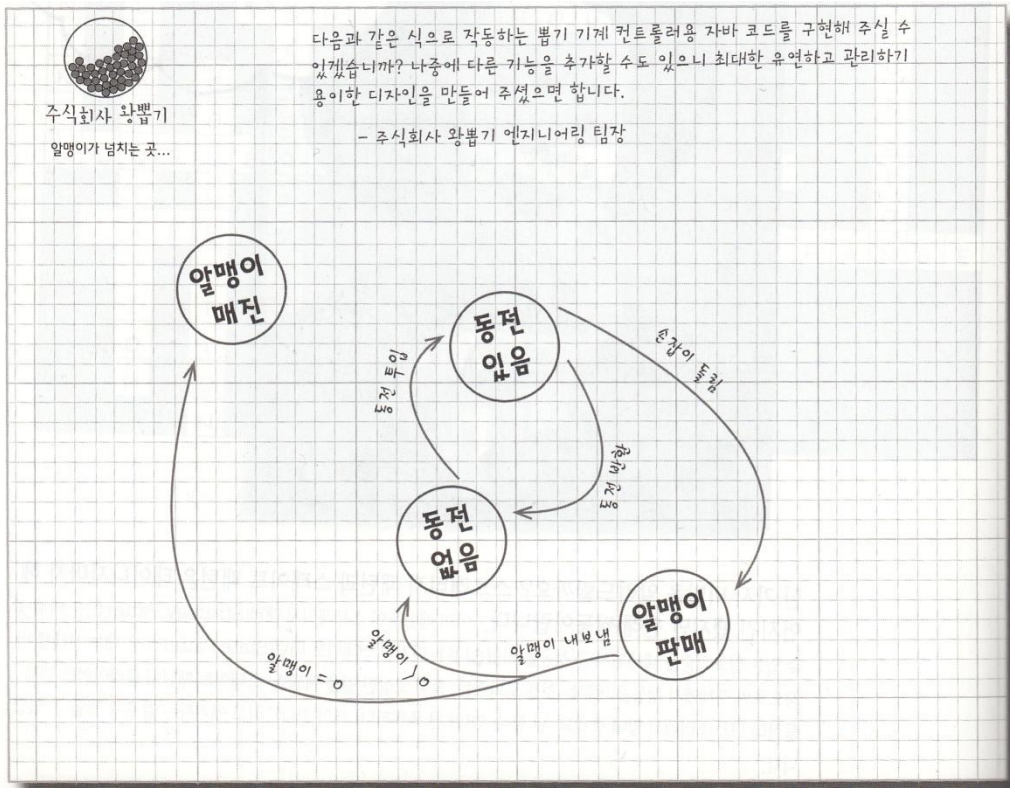
10. STATE PATTERN

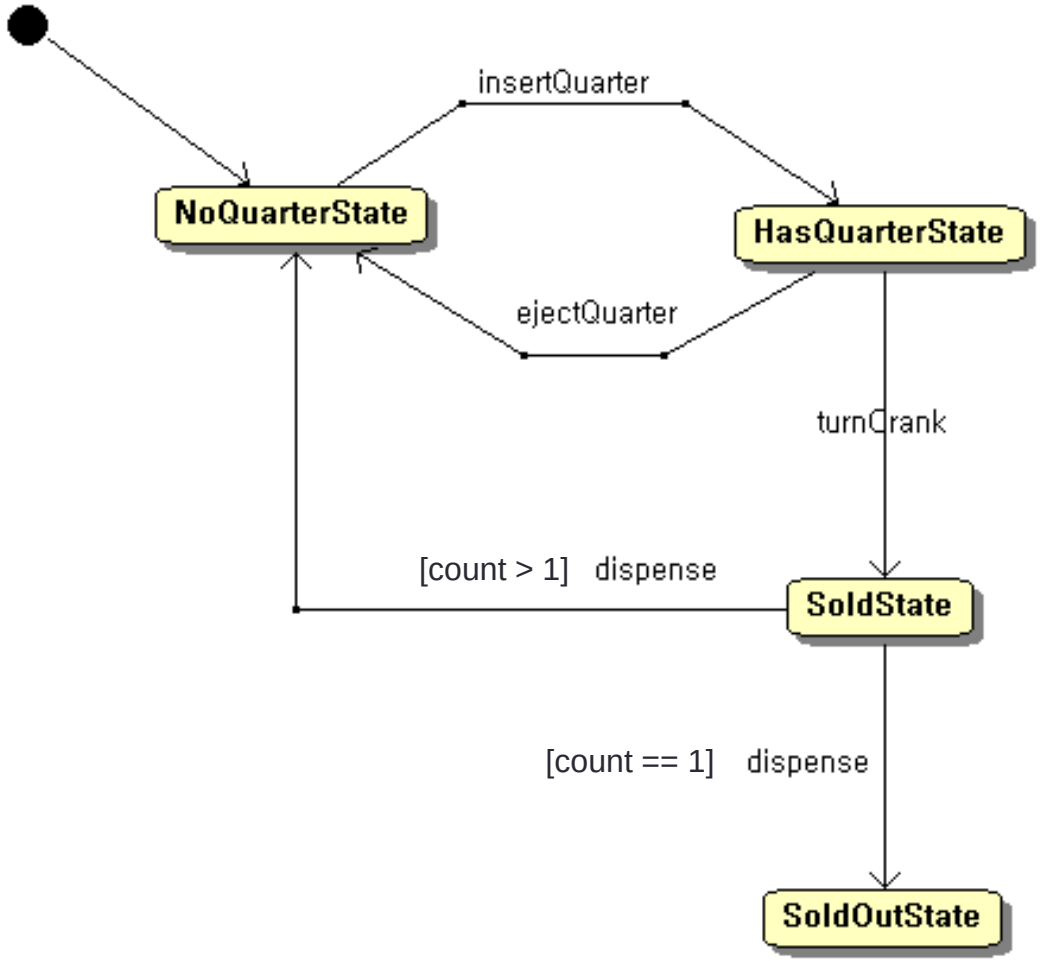
동의대학교 컴퓨터소프트웨어공학과

Gumball Machine

- 세계 최초 자바 기반 껌 판매기!!!



상태 기계 다이어그램 (State Machine Diagram)



상태 기계의 기초

- 현재 상태를 표현하기 위한 인스턴스 변수 (attribute)
  - final static int SOLD_OUT = 0;
  - final static int NO_QUARTER = 1;
  - final static int HAS_QUARTER = 2;
  - final static int SOLD = 3;

- 둥그런 껌 판매기에서 가능한 행동 (operation)
  - 동전 투입 : insertQuarter()
  - 동전 반환 : ejectQuarter()
  - 손잡이 돌림 : turnCrank()
  - 알맹이 내보냄 : dispense()

참고. enum 자료형

```
enum State {
    SOLD_OUT(0), NO_QUARTER(1),
    HAS_QUARTER(2), SOLD(3);

    State(int value) {this.value = value};
    private final int value;
    public int value() {return value;}
}

// 사용법
public class GumballMachine {
    final static enum state = State.SOLD_OUT;
    ……
```

GumballMachine 구현

```
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("You can't insert another quarter");
        } else if (state == NO_QUARTER) {
            state = HAS_QUARTER;
            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");
        }
    }
```

이 상태만 의미 있음.

```
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");
    }
}

public void turnCrank() {
    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();
    }
}
```

```java
public void dispense() {
    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;
        }
    } else if (state == NO_QUARTER) {
        System.out.println("You need to pay first");
    } else if (state == SOLD_OUT) {
        System.out.println("No gumball dispensed");
    } else if (state == HAS_QUARTER) {
        System.out.println("No gumball dispensed");
    }
}
```

```java
// 상태 기계 관점의 소스 코드
public void dispense() {
    if (state == SOLD) {
        if (count > 1) {  // guard (or condition)
            count = count - 1; // release a gumball
            state = NO_QUARTER;
        } else {  // count == 1
            count = count - 1; // release a gumball
            state == SOLD_OUT;
