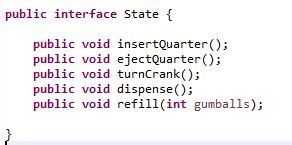
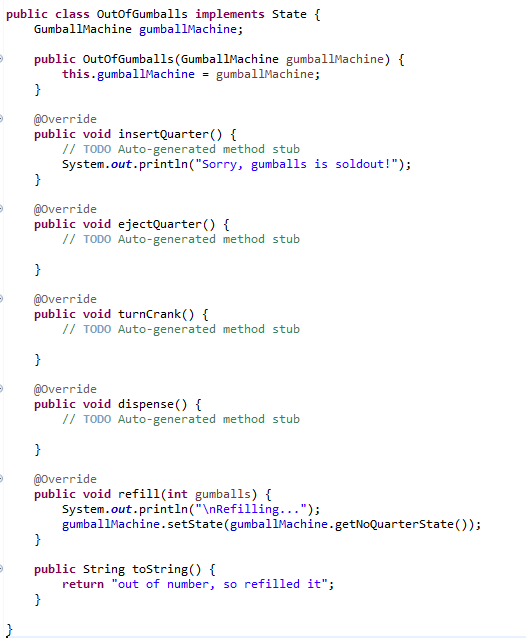
1. 아래 그림과 같이 리필(refill)하는 부분을 상태 다이어그램에 추가하였는데, 이 부분을 다음과 같이 구현하시오.
   1. GumballMachine에서 입력된 gumballs 개수 만큼 GumballMachine의 count 변수를 증가시키는**refill(int gumballs) 메소드를 추가**하시오. 리필 액션에 따른 상태 변화를 구현하기 위해, State 인터페이스와 각 상태 클래스들을 수정하시오.
   2. 기존 코드에서 어느 부분이 변경되었는 지를 기술하고, GumballMachineTestDrive 에 이를 테스트할 수 있는 코드를 추가하여 실행결과를 제출하시오.

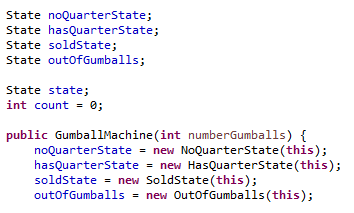


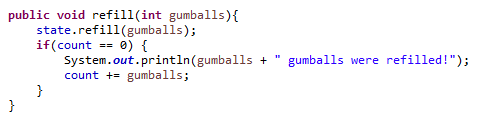
State interface에 refill 메소드를 추가하였다.



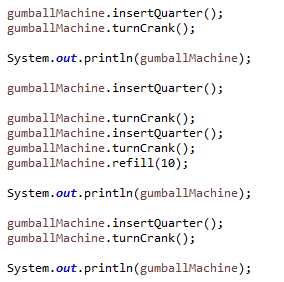
State interface를 구현한 OutOfGumballs state코드

Refill 메소드를 호출하면 리필중…을 출력하고 state를 noQuarterState로 바꾸어준다.

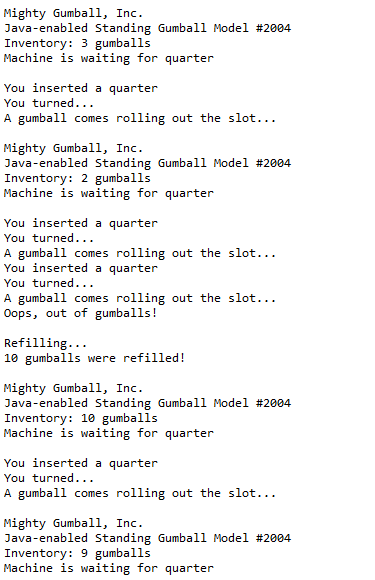




GumballMachine class에 OutofGumballs객체를 생성하였고 refill메소드를 만들어주었다. 여기서의 refill 메소드는 outofgumballs class의 refill과는 다른 메소드이며 내용은 outofgumballs class의 refill 메소드를 호출하고 남아있는 gumball이 0개 이면 입력한 개수만큼 리필되었다는 메시지와 함께 count를 입력받은 gumball 개수만큼 올려준다.

