고급객체지향 프로그래밍
강의노트 #10

State Pattern

조용주
ycho@smu.ac.kr

스테이트 패턴 (State Pattern)

- 목적
  - Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
  - Also known as Objects for States
  - 객체의 내부 상태가 바뀔 때 객체의 동작을 변경할 수 있도록 함. 객체는 자신의 클래스를 바꾸는 것처럼 보임.

용어 설명

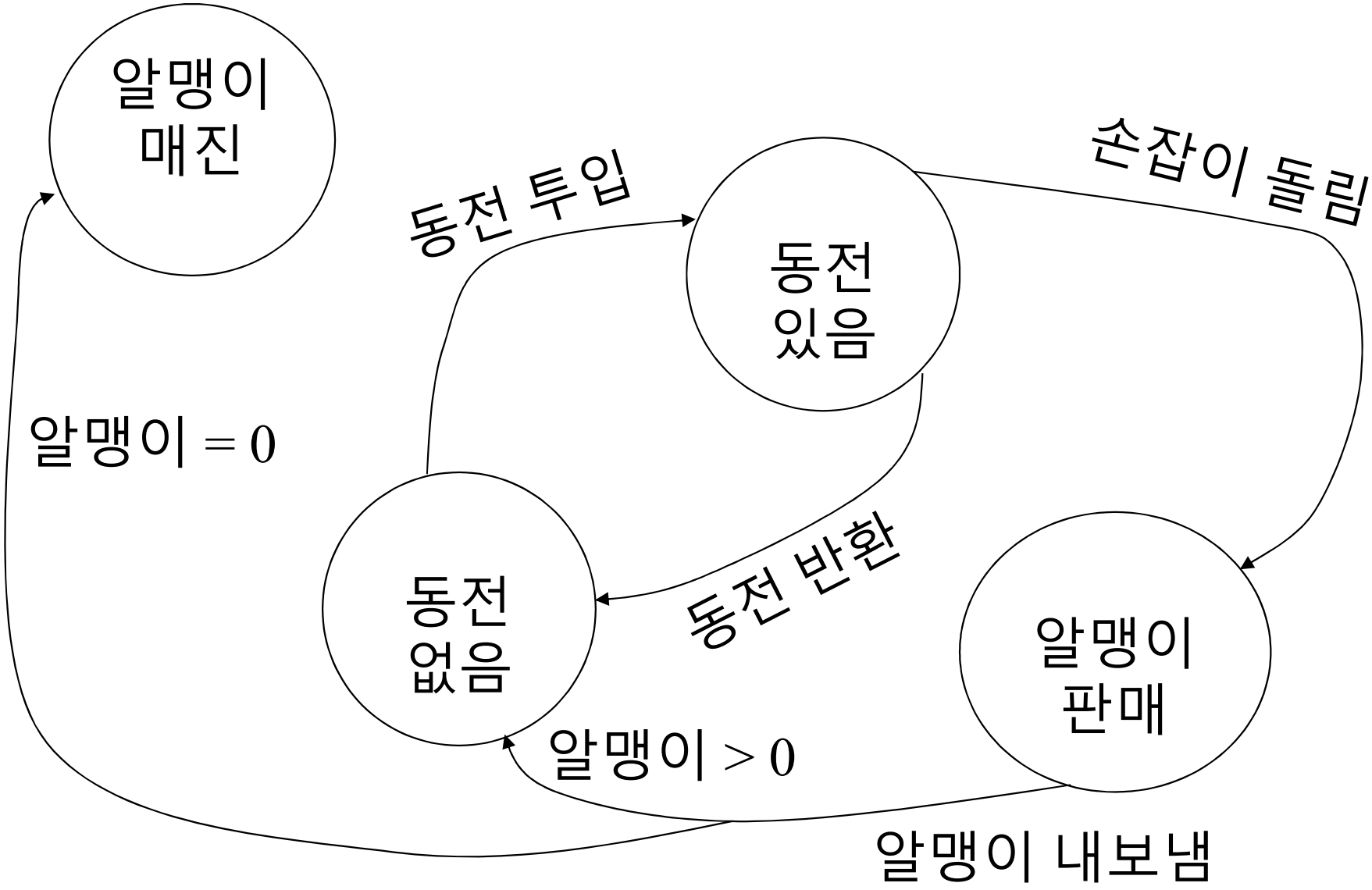
구분	설명
State(상태)	시점에 따라 특정 상태에 있어야 함. 처음에 가지게 되는 초기 상태(state) 또는 상황에 따라 여러 상태 가운데 한 상태를 가질 수 있음
Transition(전이)	외부 입력에 따라 가능한 상태로 전환

