

스테이트 패턴

이관우

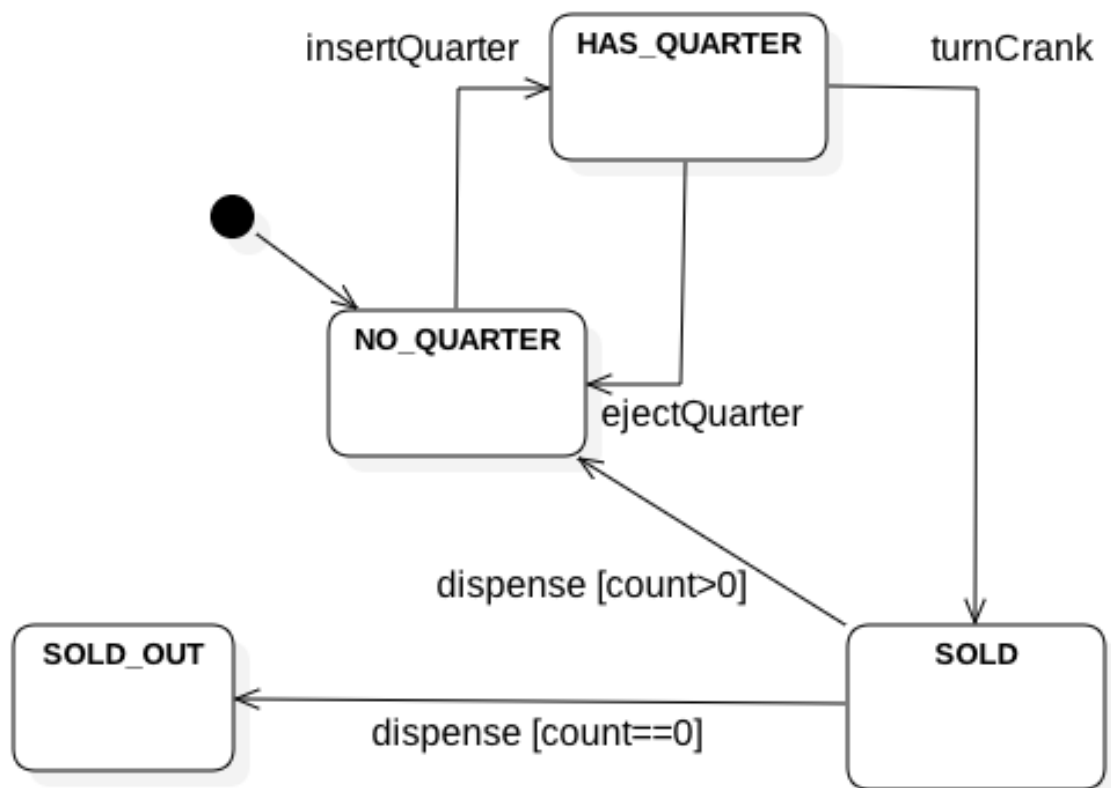
kwlee@hansung.ac.kr

학습 목표

- 스테이트 패턴이 다루는 문제를 이해.
- 스테이트 패턴의 정의 및 작동 방식 이해
- 스테이트 패턴과 스트래티지 패턴과의 차이점 이해

Gumball 기계 요구사항

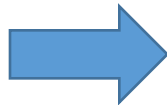
- 다음과 같은 식으로 작동하는 Gumball 기계
- 새로운 기능 추가가 용이하도록 디자인



상태 기계의 구현

상태

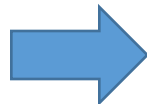
SOLD_OUT
NO_QUATER
HAS_QUATER
SOLD



```
final static int SOLD_OUT = 0;  
final static int NO_QUATER = 1;  
final static int HAS_QUATER = 2;  
final static int SOLD = 3;  
  
int state = SOLD_OUT;
```

액션

insert quarter



eject quarter

turn crank

```
public void insertQuarter() {  
    if (state == HAS_QUATER) {  
        System.out.println("You can't insert another  
                             quarter");  
    } else if (state == NO_QUATER) {  
        state = HAS_QUATER;  
        System.out.println("You inserted a quarter");  
    } else if (state == SOLD_OUT) {  
        System.out.println("You can't insert a quarter,  
                             the machine is sold out");  
    } else if (state == SOLD) {  
        System.out.println("Please wait, we're already  
                             giving you a gumball");  
    }  
}
```

Gumball 상태 기계 구현 코드

```
public class GumballMachine {
```

```
    final static int SOLD_OUT = 0;  
    final static int NO_QUARTER = 1;  
    final static int HAS_QUARTER = 2;  
    final static int SOLD = 3;
```



네 가지 상태

```
    int state = SOLD_OUT;  
    int count = 0;
```

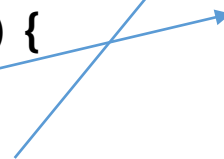
```
    public GumballMachine(int count) {  
        this.count = count;  
        if (count > 0) {  
            state = NO_QUARTER;  
        }  
    }  
}
```



현재 상태



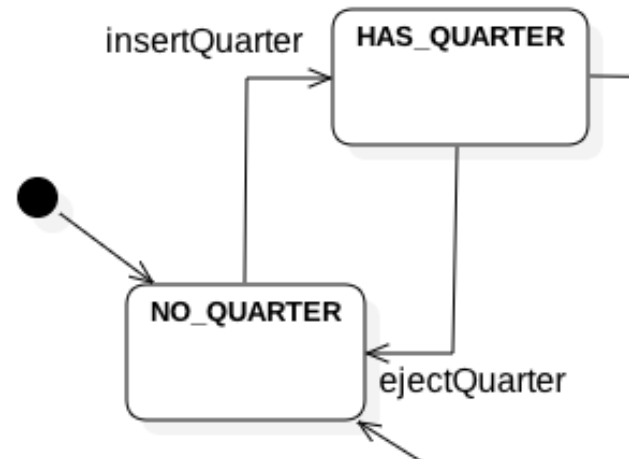
Gumball 개수



Gumball 상태 기계 구현 코드

// 동전을 반환 받으려고 하는 경우

```
public void ejectQuarter() {  
    if (state == HAS_QUARTER) {  
        System.out.println("Quarter returned");  
        state = NO_QUARTER;  
  
    } else if (state == NO_QUARTER) {  
        System.out.println("You haven't inserted a quarter");  
  
    } else if (state == SOLD) {  
        System.out.println("Sorry, you already turned the crank");  
  
    } else if (state == SOLD_OUT) {  
        System.out.println("You can't eject, you haven't inserted a quarter yet");  
    }  
}
```



