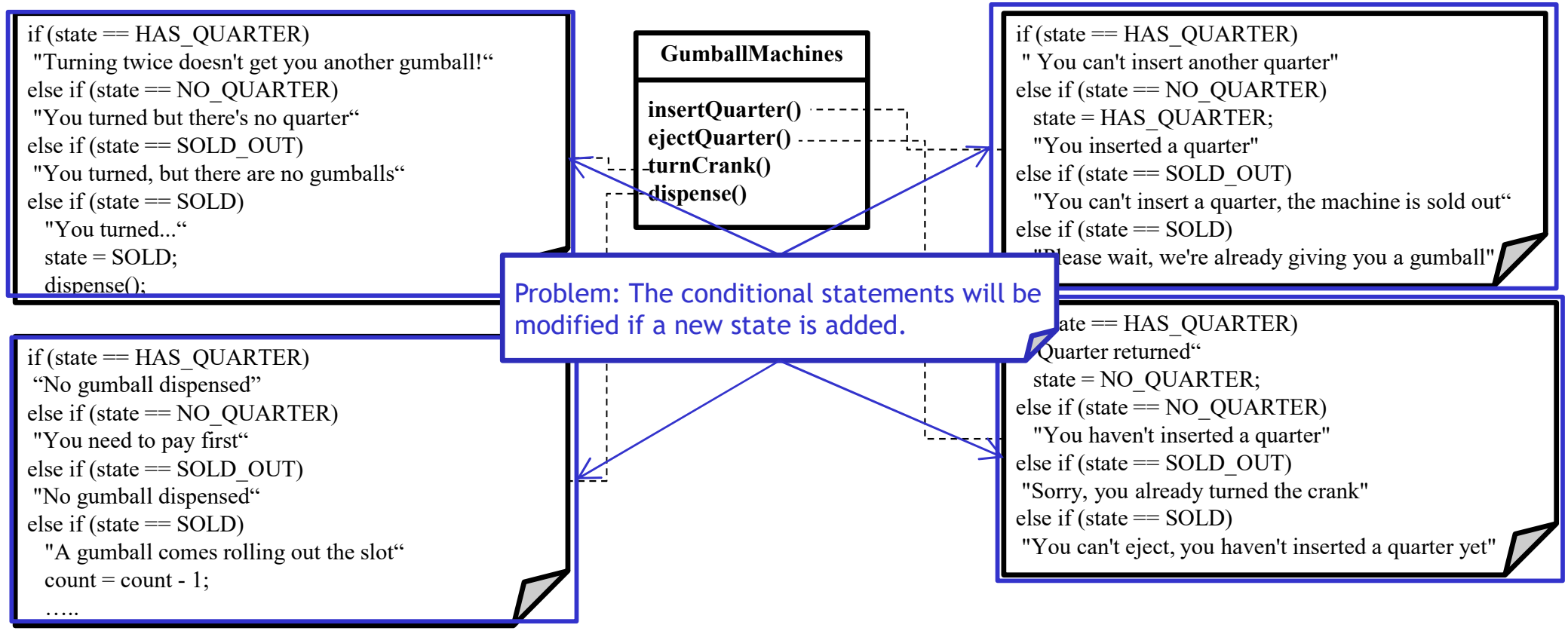
GumballMachines

insertQuarter()
ejectQuarter()
turnCrank()
dispense()

