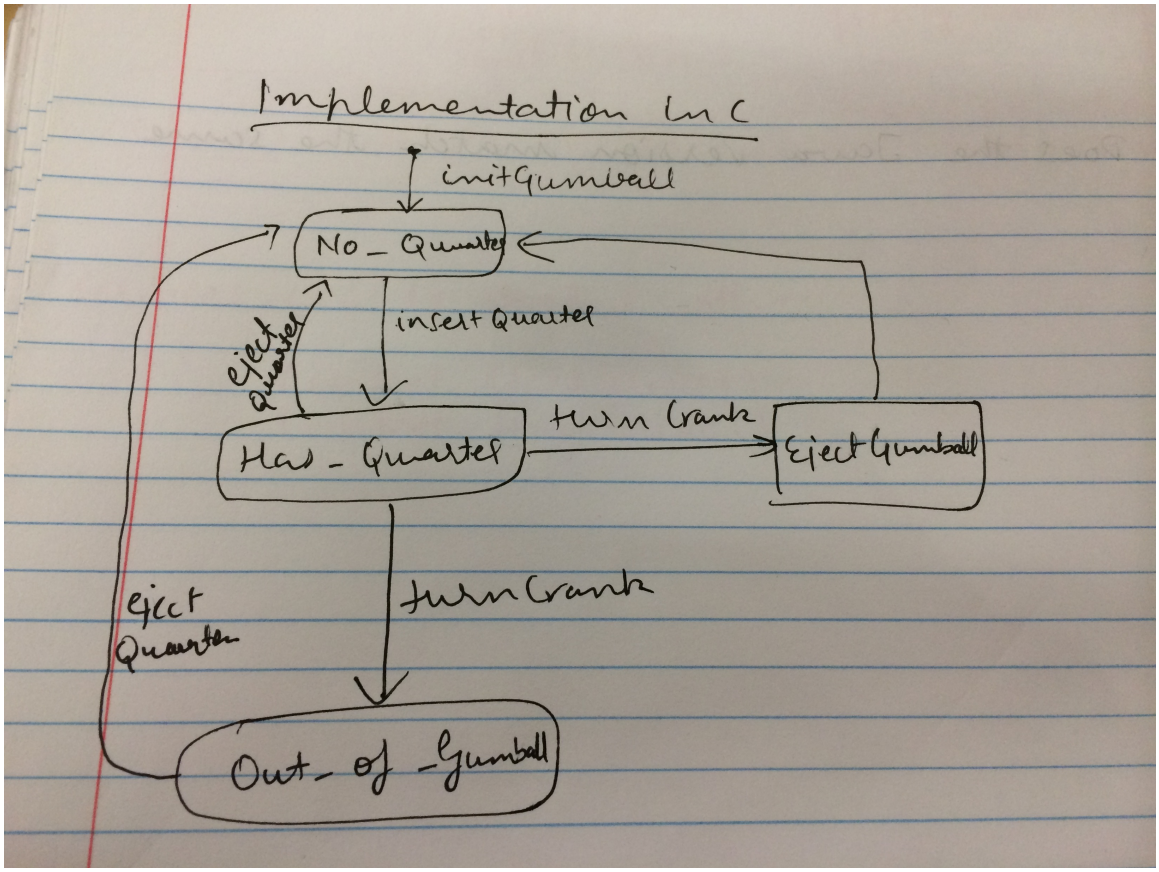
Lab2

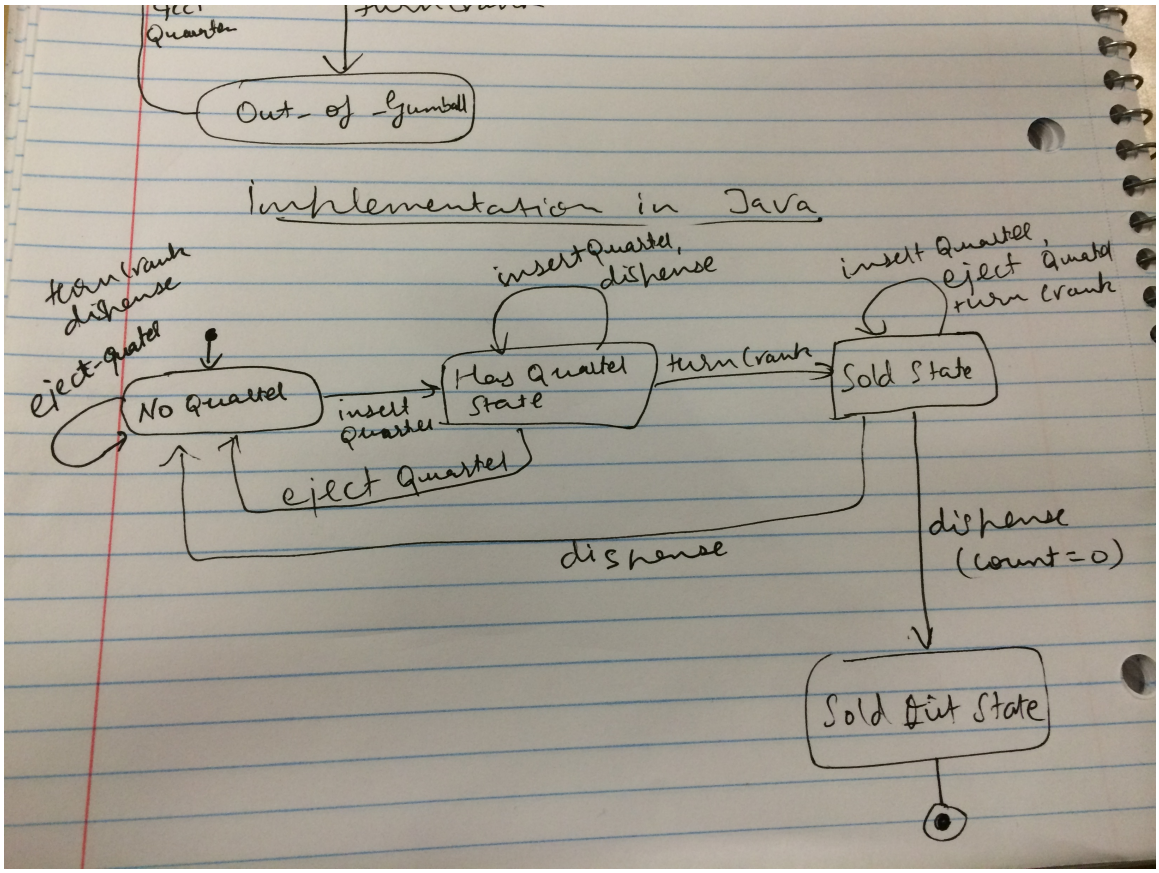
Manav Pavitra Singh
009328254

Does the Java version match the same State Machine as the C (v3) version? If not, what is the difference?

Difference:

C code for gumball state machine	Java code for the gumball state machine
Uses structures and pointers	Uses oops principle
Does not have the dispense method	Have dispense method
C is less scalable	Java is more scalable and structured





Interface as according to the modified needs:

/*New functions are added in the interface according to the modified needs and ensuring the test cases designed by others.*/

```
public interface State {

        public void insertQuarter();
        public void ejectQuarter();
        public void turnCrank();
        public void dispense();

        public void insertDime();
        public void insertNickel();
        public boolean isGumballInSlot();
        public void takeGumballFromSlot();
}
```

GumballMachineTestDrive class:

```
/* Class containing the main method */
public class GumballMachineTestDrive {

	public static void main(String[] args) {
		GumballMachine gumballMachine = new GumballMachine(5);

		System.out.println(gumballMachine);
		gumballMachine.insertQuarter();

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


	}
}
```

GumballMachine class:

```
/*  GumballMachine class containing all the method definition of the interface. The
logical flow of states start from here*/
public class GumballMachine {

	State soldOutState;
	State noFiftyState;
	State hasFiftyState;
	State soldState;

	State state = soldOutState;
	int count = 0;
	public static int amount = 0;

	public GumballMachine(int numberGumballs) {
		soldOutState = new SoldOutState(this);
		noFiftyState = new NoFiftyState(this);
		hasFiftyState = new HasFiftyState(this);
		soldState = new SoldState(this);

		this.count = numberGumballs;

		if (numberGumballs > 0) {
			state = noFiftyState;
		}
	}

	public void insertQuarter() {
```

```
		state.insertQuarter();
	}

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

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


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


	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 = noFiftyState;
	}

public State getState() {
   return state;
}
```

```
    public State getSoldOutState() {
        return soldOutState;
    }

    public State getNoFiftyState() {
        return noFiftyState;
    }

