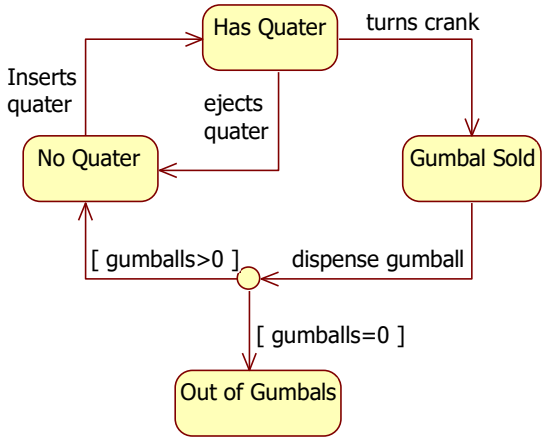
The State Pattern

Lab 1: Gumball Machine

– Chúng ta cần tạo một bộ điều khiển máy bán kẹo. Đây là cách bộ điều khiển cần để hoạt động.



1. Thiết kế ban đầu của hệ thống:

– Dùng các biến số nguyên để lưu các giá trị trạng thái và trạng thái hiện thời.

– Tạo từng phương thức cho từng hành động của máy **insertQuater()**, **ejectQuater()**, **turnCrank()**, **dispense()**. Dùng điều kiện để xác định ứng xử tương ứng với từng trạng thái

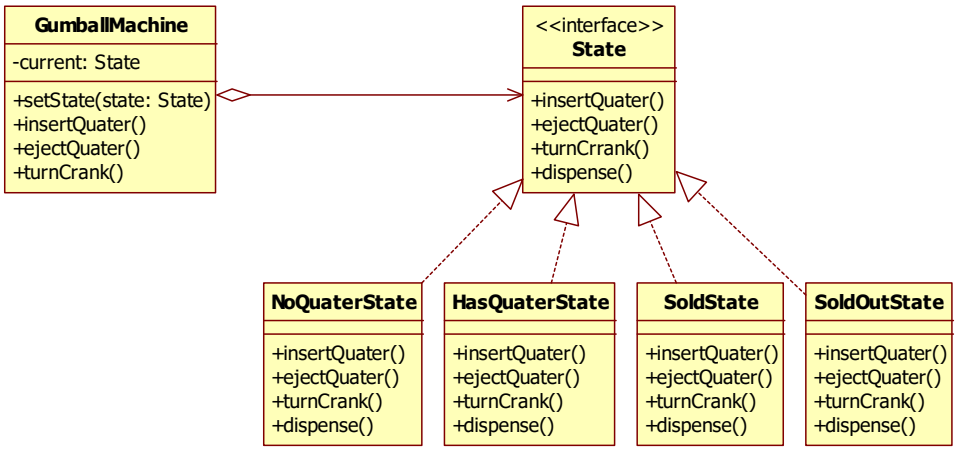
```
public class GumballMachine {
   private final static int SOLD_OUT = 0;
   private final static int NO_QUARTER = 1;
   private final static int HAS_QUARTER = 2;
   private final static int SOLD = 3;

   private int state = SOLD_OUT;
   private 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");
      }
   }

   // other methods
```

```
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();
   }
}

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");
   }
}

public void refill(int numGumBalls) {
   this.count = numGumBalls;
   state = NO_QUARTER;
}

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();
    }
}
```

```
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);
    }
}
```

2. Thiết kế lại hệ thống dùng mẫu State



– Tạo interface **State** với các phương thức hành động của máy: **insertQuater()**, **ejectQuater()**, **turnCrank()**, **dispense().**

```
public interface State {
   public void insertQuarter();
   public void ejectQuarter();
   public void turnCrank();
   public void dispense();
}
```

– Tạo các lớp trạng thái cụ thể **NoQuaterState**, **HasQuaterState**, **SoldState**, **SoldOutState** hiện thực interface **State** và cài đặt từng hành động tương ứng với từng trạng thái.