        }
    } else if (state == NO_QUARTER) {
    // 이하 생략
}
```

StringBuffer 대신 StringBuilder 사용!

```
    public String toString() {
        StringBuffer result = new StringBuffer();
        result.append("\nMighty Gumball, Inc.");
        result.append("\nJava-enabled Standing Gumball Model #2004\n");
        result.append("Inventory: " + count + " gumball");
        if (count != 1) {
            result.append("s");
        }
        result.append("\nMachine is ");
        if (state == SOLD_OUT) {
            result.append("sold out");
        } else if (state == NO_QUARTER) {
            result.append("waiting for quarter");
        } else if (state == HAS_QUARTER) {
            result.append("waiting for turn of crank");
        } else if (state == SOLD) {
            result.append("delivering a gumball");
        }
        result.append("\n");
        return result.toString();
    }
}
```

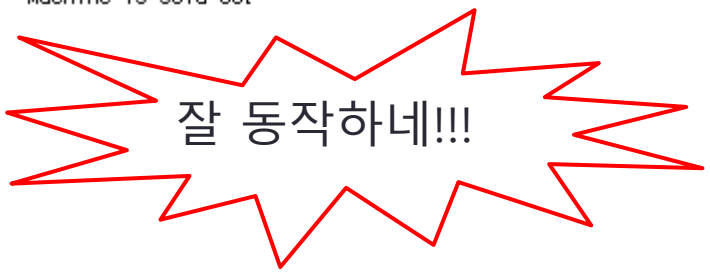
테스트 코드

```
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);

        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        gumballMachine.ejectQuarter();

        System.out.println(gumballMachine);

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

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

실행 결과

```
Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 5 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
Quarter returned
You turned but there's no quarter

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot
You inserted a quarter
You turned...
A gumball comes rolling out the slot
You haven't inserted a quarter
```

```
Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 2 gumballs
Machine is waiting for quarter

You inserted a quarter
You can't insert another quarter
You turned...
A gumball comes rolling out the slot
You inserted a quarter
You turned...
A gumball comes rolling out the slot
Oops, out of gumballs!
You can't insert a quarter, the machine is sold out
You turned, but there are no gumballs

