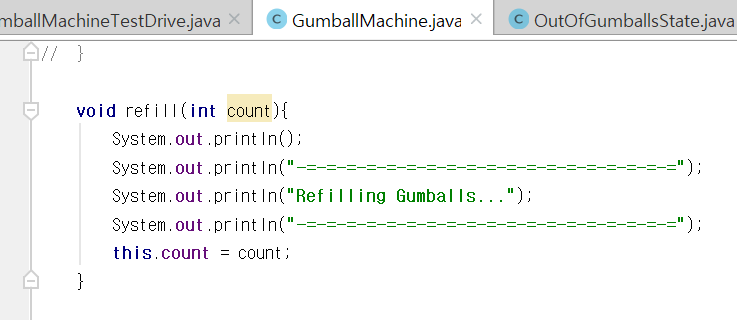
**실습 8**

1494047 김건우

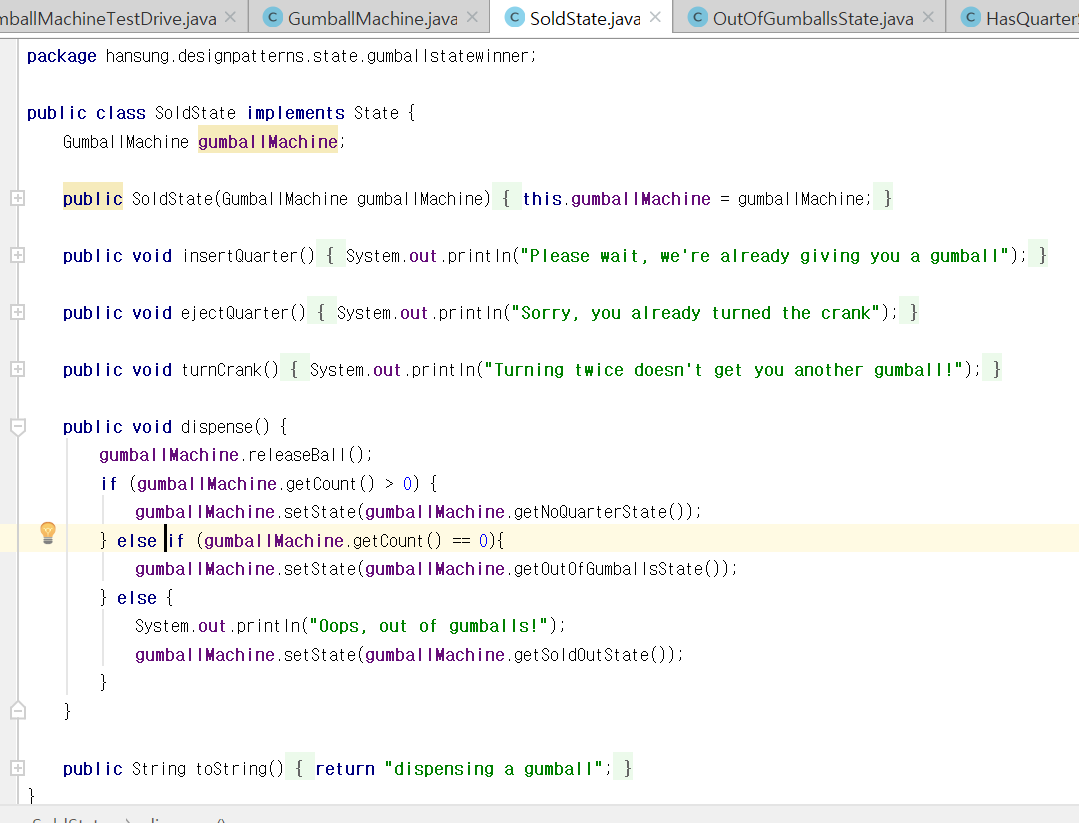
1. 아래 그림과 같이 리필(refill) 하는 부분을 상태 다이어그램에 추가하였는데 , 이 부분을 다음과 같이 구현하시오 .

A. GumballMachine에서 입력된 gumballs 개수 만큼 GumballMachine의 count 변수를 증가시키는 refill(int gumballs) 메소드를 추가하시오.



GumballMachine에 refill메소드를 추가해 주었다. Int로 지정된 Gumball개수를 추가시키기 위해 count를 파라미터로 받도록 지정해 두었다.