    public State getHasFiftyState() {
        return hasFiftyState;
    }

    public State getSoldState() {
        return soldState;
    }

    public boolean isGumballInSlot(){
        return state.isGumballInSlot();
    }

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

        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();
        }
}
```

HasFiftyState class:

```
/* This state is entered when the amount is equal to or greater than fifty cents */

import java.util.Random;

public class HasFiftyState implements State {
```

```
        GumballMachine gumballMachine;

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

        public void insertQuarter() {
           System.out.println("You inserted another quarter");
           GumballMachine.amount += 25;
           gumballMachine.setState(gumballMachine.getHasFiftyState());

        }

        public void insertNickel() {
                System.out.println("You inserted a nickel");
                GumballMachine.amount += 5;

                gumballMachine.setState(gumballMachine.getHasFiftyState());



        }

        public void insertDime() {
                System.out.println("You inserted a dime");
           GumballMachine.amount += 10;

                gumballMachine.setState(gumballMachine.getHasFiftyState());


        }

        public void ejectQuarter() {
                System.out.println("Quarter returned");

        }

        public void turnCrank() {
                System.out.println("You turned...");
                gumballMachine.setState(gumballMachine.getSoldState());
        }

    public void dispense() {
       System.out.println(" Flag ");
    }
```

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

        public boolean isGumballInSlot(){
            return false;
          }

        public void takeGumballFromSlot(){
          System.out.println("No gumballs left");
          }
}
```

NoFiftyState class:

```
/* If the amount is less than 50 cents, this state is entered */
public class NoFiftyState implements State {
   GumballMachine gumballMachine;

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

        public void insertQuarter() {
                System.out.println("You inserted a quarter");

                GumballMachine.amount += 25;

                if (GumballMachine.amount < 50)
                gumballMachine.setState(gumballMachine.getNoFiftyState());
                else
                gumballMachine.setState(gumballMachine.getHasFiftyState());


