CMPE 202
Lab #2

Gumball State Machine Lab

The Gumball Machines

Implementation in C (v1) - Typical Solution

main.c

```c
#include <stdio.h>

int has_quarter = 0 ;
int num_gumballs = 1 ;

void insert_quarter( int coin )
{
    if ( coin == 25 )
       has_quarter = 1 ;
    else
        has_quarter = 0 ;
}

void turn_crank()
{
    if ( has_quarter )
    {
        if ( num_gumballs > 0 )
        {
            num_gumballs-- ;
            has_quarter = 0 ;
            printf( "Thanks for your quarter.  Gumball Ejected!\n" ) ;
        }
        else
        {
            printf( "No More Gumballs!  Sorry, can't return your quarter.\n" ) ;
        }
    }
    else
    {
        printf( "Please insert a quarter\n" ) ;
    }
}

int main(int argc, char **argv)
{
    printf("Simple Gumball Machine - Version 1\n");
    insert_quarter( 25 ) ;
    turn_crank() ;
    insert_quarter( 25 ) ;
    turn_crank() ;
    insert_quarter( 10 ) ;
    turn_crank() ;
    return 0;
}
```

Implementation in C (v2) - Abstract Data Type

gumball.h

```c

typedef struct
{
    int num_gumballs ;
    int has_quarter ;
} GUMBALL ;

extern void init_gumball( GUMBALL *ptr, int size ) ;
extern void turn_crank( GUMBALL *ptr );
extern void insert_quarter( GUMBALL *ptr, int coin );
```

*main.c

```c
#include <stdio.h>
#include "gumball.h"

int main(int argc, char **argv)
{
    GUMBALL m1[1] ;
    GUMBALL m2[1] ;

    /* init gumball machines */
    init_gumball( m1, 1 ) ;
    init_gumball( m2, 10 ) ;

    printf("Simple Gumball Machine - Version 2\n");

    insert_quarter( m1, 25 ) ;
    turn_crank( m1 ) ;
    insert_quarter( m1, 25 ) ;
    turn_crank( m1 ) ;
    insert_quarter( m1, 10 ) ;
    turn_crank( m1 ) ;

    insert_quarter( m2, 25 ) ;
    turn_crank( m2 ) ;
    insert_quarter( m2, 25 ) ;
    turn_crank( m2 ) ;
    insert_quarter( m1, 10 ) ;
    turn_crank( m2 ) ;

    return 0;
}
```

gumball.c

```c

#include <stdio.h>
#include "gumball.h"

void init_gumball( GUMBALL *ptr, int size )
{
    ptr->num_gumballs = size ;
    ptr->has_quarter = 0 ;
}

void turn_crank( GUMBALL *ptr )
{
    if ( ptr->has_quarter )
    {
        if ( ptr->num_gumballs > 0 )
        {
            ptr->num_gumballs-- ;
            ptr->has_quarter = 0 ;
            printf( "Thanks for your quarter.  Gumball Ejected!\n" ) ;
        }
        else
        {
            printf( "No More Gumballs!  Sorry, can't return your quarter.\n" ) ;
        }
    }
    else
    {
        printf( "Please insert a quarter\n" ) ;
    }
}

void insert_quarter( GUMBALL *ptr, int coin )
{
    if ( coin == 25 )
        ptr->has_quarter = 1 ;
    else
        ptr->has_quarter = 0 ;
}
```

Implementation in C (v3) - State Transition Tables

gumball.h

```c

typedef struct
{
    int num_gumballs ;
    int has_quarter ;
    int current_state ; /* 0 = OUT_OF_GUMBALL,  1 = NO_QTR, 2 = HAS_QTR, 3 = EJECT_GUMBALL */
} GUMBALL ;

extern void init_gumball( GUMBALL *ptr, int size ) ;
extern void turn_crank( GUMBALL *ptr );
extern void insert_quarter( GUMBALL *ptr );
extern void eject_quarter( GUMBALL *ptr );

```

main.c

```c
#include <stdio.h>
#include "gumball.h"

int main(int argc, char **argv)
{
    GUMBALL m1[1] ;
    GUMBALL m2[1] ;

    /* init gumball machines */
    init_gumball( m1, 1 ) ;
    init_gumball( m2, 10 ) ;

    printf("Simple Gumball Machine - Version 3\n");

    insert_quarter( m1 ) ;
    turn_crank( m1 ) ;
    insert_quarter( m1 ) ;
    turn_crank( m1 ) ;
    insert_quarter( m1 ) ;
    turn_crank( m1 ) ;

    insert_quarter( m2 ) ;
    turn_crank( m2 ) ;
    turn_crank( m2 ) ;
    insert_quarter( m2 ) ;
    eject_quarter( m2 ) ;

    return 0;
}
```