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 0 gumballs
Machine is sold out
```



기능을 추가하려면?

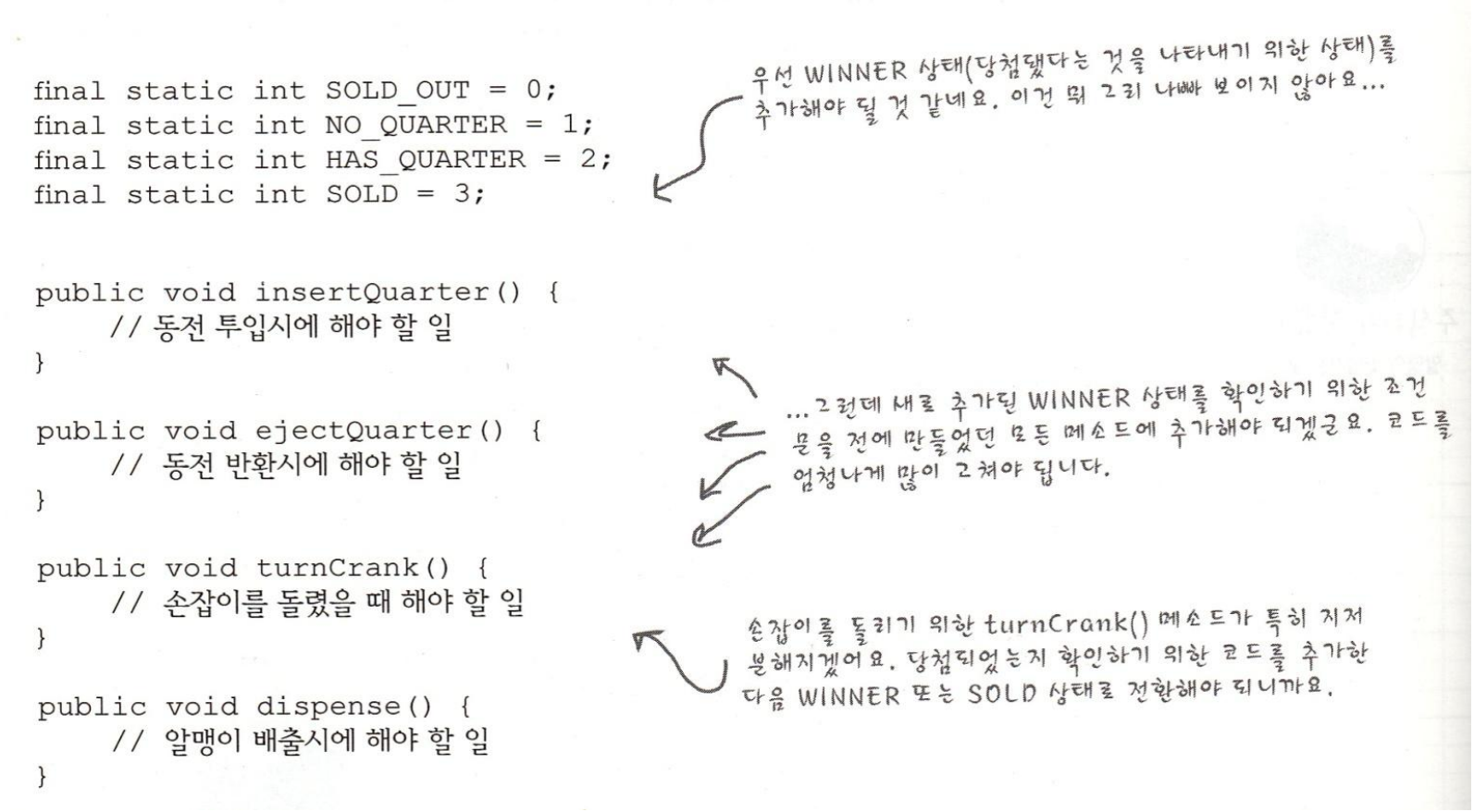
- 고객의 현재 생각
  - "지금 구현한 기계에 게임 기능을 추가하면 어떨까?"
  - "10분의 1의 확률로 1+1으로 Gumball을 주면 고객이 좋아 할 텐데. 물론 매출도 오르겠지!!!"
  - "자바로 구현했으니 쉽게 고칠 수 있겠지?"

- 개발자의 생각
  - 가능한 적게 수정하면서 쉽게 할 수 있을까?
  - 새 기능에서 요구되는 상태(WinnerState)를 어떻게 반영할 것인가?
  - 기존 코드를 검토해 봐야 어떨지 알 수 있을 것 같은데...

기존 코드 검토

- 지저분한 상태
  - "이대로는 안 될 것 같아. 이 코드 이용해서 수정하려면 고생할 것 같은데..."
  - "그래, 결심했어. 좀더 유연한 구조로 다시 코딩 해야지."
    → REFACTORING!!!