Gumball 상태 기계 구현 코드

```

public void turnCrank() {                                     // 손잡이를 돌리는 경우

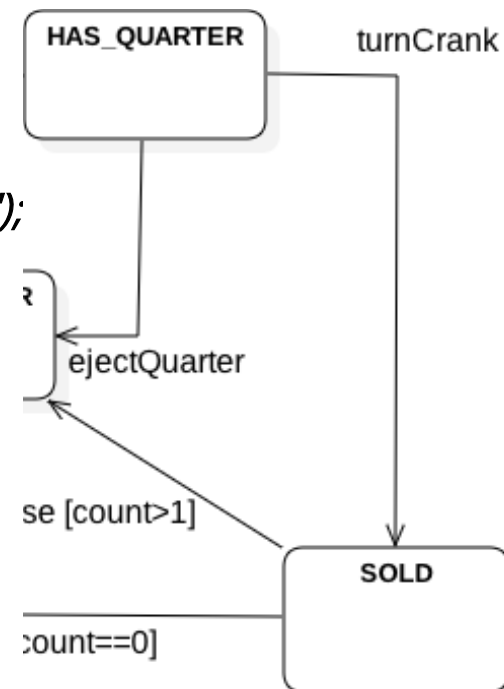
    if (state == SOLD) {
        System.out.println("Turning twice doesn't get you another gumball!");

    } else if (state == NO_QUARTER) {
        System.out.println("You turned but there's no quarter");

    } else if (state == SOLD_OUT) {
        System.out.println("You turned, but there are no gumballs");

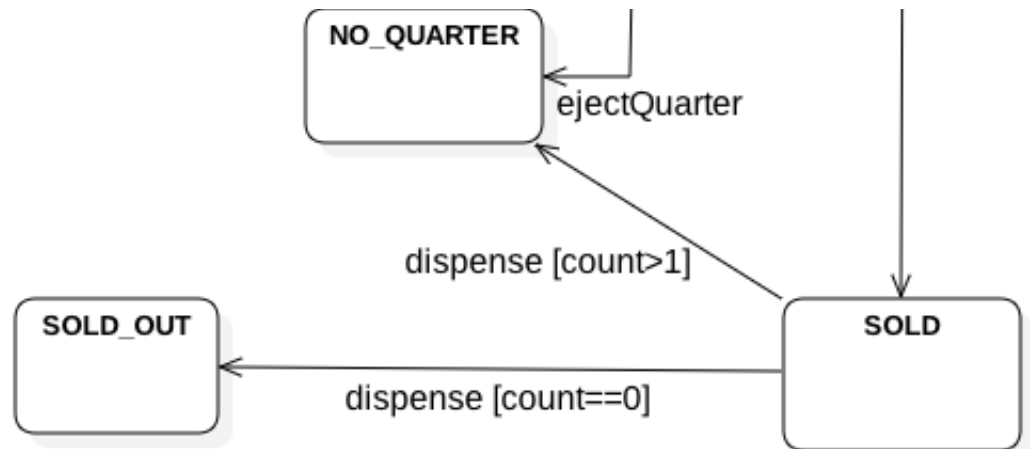
    } else if (state == HAS_QUARTER) {
        System.out.println("You turned...");
        state = SOLD;
        dispense();
    }
}

```



Gumball 상태 기계 구현 코드

```
private void dispense() {                                     // Gumball 내보내기 (내부적 호출)
    if (state == SOLD) {
        System.out.println("A gumball comes rolling out the slot");
        count = count - 1;
        if (count == 0) {
            System.out.println("Oops, out of gumballs!");
            state = SOLD_OUT;
        } else {
            state = NO_QUARTER;
        }
    }
}
```

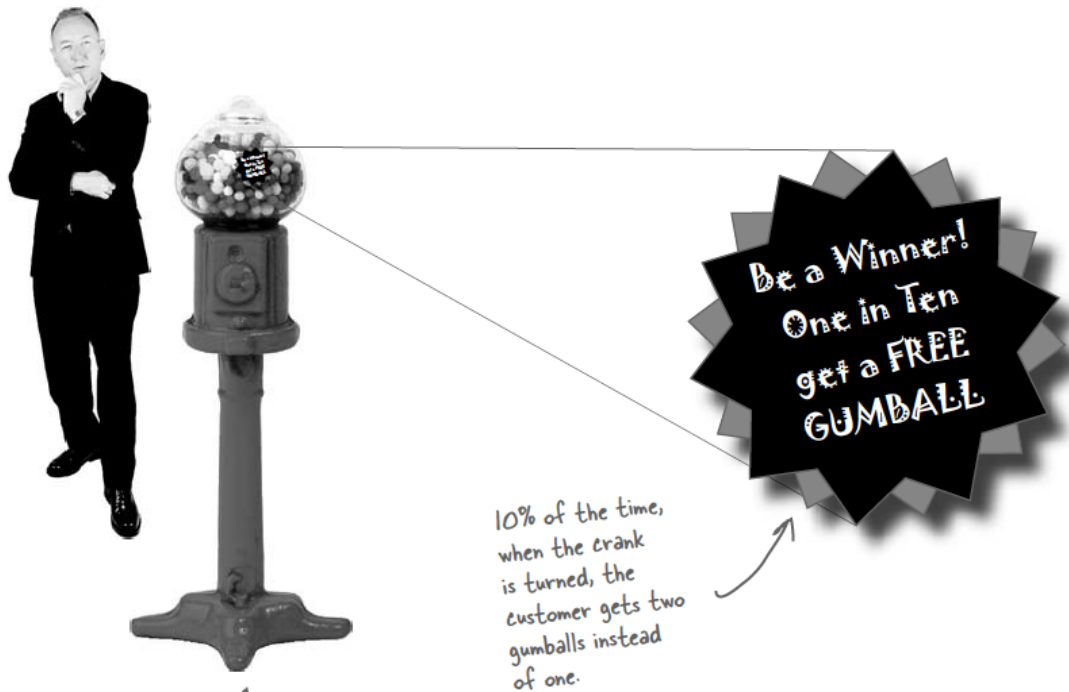


코드 테스트

```
public class GumballMachineTestDrive {  
  
    public static void main(String[] args) {  
        GumballMachine gumballMachine = new GumballMachine(5);  
  
        System.out.println(gumballMachine);  
  
        gumballMachine.insertQuarter();  
        gumballMachine.turnCrank();  
  
        System.out.println(gumballMachine);  
  
        gumballMachine.insertQuarter();  
        gumballMachine.ejectQuarter();  
        gumballMachine.turnCrank();  
  
        System.out.println(gumballMachine);  
        ...  
    }  
}
```