B. 리필 액션에 따른 상태 변화를 구현하기 위해, State 인터페이스와 각 상태 클래스들을 수정하시오 .

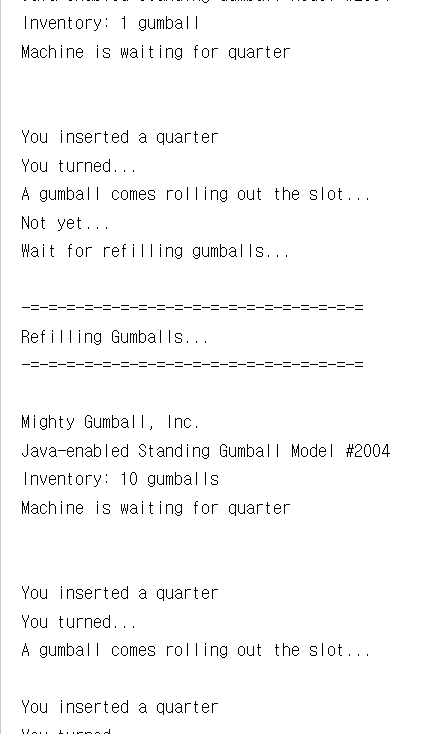


SoldState를 제외 하고서는 인터페이스를 포함하여 변경된 클래스가 없다.

SolidState는 count가 0이 되었을 때 State를 OutOfGumballState로 보내주어야 하기 때문에 수정해 주었다.

그 외의 클래스들은 변경점이 없다.

C. 기존 코드에서 어느 부분이 변경되었는 지를 기술하고 , GumballMachineTestDrive 에 이를 테스트할 수 있는 코드를 추가하여 실행결과를 제출하시오.



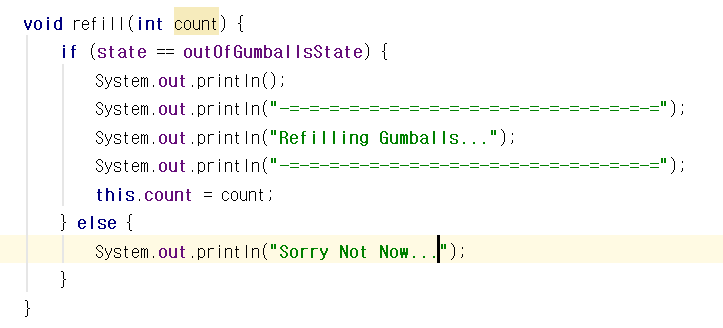
새로 추가된 클래스 OutOfGumballs = SoldOut 결과

기존 코드에 비해 변경된 점은 세가지가 있다. 첫번째로 기존 SoldState에서 count가 0이 되었을 경우 State를 OutOfGumballs로 바꿔주는 클래스와 새로 추가된 OutOfGumballsState, 그리고 GumballMachine에 추가된 refill메소드이다.

Sold에서 count가 0이 되면 OutOfGumballs에서 미리 지정된 count개수 만큼 GumballMachine의 refill메소드가 count를 증가시킨다. 그 후 다시 NoQuarterState로 변경하여 다시 정상적으로 기계가 작동할 수 있는 상태를 만들어 놓는다.

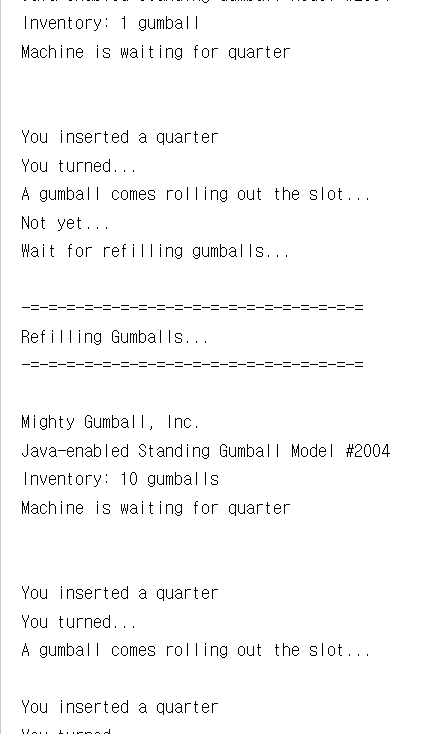
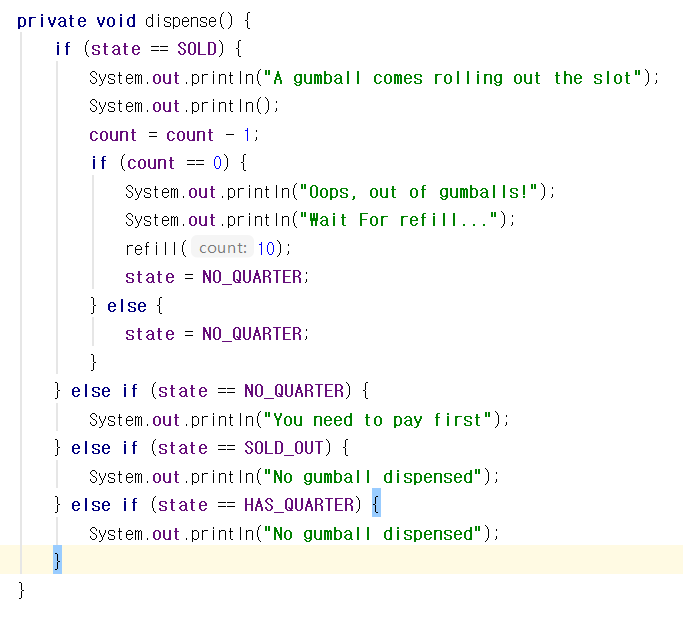
2. 1 번에서 구현한 코드에서는 현재 상태 전이를 각 상태 클래스에서 담당합니다. 위 그림과 같은 상태 전이를 GumballMachine 에서 담당하도록 구현해 봅니다.