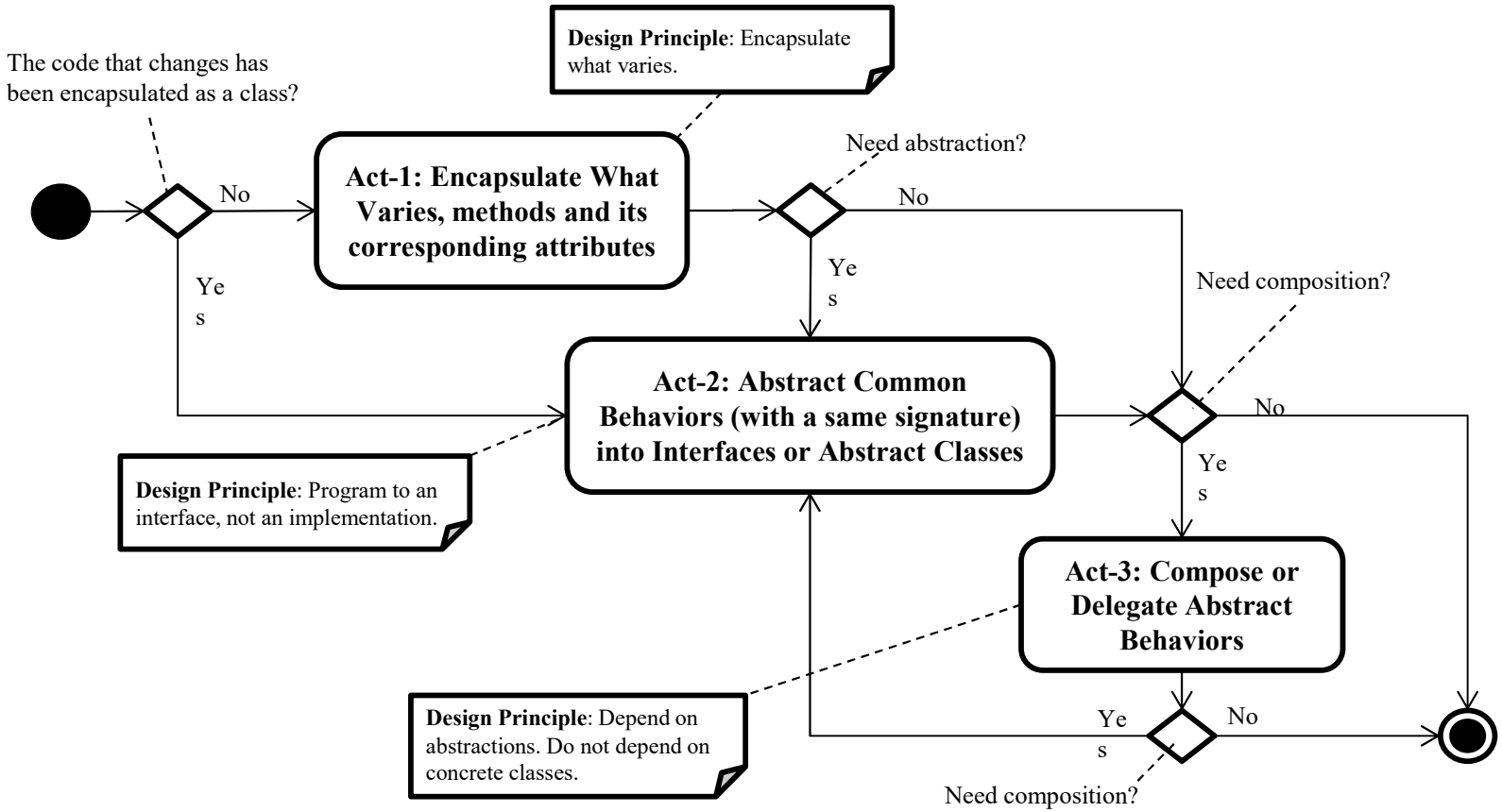
```
if (state == HAS_QUARTER)
 " You can't insert another quarter"
else if (state == NO_QUARTER)
   state = HAS_QUARTER;
   "You inserted a quarter"
else if (state == SOLD_OUT)
   "You can't insert a quarter, the machine is sold out“
else if (state == SOLD)
   "Please wait, we're already giving you a gumball"
```

```
if (state == HAS_QUARTER)
  "Quarter returned“
   state = NO_QUARTER;
else if (state == NO_QUARTER)
   "You haven't inserted a quarter"
else if (state == SOLD_OUT)
 "Sorry, you already turned the crank"
else if (state == SOLD)
 "You can't eject, you haven't inserted a quarter yet"
```