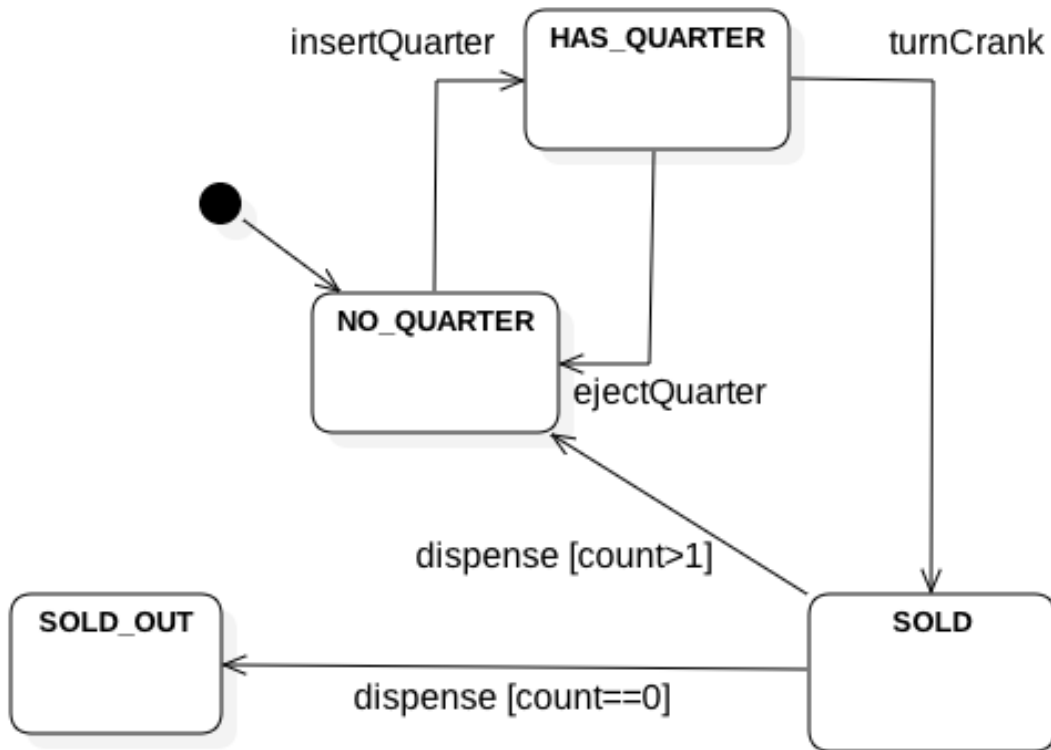
변경 요청

- 열번에 한번 꼴로 Gumball을 하나더 받을 수 있도록 수정



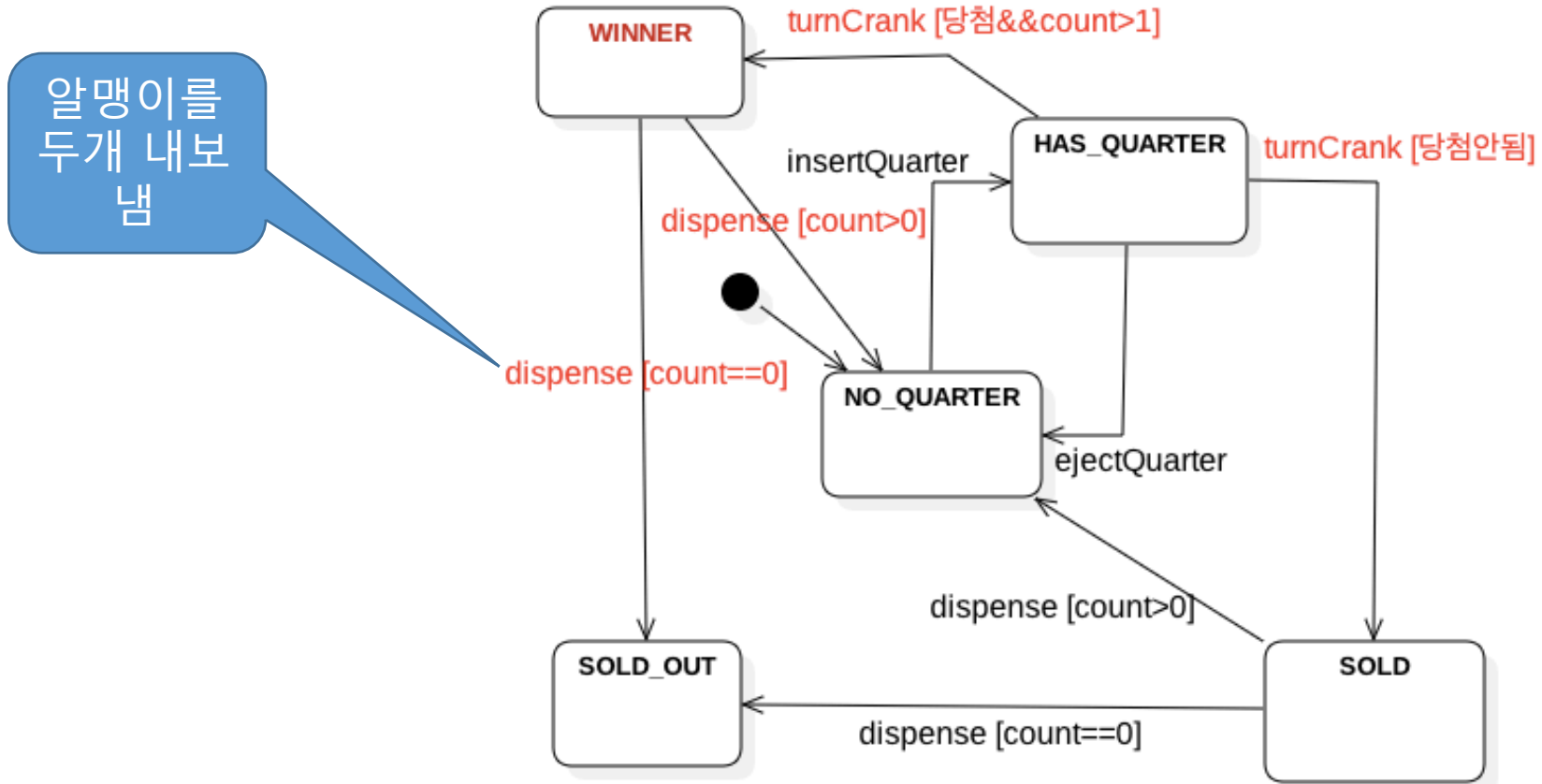
상태 다이어그램을 그려 봅시다.

