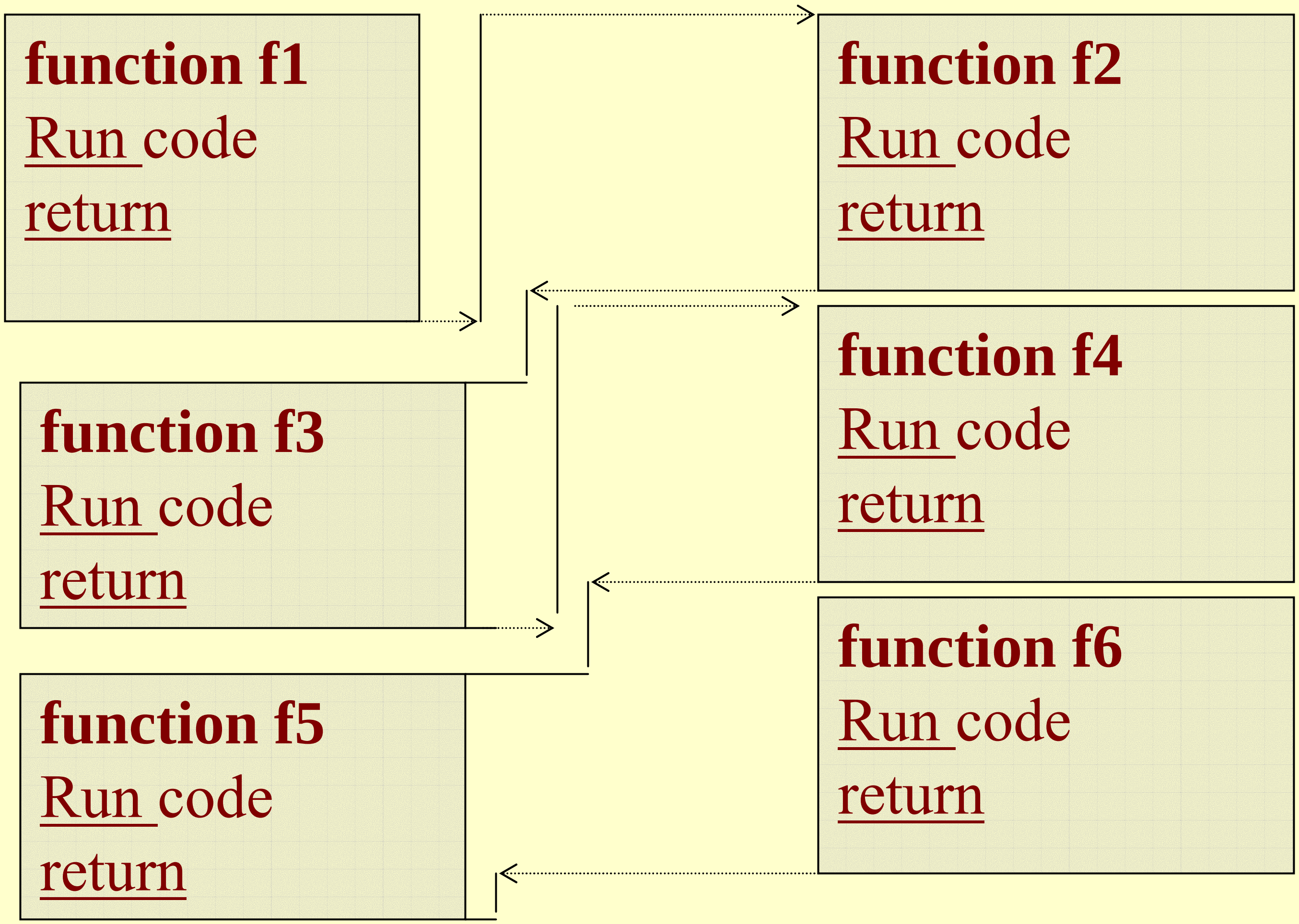
PROGRAM MODELING CONCEPTS :
Lesson-1: PROGRAM MODELS