        }

        public void insertNickel() {
                System.out.println("You inserted a nickel");
                GumballMachine.amount += 5;

                if (GumballMachine.amount < 50)
                gumballMachine.setState(gumballMachine.getNoFiftyState());
                else
                gumballMachine.setState(gumballMachine.getHasFiftyState());
```

```
        }

        public void insertDime() {
                System.out.println("You inserted a dime");

           GumballMachine.amount += 10;

                if (GumballMachine.amount < 50)
                gumballMachine.setState(gumballMachine.getNoFiftyState());
                else
                gumballMachine.setState(gumballMachine.getHasFiftyState());


        }

        public void ejectQuarter() {

                System.out.println("You haven't inserted a quarter");
        }

        public void turnCrank() {
                System.out.println("You turned, but there's not enough amount");
         }

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

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

         public boolean isGumballInSlot(){
        return false;
     }

   public void takeGumballFromSlot(){
     System.out.println("Take your second gumball");

   }
}
```

SoldState class:

```
/* This state is entered when there is at least one gumball in the machine and the
user has turned the crank from any of the previous states. The dispenser method is
called upon then */

public class SoldState implements State {

   GumballMachine gumballMachine;
   int sold_state_dispensed;
   int remaining_balance;

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

   public void insertQuarter() {
      System.out.println("You inserted a quarter");
      GumballMachine.amount += 25;
   }

   public void insertNickel() {
      System.out.println("You inserted a nickel");
      GumballMachine.amount += 5;
   }

   public void insertDime() {
      System.out.println("You inserted a dime");
      GumballMachine.amount += 10;
   }

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

   public void turnCrank() {

         if(GumballMachine.amount >= 50 && gumballMachine.getCount()>2)
         {

            System.out.println("Collect your second gumball");
         }
         else

         {
            System.out.println("Turning twice doesn't give you second gumball");
            gumballMachine.setState(gumballMachine.getNoFiftyState());
         }
```

```
    }

    public void dispense() {

        gumballMachine.releaseBall();
        sold_state_dispensed = 1;
        System.out.println("remaining balance "+ (GumballMachine.amount - 50));
        GumballMachine.amount = GumballMachine.amount - 50;


        gumballMachine.setState(gumballMachine.getSoldState());




    }

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

    public boolean isGumballInSlot(){

          if(sold_state_dispensed == 1)
          return true;
          else
          return false;
       }

    public void takeGumballFromSlot(){
        sold_state_dispensed = 0;
         if (gumballMachine.getCount() > 0) {
            System.out.println("count ="+gumballMachine.getCount());
            gumballMachine.setState(gumballMachine.getNoFiftyState());
        } else {
            System.out.println("Oops, out of gumballs!");
            gumballMachine.setState(gumballMachine.getSoldOutState());
        }

  }
}
```

SoldOutState class:

```
/* This state is called when the gumball machine is in sold out state i.e. there isn't any gumball left in the machine */
public class SoldOutState implements State {
   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";
        }

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

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

        public boolean isGumballInSlot(){
             return false;
          }

        public void takeGumballFromSlot(){
          System.out.println("No gumballs left");
          }
```