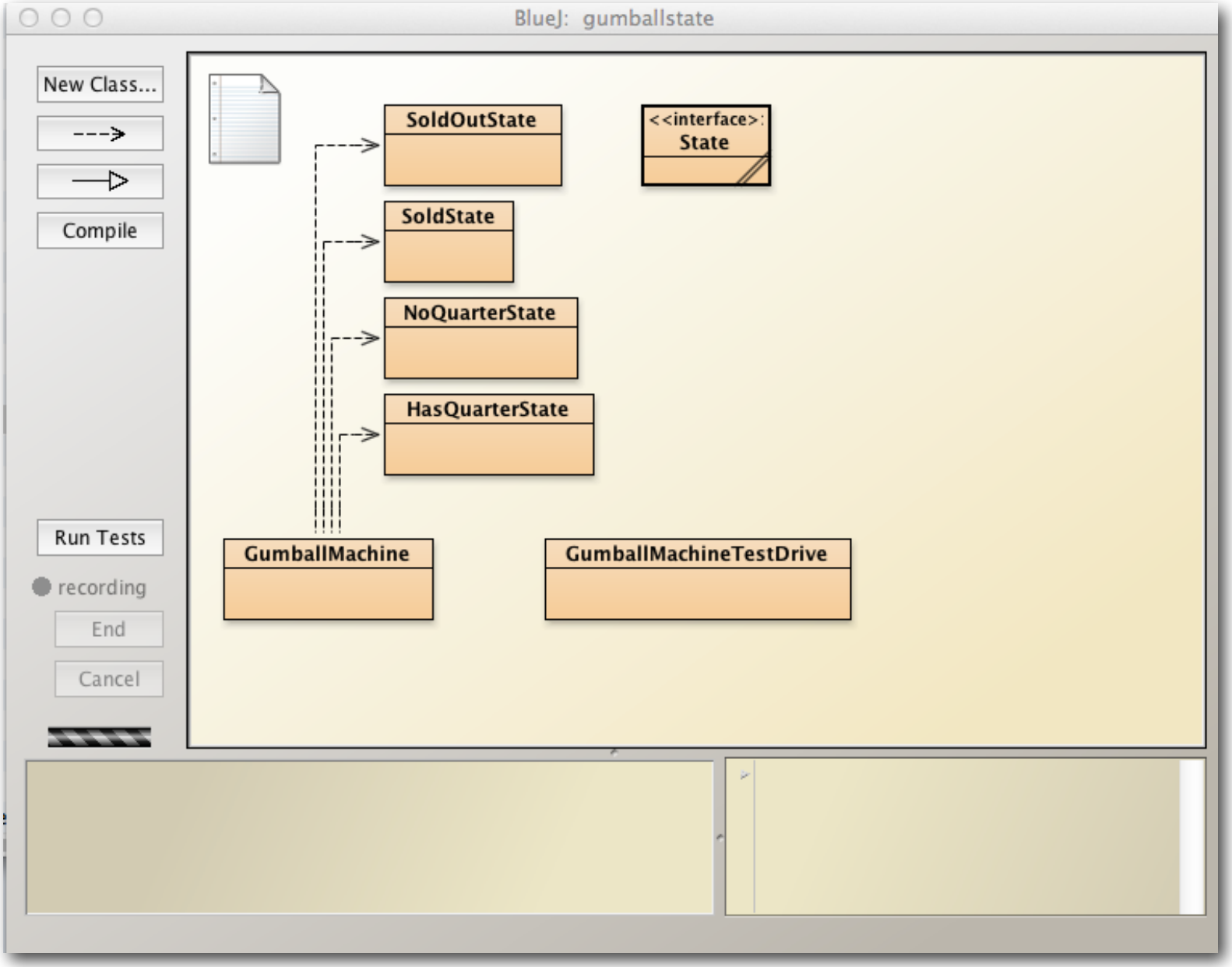
Implementation in C (v3) - State Transition Tables

