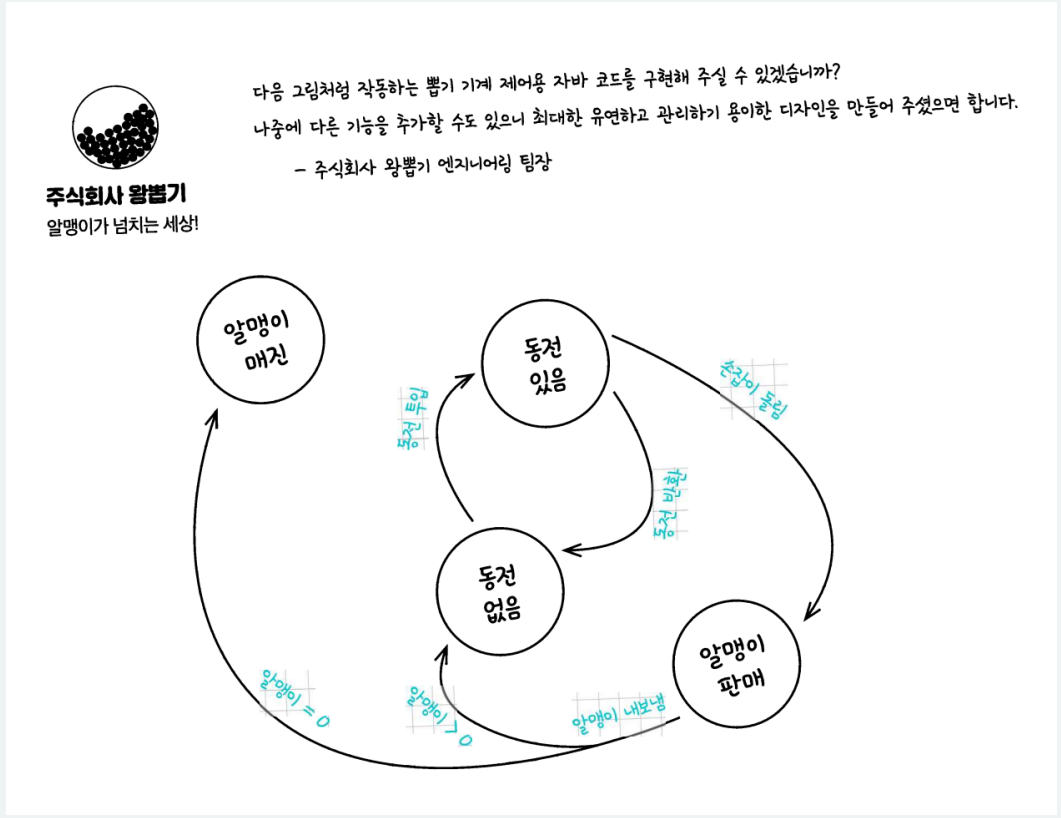
헤드퍼스트 디자인패턴

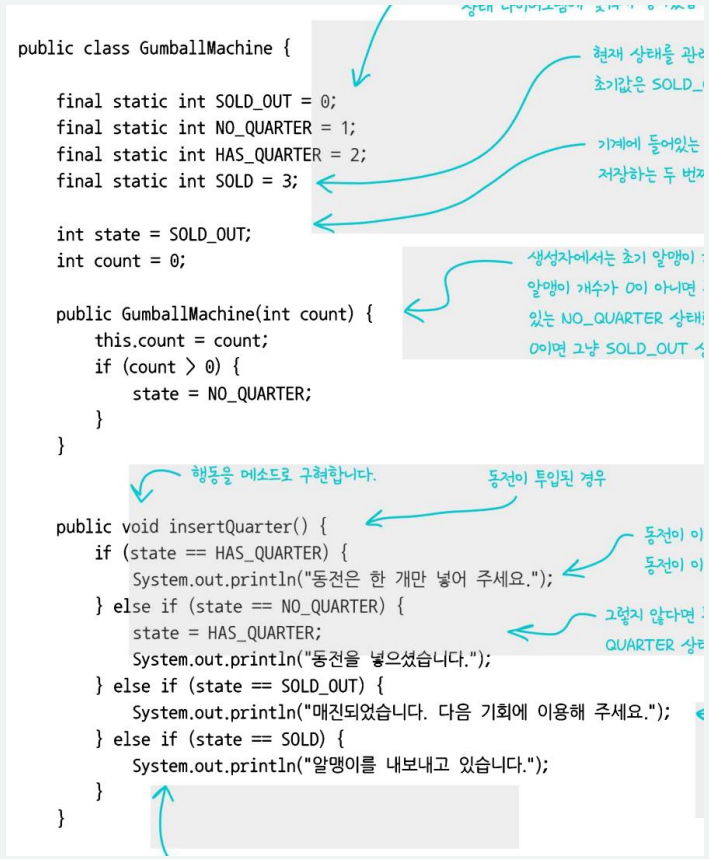
상태 패턴

프로그램 요구사항

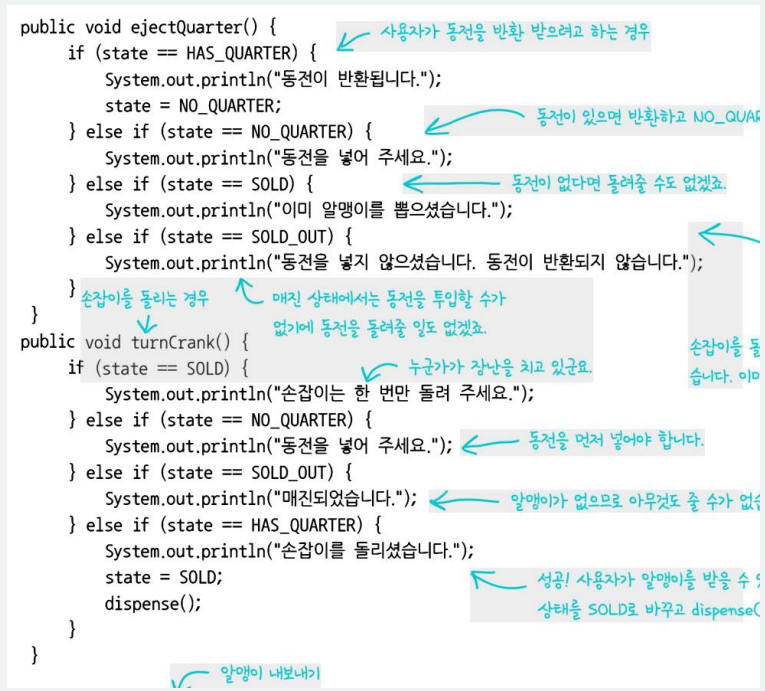


상태 값 리스트	행동 값 리스트
Sold Out	동전 투입
No Quarter	동전 반환
Has Quarter	손잡이 돌림
Sold	알맹이 내보냄

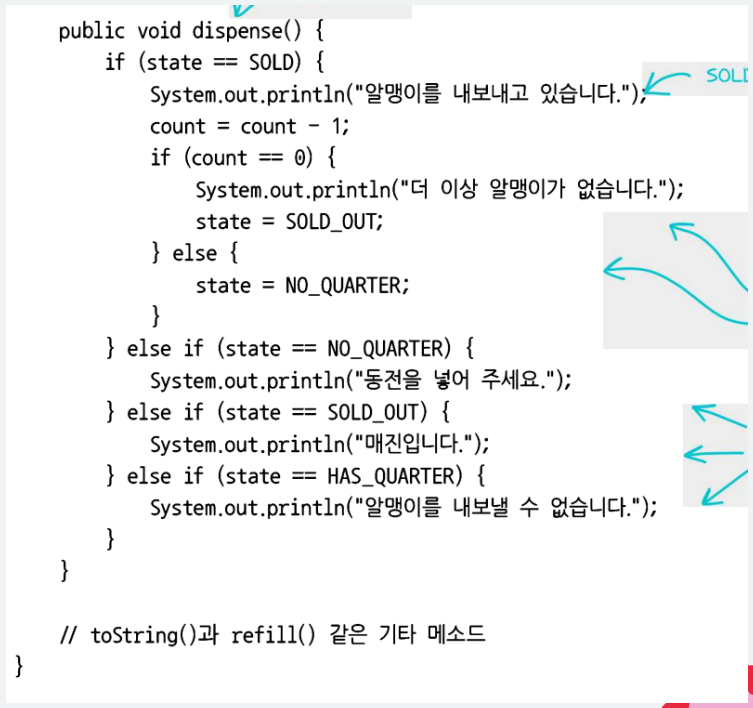
평소 코딩 방식



```
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;

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

    public void insertQuarter() {
        if (state == HAS_QUARTER) {
            System.out.println("동전은 한 개만 넣어 주세요.");
        } else if (state == NO_QUARTER) {
            state = HAS_QUARTER;
            System.out.println("동전을 넣으셨습니다.");
        } else if (state == SOLD_OUT) {
            System.out.println("매진되었습니다. 다음 기회에 이용해 주세요.");
        } else if (state == SOLD) {
            System.out.println("알맹이를 내보내고 있습니다.");
        }
    }
```