```
}
```

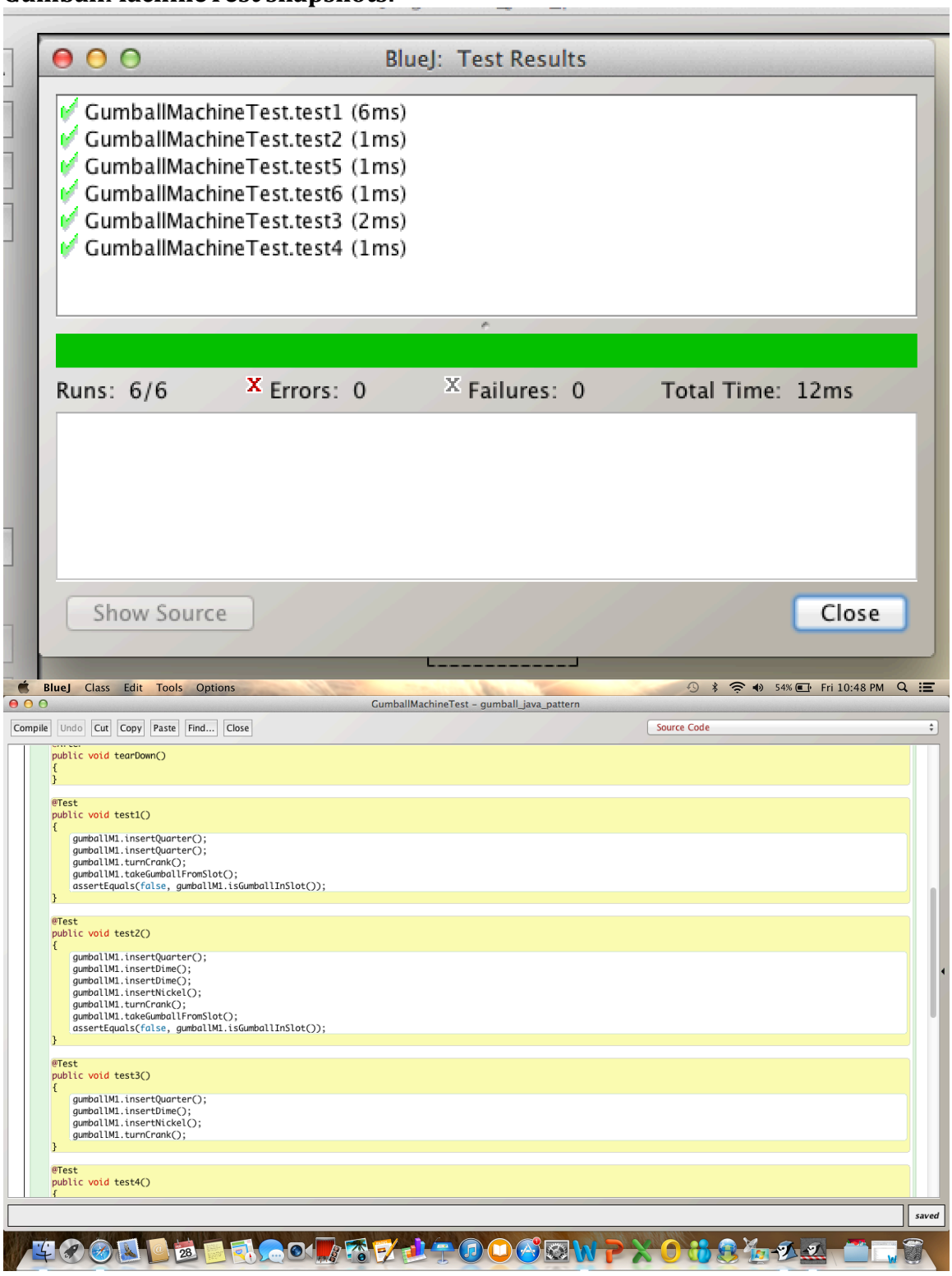
GumballMachineTest class:

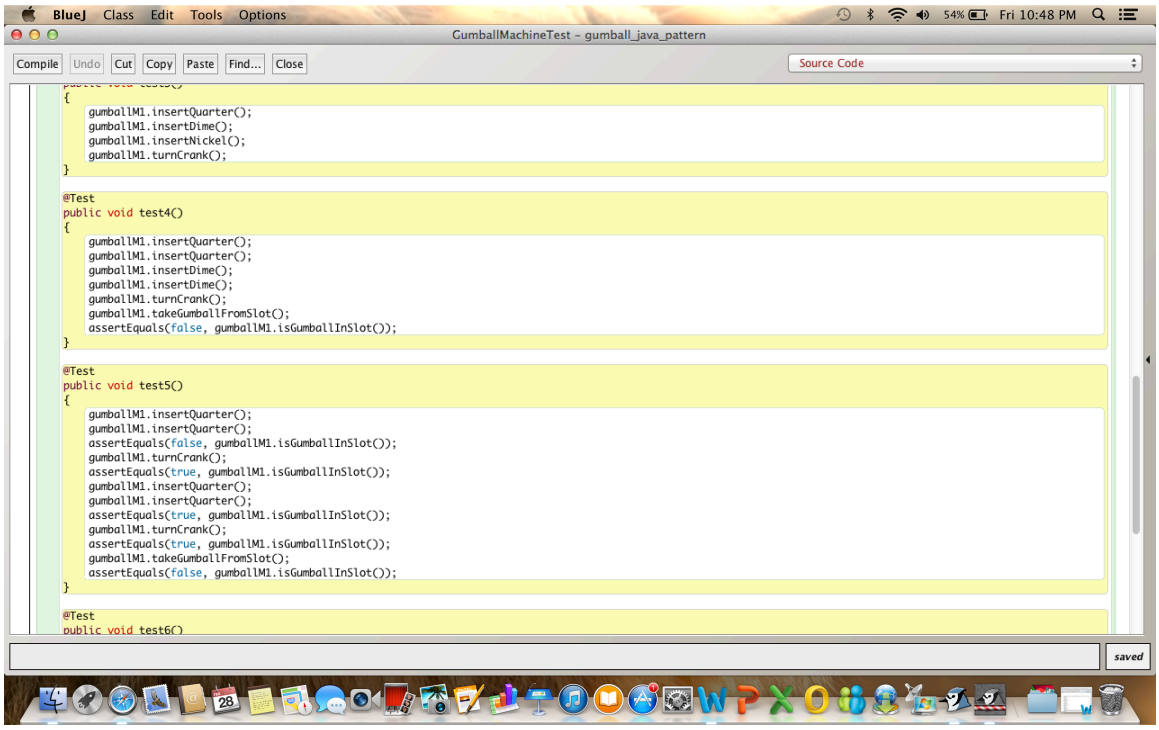
```
/* Gumball machine test class showing the various test cases*/
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

/**
 * The test class GumballMachineTest.
 *
 * @author  (your name)
 * @version (a version number or a date)
 */
public class GumballMachineTest
{
   private GumballMachine gumballM1;

   /**
    * Default constructor for test class GumballMachineTest
    */
   public GumballMachineTest()
   {
   }

   /**
    * Sets up the test fixture.
    *
    * Called before every test case method.
    */
   @Before
   public void setUp()
   {
      gumballM1 = new GumballMachine(5);
   }