```
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() {
    // 동전 투입시에 해야 할 일
}

public void ejectQuarter() {
    // 동전 반환시에 해야 할 일
}

public void turnCrank() {
    // 손잡이를 돌렸을 때 해야 할 일
}

public void dispense() {
    // 알맹이 배출시에 해야 할 일
}
```

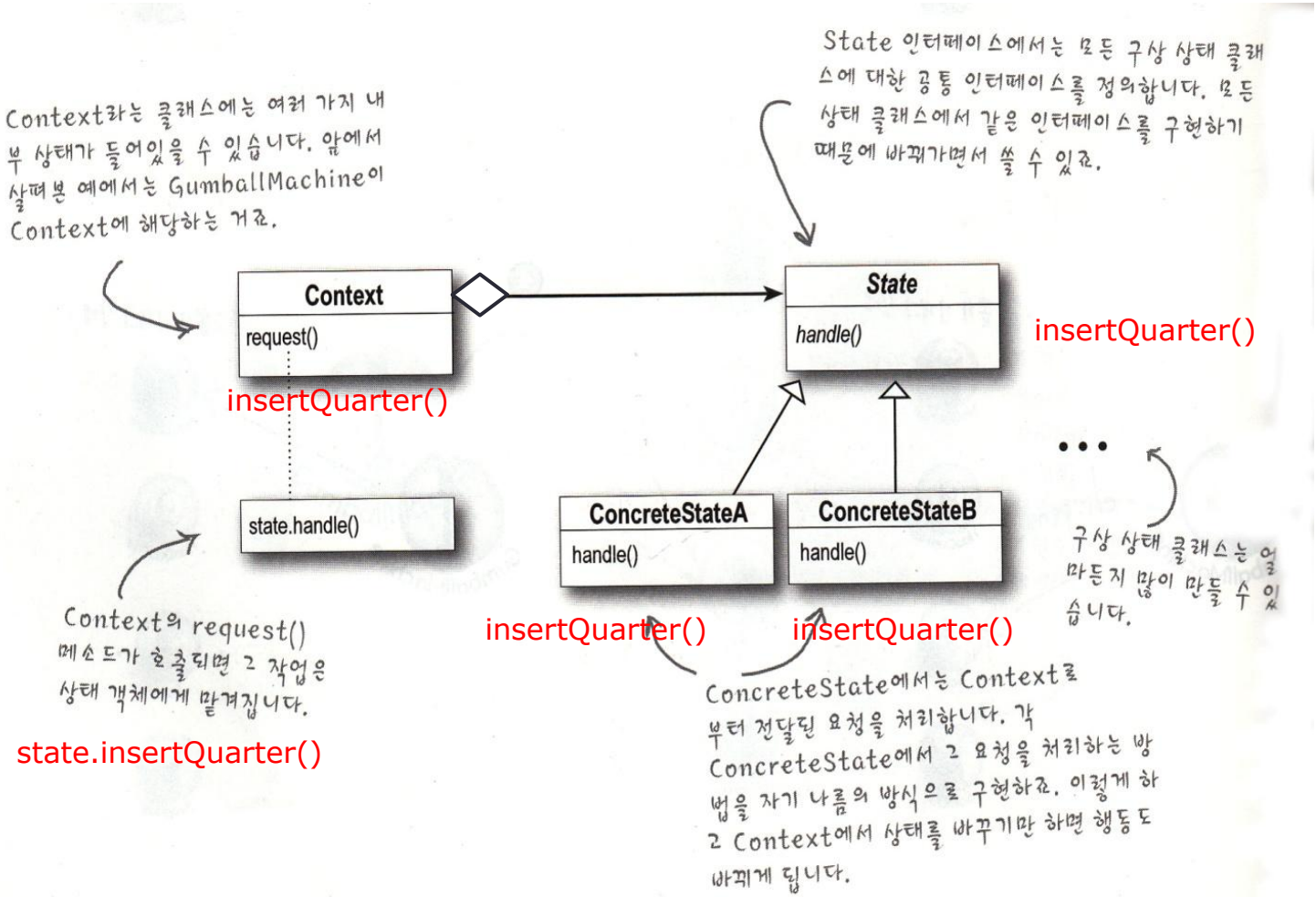


State Pattern을 활용하자.

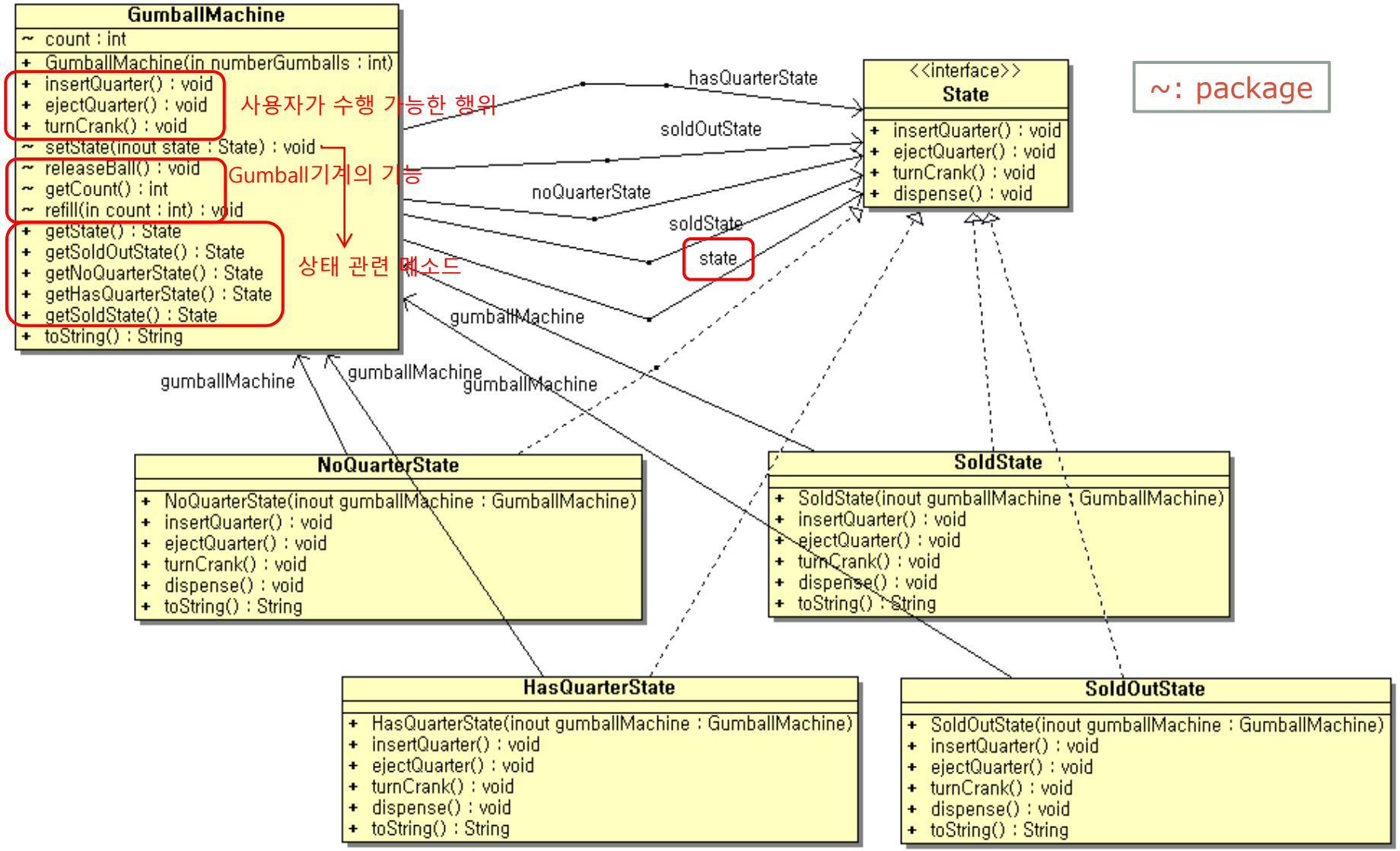
- 문제점
  - 객체의 행위는 상태에 따라 다르고, 메소드는 조건에 따라 상태에 의존적인 동작을 반영하는 case 로직을 포함한다. 조건부 로직에 대한 다른 대안은 있는가?
- 해결책
  - 각 상태를 위해 state 클래스를 생성하고, 각 클래스는 공통 인터페이스를 구현하라. ***상태 의존적인 오퍼레이션을 context object에서 현재 state 객체로 위임***하라. ***Context object가 항상 현재 상태를 반영하는 state 객체를 가리키도록*** 확실히 하라.

- State Pattern을 사용하면 ***객체의 내부 상태가 바뀜에 따라서 객체의 행동을 바꿀 수 있습니다***. 마치 객체의 클래스가 바뀌는 것과 같은 결과를 얻을 수 있습니다.