- 예
  - 게임 캐릭터: 걷는 상태, 뛰는 상태, 멈춘 상태
  - 가전 제품: on, off, sleep…
  - 지하철 개찰구: 열림, 잠금

디자인 패턴 요소

요소	설명
이름	스테이트 (State)
문제	상태(state)가 여러 개 있고, if 문으로 상태를 통제
해결방안	상태를 한 곳에서 관리
결과	변경 최소화

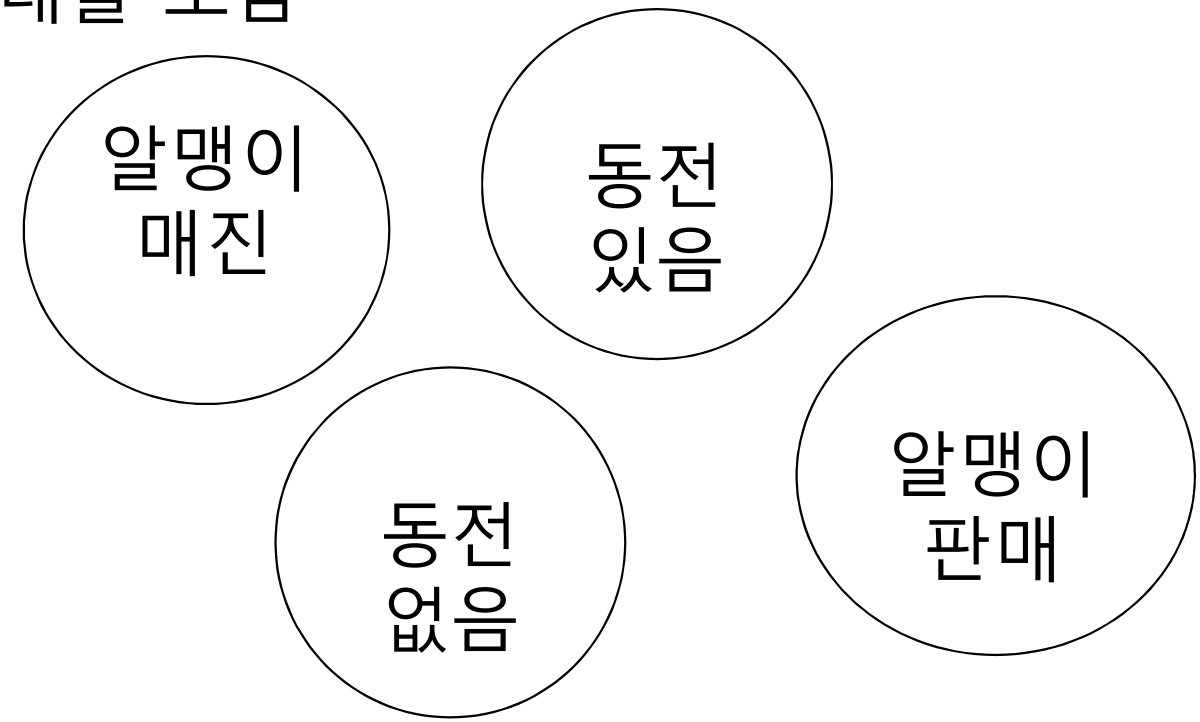
사례 1 뽑기 기계



상태기계: Finite State Machine

사례 1 뽑기 기계

- 상태 다이어그램 → 코드
  - 상태를 모음



사례 1 뽑기 기계

- 현재 상태를 저장하기 위한 인스턴스 변수를 만들고 각 상태의 값을 정의

```
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;
```