A. 예제 코드로 주어진 gumball 프로젝트를 바탕으로 진행합니다. GumballMachine 에서 입력된 gumballs개수 만큼 GumballMachine 의 count 변수를 증가시키는 refill(intgumballs) 메소드를 추가하고, 리필 액션에 따른 상태 변화를 구현하기 위해 코드를 수정해 보세요.



새로 추가된 리필 메소드 (outOfGumballState 일때만 refill 호출 가능)

B. 기존 코드에서 어느 부분이 변경되었는 지를 기술하고, GumballMachineTestDrive 에 이를 테스트할 수 있는 코드를 추가하여 실행결과를 제출하시오.



기존 코드와 다르게 변경된 부분은 refill메소드가 들어가는 dispense 가 유일하게 변경된 부분이다.

Count가 0일 때 refill메소드를 호출하여 10 만큼 count를 증가시키고 다시 State를 NO\_QUARTER로 돌려 놓는다.

3. 강의와 실습을 통해서 상태전이는 두가지 방식으로 구현 가능함을 배웠습니다.

A. 스테이트 패턴의 각 상태 클래스에서 상태전이 담당

B. GumballMachine 과 같은 상태기계를 나타내는 클래스에서 담당

이 두가지 방식의 장점과 단점에 대해서 구체적인 예를 통해서 토의하세요. (힌트, 강의자료에는 상태 다이어그램에서 Winner 상태가 추가된 경우이고, 실습에서는 상태 간에 전이를 야기시키는 이벤트가 추가된 경우입니다. 이 두가지 경우에 각 방식의 차이점을 비교해 보세요)

|  |  |  |
| --- | --- | --- |
|  | 장점 | 단점 |
| 상태 클래스  에서 담당  (Winner) | 상태 변환에 따른 행위 수행 변경이 손쉽게 이루어질 수 있다.  수행할 행위를 결정하기 위해 객체 내부에서 상태 값을 비교하는 문장을 없앨 수 있다.  특정 상태와 관련된 행위들을 하나의 객체로 모아주는 역할을 한다. 이를 통해 State 패턴은 동일한 상태에서 이루어지는 행위를 캡슐화시켜 유지, 보수할 수 있게 해준다.  State 패턴을 적용할 경우에는 객체의 상태 전환이 명백히 드러난다.  **특정 State에서의 Event를 특정할 수 있다. 이번 실습에서 winner나 refill같은 경우 관리자가 원한 State이외에서는 호출 되지 않고 오류 메세지를 프린트하는 방식의 코딩이 가능하다.** | 불필요하게 클래스가 많아지는 문제가 생길 수 있다. 그러므로 객체 내부에서 저장, 관리하고 있는 상태의 종류가 더 추가될 가능성이 있는지, 특정 상태에서 수행해야 할 행위들을 하나로 모을 필요가 있는지 등을 고려해야 된다. |
| 상태기계 클래스에서 담당  (Gumball) | 클래스 하나로 코딩이 가능하여 코드를 이해하는데 시간이 적게 이용 될 수 있다.  Event 별로 캡슐화가 되어 있다. | 클래스 하나에 모든 state가 들어있다보니 현재 어떤 state인지 파악하기가 힘들다.  **이벤트마다 현재 어떤 State인지 비교를 해주어야 하는 코드가 필요하다.** |