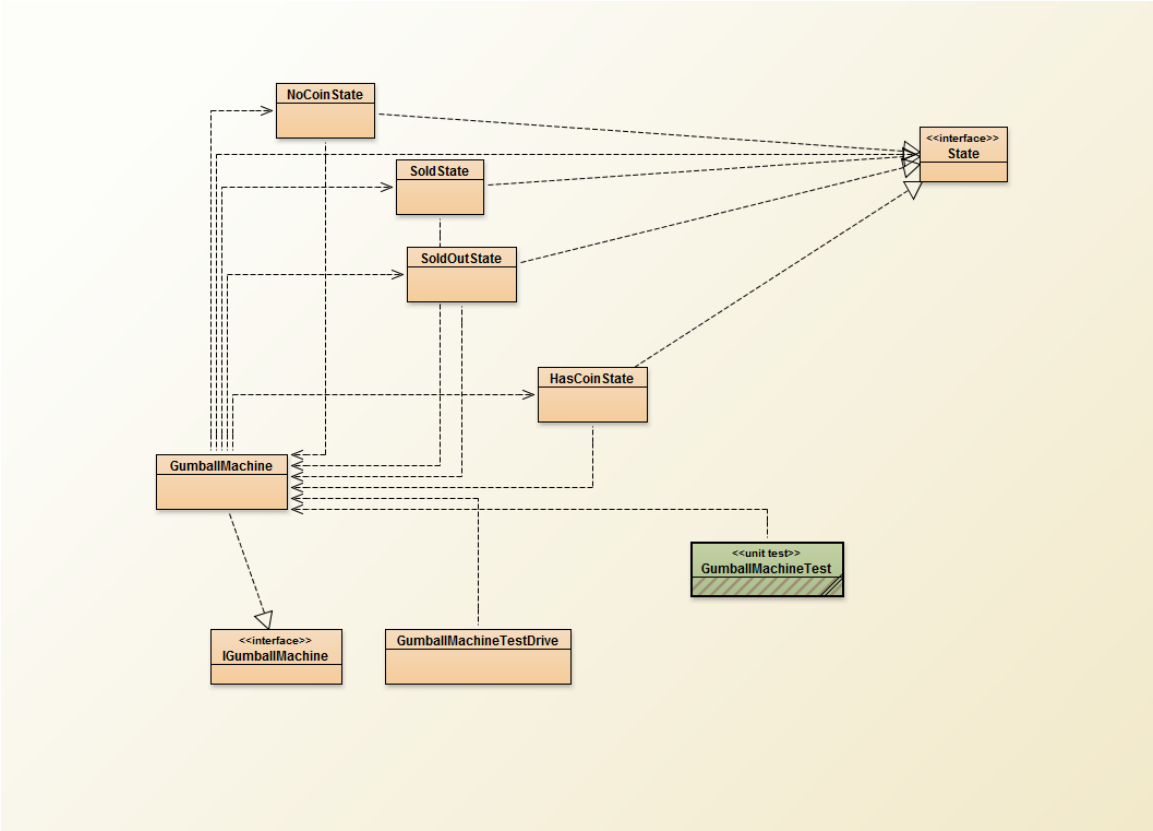
GumballMachine using BlueJ

Deenash Arivazhagan Senthamilselvi (009877114)
deenash.arivazhagansenthamilselvi@sjsu.edu
CMPE 202 – SOFTWARE SYSTEMS ENGINEERING

GumballMachine in BlueJ



Code Definition

State Interface:

Abstract class for NoCoinState, HasCoinState, SoldState, SoldOutState.

```
public interface State {

	public void insertQuarter();
	public void ejectQuarter();
	public void turnCrank();
	public void dispense();
	public void insertDime();
	public void insertNickel();
}
```

NoCoinState

Maintains the state in NoCoinState till the total value of the coin reaches >= 50 cents and then changes to HasCoinState.

EjectQuarter() will dispense all the coins inserted.

```
public class NoCoinState implements State {
    GumballMachine gbMac;

    public NoCoinState(GumballMachine gbMac) {
        this.gbMac = gbMac;
    }

      public void insertQuarter() {

            System.out.println("You inserted a quarter");
            gbMac.setMoney_count(gbMac.getMoney_count()+25);
            System.out.println("The total money inserted:
"+gbMac.getMoney_count()+ " cents");
            if(gbMac.getMoney_count()>=50)
            {
                  gbMac.setState(gbMac.getHasCoinState());
            }
            else
            {
                  System.out.println("You have to insert "+ (50-
gbMac.getMoney_count())+ " more cents to get a Gumball.");
            }
      }


      public void insertDime(){
            System.out.println("You inserted a dime");
            gbMac.setMoney_count(gbMac.getMoney_count()+10);
            System.out.println("The total money inserted:
"+gbMac.getMoney_count()+ " cents");
            if(gbMac.getMoney_count()>=50)
            {
                  gbMac.setState(gbMac.getHasCoinState());
            }
            else
            {
                  System.out.println("You have to insert "+ (50-
gbMac.getMoney_count())+ " more cents to get a Gumball.");
            }
      }

      public void insertNickel(){
            System.out.println("You inserted a Nickel.");
            gbMac.setMoney_count(gbMac.getMoney_count()+5);
            System.out.println("The total money inserted:
"+gbMac.getMoney_count()+ " cents");
            if(gbMac.getMoney_count()>=50)
            {
```

```
                        gbMac.setState(gbMac.getHasCoinState());
                }
                else
                {
                        System.out.println("You have to insert "+ (50-
gbMac.getMoney_count())+ " more cents to get a Gumball.");
                }
        }

        public void ejectQuarter() {
                if(gbMac.getMoney_count()<25)
                        System.out.println("You haven't inserted a
quarter / Not sufficient money is inserted.");
                else
                {
                        System.out.println("You can take back your
Quarter.");
                        gbMac.setMoney_count(gbMac.getMoney_count()-
25);
                        System.out.println("The total money present:
"+gbMac.getMoney_count()+ " cents");
                }
        }

        public void turnCrank() {
                System.out.println("You turned, but there's no
enough money");
         }

        public void dispense() {
                System.out.println("You need to pay first");
        }

        public String toString() {
                return "waiting for two quarters";
        }

                }
```