- Q: 1장에서 학습한 Strategy Pattern과는 어떤 차이가 있을 까요?
  → 교재 pp.448-449 (1판: pp.456-457) 참조


Context라는 클래스에는 여러 가지 내부 상태가 들어있을 수 있습니다. 앞에서 살펴본 예에서는 GumballMachine이 Context에 해당하는 거죠.
State 인터페이스에서는 모든 구상 상태 클래스에 대한 공통 인터페이스를 정의합니다. 모든 상태 클래스에서 같은 인터페이스를 구현하기 때문에 바꿔가면서 쓸 수 있죠.
Context
request()
State
handle()
insertQuarter()
insertQuarter()
state.handle()
ConcreteStateA
handle()
ConcreteStateB
handle()
...
구상 상태 클래스는 얼마든지 많이 만들 수 있습니다.
Context의 request() 메소드가 호출되면 그 작업은 상태 객체에게 맡겨집니다.
state.insertQuarter()
insertQuarter()
insertQuarter()
ConcreteState에서는 Context로부터 전달된 요청을 처리합니다. 각 ConcreteState에서 그 요청을 처리하는 방법을 자기 나름의 방식으로 구현하죠. 이렇게 하고 Context에서 상태를 바꾸기만 하면 행동도 바뀌게 됩니다.

GumballMachineState 구현



State 관련