- 열번에 한번 꼴로 알맹이를 하나씩 더 주는 Gumball 기계를 위한 상태 다이어그램을 수정해 보세요.



상태 다이어그램을 그려 봅시다.

- 열번에 한번 꼴로 알맹이를 하나씩 더 주는 Gumball 기계를 위한 상태 다이어그램을 수정해 보세요



변경 사항 검토

```
final static int SOLD_OUT = 0;  
final static int NO_QUARTER = 1;  
final static int HAS_QUARTER = 2;  
final static int SOLD = 3;
```

```
public void insertQuarter() {  
    // insert quarter code here  
}
```

```
public void ejectQuarter() {  
    // eject quarter code here  
}
```

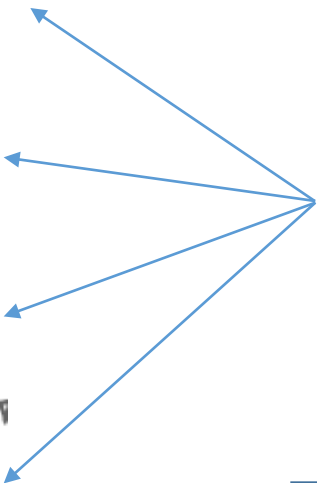
```
public void turnCrank() {  
    // turn crank code here  
}
```

```
public void dispense() {  
    // dispense code here  
}
```

WINNER 상태 추가



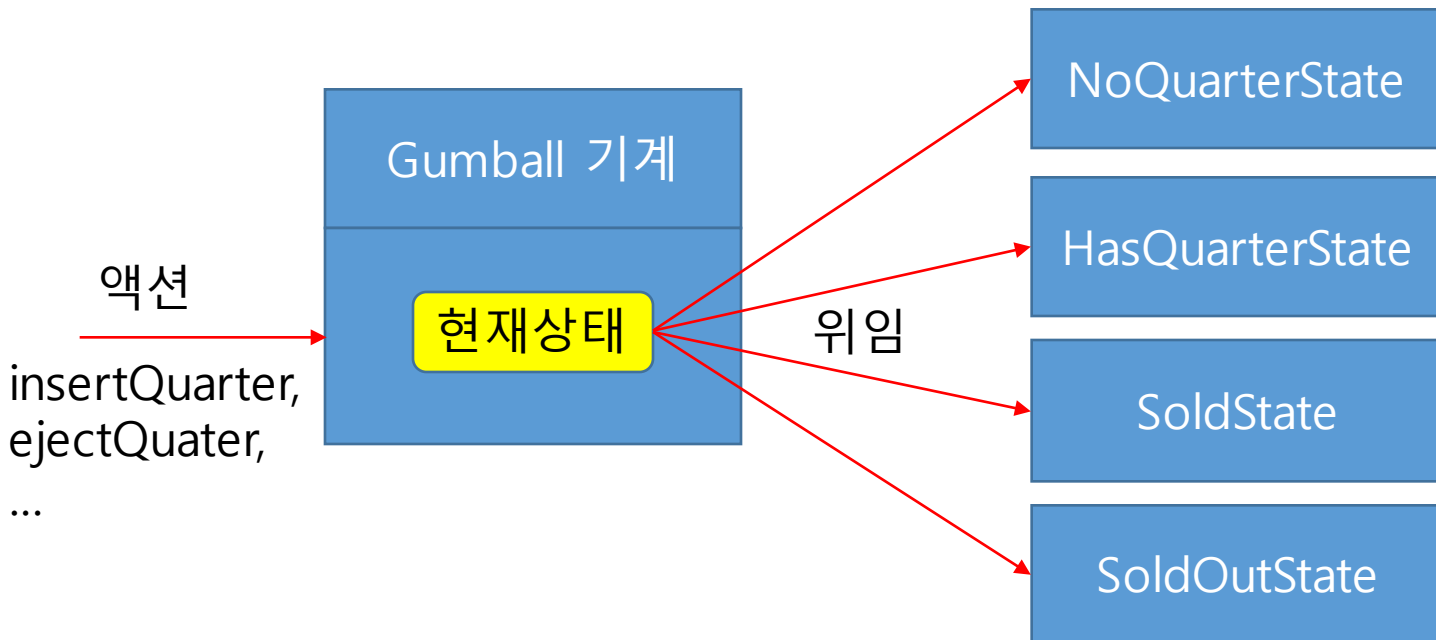
WINNER 상태 추가에
따른 새로운 조건 검사
를 모든 메소드에 추가
해야함



