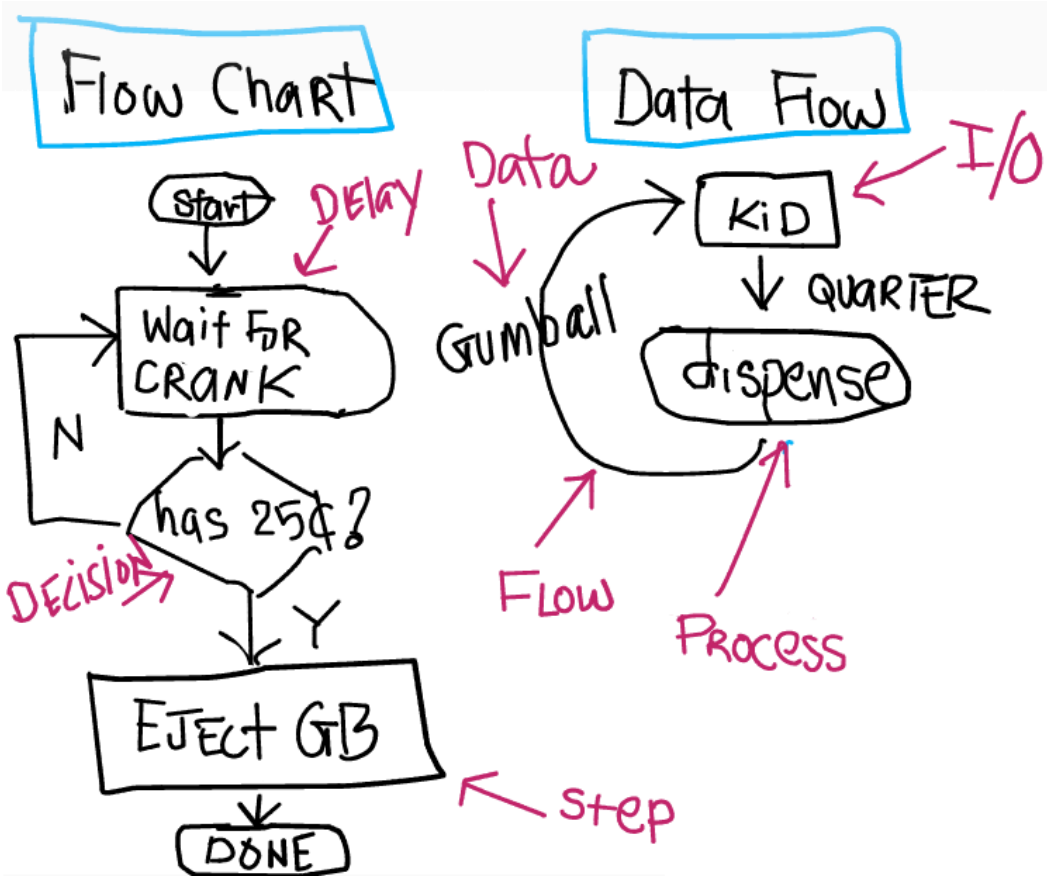
CMPE 202

"Classical" Procedural Programming
vs.
Object-Oriented Programming

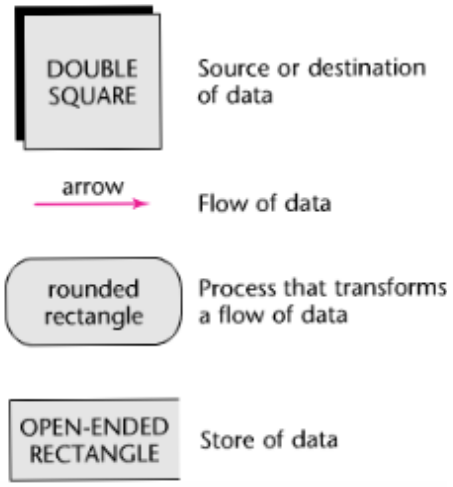
The Gumball Machine

Basic Requirements:

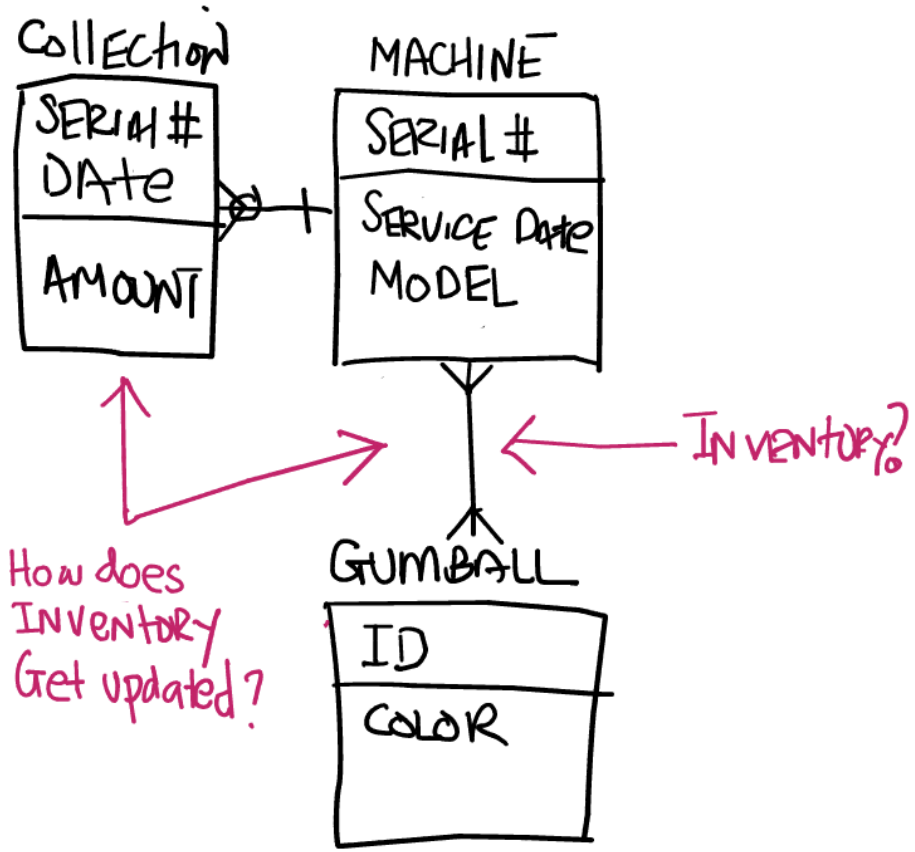
1. Insert a quarter, crank, get a gumball
2. If no more gumball, lose the quarter



	Terminator	A start or stop point in a process
	Process	A computation step or an operation
	Decision	Decision making or branching
	Delay	A waiting period
	Data I/O	Input or output operation

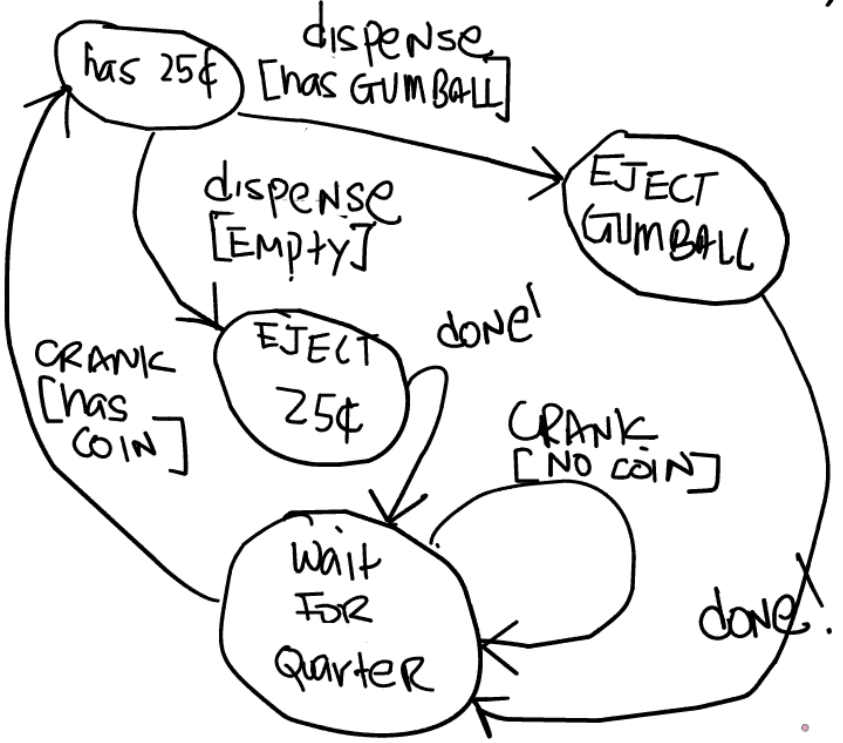


Data Model



MACHINE LOGIC AS FSM.