```java
public interface State {

    public void insertQuarter();

    public void ejectQuarter();

    public void turnCrank();

    public void dispense();
}
```

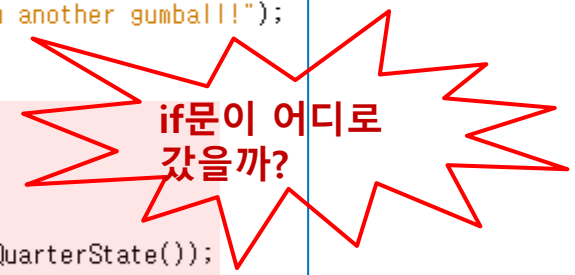
```java
public class NoQuarterState implements State {

    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");
    }

    public String toString() {
        return "waiting for quarter";
    }
}
```

상태 전환 이루어짐!

if문이 어디로 갔을까?

```java
public class HasQuarterState implements State {

    GumballMachine gumballMachine;

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

    public void insertQuarter() {
        System.out.println("You can't insert another quarter");
    }

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

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

    public void dispense() {
        System.out.println("No gumball dispensed");
    }

    public String toString() {
        return "waiting for turn of crank";
    }
}
```

```java
public class SoldState implements State {

    GumballMachine gumballMachine;

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

    public void insertQuarter() {
        System.out.println("Please wait, we're already giving you a gumball");
    }

    public void ejectQuarter() {
        System.out.println("Sorry, you already turned the crank");
    }

    public void turnCrank() {
        System.out.println("Turning twice doesn't get you another gumball!");
    }

    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());
        }
    }

    @Override
    public String toString() {
        return "dispensing a gumball";
    }
}
```



```java
public class SoldOutState implements State {

    GumballMachine gumballMachine;

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

    public void insertQuarter() {
        System.out.println("You can't insert a quarter, the machine is sold out");
    }

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

    public void turnCrank() {
        System.out.println("You turned, but there are no gumballs");
    }

    public void dispense() {
        System.out.println("No gumball dispensed");
    }

    @Override
    public String toString() {
        return "sold out";
    }
}
```

```
public class GumballMachine {

    State soldOutState;
    State noQuarterState;
    State hasQuarterState;
    State soldState;
    State state = soldOutState;
    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;
        }
    }

    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("A gumball comes rolling out the slot...");
        if (count != 0) {
            count = count - 1;
        }
    }

    int getCount() {
        return count;
    }

    void refill(int count) {
        this.count = count;
        state = noQuarterState;
    }

    public State getState() {
        return state;
    }

    public State getSoldOutState() {
        return soldOutState;
    }

    public State getNoQuarterState() {
        return noQuarterState;
    }
```

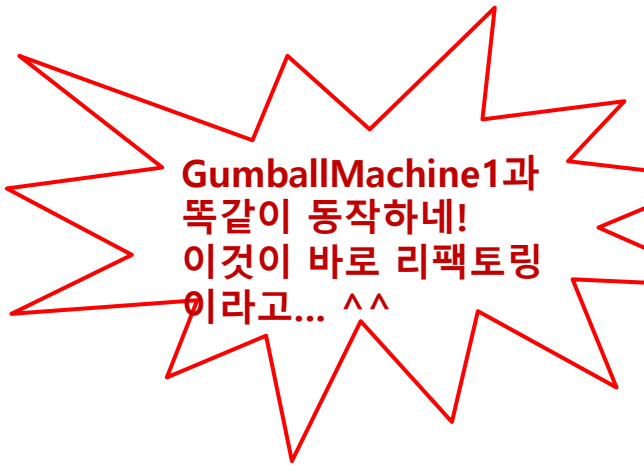
this.count += count; // 더 현실적

```
    public State getHasQuarterState() {
        return hasQuarterState;
    }

    public State getSoldState() {
        return soldState;
    }

    @Override
    public String toString() {
        StringBuilder result = new StringBuilder();
        result.append("\nMighty Gumball, Inc.");
        result.append("\nJava-enabled Standing Gumball Model #2004");
        result.append("\nInventory: " + count + " gumball");
        if (count != 1) {
            result.append("s");
        }
        result.append("\n");
        result.append("Machine is " + state + "\n");
        return result.toString();
    }
}
```

테스트 코드