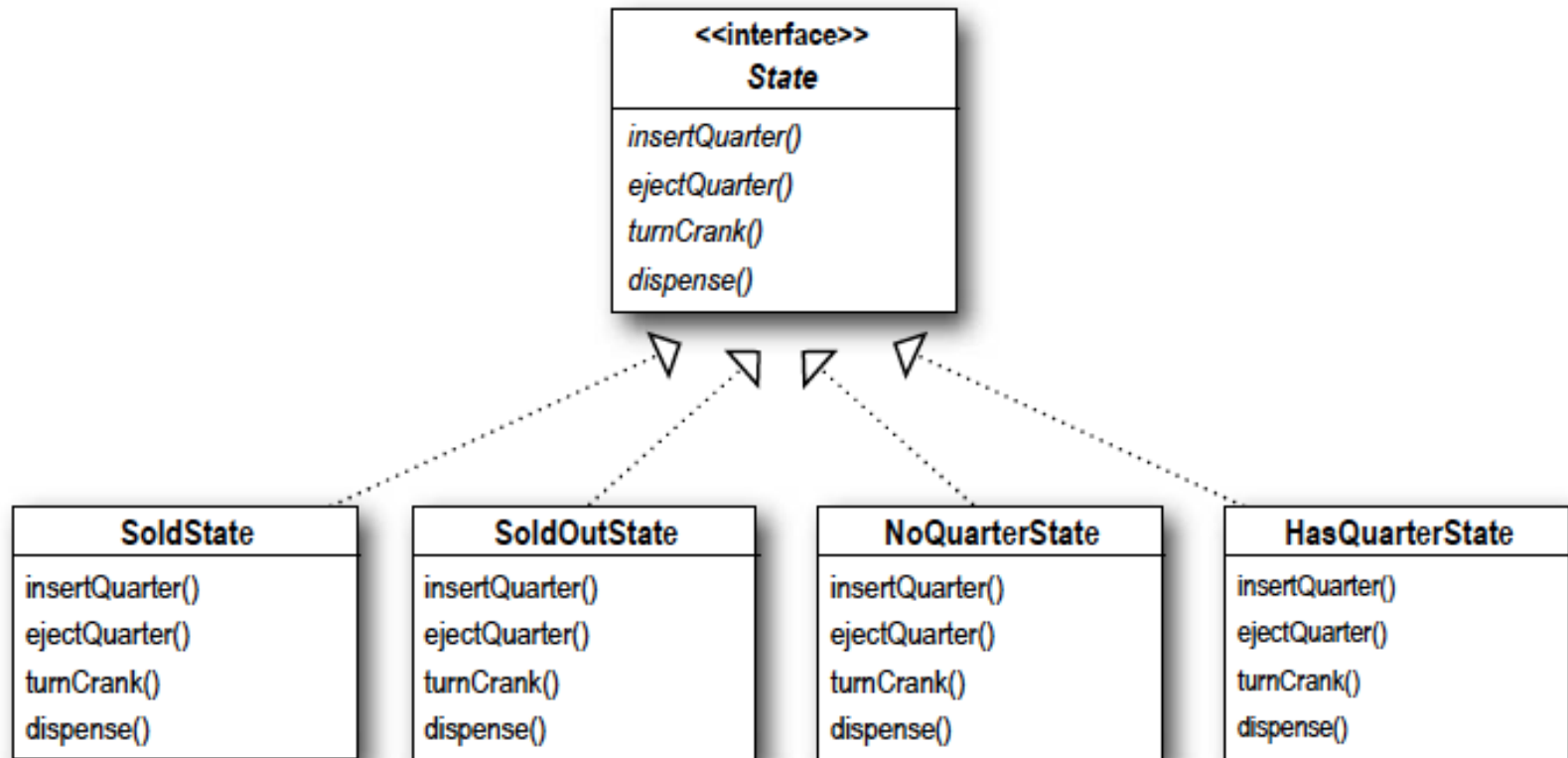
상태 변화가 캡슐화 안됨

새로운 디자인: 상태 변화를 캡슐화

- 각 상태는 별도의 클래스에서 담당 (캡슐화)
- Gumball 기계의 현재 상태에 따라, 해당 상태를 담당하는 클래스에 모든 작업 처리를 위임

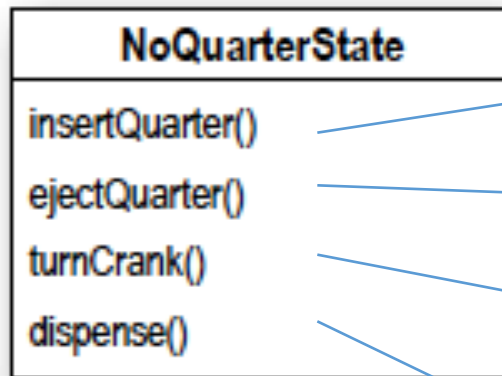


State 인터페이스 및 클래스 정의



상태 클래스 구현

- 각 메소드가 호출되었을 때 무엇을 해야 할까요?



HasQuarterState로 이동

"You haven't inserted a quarter"
메시지 출력

"You turned but there's no quarter"
메시지 출력

"You need to pay first" 메시지 출력

NoQuarterState 구현

```
public class NoQuarterState implements State {
    GumballMachine gumballMachine;

    public NoQuarterState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    public void insertQuarter() {
        System.out.println("You inserted a quarter");
        gumballMachine.setState(gumballMachine.getHasQuarterState());
    }

    public void ejectQuarter() {
        System.out.println("You haven't inserted a quarter");
    }

    public void turnCrank() {
        System.out.println("You turned, but there's no quarter");
    }

    public void dispense() {
        System.out.println("You need to pay first");
    }

    ...
}
```

GumBallMachine 클래스 구현

```
public class GumballMachine {  
    State soldOutState;  
    State noQuarterState;  
    State hasQuarterState;  
    State soldState;  
  
    State state;  
    int count = 0;  
  
    public GumballMachine(int numberGumballs) {  
        soldOutState = new SoldOutState(this);  
        noQuarterState = new NoQuarterState(this);  
        hasQuarterState = new HasQuarterState(this);  
        soldState = new SoldState(this);  
  
        this.count = numberGumballs;  
        if (numberGumballs > 0)  
            state = noQuarterState;  
        else  
            state = soldOutState;  
    }  
}
```

GumBallMachine 클래스 구현

```
public class GumBallMachine {  
    State state;  
    ...  
    public void insertQuarter() { state.insertQuarter(); }  
  
    public void ejectQuarter() { state.ejectQuarter(); }  
  
    public void turnCrank() {  
        state.turnCrank();  
        state.dispense();  
    }  
    void setState(State state) { this.state = state; }  
  
    void releaseBall() { // SoldState의 dispense() 메소드에서 호출  
        System.out.println("A gumball comes rolling out the slot...");  
        if (count != 0) {  
            count = count - 1;  
        }  
    }  
    ...  
}
```

Gumball 기계의 수정

```
public class GumballMachine {  
  
    final static int SOLD_OUT = 0;  
    final static int NO_QUARTER = 1;  
    final static int HAS_QUARTER = 2;  
    final static int SOLD = 3;  
  
    int state = SOLD_OUT;  
    int count = 0;  
}
```