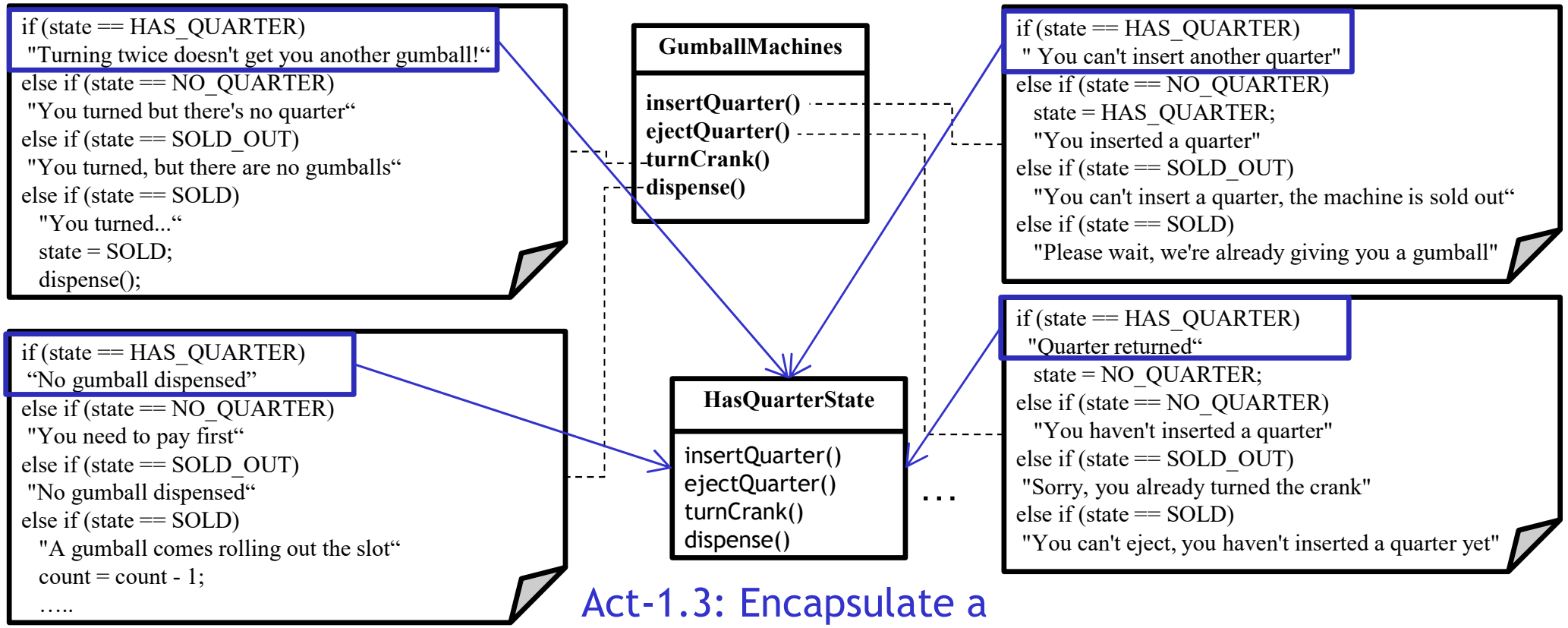
Problems with Initial Design



Design Process for Change



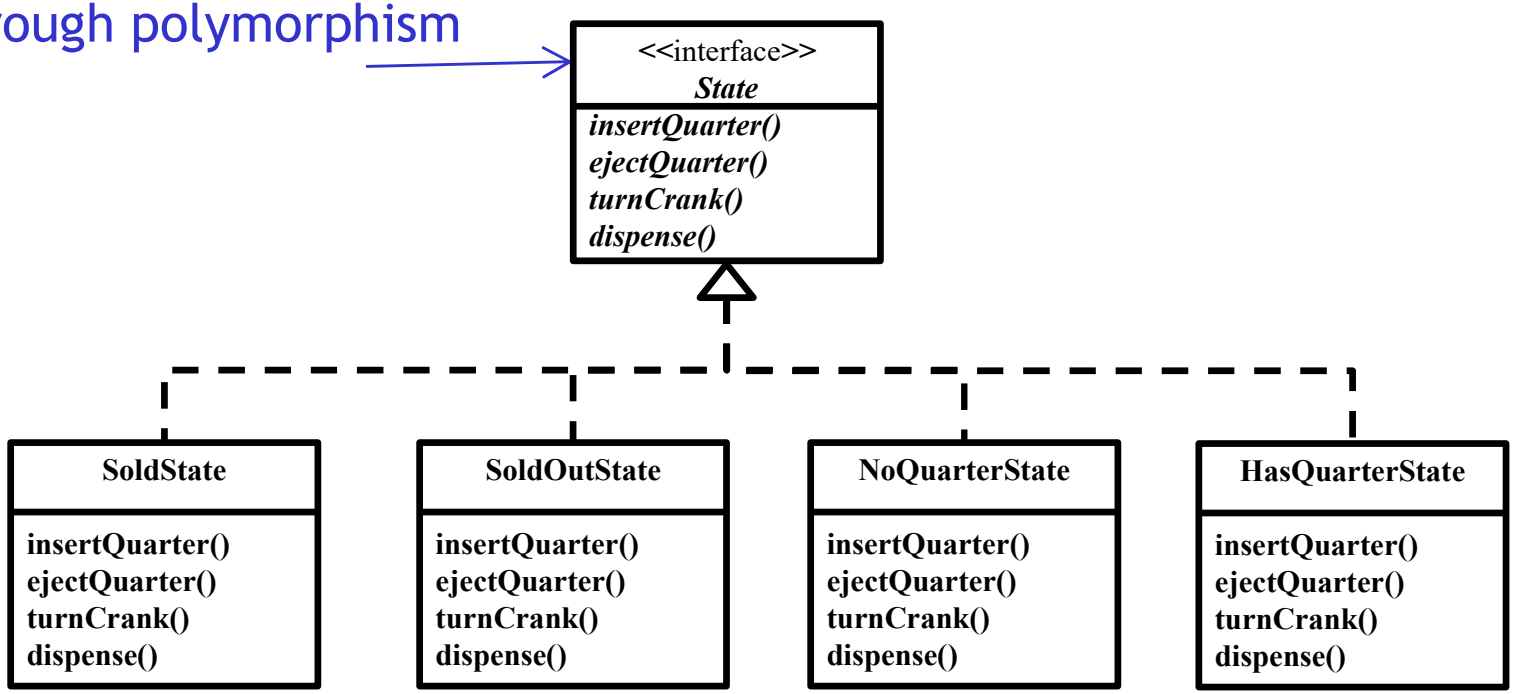
Act-1: Encapsulate What Varies



Act-1.3: Encapsulate a part of a method body into a concrete class

