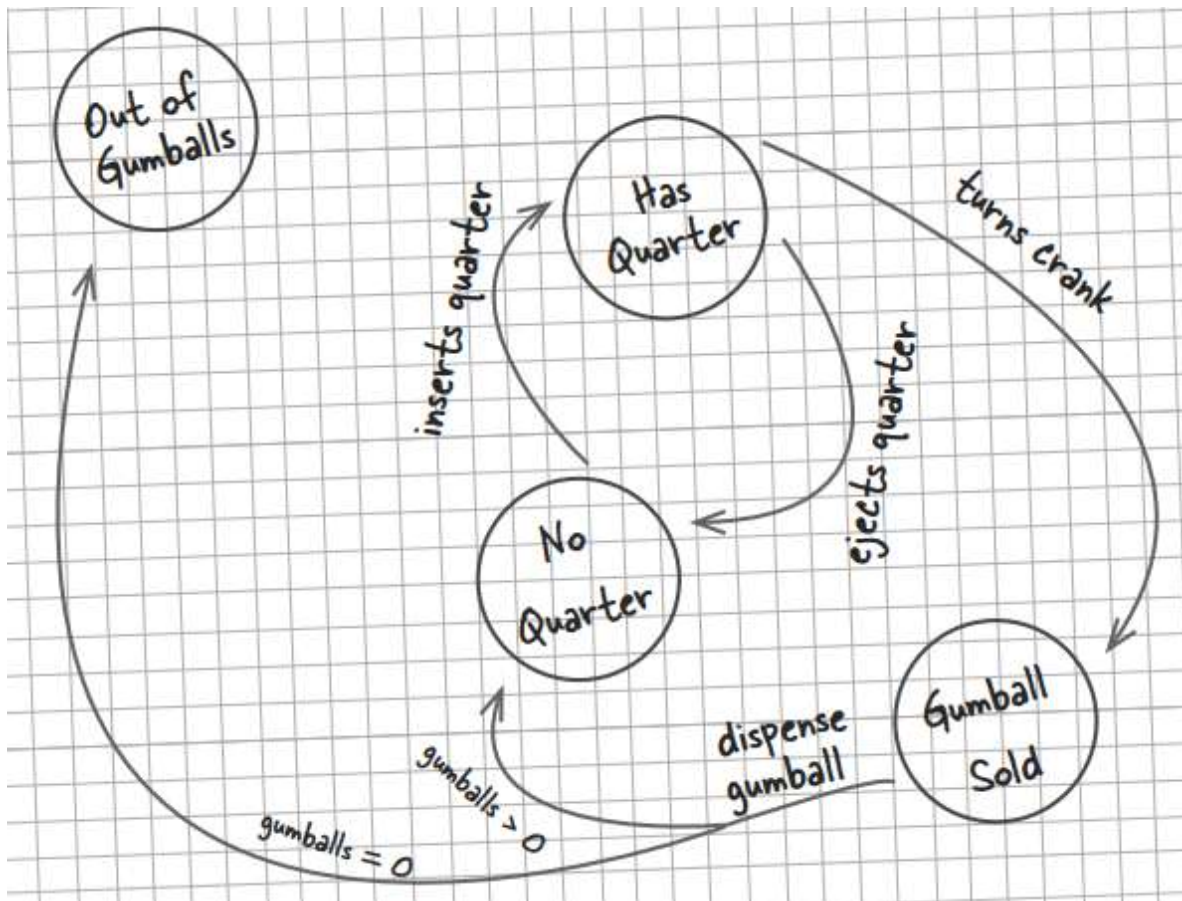


Gumball Machine

La empresa Mighty Gumball Inc. nos ha dado la responsabilidad de implementar el software para sus máquinas de goma de mascar.

Los especialistas de Mighty Gumball Inc. esperan que el controlador de la máquina de goma maneje la siguiente lógica. Ellos esperan agregar más comportamiento en el futuro, entonces se necesita mantener el diseño lo más flexible y mantenible posible.



La primera versión de la aplicación

```
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;  
        }  
    }  
}
```

Here are the four states; they match the states in Mighty Gumball's state diagram.

Here's the instance variable that is going to keep track of the current state we're in. We start in the `SOLD_OUT` state.

We have a second instance variable that keeps track of the number of gumballs in the machine.

The constructor takes an initial inventory of gumballs. If the inventory isn't zero, the machine enters state `NO_QUARTER`, meaning it is waiting for someone to insert a quarter, otherwise it stays in the `SOLD_OUT` state.

Now we start implementing the actions as methods....

```
    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");  
        }  
    }  
}
```

When a quarter is inserted, if....

a quarter is already inserted we tell the customer;

otherwise we accept the quarter and transition to the `HAS_QUARTER` state.

If the customer just bought a gumball he needs to wait until the transaction is complete before inserting another quarter.

and if the machine is sold out, we reject the quarter.

```

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");
    }
}

```

Now, if the customer tries to remove the quarter...

If there is a quarter, we return it and go back to the NO_QUARTER state.

Otherwise, if there isn't one we can't give it back.

You can't eject if the machine is sold out, it doesn't accept quarters!

If the customer just turned the crank, we can't give a refund; he already has the gumball!

The customer tries to turn the crank...

```

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();
    }
}

```

Someone's trying to cheat the machine.

We need a quarter first.

We can't deliver gumballs; there are none.

Success! They get a gumball. Change the state to SOLD and call the machine's dispense() method.

Called to dispense a gumball.

```

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");
    }
}

```

We're in the SOLD state; give 'em a gumball!

Here's where we handle the "out of gumballs" condition: If this was the last one, we set the machine's state to SOLD_OUT; otherwise, we're back to not having a quarter.

None of these should ever happen, but if they do, we give 'em an error, not a gumball.

// other methods here like toString() and refill()

Sabíamos que pasaría...un requerimiento de cambio

El CEO de Mighty Gumball, Inc. piensa que convertir “comprar un goma” en un juego incrementaría significativamente las ventas. Para esto, desea que agreguemos la siguiente lógica: “el 10% de las veces que se gira la manivela, el cliente recibe 2 gomas en vez de 1”.

Analicemos cómo implementar este cambio.....

```
final static int SOLD_OUT = 0;
final static int NO_QUARTER = 1;
final static int HAS_QUARTER = 2;
final static int SOLD = 3;
```

First, you'd have to add a new WINNER state here. That isn't too bad...

```
public void insertQuarter() {
    // insert quarter code here
}
```

```
public void ejectQuarter() {
    // eject quarter code here
}
```

... but then, you'd have to add a new conditional in every single method to handle the WINNER state; that's a lot of code to modify.

```
public void turnCrank() {
    // turn crank code here
}
```

```
public void dispense() {
    // dispense code here
}
```

turnCrank() will get especially messy, because you'd have to add code to check to see whether you've got a WINNER and then switch to either the WINNER state or the SOLD state.

¿Cuáles son los problemas de nuestra implementación?