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

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

   public void insertQuarter() {
      // TODO
   }

   public void ejectQuarter() {
      // TODO
   }

   public void turnCrank() {
      // TODO
   }

   public void dispense() {
      // TODO
   }

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

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

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

   public void insertQuarter() {
      // TODO
   }

   public void ejectQuarter() {
      // TODO
   }

   public void turnCrank() {
      // TODO
   }

   public void dispense() {
      // TODO
   }

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

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

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

   public void insertQuarter() {
      // TODO
   }

   public void ejectQuarter() {
      // TODO
   }

   public void turnCrank() {
      // TODO
   }

   public void dispense() {
      gumballMachine.releaseBall();

      // TODO
   }

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

```
public class SoldOutState implements State {
   private 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");
    }

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

- Trong lớp **GumballMachine**
  - Định nghĩa các biến **State** để biểu diễn các giá trị trạng thái và trạng thái hiện thời của máy.
  - Tạo phương thức **setState()** để đặt trạng thái cho máy.
  - Tạo các phương thức hành động cho máy và ủy quyền thực hiện cho phương thức tương ứng trong trạng thái hiện thời.

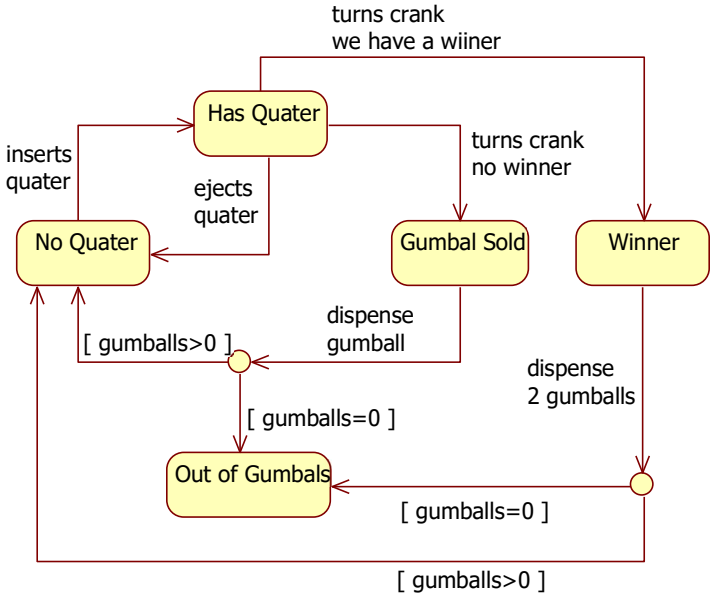
```
public class GumballMachine {
    private State soldOutState;
    private State noQuarterState;
    private State hasQuarterState;
    private State soldState;
    private State state = soldOutState;
    private int count = 0;

    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;
    }

    public State getHasQuarterState() {
        return hasQuarterState;
    }

    public State getSoldState() {
        return soldState;
    }

    public String toString() {
        StringBuffer result = new StringBuffer();
        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();
    }
}
```

Viết lớp Test thiết kế

```
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);
   }
}
```

3. **<u>Yêu cầu mới</u>**: Chuyển việc mua kẹo thành một trò chơi! 10% thời gian khi kéo tay quay (turn crank), khách hàng sẽ được 2 kẹo thay vì 1!
   Lược đồ trạng thái hoạt động của máy như sau:



- Thêm lớp trạng thái **WinnerState** cài đặt ứng xử của máy như trên

```
public class WinnerState implements State {
   private 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!");
   }
   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());
               }
            }
         }

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

- Hiệu chỉnh ứng xử **turnCrank** trong lớp **HasQuaterState** để cập nhật ứng xử trạng thái như trên.

```
import java.util.Random;

public class HasQuarterState implements State {
   private Random randomWinner = new Random(System.currentTimeMillis());
   private GumballMachine gumballMachine;

   ...

   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());
      }
   }

   ...
}
```

- Thêm biến **winnerState** và phương thức **getWinnerState**() trong lớp **GumballMachine** để lưu và truy xuất trạng thái **winner**.

```
public class GumballMachine {
   private State soldOutState;
   private State noQuarterState;
   private State hasQuarterState;
   private State soldState;
   private State winnerState;

   private State state = soldOutState;
   int count = 0;

   public GumballMachine(int numberGumballs) {
      soldOutState = new SoldOutState(this);
      noQuarterState = new NoQuarterState(this);
```

```java
        hasQuarterState = new HasQuarterState(this);
        soldState = new SoldState(this);
        winnerState = new WinnerState(this);

        this.count = numberGumballs;
        if (numberGumballs > 0) {
            state = noQuarterState;
        }
    }

    ...

    public State getWinnerState() {
        return winnerState;
    }

    ...
}
```

```java
public class GumballMachineTestDrive {

    public static void main(String[] args) {
        GumballMachine gumballMachine = new GumballMachine(10);

        System.out.println(gumballMachine);

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

        System.out.println(gumballMachine);

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

        System.out.println(gumballMachine);

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

        System.out.println(gumballMachine);

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

        System.out.println(gumballMachine);

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