HasCoinState:

State changes to HasCoinState when the total coin inserted reaches 50 cents. Throws error statement on inserting more coins after this state is reached. Turning the Crank will initiate ejecting the Gumball from the Machine and set “isGumballInSlot” variable to true and changes the state to SoldState.

```
import java.util.Random;

public class HasCoinState implements State {
        GumballMachine gbMac;

        public HasCoinState(GumballMachine gbMac) {
                this.gbMac = gbMac;
        }

        public void insertQuarter() {
```

```
            System.out.println("You can't insert another
quarter");
      }

      public void insertDime(){
            System.out.println("You can't insert another dime");
      }

      public void insertNickel(){
            System.out.println("You can't insert another
nickel");
      }

      public void ejectQuarter() {
            System.out.println("Quarter returned");
            gbMac.setMoney_count(gbMac.getMoney_count()-25);
            gbMac.setState(gbMac.getNoCoinState());
      }

      public void turnCrank() {
            System.out.println("You turned...");
            gbMac.gumballInSlot=true;
            gbMac.setState(gbMac.getSoldState());
      }

    public void dispense() {
        System.out.println("No gumball dispensed");
    }

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

SoldState:

Releases a gumball from the Machine and returns the extra money inserted.
Decreases the Gumball count by one.
Sets the GumballMachine to 'SoldOutState' if no more gumballs are in the machine or to the 'NoCoinState'.

```
public class SoldState implements State {

    GumballMachine gbMac;

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

      public void insertQuarter() {
            System.out.println("Please wait, we're already
giving you a gumball");
      }
```

```
	public void ejectQuarter() {
		System.out.println("Sorry, you already turned the
crank");
	}

	public void insertDime(){
		System.out.println("Please wait, we're already
giving you a gumball");
	}

	public void insertNickel(){
		System.out.println("Please wait, we're already
giving you a gumball");
	}

	public void turnCrank() {
		System.out.println("Turning twice doesn't get you
another gumball!");
	}

	public void dispense() {

		gbMac.releaseBall();
		//gbMac.takeGumballFromSlot();
		gbMac.setMoney_count(gbMac.getMoney_count()-50);
		gbMac.releaseMoney();
		if (gbMac.getCount() > 0) {
			gbMac.setState(gbMac.getNoCoinState());
		} else {
			System.out.println("Oops, out of gumballs!");
			gbMac.setState(gbMac.getSoldOutState());
		}
	}

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

SoldOutState:

This state is set on Gumball count falls to zero. Throws appropriate error messages on inserting a coin or Turning the crank.

```
public class SoldOutState implements State {
    GumballMachine gbMac;

    public SoldOutState(GumballMachine gbMac) {
        this.gbMac = gbMac;
    }

      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 insertDime(){
            System.out.println("You can't insert a dime, the
machine is sold out");
      }

      public void insertNickel(){
            System.out.println("You can't insert a nickel, the
machine is sold out");
      }

      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";
      }
}
```

IGumballMachine

Defines the GumballMachine interface.

```
public interface IGumballMachine {
    void insertQuarter( ) ;
    void insertDime( ) ;
    void insertNickel( );
    void turnCrank();
    boolean isGumballInSlot();
    void takeGumballFromSlot();
}
```

GumballMachine:

Initiates the Gumball with the SoldOutState or with NoCoinState. Provides functionalities for releaseGumball and releaseMoney.
takeGumballMahineFromSlot() is used to reset the isGumballMachineInSlot to false depending on the current state.

```
public class GumballMachine implements IGumballMachine {

	State soldOutState;
	State noCoinState;
	State hasCoinState;
	State soldState;

	State state = soldOutState;
	int count = 0;
	int money_count=0;
	boolean gumballInSlot=false;


	public GumballMachine(int numberGumballs) {
		soldOutState = new SoldOutState(this);
		noCoinState = new NoCoinState(this);
		hasCoinState = new HasCoinState(this);
		soldState = new SoldState(this);

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

	public void insertQuarter() {
		state.insertQuarter();
	}

	public void insertDime(){
		state.insertDime();
	}

	public void insertNickel(){
		state.insertNickel();
	}

	public void ejectQuarter() {
		state.ejectQuarter();
	}

	public void turnCrank() {
		state.turnCrank();
		state.dispense();
	}


	void setState(State state) {
		this.state = state;
	}

	void releaseBall() {
		if(this.gumballInSlot==true)
		{
```

```
                    System.out.println("A gumball comes rolling
out the slot...");
                    if (count != 0) {
                          count = count - 1;
                    }
              }
              else
                    System.out.println("No Gumball to release.");
        }

        void releaseMoney(){
              if(money_count>0)
              {
                    int tempMoney=money_count;
                    System.out.println("You can have your
"+money_count+" cents back.");
                    money_count=0;
              }
              else
                    System.out.println("Your must have inserted
two Quarters exactly. There is no Change.");

        }

        int getCount() {
              return count;
        }

        void refill(int count) {
              this.count = count;
              state = noCoinState;
        }

      public State getState() {
          return state;
      }

      public State getSoldOutState() {
          return soldOutState;
      }

      public State getNoCoinState() {
          return noCoinState;
      }

      public State getHasCoinState() {
          return hasCoinState;
      }

      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();
          }


          public boolean isGumballInSlot() {

                return gumballInSlot;
          }


          public void takeGumballFromSlot() {
                if(gumballInSlot==true)
                {
                      System.out.println("Removing Gumball from the
GumballMachine Slot");
                      gumballInSlot=false;
                }
                else
                      System.out.println("No Gumball in Slot.");
          }

          public int getMoney_count() {
                return money_count;
          }

          public void setMoney_count(int money_count) {
                this.money_count = money_count;
          }

          public void setGumballInSlot(boolean gumballInSlot) {
                this.gumballInSlot = gumballInSlot;
          }
}
```

GumballMachineTestDrive:

Sample Program case to run the GumbalMachine Program in correct order of Execution and to traverse through the various order to check the error messages.

```
public class GumballMachineTestDrive {

    public static void main(String[] args) {

        GumballMachine gbMac = new GumballMachine(5);
```

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

        gbMac.insertQuarter();

        gbMac.turnCrank();

        System.out.println(gbMac);

        gbMac.insertQuarter();

        gbMac.turnCrank();

        System.out.println("\n\n\nGumball count remaining:
"+gbMac.getCount()+" \n\n\n");

        gbMac.insertQuarter();

        gbMac.turnCrank();

        System.out.println(gbMac);



            gbMac = new GumballMachine(5);

            System.out.println(gbMac);

            gbMac.insertQuarter();
            gbMac.insertDime();
            gbMac.insertDime();
            gbMac.insertDime();
            gbMac.insertQuarter();
            gbMac.turnCrank();
            gbMac.insertDime();
            gbMac.ejectQuarter();
            System.out.println(gbMac);

    }
}
```

JUnit TestCase:

```
public class GumballMachineTest {

      GumballMachine gm;

      @Before
      public void setUp() throws Exception {
      }

      @After
      public void tearDown() throws Exception {
      }

      @Test
      public void test1() {
            gm=new GumballMachine(2);
            gm.insertQuarter();
            gm.insertDime();
            gm.insertNickel();
            gm.turnCrank();
            assertEquals(false, gm.isGumballInSlot());
      }


    @Test
    public void test2()
    {
      gm=new GumballMachine(2);
        gm.insertQuarter();
        gm.insertQuarter();
        gm.turnCrank();
        assertEquals(true, gm.isGumballInSlot());
        gm.takeGumballFromSlot();
        assertEquals(false, gm.isGumballInSlot());
    }

    @Test
    public void test3()
    {
      gm=new GumballMachine(5);
        gm.insertQuarter();
        gm.insertDime();
        gm.insertQuarter();
        gm.turnCrank();
        assertEquals(4, gm.getCount());
    }