- 이 시스템에서 있을 수 있는 모든 행동을 모음

동전 반환

동전 투입

손잡이 돌림

알맹이 내보냄

사례 1 뽑기 기계

```
public void insertQuarter() {
  if (state == HAS_QUARTER) {
    System.out.println("동전은 한 개만 넣어주세요");
  } else if (state == SOLD_OUT) {
    System.out.println("매진되었습니다. 다음 기회에
이용해주세요");
  } else if (state == SOLD) {
    System.out.println("잠깐만 기다려 주세요. 알맹이
가 배출되고 있습니다");
  } else if (state = NO_QUARTER) {
    state = HAS_QUARTER;
    System.out.println("동전이 투입되었습니다");
  }
}
```

사례 1 뽑기 기계

```
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;
    }
  }
```

사례 1 뽑기 기계

```
  public void insertQuarter() {
    if (state == HAS_QUARTER) {
      System.out.println("동전은 한 개만 넣어주세요");
    } else if (state == SOLD_OUT) {
      System.out.println("매진되었습니다. 다음 기회에
이용해주세요");
    } else if (state == SOLD) {
      System.out.println("잠깐만 기다려 주세요. 알맹이
가 배출되고 있습니다");
    } else if (state = NO_QUARTER) {
      state = HAS_QUARTER;
      System.out.println("동전이 투입되었습니다");
    }
  }
```

사례 1 뽑기 기계

```
  public void ejectQuarter() {
    if (state == HAS_QUARTER) {
      System.out.println("동전이 반환됩니다");
      state = NO_QUARTER;
    } else if (state == NO_QUARTER) {
      System.out.println("ž ã"Ĥ  넣어주세요");
    } else if (state == SOLD) {
      System.out.println("이미 알맹이를 뽑으셨습니다
");
    } else if (state = SOLD_OUT) {
      System.out.println("동전을 넣지 않으셨습니다. 동
전이 반환되지 않습니다");
    }
  }
```

사례 1 뽑기 기계

```
  public void turnCrank() {
    if (state == SOLD) {
      System.out.println("손잡이는 한 번만 돌려주세요
");
    } else if (state == NO_QUARTER) {
      System.out.println("ž ã"H  넣어주세요");
    } 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("알맹이가 나가고 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 넣어주세요");
    } else if (state == SOLD_OUT) {
      System.out.println("매진입니다");
    } else if (state = HAS_QUARTER) {
      System.out.println("알맹이가 나갈 수 없습니다
");
    }
  }
  // 기타 메소드
}
```