```
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);
    }
}
```

실행 결과

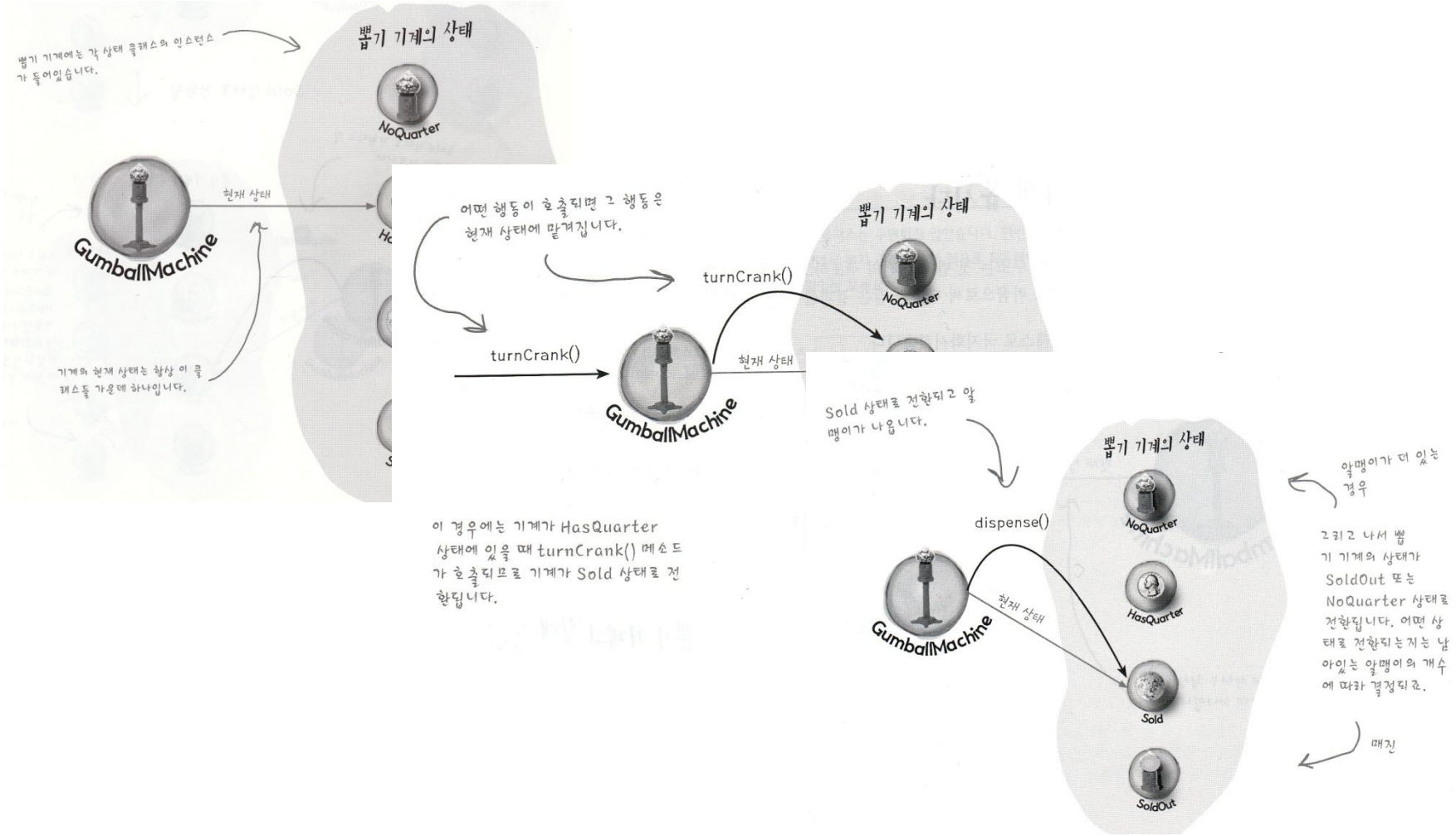



```
Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 5 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...
You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 2 gumballs
Machine is waiting for quarter
```

지금까지 한 일을 정리해 봅시다.

- GumballMachine1은 지저분한 구조 !!!
- 방금 구현한 GumballMachine2는
  - 각 상태의 행동을 별개의 클래스로 국지화시켰습니다.
  - 관리하기 힘든 골칫덩어리 if 선언문을 없앴습니다.
  - 각 상태를 변경에 대해서는 닫혀 있도록 하면서도 GumballMachine 자체는 새로운 상태 클래스를 추가하는 확장에 대해서 열려 있도록 고쳤습니다. (OCP)
  - 처음에 주식회사 왕뽑기에서 제시했던 다이어그램에 훨씬 가까우면서 더 이해하기 좋은 코드 베이스와 클래스 구조를 만들었습니다.

GumballMachine의 상태 변화

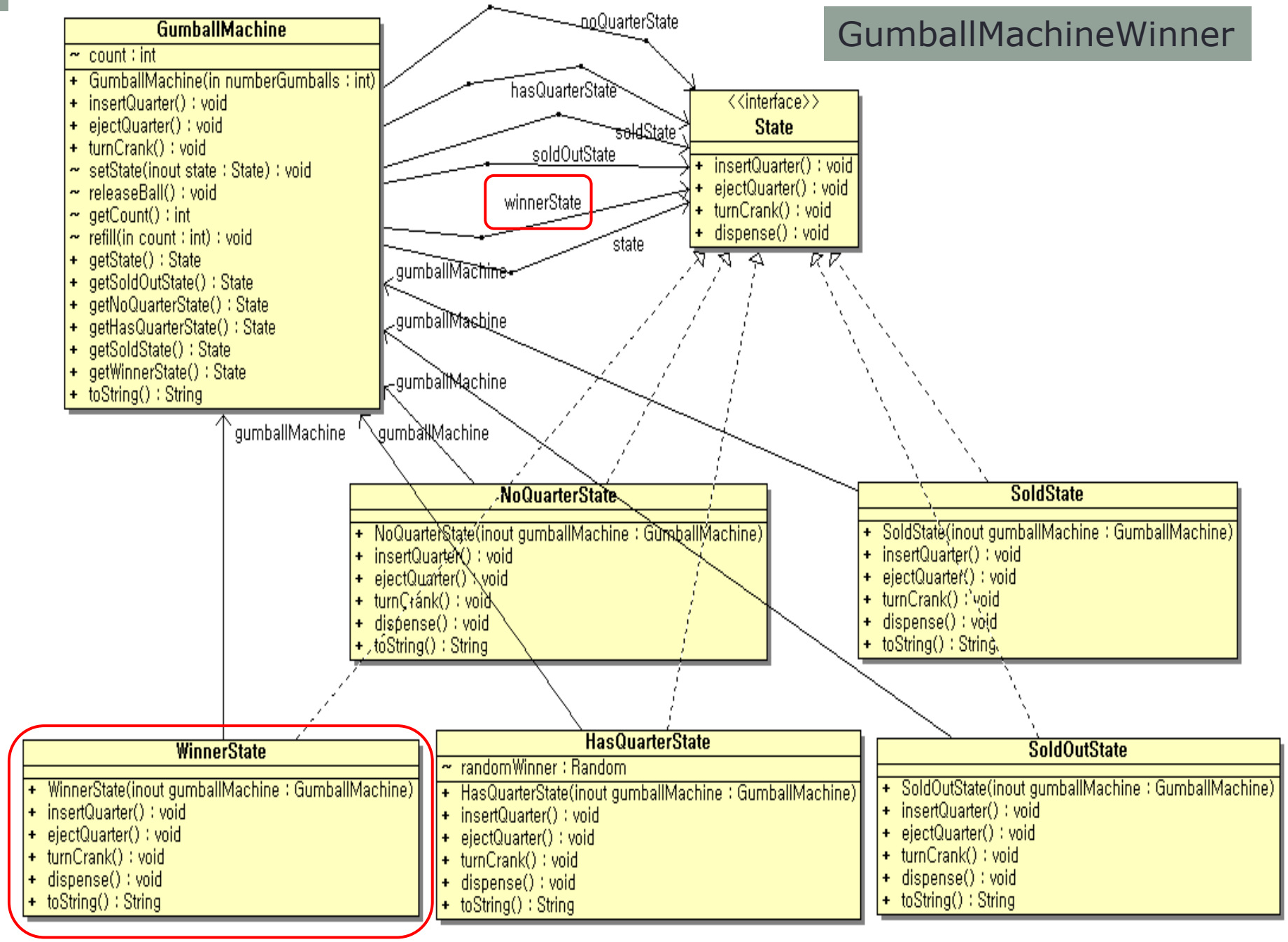


이제 무엇을 해야 하나?

- 뽑기 기계에 공짜 알맹이 당첨 기능을 추가해 봅시다.
  →"1+1"은 즐거워!!!

- 수정되는 부분
  - WinnerState 클래스 (State 인터페이스 구현) 추가
  - GumballMachine → State 연관 추가 : winnerState
  - WinnerState 구현
  - 확률적으로 "1+1"이 되도록 HasQuarterState 수정
    → 나머지 상태 클래스는 수정 필요 없음.

GumballMachineWinner



WinnerState 클래스 추가