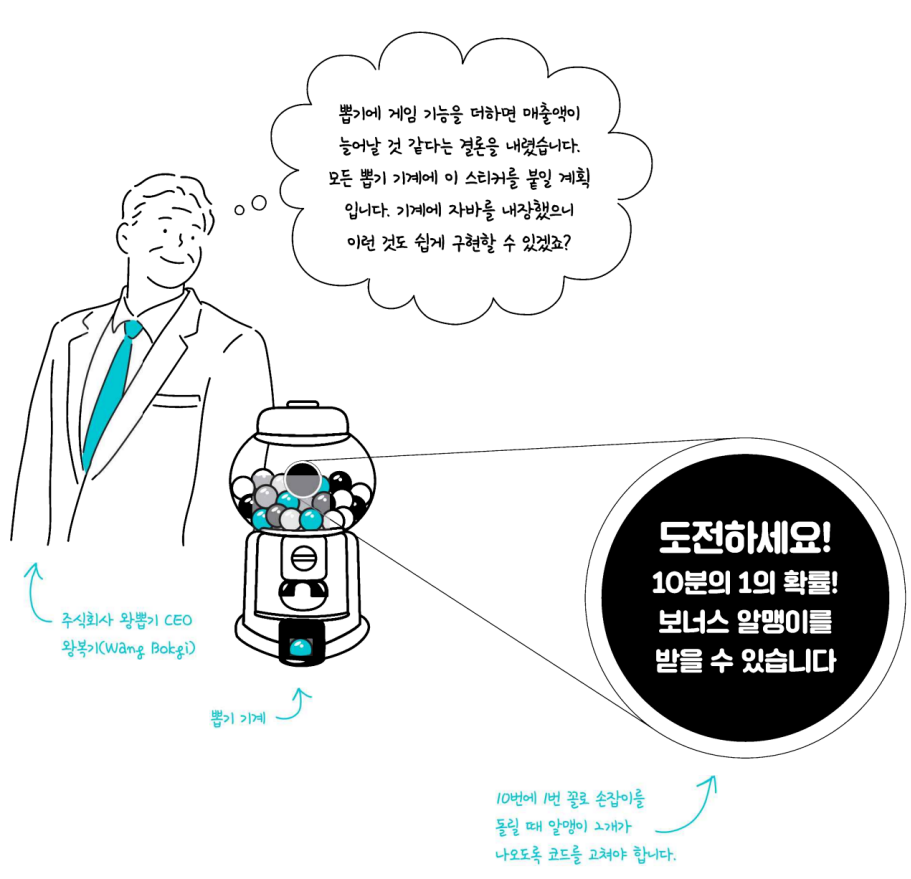
```
public void ejectQuarter() {
    if (state == HAS_QUARTER) {
        System.out.println("동전이 반환됩니다.");
        state = NO_QUARTER;
    } else if (state == NO_QUARTER) {
        System.out.println("동전을 넣어 주세요.");
    } else if (state == SOLD) {
        System.out.println("이미 알맹이를 뽑으셨습니다.");
    } else if (state == SOLD_OUT) {
        System.out.println("동전을 넣지 않으셨습니다. 동전이 반환되지 않습니다.");
    }
}
public void turnCrank() {
    if (state == SOLD) {
        System.out.println("손잡이는 한 번만 돌려 주세요.");
    } else if (state == NO_QUARTER) {
        System.out.println("동전을 넣어 주세요.");
    } else if (state == SOLD_OUT) {
        System.out.println("매진되었습니다.");
    } else if (state == HAS_QUARTER) {
        System.out.println("손잡이를 돌리셨습니다.");
        state = SOLD;
        dispense();
    }
}
```



```
    public void dispense() {
        if (state == SOLD) {
            System.out.println("알맹이를 내보내고 있습니다.");
            count = count - 1;
            if (count == 0) {
                System.out.println("더 이상 알맹이가 없습니다.");
                state = SOLD_OUT;
            } else {
                state = NO_QUARTER;
            }
        } else if (state == NO_QUARTER) {
            System.out.println("동전을 넣어 주세요.");
        } else if (state == SOLD_OUT) {
            System.out.println("매진입니다.");
        } else if (state == HAS_QUARTER) {
            System.out.println("알맹이를 내보낼 수 없습니다.");
        }
    }

    // toString()과 refill() 같은 기타 메소드
}
```