gumball.c

```c

#include <stdio.h>
#include "gumball.h"

void OUT_OF_GUMBALL( GUMBALL *ptr )
{
    printf( "No More Gumballs! \n" ) ;
    ptr->current_state = 0 ; /* No Gumballs! */
}

void NO_QTR( GUMBALL *ptr )
{
    printf( "No Quarter Inserted! \n" ) ;
    ptr->current_state = 2 ; /* No Quarter! */
}

void HAS_QTR( GUMBALL *ptr )
{
    printf( "Quarter Inserted! \n" ) ;
    ptr->current_state = 1 ; /* Has Quarter! */
    ptr->has_quarter = 1 ;
}

void EJECT_GUMBALL( GUMBALL *ptr )
{
    if( ptr->num_gumballs>0 )
    {
        printf( "Your Gumball has been ejected!\n" ) ;
        ptr->has_quarter = 0 ;
        ptr->num_gumballs-- ;
    }
    if( ptr->num_gumballs <= 0 )
        ptr->current_state = 0 ; /* Out of Gumballs! */
    else
        ptr->current_state = 2 ; /* No Quarter */
}

void (*machine[3][4]) (GUMBALL *ptr) = {
                        /* OUT_OF_GUMBALL,     HAS_QTR,        NO_QTR,     EJECT_GUMBALL  */
    /* crank */         {OUT_OF_GUMBALL,       EJECT_GUMBALL,  NO_QTR,     0            },
    /* insert qtr */    {OUT_OF_GUMBALL,       HAS_QTR,        HAS_QTR,    0            },
    /* eject qtr */     {OUT_OF_GUMBALL,       NO_QTR,         NO_QTR,     0            }
} ;

```

```c

void init_gumball( GUMBALL *ptr, int size )
{
    ptr->num_gumballs = size ;
    ptr->has_quarter = 0 ;
    ptr->current_state = 2 ; /* No Quarter */
}

void turn_crank( GUMBALL *ptr )
{
    machine[0][ptr->current_state](ptr) ;
}

void insert_quarter( GUMBALL *ptr )
{
    machine[1][ptr->current_state](ptr) ;
}

void eject_quarter( GUMBALL *ptr )
{
    machine[2][ptr->current_state](ptr) ;
}

```

Implementation in Java using State Pattern



Implementation in Java using State Pattern

GumballMachine – gumballstate

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```java
package gumballstate;

public class GumballMachine {

    State soldOutState;
    State noQuarterState;
    State hasQuarterState;
    State soldState;

    State state = soldOutState;
    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;
    }

```

saved

Implementation in Java using State Pattern

GumballMachine – gumballstate

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```java
    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");
```

saved

Implementation in Java using State Pattern

State – gumballstate

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```java
package gumballstate;

public interface State {

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

Class compiled – no syntax errors

saved

Implementation in Java using State Pattern

HasQuarterState – gumballstate

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```java
package gumballstate;

import java.util.Random;

public class HasQuarterState implements State {
    GumballMachine gumballMachine;

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

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

    public void ejectQuarter() {
        System.out.println("Quarter returned");
        gumballMachine.setState(gumballMachine.getNoQuarterState());
    }

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

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

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

saved

Implementation in Java using State Pattern

GumballMachineTestDrive – gumballstate

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```java
package gumballstate;

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);
    }
}
```

saved

Greenfoot Gumball Machine State Pattern Lab

- Review the code for the Java Implementation (Java State Pattern) and the State Machine in C (v3). Draw State Diagrams for both versions in Astah UML.

- Modify the “Java State Pattern Version” version to support the new requirement as follows:

  - New Requirements:
    - Machine accepts Dimes and Nickels (in addition to Quarters)
    - Cost for a Gumball has increased to 50 cents

Greenfoot Gumball Machine State jUnit Lab

1. The "**GumballMachine**" class must be modified to implement the following interface (as discussed in class):

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

2. You must implement at least 5 test cases for the jUnit Test Class that tests GumballMachine against the contract defined by the IGumballMachine Interface.

3. Please note that there are some conditions not explicitly covered by the interface. As such, please write your code with the following assumptions to make sure that jUnit test cases written by other students will test your code with the same assumptions.

- Given an inventory of 2 or more gumballs, when I insert more than 50 cents into the Gumball Machine and turn the crank, then only one Gumball is returned in the gumball slot along with the change (the amount > 50 cents).

- Given an inventory of 2 or more gumballs, when I insert 50 cents and turn the crank to get a gumball in the slot, and then proceed to insert 50 more cents before taking the gumball and turn the crank a second time, then there should be two gumballs in the slot and isGumballInSlot returns "true" and when I take the gumballs from the slot, both of the gumballs will be removed and isGumballInSlot would then return "false".

- Given that less than 50 cents is currently in the Gumball Machine, and I insert a coin (quarter, dime, nickel) such that the total amount is still less than 50 cents, when I turn the crank on the Gumball Machine the Gumball Machine remembers how much money I have put into the machine but does not eject a gumball.