사례 1 뽑기 기계

```
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();
    gumballMachine.ejectQuarter();

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

사례 1 뽑기 기계

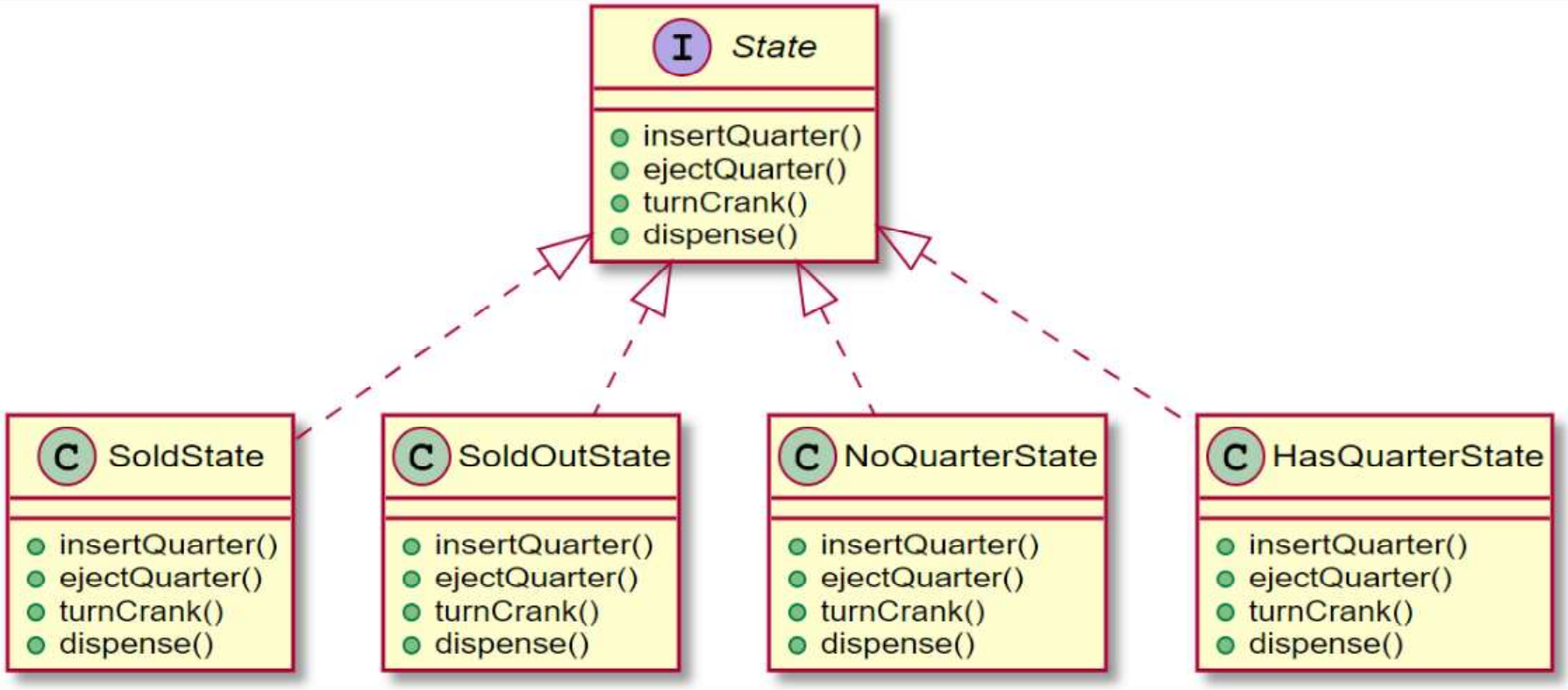
```
        gumballMachine.insertQuarter();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

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

사례 1 뽑기 기계

- 수정 요청이 들어옴
  - 10번에 한 번 꼴로 손잡이를 돌릴 때 알맹이 두 개가 나오도록 고쳐야 함
    - WINNER 상태(당첨됐다는 것을 나타냄)를 추가해야 함
    - 추가된 WINNER 상태를 확인하기 위한 조건문을 모든 메소드에 추가해야 함 → 코드를 많이 고쳐야 함