갬블머신이라는 메인 클래스에 각 행동 메소드에서 상태에 대한 값들을 처리하는 형태로 작성

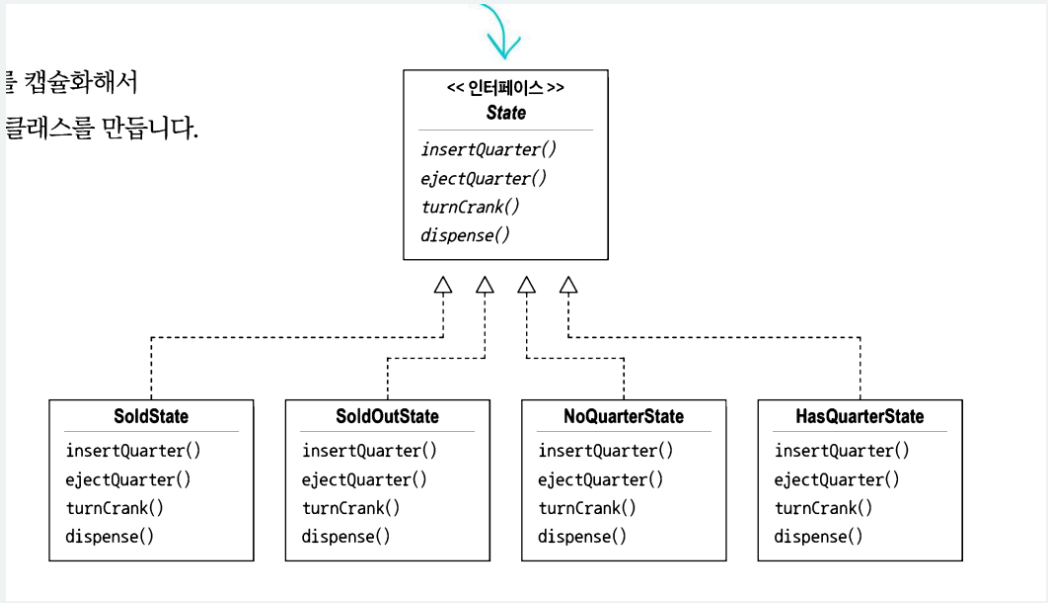
??: 이거에 추가 기능 넣는 것은 어렵지 않지?
나: (ㅅㅂ 줫같네) 네 가능하죠



위에서 나름대로 괜찮은 코드를 만들었다고 생각했지만, 코드를 확장하는 일은 쉽지 않습니다.

이대로 가다가는 새로운 기능을 추가해 달라는 요청을 받을 때 마다 모든 코드를 수정해야 하며 버그가 많아질 것 같습니다.