정적 정수 변수 부분을
새로 만든 상태 클래스로
대체

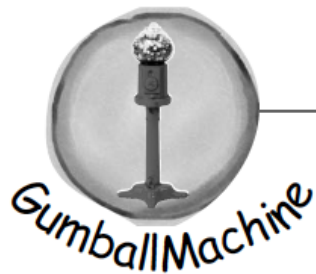
이전 코드

```
public class GumballMachine {  
  
    State soldOutState;  
    State noQuarterState;  
    State hasQuarterState;  
    State soldState;  
  
    State state = soldOutState;  
    int count = 0;  
}
```

새로운 코드

정리

Gumball 기계에는
각 상태 클래스의 인
스턴스가 있음



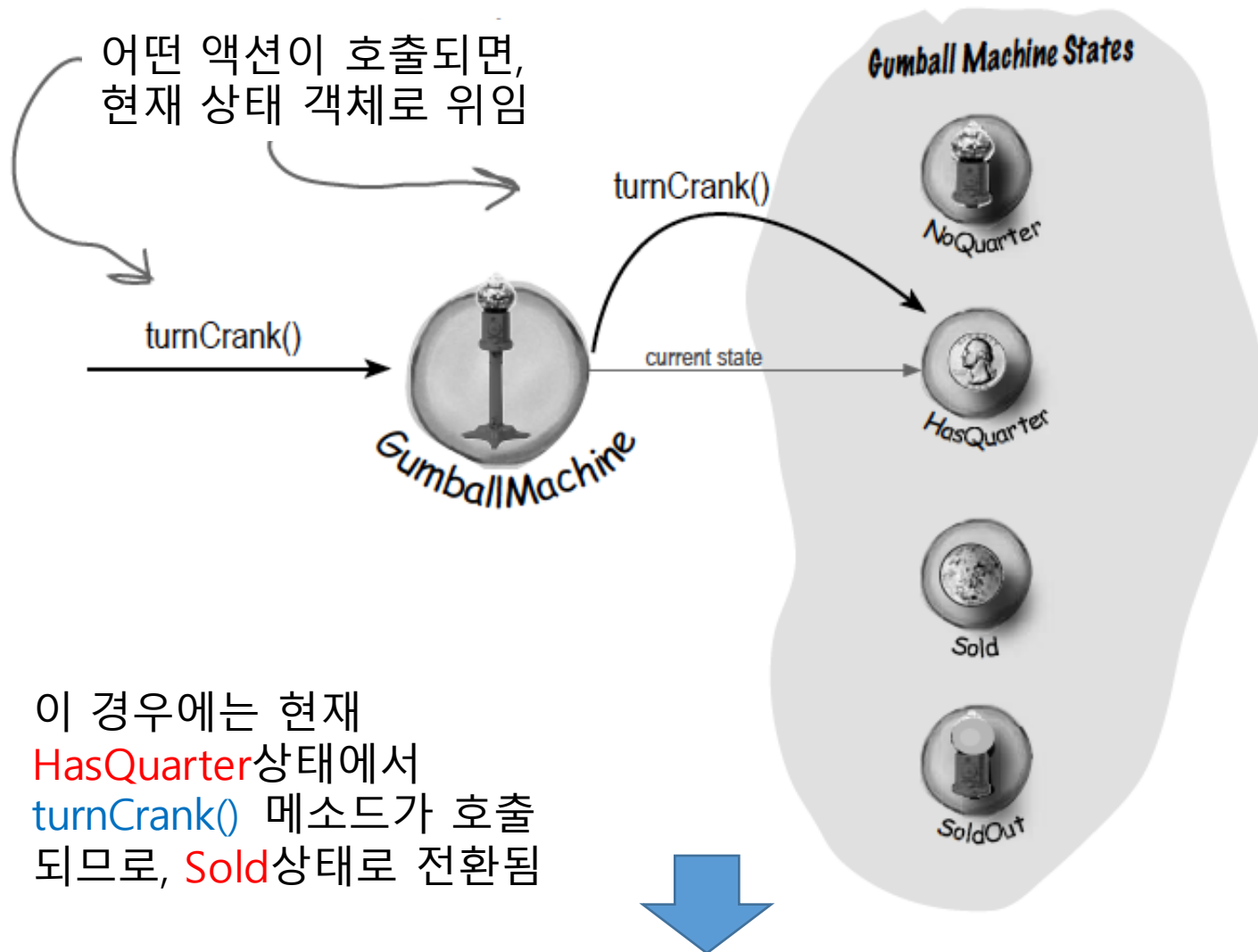
current state

Gumball Machine States



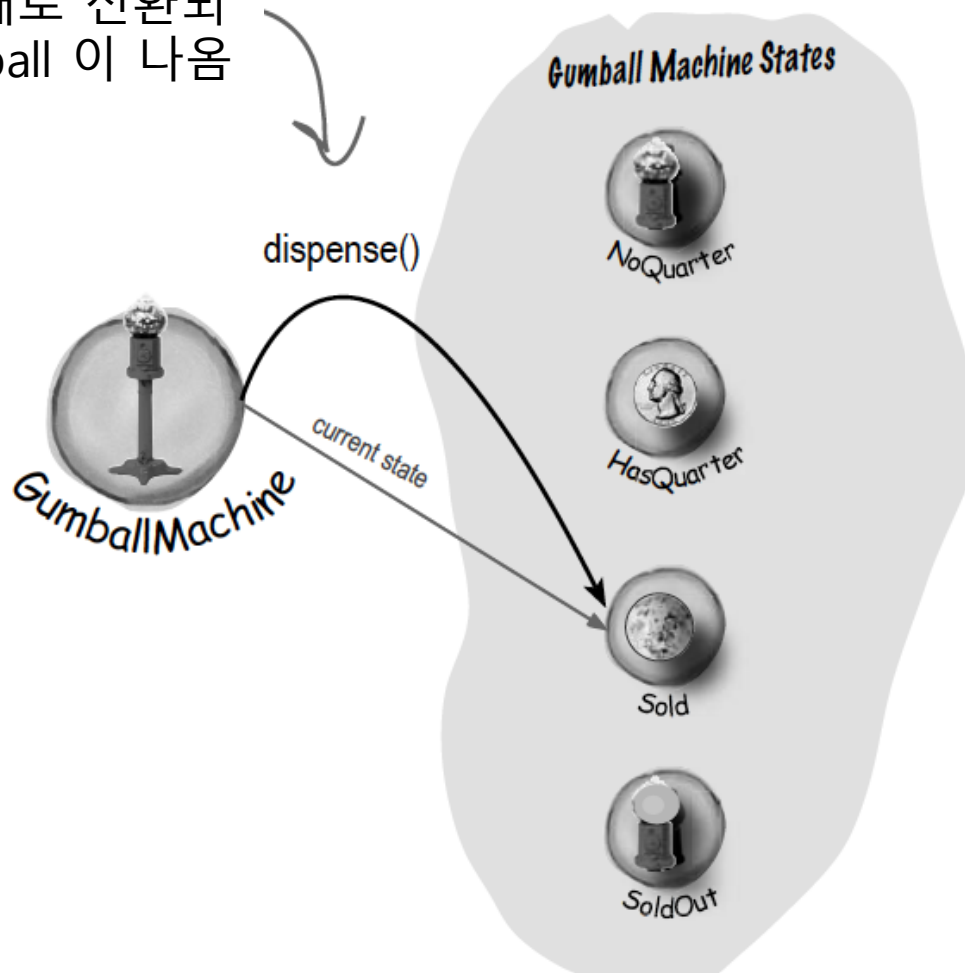
Gumball 기계의 현재
상태는 항상 이 인스
턴스 중에 하나임

정리



정리

Sold 상태로 전환되고 Gumball 이 나옴



more gumballs

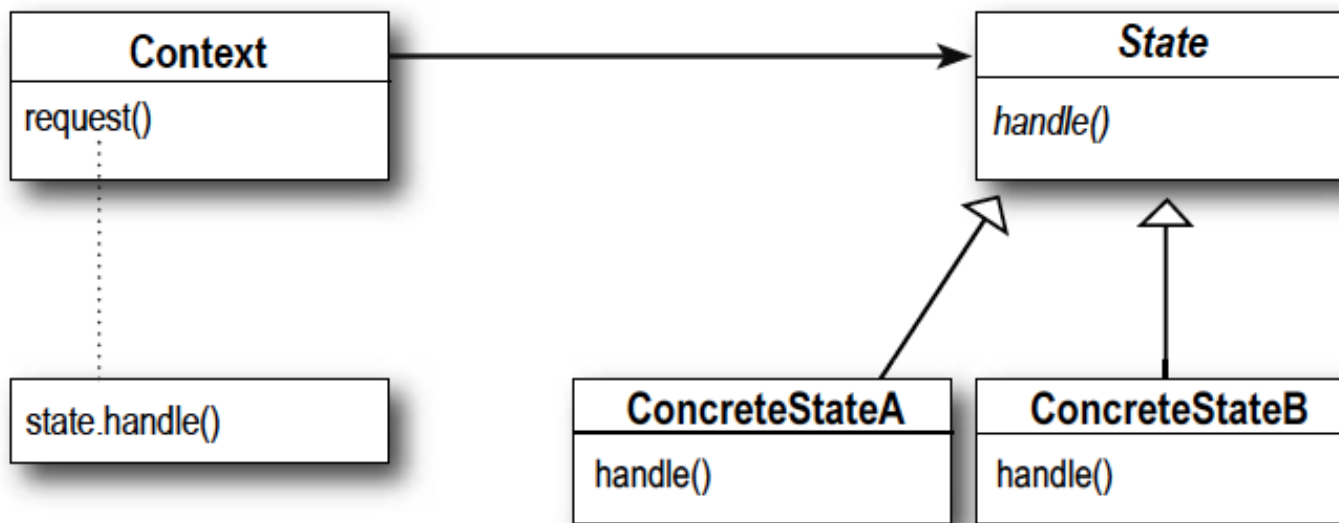
그리고 나서
Gumball 기계의 상태가 **SoldOut** 또는 **NoQuarter** 상태로 전환 됩니다.
어떤 상태로 전환 되는 지는 남아 있는 Gumball 개수에 따라 결정 됩니다

sold out

스태이트 패턴

- 정의

- 상태 기계의 각 상태를 State 추상 클래스의 서브 클래스로 구현
- 상태 기계의 상태 전이는 현재 상태를 나타내는 State 추상 클래스 타입의 객체 메소드를 호출하는 것으로 구현됨



Context의 request() 메소드의 호출은 상태 객체로 위임됨

각 상태를 나타내는 ConcreteState 객체는 Context로부터 전달된 요청을 처리

공짜 Gumball 당첨 기능을 구현

```
public class GumballMachine {
```

```
    State soldOutState;  
    State noQuarterState;  
    State hasQuarterState;  
    State soldState;
```

```
    State winnerState;
```

WinnerState를 추가하고
생성자내에서 초기화

```
    State state = soldOutState;  
    int count = 0;
```

```
    public GumballMachine(int numberGumballs) {
```

```
        ...
```

```
        winnerState = new WinnerState(this);
```

```
        ...
```

```
    }
```

공짜 Gumball 당첨 기능을 구현

```
public class WinnerState implements State {
    GumballMachine gumballMachine;
    ...
    public void insertQuarter() {.. }
    public void ejectQuarter() {.. }
    public void turnCrank() {.. }
    public void dispense() {
        gumballMachine.releaseBall();
        if (gumballMachine.getCount() == 0) {
            gumballMachine.setState(gumballMachine.getSoldOutState());
        } else {
            gumballMachine.releaseBall();           //추가 Gumball 제공
            System.out.println("YOU'RE A WINNER! ...");
            if (gumballMachine.getCount() > 0) {
                gumballMachine.setState(gumballMachine.getNoQuarterState());
            } else {
                System.out.println("Oops, out of gumballs!");
                gumballMachine.setState(gumballMachine.getSoldOutState());
            }
        }
    }
    ...
}
```