    @Test
    public void test4()
    {
      gm=new GumballMachine(2);
        gm.insertQuarter();
        gm.insertQuarter();
```

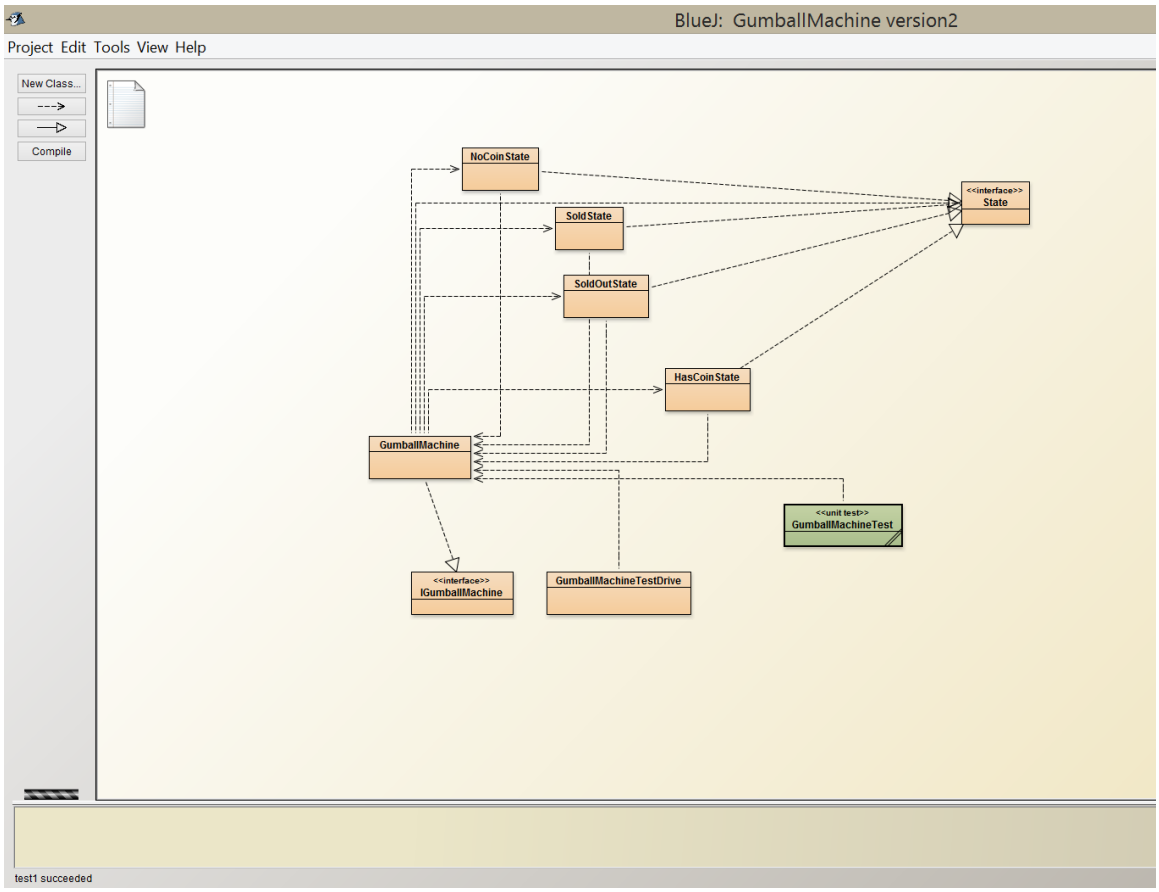
```
    gm.insertNickel();
    gm.turnCrank();
    gm.takeGumballFromSlot();
    assertEquals(0, gm.getMoney_count());
}

@Test
public void test5()
{
  gm=new GumballMachine(2);
    gm.insertDime();
    gm.insertQuarter();
    gm.turnCrank();
    gm.insertDime();
    gm.insertNickel();
    gm.turnCrank();
    gm.takeGumballFromSlot();
    gm.insertQuarter();
    gm.turnCrank();
    assertEquals(false, gm.isGumballInSlot());
}