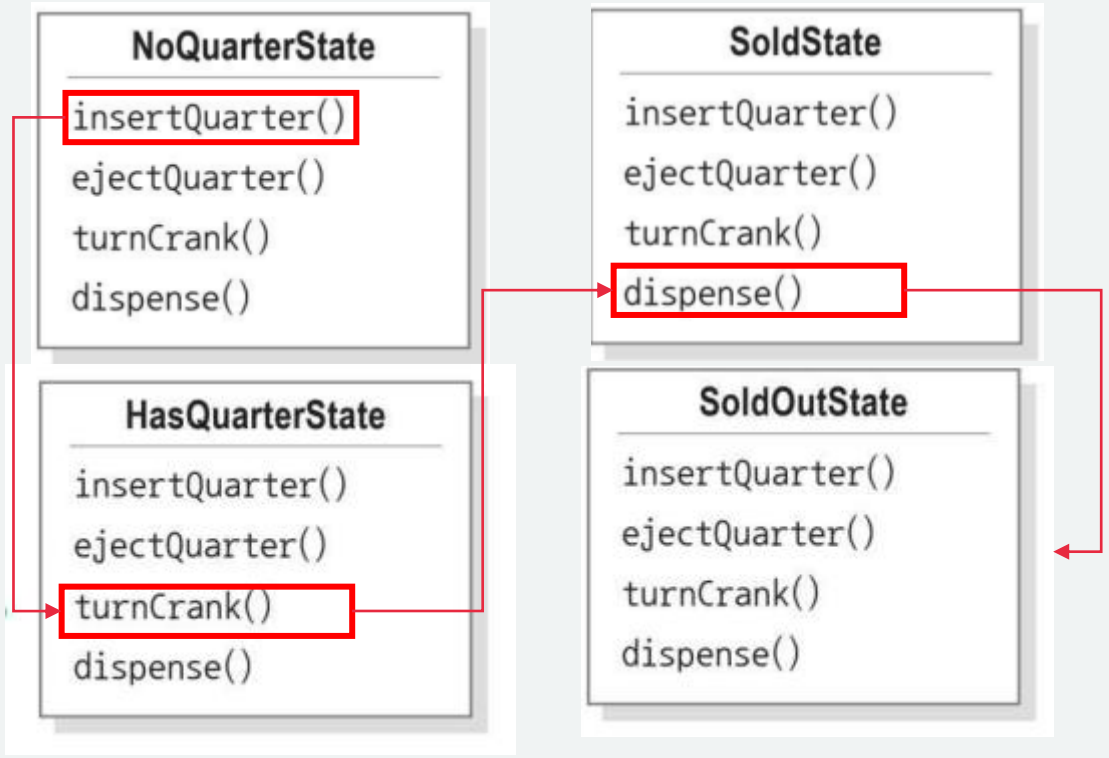
디자인 패턴 도입하기



State 인터페이스를 구현합니다.

모든 상태들을 캡슐화 하여 State 인터페이스를 구현하는 클래스를 구현합니다.

상태 별 행동을 생각해봅시다.



NoQuarterState::
 insertQuarter: HasQuarterState로 전환
 ejectQuarter: 동전을 넣어달라는 메시지 출력

HasQuarterState::
 turnCrank: SoldState 전환

SoldState::
 insertQuarter: 알맹이를 보내고 있다는 메시지
 dispense: 알맹이 개수를 확인하고 0이하면 Soldoutstate로 전환 아니면 NoQuarterState

```
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;
        }
    }

    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() {
        System.out.println("알맹이를 내보내고 있습니다.");
        if (count > 0) {
            count = count - 1;
        }
    }
    // 상태 객체를 위한 게터 메소드 등의 기타 메소드...
}
```

모든 상태 객체를 선언합니다.

상태 인스턴스 변수(State 형식)

count 인스턴스 변수에는 알맹이 개수가 저장됩니다.
처음에는 비어 있으니까 0으로 설정합니다.

생성자 알맹이의 초기 개수를
인자로 받아서 인스턴스 변수에 저장합니다.

그리고 State 인스턴스도 각각 하나씩 생성합니다.

알맹이 개수가 0개보다 많으면 state를 NoQuarterState로 설정합니다.

메소드를 구현하는 부분.
현재 상태가 작업을 처리하게 만듭니다.
구현하기 정말 쉽죠.

dispense() 메소드를 구현하지 않아도 됩니다.
내부에서만 필요한 행동이니까요. 사용자가 직접 기계에 알맹이를
내놓으라고 요청할 수는 없습니다. 물론 상태 객체는
turnCrank() 메소드에서 dispense() 메소드를 호출하기도 합니다.

이 메소드를 사용하면 이 안에 들어있는 State 객체를 비롯한
다른 객체에서 뽑기 기계의 상태를 다른 상태로 전환할 수 있습니다.

알맹이를 내보내고 count 인스턴스 변수의 값을 1 줄이는
보조 메소드인 releaseBall() 메소드를 지원합니다.