공짜 Gumball 당첨 기능을 구현

```
public class HasQuarterState implements State {  
    Random randomWinner = new Random(System.currentTimeMillis());  
    GumballMachine gumballMachine;  
  
    ...  
    public void insertQuarter() {... }  
  
    public void ejectQuarter() {... }  
  
    public void turnCrank() {  
        System.out.println("You turned...");  
        int winner = randomWinner.nextInt(10);  
        if ((winner == 0) && (gumballMachine.getCount() > 1)) {  
            gumballMachine.setState(gumballMachine.getWinnerState());  
        } else {  
            gumballMachine.setState(gumballMachine.getSoldState());  
        }  
    }  
  
    public void dispense() {... }  
}
```

테스트해보죠..

```
public class GumballMachineTestDrive {  
  
    public static void main(String[] args) {  
        GumballMachine gumballMachine =  
            new GumballMachine(10);  
  
        System.out.println(gumballMachine);  
  
        gumballMachine.insertQuarter();  
        gumballMachine.turnCrank();  
        gumballMachine.insertQuarter();  
        gumballMachine.turnCrank();  
  
        System.out.println(gumballMachine);  
  
        gumballMachine.insertQuarter();  
        gumballMachine.turnCrank();  
        gumballMachine.insertQuarter();  
        gumballMachine.turnCrank();  
  
        ...  
    }  
}
```

질문

- WinnerState가 꼭 있어야 하나요? 그냥 SoldState에서 Gumball을 두개 내보내도록 하면 안되나요?

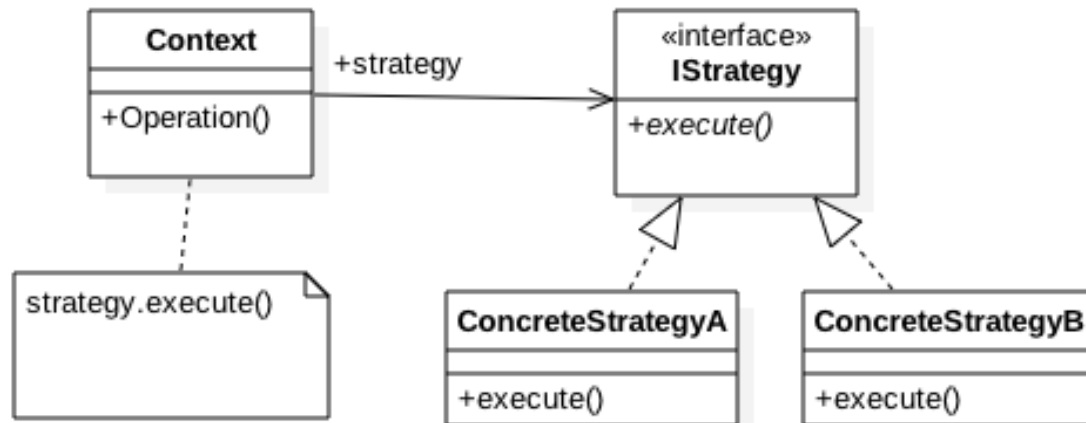
https://github.com/kwanulee/PatternExample/blob/master/state/gumballstatewinner_alternative/src/hansung/designpatterns/state/gumballstatewinner/SoldState.java#L26-L39

예제 프로젝트

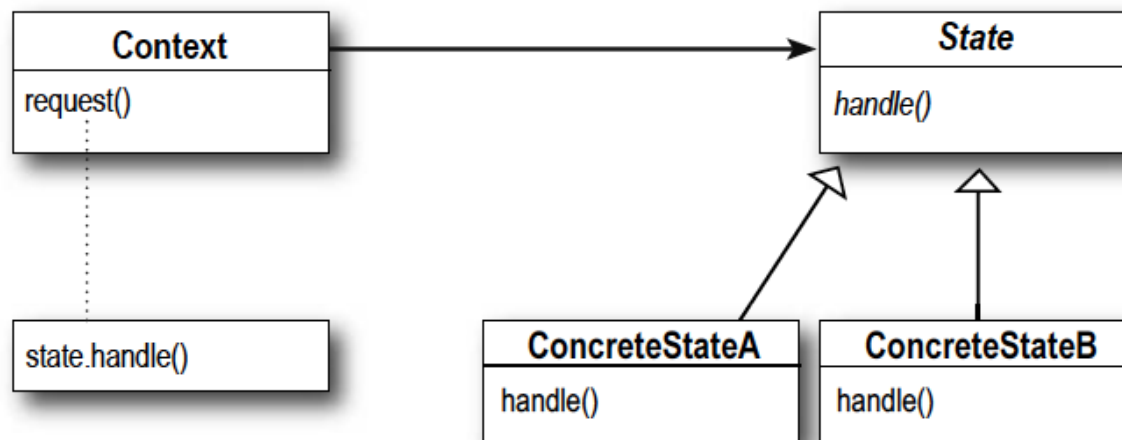
https://github.com/kwanulee/PatternExample/tree/master/state/gumballstatewinner_alternative

스트래티지 패턴과의 비교

스트래티지
패턴



스테이트
패턴



스트래티지 패턴과의 비교

• 스트래티지 패턴 목적

- Context의 특정 오퍼레이션이 다양한 알고리즘 (ConcreteStrategy 객체) 으로 교체될 필요가 있는 경우에 사용됨
 - 예, 오리의 나는 알고리즘 교체

• 스트래티지 패턴 동작 방식

- Context에 특정 ConcreteStrategy 객체를 설정
- 클라이언트로부터 Context의 operation() 메소드가 호출될 때, **설정된 ConcreteStrategy 객체의 execute() 메소드 호출**

• 스테이트 패턴 목적

- Context의 내부 상태에 따라 동일한 액션을 다른 방식으로 처리해야 할 때 사용됨
 - 예, turnCrank 동작이 Gumball 기계의 상태에 따라 다른 방식으로 처리됨

• 스테이트 패턴 동작 방식

- Context에 관련 ConcreteState 객체를 설정.
- 클라이언트로부터 Context의 request() 메소드가 호출될 때, **현재 상태의 ConcreteState 객체의 handle() 메소드 호출**

핵심 정리

- 스테이트 패턴을 이용하면 상태 기계의 각 상태를 클래스를 이용하여 표현
- Context 객체에서는 현재 상태 객체에게 액션 요청을 위임함
- 각 상태를 클래스로 캡슐화함으로써 나중에 변경시켜야 하는 내용을 국지화 시킬 수 있음
- 스테이트 패턴과 스트래티지 패턴을 구조는 동일하지만, 그 용도는 서로 다름