   /**
    * Tears down the test fixture.
    *
    * Called after every test case method.
    */
   @After
   public void tearDown()
   {
```

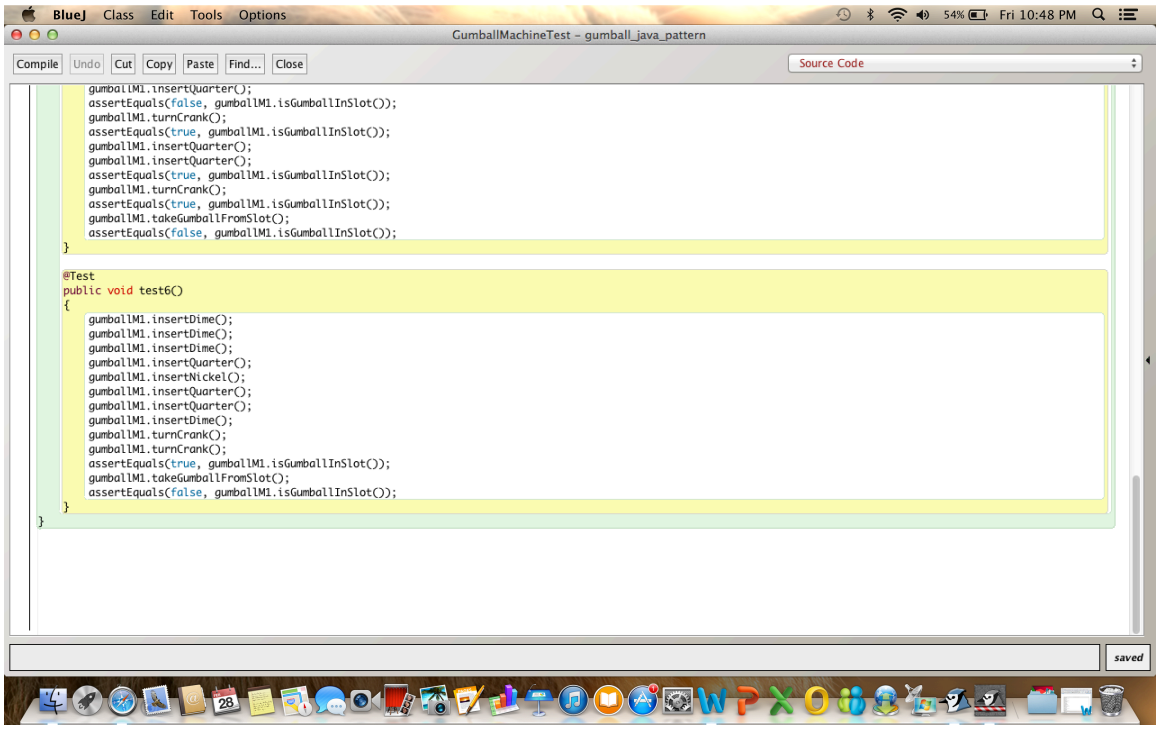
```
  }

  @Test
/* 50 cents , and the crank is turned, since the takeGumballFromSlot() is called,
hence isGumballInSlot returns false */
  public void test1()
  {
    gumballM1.insertQuarter();
    gumballM1.insertQuarter();
    gumballM1.turnCrank();
    gumballM1.takeGumballFromSlot();
    assertEquals(false, gumballM1.isGumballInSlot());
  }

  @Test
/* 50 cents in different coins, same behavior */
  public void test2()
  {
    gumballM1.insertQuarter();
    gumballM1.insertDime();
    gumballM1.insertDime();
    gumballM1.insertNickel();
    gumballM1.turnCrank();
    gumballM1.takeGumballFromSlot();
    assertEquals(false, gumballM1.isGumballInSlot());
  }

  @Test
/* less than 50 cents, as per the requirement, it remembers the amount but does not
eject the gumball */
  public void test3()
  {
    gumballM1.insertQuarter();
    gumballM1.insertDime();
    gumballM1.insertNickel();
    gumballM1.turnCrank();
  }

  @Test
/* more than 50 cents, gumball is ejected and change is returned in this case, which
fulfills our another requirement */
  public void test4()
  {
    gumballM1.insertQuarter();
    gumballM1.insertQuarter();
    gumballM1.insertDime();
```

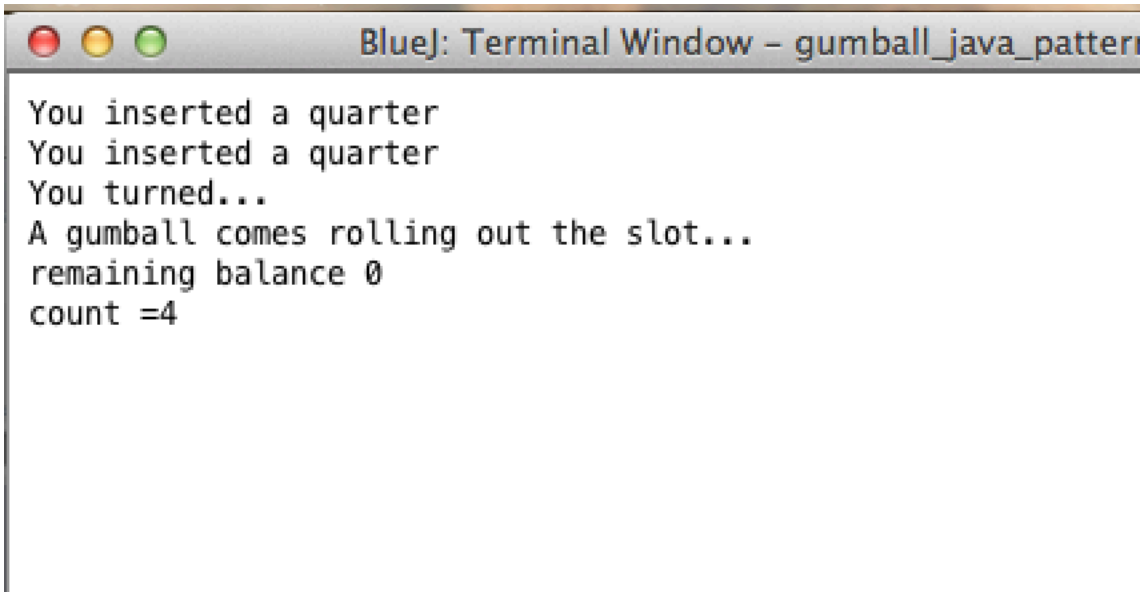
```
        gumballM1.insertDime();
        gumballM1.turnCrank();
        gumballM1.takeGumballFromSlot();
        assertEquals(false, gumballM1.isGumballInSlot());
    }