        }
```

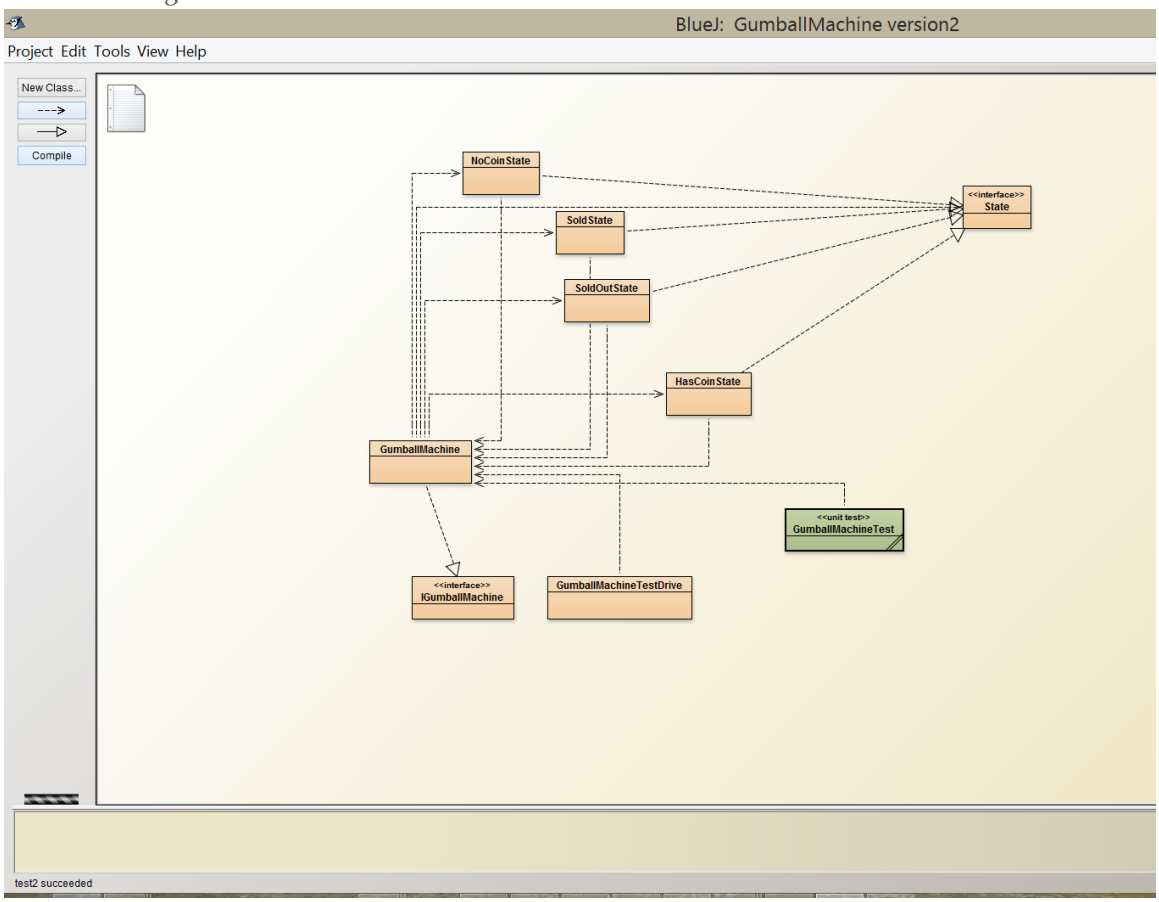
Case 1:

Executes the GumBall Machine in correct order and checks for gumballInSlot condition variable that has to be set to false.



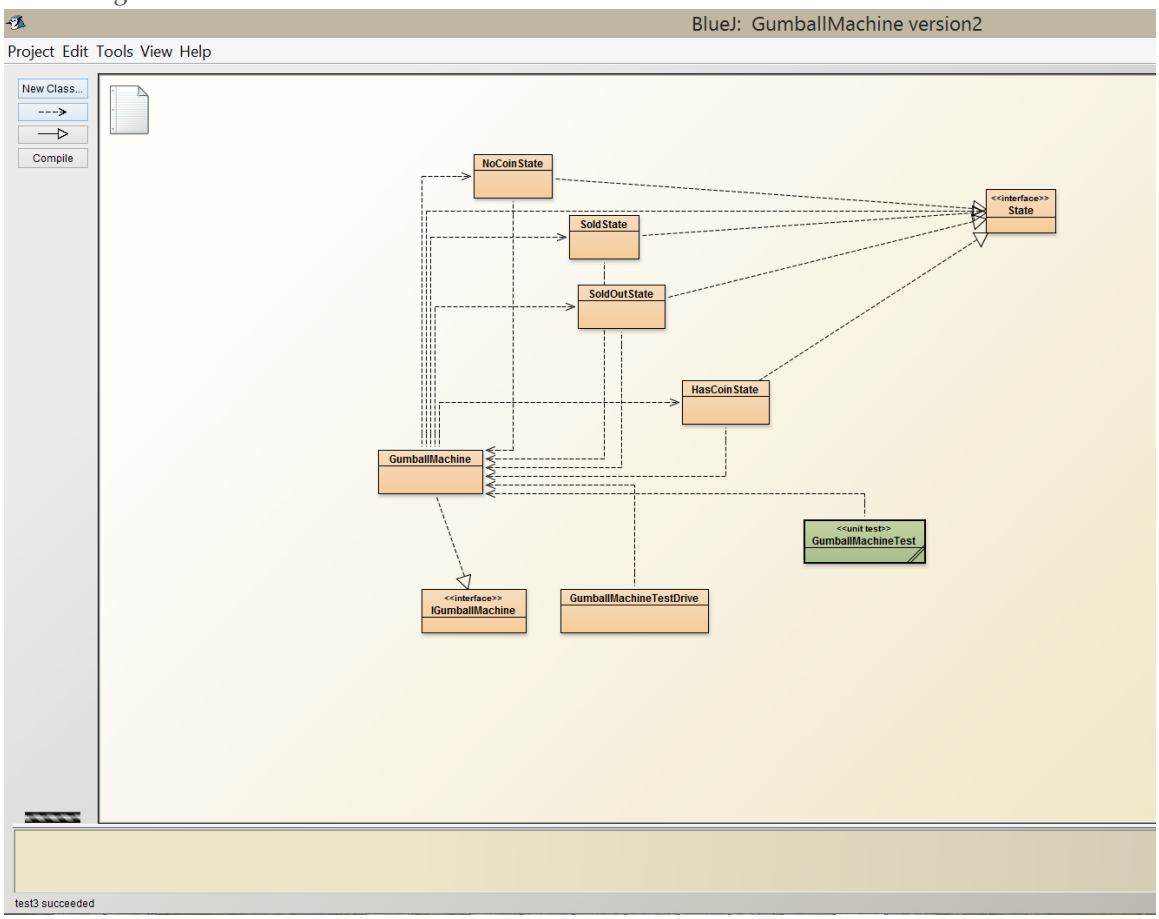
Case 2:

Runs the GumballMachine only using two quarters and checks the