(Finite State Machine)



A "NICE" GUMBALL MACHINE!

Implementation in C (version 1) - 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;
}
```

Problems:

1. Global variables are not a good practice! Why?
2. Can only have one gumball machine at a time.
3. What about a 50¢ gumball machine?
4. How about the "nice" gumball machine that returns coins?

Implementation in C (version 2) - Abstract Data Type

gumball.h

```
1
2   typedef struct
3   {
4       int num_gumballs ;
5       int has_quarter ;
6   } GUMBALL ;
7
8   extern void init_gumball( GUMBALL *ptr, int size ) ;
9   extern void turn_crank( GUMBALL *ptr );
10  extern void insert_quarter( GUMBALL *ptr, int coin );
```

*main.c

```
1   #include <stdio.h>
2   #include "gumball.h"
3
4   int main(int argc, char **argv)
5   {
6       GUMBALL m1[1] ;
7       GUMBALL m2[1] ;
8
9       /* init gumball machines */
10      init_gumball( m1, 1 ) ;
11      init_gumball( m2, 10 ) ;
12
13      printf("Simple Gumball Machine - Version 2\n");
14
15      insert_quarter( m1, 25 ) ;
16      turn_crank( m1 ) ;
17      insert_quarter( m1, 25 ) ;
18      turn_crank( m1 ) ;
19      insert_quarter( m1, 10 ) ;
20      turn_crank( m1 ) ;
21
22      insert_quarter( m2, 25 ) ;
23      turn_crank( m2 ) ;
24      insert_quarter( m2, 25 ) ;
25      turn_crank( m2 ) ;
26      insert_quarter( m1, 10 ) ;
27      turn_crank( m2 ) ;
28
29      return 0;
30  }
```

gumball.c

```
1
2   #include <stdio.h>
3   #include "gumball.h"
4
5   void init_gumball( GUMBALL *ptr, int size )
6   {
7       ptr->num_gumballs = size ;
8       ptr->has_quarter = 0 ;
9   }
10
11  void turn_crank( GUMBALL *ptr )
12  {
13      if ( ptr->has_quarter )
14      {
15          if ( ptr->num_gumballs > 0 )
16          {
17              ptr->num_gumballs-- ;
18              ptr->has_quarter = 0 ;
19              printf( "Thanks for your quarter.  Gumball Ejected!\n" ) ;
20          }
21          else
22          {
23              printf( "No More Gumballs!  Sorry, can't return your quarter.\n" ) ;
24          }
25      }
26      else
27      {
28          printf( "Please insert a quarter\n" ) ;
29      }
30  }
31
32  void insert_quarter( GUMBALL *ptr, int coin )
33  {
34      if ( coin == 25 )
35          ptr->has_quarter = 1 ;
36      else
37          ptr->has_quarter = 0 ;
38  }
```

ADT --> think of the GB Machine in terms of its operations. (insert quarter, turn crank)
Data Encapsulation --> hide implementation details. Not supported in C Language

Implementation in Java (BlueJ)

```
1
2  public class GumballMachine
3  {
4
5      private int num_gumballs;
6      private boolean has_quarter;
7
8      public GumballMachine( int size )
9      {
10         // initialise instance variables
11         this.num_gumballs = size;
12         this.has_quarter = false;
13     }
14
15     public void insertQuarter(int coin)
16     {
17         if ( coin == 25 )
18             this.has_quarter = true ;
19         else
20             this.has_quarter = false ;
21     }
22
23     public void turnCrank()
24     {
25         if ( this.has_quarter )
26         {
27             if ( this.num_gumballs > 0 )
28             {
29                 this.num_gumballs-- ;
30                 this.has_quarter = false ;
31                 System.out.println( "Thanks for your quarter.  Gumball Ejected!" ) ;
32             }
33             else
34             {
35                 System.out.println( "No More Gumballs!  Sorry, can't return your quarter." ) ;
36             }
37         }
38         else
39         {
40             System.out.println( "Please insert a quarter" ) ;
41         }
42     }
43 }
44
```

Native support for
Encapsulation
in OO Languages
(like Java)