    @Test
/* 100 cents, This test case fulfills the requirement of isGumballInSlot() function,
which returns accordingly if the take gumball isn't called first. The output gives you
two gumballs in this case */
    public void test5()
    {
        gumballM1.insertQuarter();
        gumballM1.insertQuarter();
        assertEquals(false, gumballM1.isGumballInSlot());
        gumballM1.turnCrank();
        assertEquals(true, gumballM1.isGumballInSlot());
        gumballM1.insertQuarter();
        gumballM1.insertQuarter();
        assertEquals(true, gumballM1.isGumballInSlot());
        gumballM1.turnCrank();
        assertEquals(true, gumballM1.isGumballInSlot());
        gumballM1.takeGumballFromSlot();
        assertEquals(false, gumballM1.isGumballInSlot());
    }

    @Test
/* more than 100, crank turned consecutive two times after inserting all the
amount, gives two gumballs and the change */
    public void test6()
    {
        gumballM1.insertDime();
        gumballM1.insertDime();
        gumballM1.insertDime();
        gumballM1.insertQuarter();
        gumballM1.insertNickel();
        gumballM1.insertQuarter();
        gumballM1.insertQuarter();
        gumballM1.insertDime();
        gumballM1.turnCrank();
        gumballM1.turnCrank();
        assertEquals(true, gumballM1.isGumballInSlot());
        gumballM1.takeGumballFromSlot();
        assertEquals(false, gumballM1.isGumballInSlot());
    }
}
```

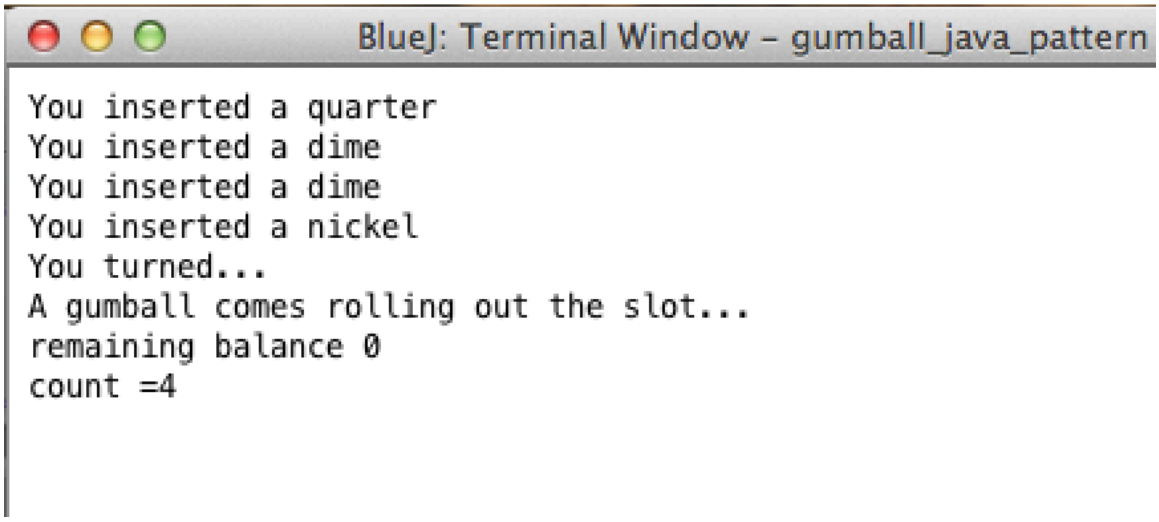
GumballMachineTest snapshots:

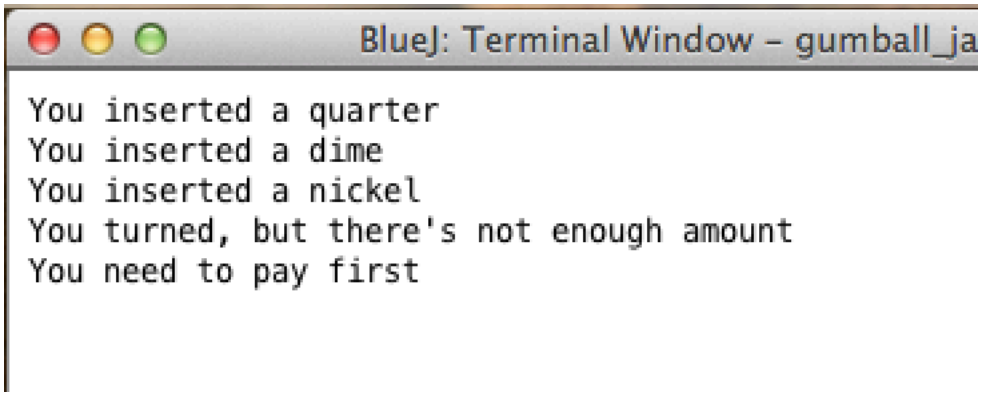


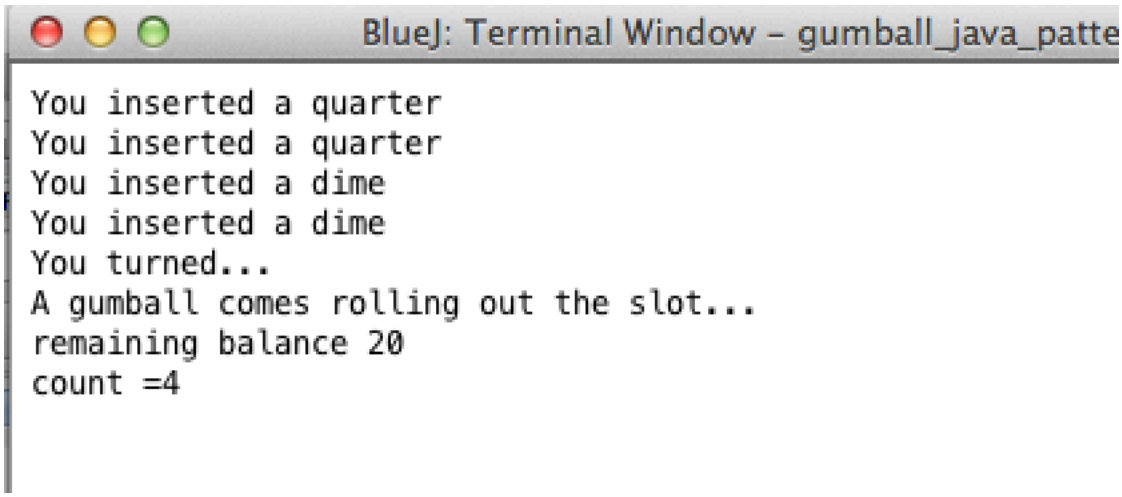

BlueJ Class Edit Tools Options
54% Fri 10:48 PM
GumballMachineTest - gumball_java_pattern