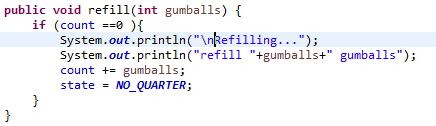


Test 코드



결과

1. 1번에서 구현한 코드에서는 현재 상태 전이를 각 상태 클래스에서 담당합니다. 위 그림과 같은 상태 전이를 GumballMachine에서 담당하도록 구현해 봅니다.
   1. GumballMachine에서 입력된 gumballs 개수 만큼 GumballMachine의 count 변수를 증가시키는**refill(int gumballs) 메소드를 추가**하고, 리필 액션에 따른 상태 변화를 구현하기 위해 코드를 수정해 보세요.
   2. 기존 코드에서 어느 부분이 변경되었는 지를 기술하고, GumballMachineTestDrive 에 이를 테스트할 수 있는 코드를 추가하여 실행결과를 제출하시오.

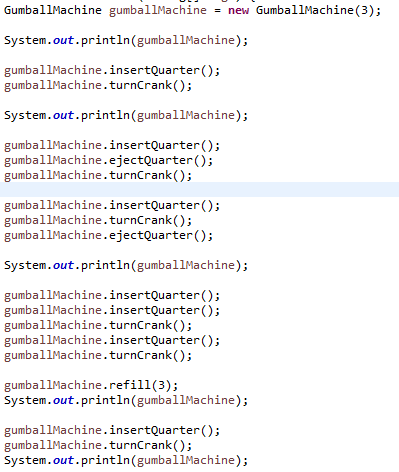


GumballMachine class에 refill메소드를 정의하였다.

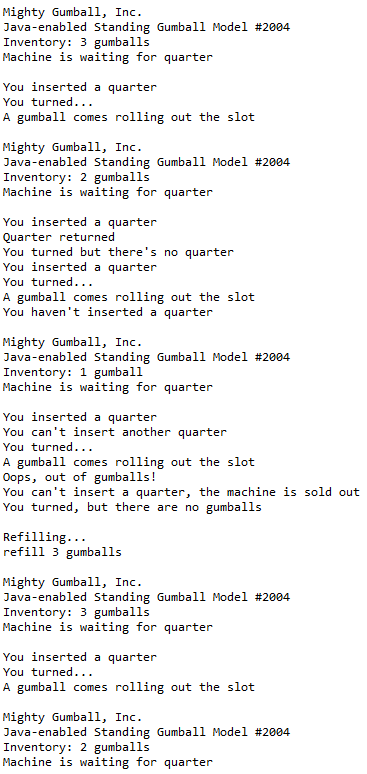
1번과 마찬가지로 count가 0일때 수행하도록 하였으며

Refilling… 이라는 메시지와 입력받은 개수의 gumball이 리필되었다는 메시지를 출력하고

Count를 늘려주며 state를 NO\_QUARTER로 변경한다.



TEST code



결과

1. 강의와 실습을 통해서 상태전이는 두가지 방식으로 구현 가능함을 배웠습니다.
   1. 스테이트 패턴의 각 상태 클래스에서 상태전이 담당
   2. GumballMachine과 같은 상태기계를 나타내는 클래스에서 담당

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

스테이트 패턴

장점:

각 state의 행동을 별개의 클래스로 국지화 시켜 상태별로 메소드를 관리할 수 있는 장점이 있다. 또한 상태별로 메소드의 역할을 다른 방식으로 처리하기에 매우 편리하다. 강의 자료의 winnerstate를 보면 winner상태가 구현되었을 때는 그렇지 않을 때와 달리 winner state만의 행동을 구현할 수 있다.

단점:

상태별로 클래스를 정의하여 상태를 많이 구현하다 보면 클래스의 개수가 늘어나 관리가 복잡해질 수가 있다.

GumballMarchine과 같은 상태기계를 나타내는 클래스에서 담당하는경우

장점: 클래스의 수가 많지 않아 클래스 자체의 관리가 쉽다.

단점: state별로 다른 행동을 하기 위해서는 하나의 메소드에 수많은 if else 문이 들어가야 하는 단점이 있다. 하나의 클래스에 구현이 되어 있다고 하더라도 코드가 복잡해지는 단점이 있다. 또한 각 state별로 행동이 묶여있지않다 보니 각 state별로 행동을 수정하기가 힘들며 새로운 행동을 추가할 때에도 수많은 if else문을 찾아 수정해야하는 큰 단점이 있다.