```
public class WinnerState implements State {

    GumballMachine gumballMachine;

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

    public void insertQuarter() {
        System.out.println("Please wait, we're already giving you a Gumball");
    }

    public void ejectQuarter() {
        System.out.println("Please wait, we're already giving you a Gumball");
    }

    public void turnCrank() {
        System.out.println("Turning again doesn't get you another gumball!");
    }
```

아무 일도 할 필요 없음!!!

```
    public void dispense() {
        System.out.println("YOU'RE A WINNER! You get two gumballs for your quarter");
        gumballMachine.releaseBall();
        if (gumballMachine.getCount() == 0) {
            gumballMachine.setState(gumballMachine.getSoldOutState());
        } else {
            gumballMachine.releaseBall();
            if (gumballMachine.getCount() > 0) {
                gumballMachine.setState(gumballMachine.getNoQuarterState());
            } else {
                System.out.println("Oops, out of gumballs!");
                gumballMachine.setState(gumballMachine.getSoldOutState());
            }
        }
    }

    @Override
    public String toString() {
        return "despensing two gumballs for your quarter, because YOU'RE A WINNER!";
    }
}
```

이 조건문은 실행될 수 있을까?

HasQuaterState의 turnCrank() 참고

1+1 기능 추가!

GumballMachine 클래스 수정

```
public class GumballMachine {

    State soldOutState;
    State noQuarterState;
    State hasQuarterState;
    State soldState;
    State winnerState;
    State state = soldOutState;
    int count = 0;

    public GumballMachine(int numberGumballs) {
        soldOutState = new SoldOutState(this);
        noQuarterState = new NoQuarterState(this);
        hasQuarterState = new HasQuarterState(this);
        soldState = new SoldState(this);
        winnerState = new WinnerState(this);
```

```
public State getSoldState() {
    return soldState;
}

// 3) WinnerState 객체에 대한 getter 메소드 추가
public State getWinnerState() {
    return winnerState;
}
```

HasQuarterState 클래스 수정

- Random 변수인 randomWinner 추가

```
public class HasQuarterState implements State {

    GumballMachine gumballMachine;
    java.util.Random randomWinner = new java.util.Random(System.currentTimeMillis());

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

- turnCrank() 수정
  - 1/10의 확률에 따라서 상태 달라짐.

```
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());
    }

}
```

풍선 껌이 항상 2개 이상!

실행 결과

```
Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 5 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...
You inserted a quarter
You turned...
YOU'RE A WINNER! You get two gumballs for your quarter
A gumball comes rolling out the slot...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 1 gumball
Machine is waiting for quarter
```