Compile Undo Cut Copy Paste Find... Close
Source Code
{
gumballM1.insertQuarter();
gumballM1.insertDime();
gumballM1.insertNickel();
gumballM1.turnCrank();
}
@Test
public void test4()
{
gumballM1.insertQuarter();
gumballM1.insertQuarter();
gumballM1.insertDime();
gumballM1.insertDime();
gumballM1.turnCrank();
gumballM1.takeGumballFromSlot();
assertEquals(false, gumballM1.isGumballInSlot());
}
@Test
public void test5()
{
gumballM1.insertQuarter();
gumballM1.insertQuarter();
assertEquals(false, gumballM1.isGumballInSlot());
gumballM1.turnCrank();
assertEquals(true, gumballM1.isGumballInSlot());
gumballM1.insertQuarter();
gumballM1.insertQuarter();
assertEquals(true, gumballM1.isGumballInSlot());
gumballM1.turnCrank();
assertEquals(true, gumballM1.isGumballInSlot());
gumballM1.takeGumballFromSlot();
assertEquals(false, gumballM1.isGumballInSlot());
}
@Test
public void test6()
saved

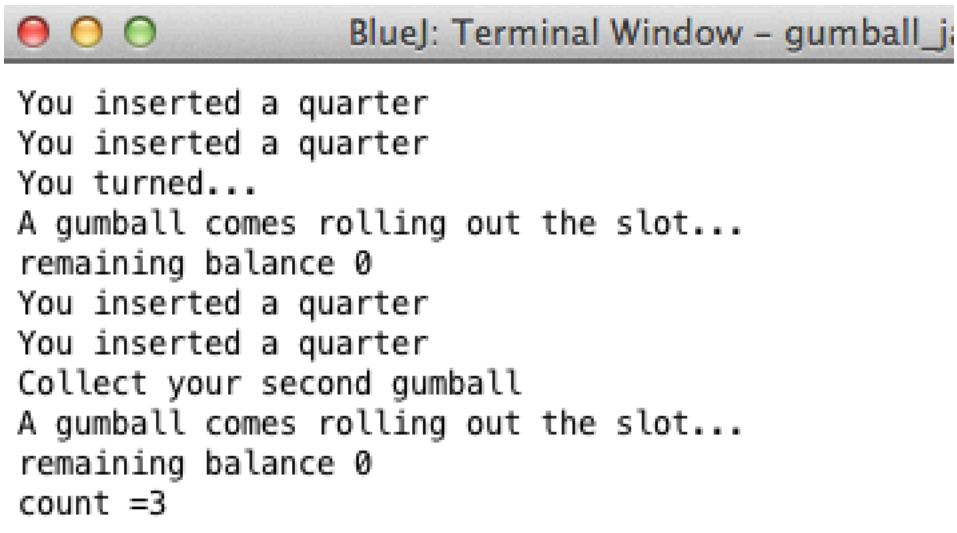

BlueJ Class Edit Tools Options
54% Fri 10:48 PM
GumballMachineTest - gumball_java_pattern
Compile Undo Cut Copy Paste Find... Close
Source Code
gumballM1.insertQuarter();
assertEquals(false, gumballM1.isGumballInSlot());
gumballM1.turnCrank();
assertEquals(true, gumballM1.isGumballInSlot());
gumballM1.insertQuarter();
gumballM1.insertQuarter();
assertEquals(true, gumballM1.isGumballInSlot());
gumballM1.turnCrank();
assertEquals(true, gumballM1.isGumballInSlot());
gumballM1.takeGumballFromSlot();
assertEquals(false, gumballM1.isGumballInSlot());
}
@Test
public void test6()
{
gumballM1.insertDime();
gumballM1.insertDime();
gumballM1.insertDime();
gumballM1.insertQuarter();
gumballM1.insertNickel();
gumballM1.insertQuarter();
gumballM1.insertQuarter();
gumballM1.insertDime();