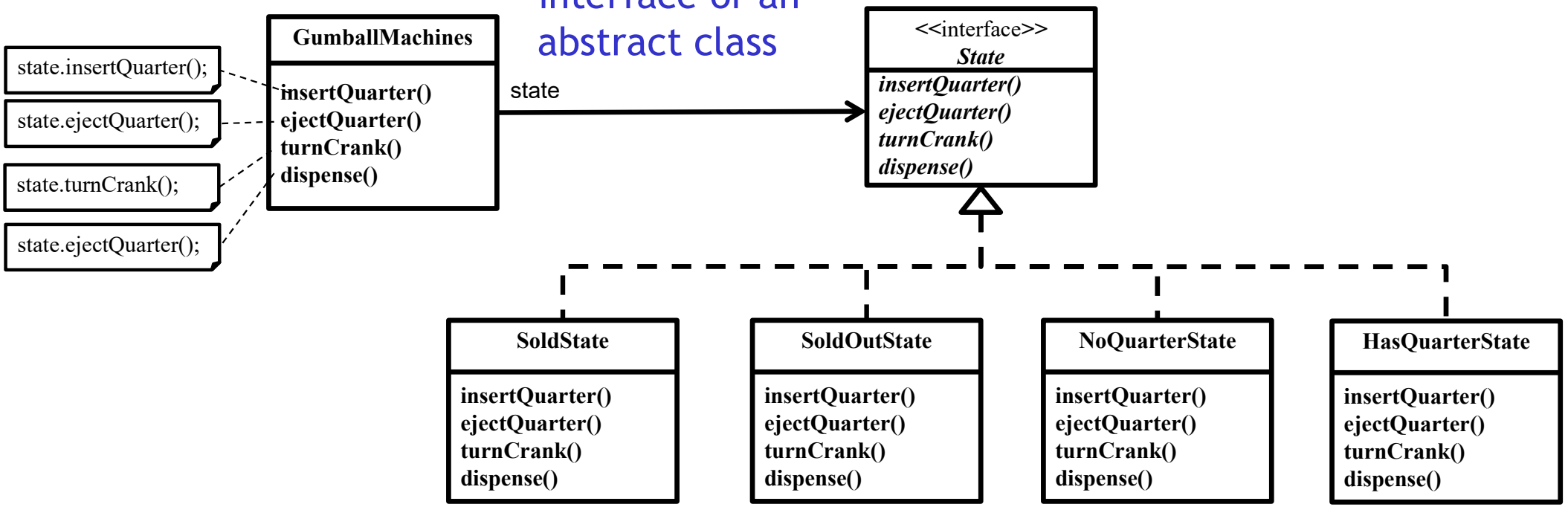
Act-2: Abstract Common Behaviors

Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism



Act-3: Compose Abstract Behaviors

Act-3.3: Delegate behavior to an interface or an abstract class



Refactored Design after Design Process