gumballInSlot condition with and without calling the isGumballInSlot function.



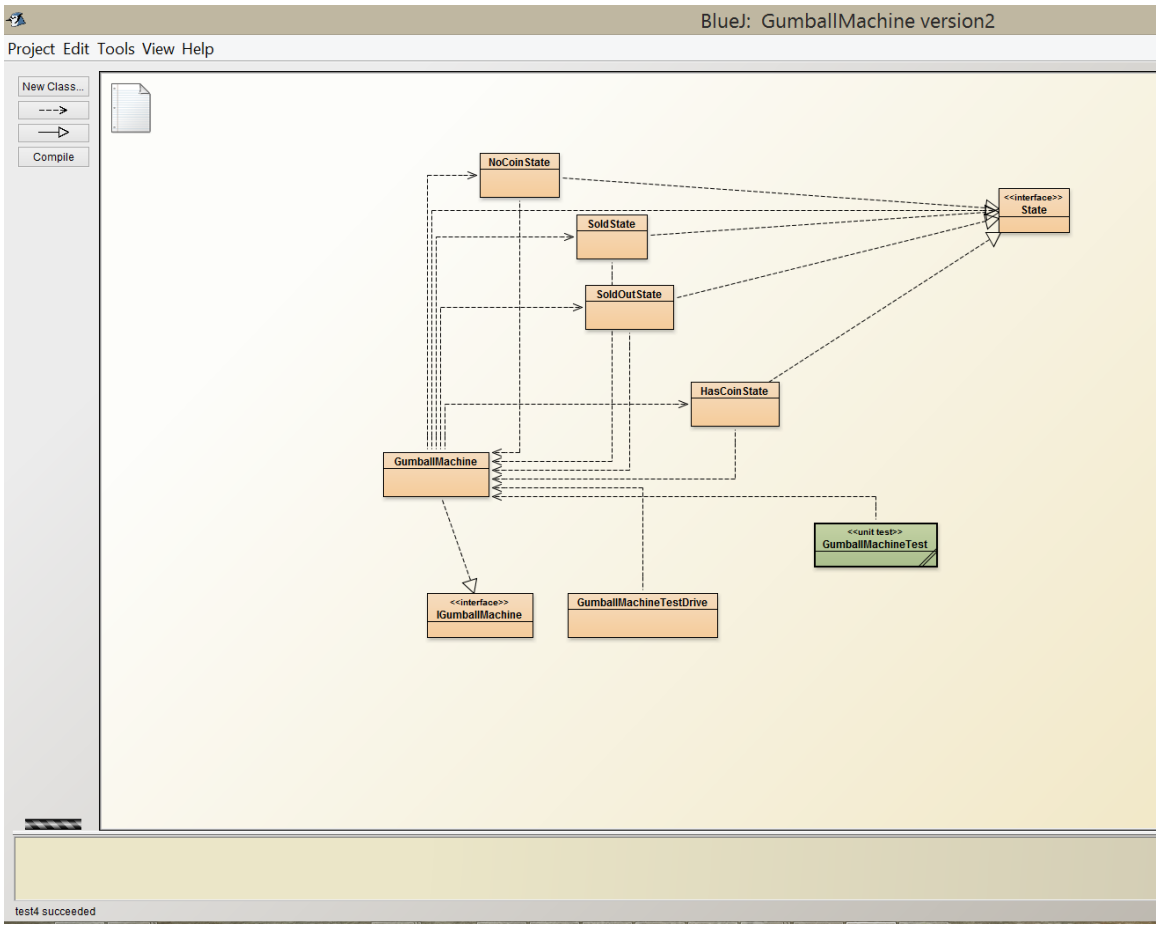
Case 3:

Initiate the Gumball Machine with 5 coins and check whether exactly only one Gumball count is reduced on turning the Crank.



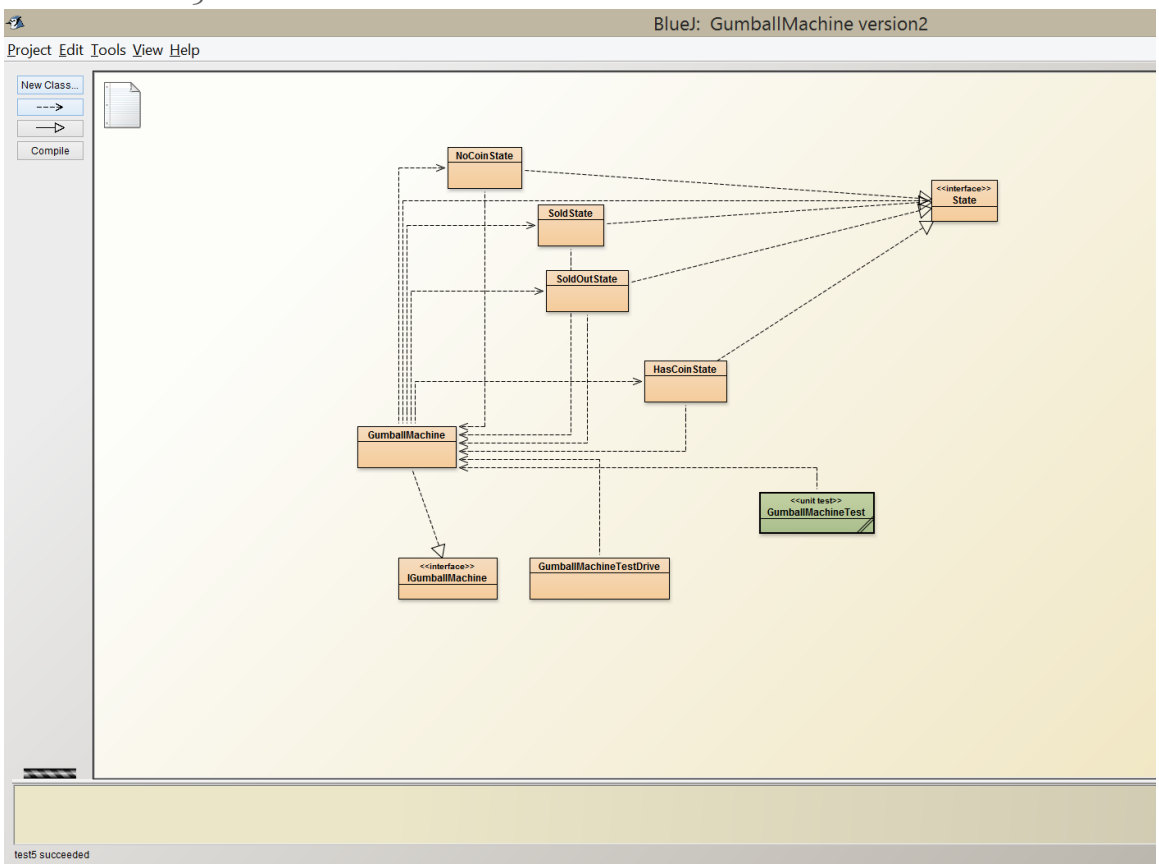
Case 4:

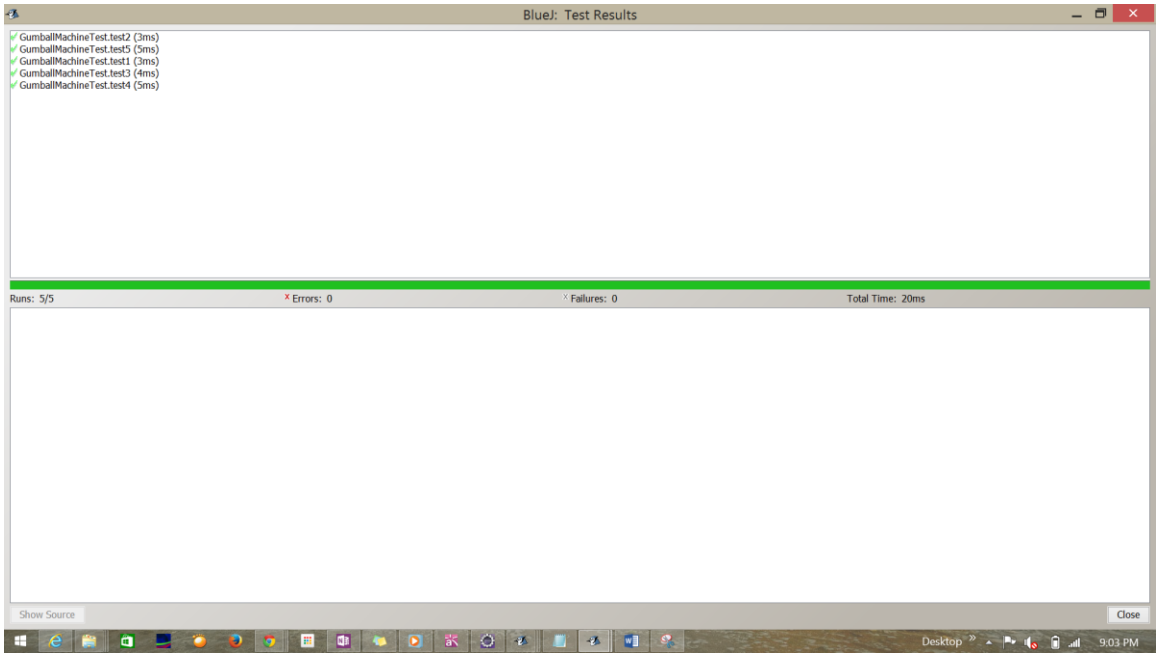
Initiates and executes the GumballMachine operation and checks whether the total coin count is reset to zero at the end of the transaction.



Case 5:

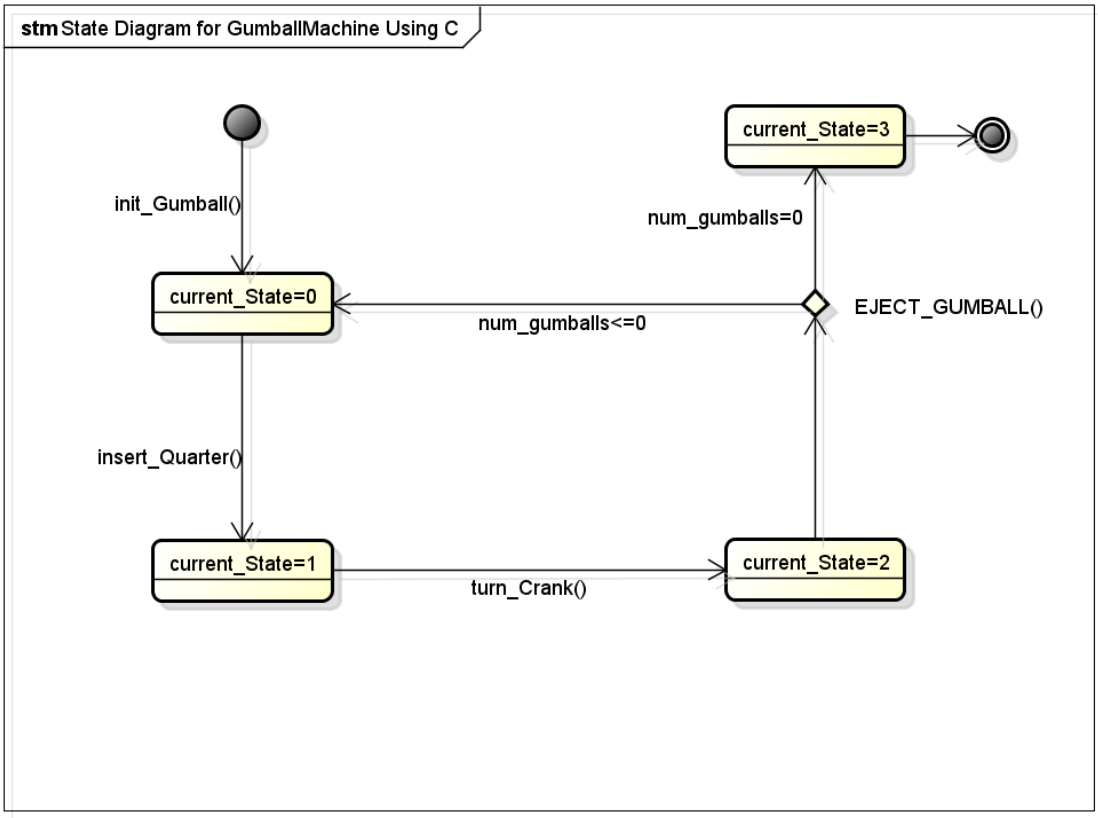
Checks whether the isGumballInSlot remains in false state after the first