각 상태 객체를 가져오는 getNoQuarterState()와 남아 있는 알맹이 개수를
구하는 getCount() 메소드 등이 여기에 들어갈 수 있습니다.

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

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

    public void insertQuarter() {
        System.out.println("동전을 넣으셨습니다.");
        gumballMachine.setState(gumballMachine.getHasQuarterState());
    }

    public void ejectQuarter() {
        System.out.println("동전을 넣어 주세요.");
    }

    public void turnCrank() {
        System.out.println("동전을 넣어 주세요.");
    }

    public void dispense() {
        System.out.println("동전을 넣어 주세요.");
    }
}
```

누군가가 동전을 넣으면 동전이 투입되었다는 메시지를 출력하고 기계의 상태를 HasQuarterState로 전환합니다.

이 부분은 잠시 후에 설명하겠습니다.

동전을 넣지도 않고 돌려 달라고 하면 안 되겠죠?

동전을 넣지 않고 알맹이를 달라고 해서도 안 되죠.

돈이 들어오기 전에는 알맹이를 내줄 수는 없습니다.

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

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

    public void insertQuarter() {
        System.out.println("동전은 한 개만 넣어 주세요.");
    }

    public void ejectQuarter() {
        System.out.println("동전이 반환됩니다.");
        gumballMachine.setState(gumballMachine.getNoQuarterState());
    }

    public void turnCrank() {
        System.out.println("손잡이를 돌리셨습니다.");
        gumballMachine.setState(gumballMachine.getSoldState());
    }

    public void dispense() {
        System.out.println("알맹이를 내보낼 수 없습니다.");
    }
}
```

상태 인스턴스를 만들 때는 GumballMachine의 레퍼런스를 전달합니다. 나중에 다른 상태로 전환할 때 이 레퍼런스가 필요하죠.

이 상태에서는 부적절한 메소드

동전을 돌려주고 NoQuarterState로 전환합니다.

사용자가 손잡이를 돌리면 setState() 메소드를 호출하고 SoldState 객체를 인자로 전달해서 뽑기 기계를 다른 상태로 전환합니다. SoldState 객체는 GumballMachine의 getSoldState() 게터 메소드(각 상태 객체마다 게터 메소드를 만들어야 합니다)를 써서 구할 수 있습니다.

이 메소드도 이 상태에서는 부적절합니다.

```
public class SoldState implements State {
    //인스턴스 변수 및 생성자

    public void insertQuarter() {
        System.out.println("알맹이를 내보내고 있습니다.");
    }

    public void ejectQuarter() {
        System.out.println("이미 알맹이를 뽑으셨습니다.");
    }

    public void turnCrank() {
        System.out.println("손잡이는 한 번만 돌려 주세요.");
    }

    public void dispense() {
        gumballMachine.releaseBall();
        if (gumballMachine.getCount() > 0) {
            gumballMachine.setState(gumballMachine.getNoQuarterState());
        } else {
            System.out.println("Oops, out of gumballs!");
            gumballMachine.setState(gumballMachine.getSoldOutState());
        }
    }
}
```

이 메소드들은 전부

중요한 부분

사용자가 동전을 넣고 손잡이를 돌렸을 때만 이 상태가 될 수 따라서 일단 뽑기 기계에서

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

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

    public void insertQuarter() {
        System.out.println("죄송합니다. 매진되었습니다.");
    }

    public void ejectQuarter() {
        System.out.println("동전을 반환할 수 없습니다. 동전을 넣지 않았습니다.");
    }

    public void turnCrank() {
        System.out.println("죄송합니다. 매진되었습니다.");
    }

    public void dispense() {
        System.out.println("알맹이를 내보낼 수 없습니다.");
    }
}
```

SoldOutStat
알맹이를 새로
할 수가 없습니

10% 확률 당첨 기능 추가 하기

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

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

    public void insertQuarter() {
        System.out.println("동전은 한 개만 넣어 주세요.");
    }

    public void ejectQuarter() {
        System.out.println("동전이 반환됩니다.");
        gumballMachine.setState(gumballMachine.getNoQuarterState());
    }

    public void turnCrank() {
        System.out.println("손잡이를 돌리셨습니다.");
        gumballMachine.setState(gumballMachine.getSoldState());
    }

    public void dispense() {
        System.out.println("알맹이를 내보낼 수 없습니다.");
    }
}
```

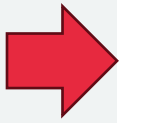
상태 인스턴스를 만들 때는 GumballMachine의 레퍼런스를 전달합니다. 나중에 다른 상태로 전환할 때 이 레퍼런스가 필요하죠.