사례 1 뽑기 기계

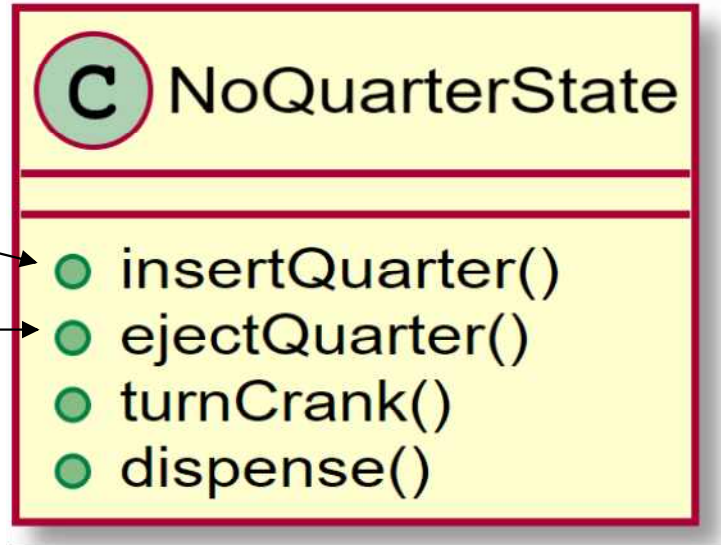
- 새로운 디자인
  - 뽑기 기계와 관련된 모든 행동에 대한 메소드가 들어있는 인터페이스 정의
  - 기계의 모든 상태에 대해 상태 클래스를 구현
  - 조건문 코드를 없애고, 상태 클래스에 모든 작업을 위임