transaction is exectuted and a coin lesser than 50 cents is inserted for the second transaction.



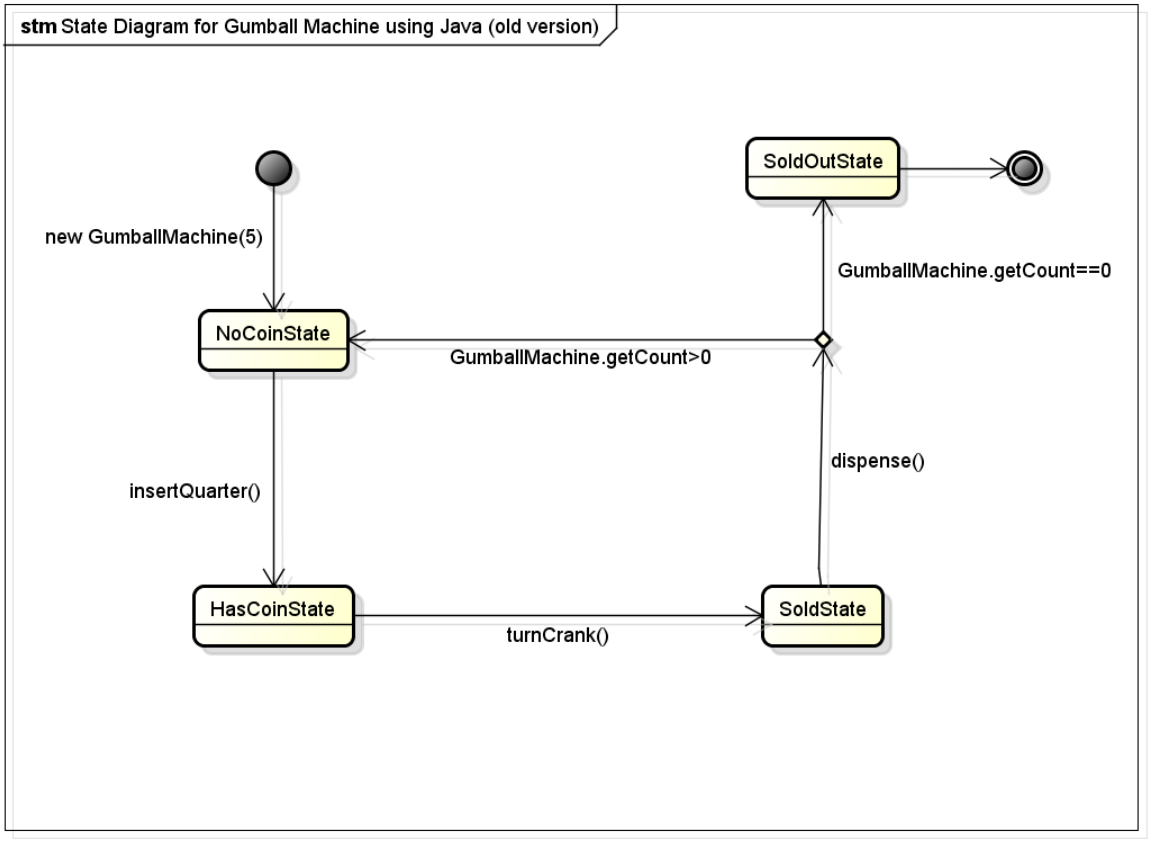


State Diagrams:

Gumball machine using C:



GumballMachine using Java (version 1 – Without Dime and Nickel)



GumballMachine Using Java (Version 2 – With Dime And Nickel)