이 상태에서는 부적절한 메소드

동전을 돌려주고 NoQuarterState로 전환합니다.

사용자가 손잡이를 돌리면 setState() 메소드를 호출하고 SoldState 객체를 인자로 전달해서 뽑기 기계를 다른 상태로 전환합니다. SoldState 객체는 GumballMachine의 getSoldState() 게터 메소드(각 상태 객체마다 게터 메소드를 만들어야 합니다)를 써서 구할 수 있습니다.

이 메소드도 이 상태에서는 부적절합니다.



```
public class HasQuarterState implements State {
    Random randomWinner = new Random(System.currentTimeMillis());
    GumballMachine gumballMachine;

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

    public void insertQuarter() {
        System.out.println("동전은 한 개만 넣어 주세요.");
    }

    public void ejectQuarter() {
        System.out.println("동전이 반환됩니다.");
        gumballMachine.setState(gumballMachine.getNoQuarterState());
    }

    public void turnCrank() {
        System.out.println("손잡이를 돌리셨습니다.");
        int winner = randomWinner.nextInt(10);
        if ((winner == 0) && (gumballMachine.getCount() > 1)) {
            gumballMachine.setState(gumballMachine.getWinnerState());
        } else {
            gumballMachine.setState(gumballMachine.getSoldState());
        }
    }

    public void dispense() {
        System.out.println("알맹이를 내보낼 수 없습니다.");
    }
}
```

우선 10% 확률로 당첨 여부를 결정하는 난수 발생기를 추가합니다.

당첨되었고 남아있

WinnerState로

만족되지 않으면 S

```
public class WinnerState implements State {

    // 인스턴스 변수 및 생성자
    // insertQuarter 오류 메시지
    // ejectQuarter 오류 메시지
    // turnCrank 오류 메시지

    public void dispense() {
        gumballMachine.releaseBall();
        if (gumballMachine.getCount() == 0) {
            gumballMachine.setState(gumballMachine.getSoldOutState());
        } else {
            gumballMachine.releaseBall();
            System.out.println("축하드립니다! 알맹이를 하나 더 받으실 수 있습니다.");
            if (gumballMachine.getCount() > 0) {
                gumballMachine.setState(gumballMachine.getNoQuarterState());
            } else {
                System.out.println("더 이상 알맹이가 없습니다.");
                gumballMachine.setState(gumballMachine.getSoldOutState());
            }
        }
    }
}
```

SoldState와 똑같습니다.

알맹이를 그냥 내보내고 NoQuarterState 또는 SoldO

알맹이가 하나 더 있으면 내보냅니다.

데모 버전 돌려보기

```
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.turnCrank();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

        System.out.println(gumballMachine);
    }
}
```

이번에도 알맹이를 5개만 넣어 두고 뽑기 기계를 돌립니다.

당첨되는 걸 봐야 하니까 동전을 넣고
손잡이를 돌리는 과정을 여러 번 반복합니다.
뽑기 기계의 상태도 종종 출력해 줘야겠죠.

```
주식회사 왕뽑기
자바로 돌아가는 최신형 뽑기 기계
남은 개수: 5개
동전 투입 대기중

동전을 넣으셨습니다.
손잡이를 돌리셨습니다.
축하드립니다! 알맹이를 하나 더 받으실 수 있습니다.
알맹이를 내보내고 있습니다.
알맹이를 내보내고 있습니다.

주식회사 왕뽑기
자바로 돌아가는 최신형 뽑기 기계
남은 개수: 3개
동전 투입 대기중

동전을 넣으셨습니다.
손잡이를 돌리셨습니다.
알맹이를 내보내고 있습니다.
동전을 넣으셨습니다.
손잡이를 돌리셨습니다.
축하드립니다! 알맹이를 하나 더 받으실 수 있습니다.
알맹이를 내보내고 있습니다.
알맹이를 내보내고 있습니다.
매진입니다.

주식회사 왕뽑기
자바로 돌아가는 최신형 뽑기 기계
남은 개수: 0개
매진
```