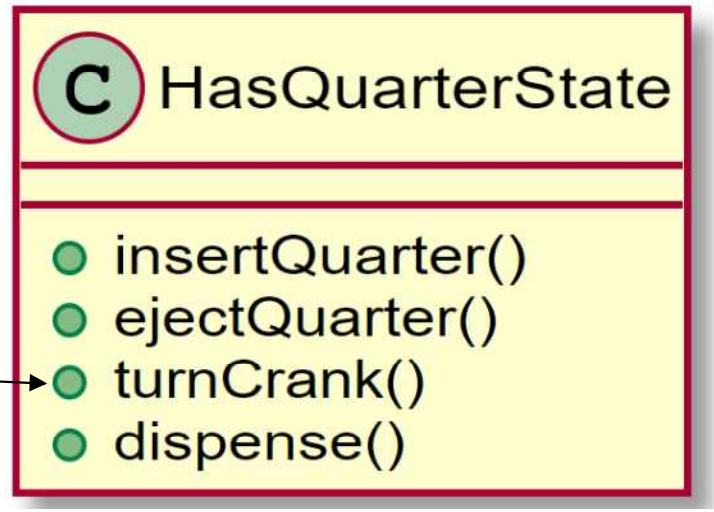
사례 1 뽑기 기계

HasQuarterState로 전환

동전을 넣어달라는
메시지를 출력



HasQuarterState로 전환

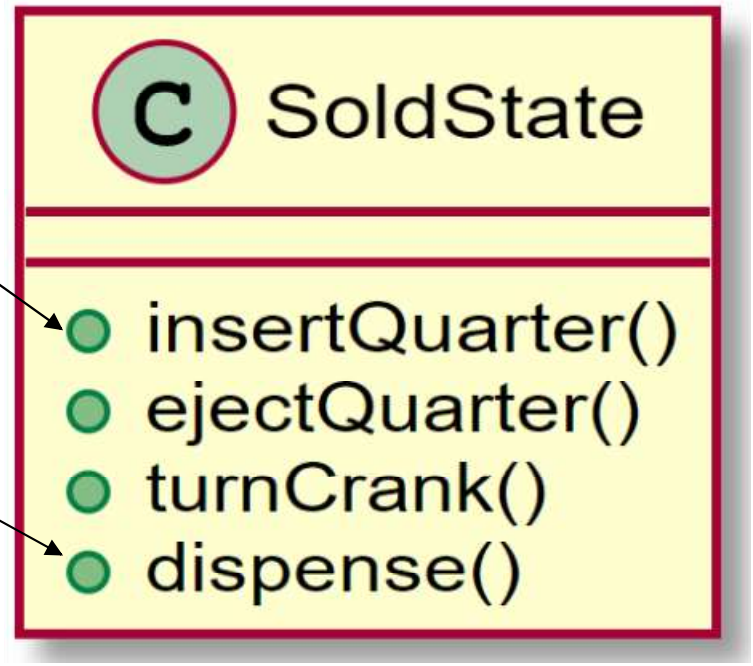


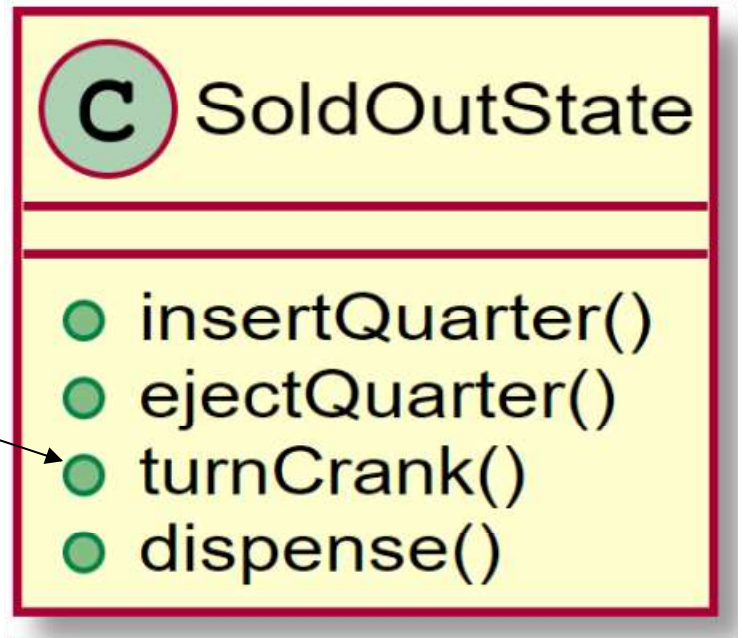
사례 1 뽑기 기계

알맹이를 내보내고 있으니
기다려 달라는 메시지를 출력

알맹이를 하나 내보냄. 알맹이
개수 > 0이면 NoQuarterState,
= 0이면 SoldOutState로 전환

매진되었음을 알림





사례 1 뽑기 기계

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

  public NoQuarterState(GumballMachine gbMachine) {
    gumballMachine = gbMachine;
  }

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

  public void ejectQuarter() {
    System.out.println("동전을 넣어주세요");
  }
```

사례 1 뽑기 기계

```
  public void turnCrank() {
    System.out.println("동전을 넣어주세요");
  }

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

사례 1 뽑기 기계

- 뽑기 기계 수정
  - 상태 표시를 정수로 하던 것을 상태 객체를 사용하는 것으로 변경

```
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;
    }
  }

  // State 객체별 Getter 메소드를 비롯한 기타 메소드
}
```

사례 1 뽑기 기계

- 다른 상태 클래스 구현

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

  public HasQuarterState(GumballMachine gbMachine) {
    gumballMachine = gbMachine;
  }