gumballM1.turnCrank();
gumballM1.turnCrank();
assertEquals(true, gumballM1.isGumballInSlot());
gumballM1.takeGumballFromSlot();
assertEquals(false, gumballM1.isGumballInSlot());
}
}
saved

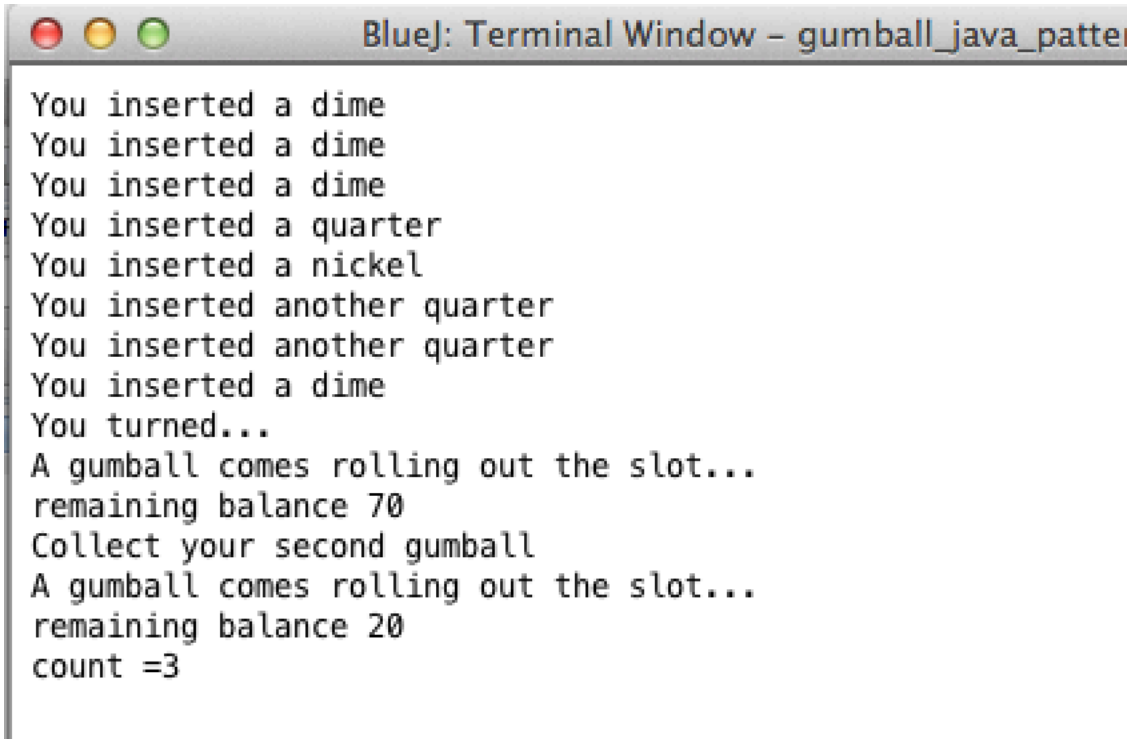

BlueJ: Terminal Window – gumball_java_patter
You inserted a quarter
You inserted a quarter
You turned...
A gumball comes rolling out the slot...
remaining balance 0
count =4


BlueJ: Terminal Window – gumball_java_pattern
You inserted a quarter
You inserted a dime
You inserted a dime
You inserted a nickel
You turned...
A gumball comes rolling out the slot...
remaining balance 0
count =4


BlueJ: Terminal Window – gumball_ja
You inserted a quarter
You inserted a dime
You inserted a nickel
You turned, but there's not enough amount
You need to pay first


BlueJ: Terminal Window – gumball_java_patte
You inserted a quarter
You inserted a quarter
You inserted a dime
You inserted a dime
You turned...
A gumball comes rolling out the slot...
remaining balance 20
count =4


BlueJ: Terminal Window – gumball_j
You inserted a quarter
You inserted a quarter
You turned...
A gumball comes rolling out the slot...
remaining balance 0
You inserted a quarter
You inserted a quarter
Collect your second gumball
A gumball comes rolling out the slot...
remaining balance 0
count =3


BlueJ: Terminal Window – gumball_java_patte
You inserted a dime
You inserted a dime
You inserted a dime
You inserted a quarter
You inserted a nickel
You inserted another quarter
You inserted another quarter
You inserted a dime
You turned...
A gumball comes rolling out the slot...
remaining balance 70
Collect your second gumball
A gumball comes rolling out the slot...
remaining balance 20
count =3