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

  public void ejectQuarter() {
    System.out.println("동전이 반환됩니다");
  }
```

사례 1 뽑기 기계

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

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

사례 1 뽑기 기계

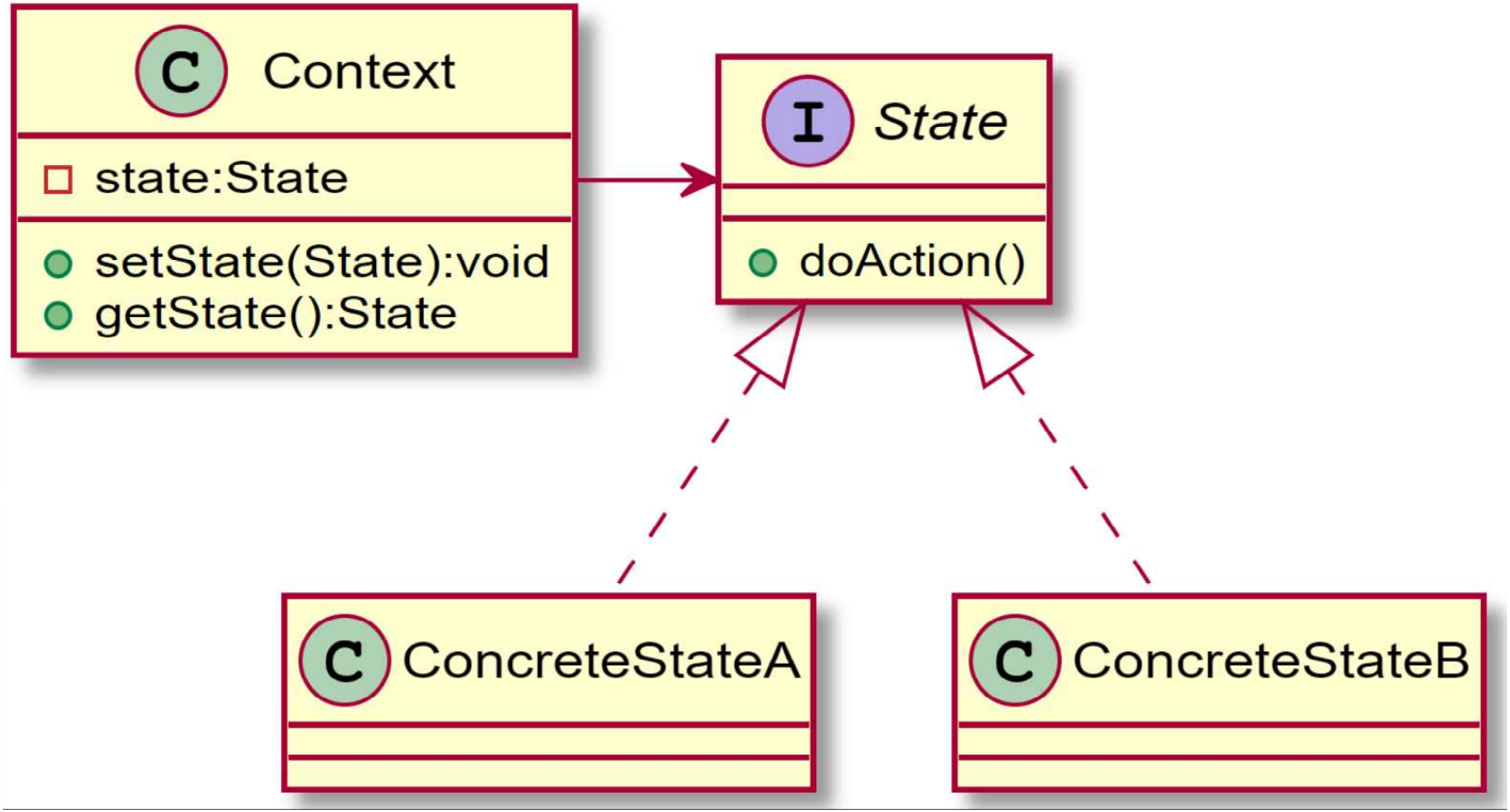
- 다른 상태 클래스 구현

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

  public SoldState(GumballMachine gbMachine) {
    gumballMachine = gbMachine;
  }

  public void insertQuarter() {
    System.out.println("잠깐만 기다려 주세요. 알맹이가
나가고 있습니다");
  }

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

사례 1 뽑기 기계

```
  public turnCrank() {
    System.out.println("손잡이는 한 번만 돌려주세요");
  }

  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());
    }
  }
}
```

스테이트 패턴


Context
state:State
setState(State):void
getState():State
State
doAction()
ConcreteStateA
ConcreteStateB

디자인 패턴 요소

요소	설명
이름	스테이트 (State)
문제	상태(state)가 여러 개 있고, if 문으로 상태를 통제
해결방안	상태를 한 곳에서 관리
결과	변경 최소화