1. Sequential Program Model

Programming using Sequential functions

- Use of multiple function calls sequentially

Programming model of Six sequential function calls



function f1

Run code

return

function f2

Run code

return

function f3

Run code

return

function f4

Run code

return

function f5

Run code

return

function f6

Run code

return

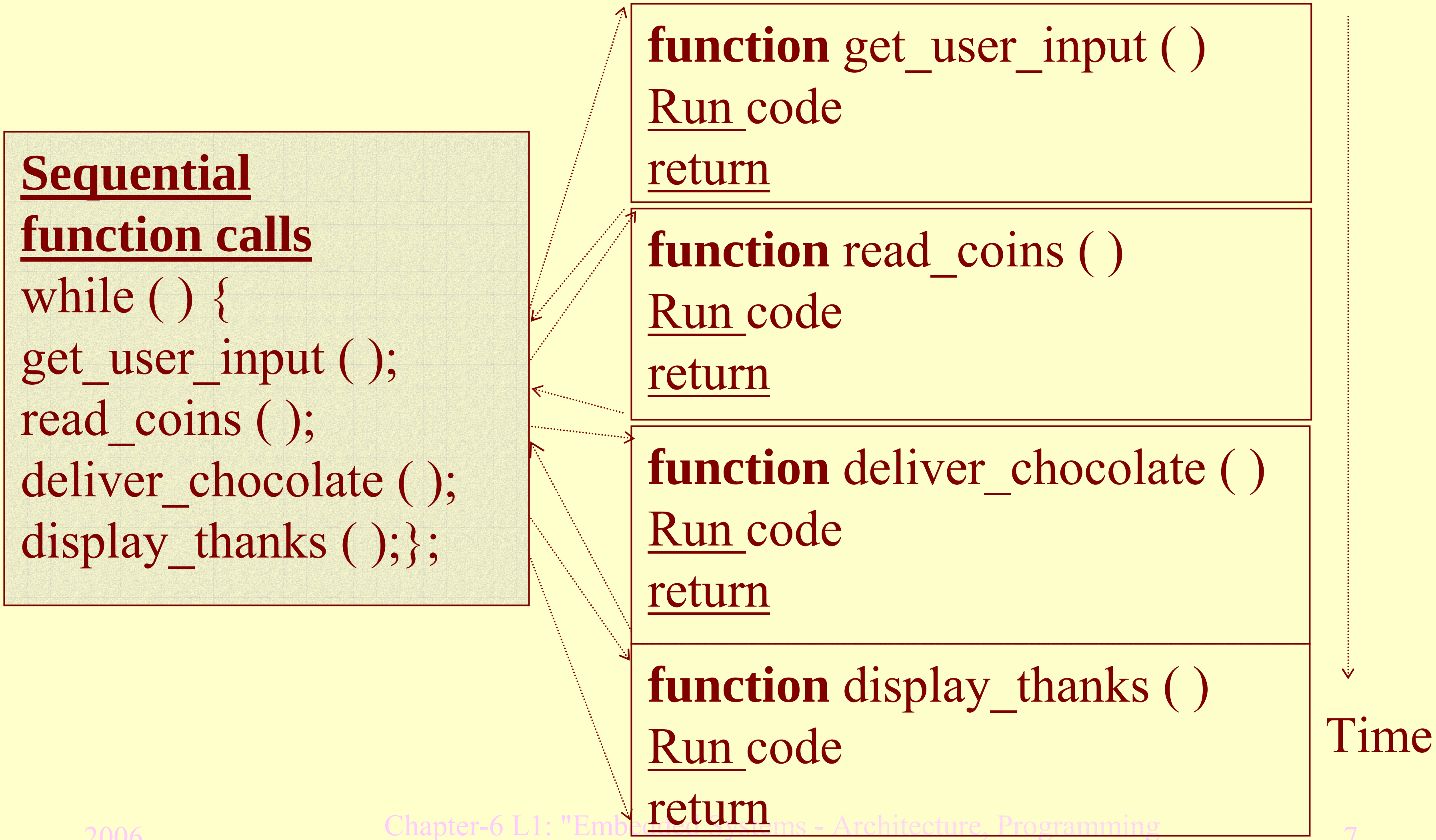
ACVM Example Sequential Program Model

- Run function get_user_input ( ) for obtaining input for choice of chocolate from child.
- Run function read_coins ( ) for reading the coins inserted into ACVM for cost of chocolate.
- Run function deliver_chocolate for delivering the chocolate

ACVM Example Sequential Program Model (contd.)

- Run function display_thanks for displaying ‘Collect the nice chocolate. Visit again!’

Sequential Programming Model of an ACVM



Sequential function calls

```
while ( ) {
get_user_input ( );
read_coins ( );
deliver_chocolate ( );
display_thanks ( );};
```

function get_user_input ( )
Run code
return

function read_coins ( )
Run code
return

function deliver_chocolate ( )
Run code
return

function display_thanks ( )
Run code
return

2. Data-Flow and control data flow Models

Data-Flow Model

- Data flow graphs (DFGs)
- Control data flow graphs (CDFGs)
- Modeling of data paths
- program flows in software.
- A program is modeled as handling the input data streams and creating output data streams

3. State Machine Model

State Machine Model

- A programming model is that there are different states and the model considers a program as a machine, which is producing the states

Example of a Mobile Phone T9 keypad key

- Key marked 5 can produce on pressing different states─ (0, 5), (1, 5), (1, j), …..
- The transition of the key occurs if it is pressed again within an interval.
- The state of the key undergoes in a cyclic fashion till during the interval same key is not pressed again
- $(1, 5) \rightarrow (1, j) \rightarrow (1, k) \rightarrow (1, l) \rightarrow (\mathrm{l}, 5) \rightarrow (1, j)$ inserted between the two elements.

4. Concurrent Processes and Inter-Process Communication Model

Concurrent Processes and Inter-Process Communication Model

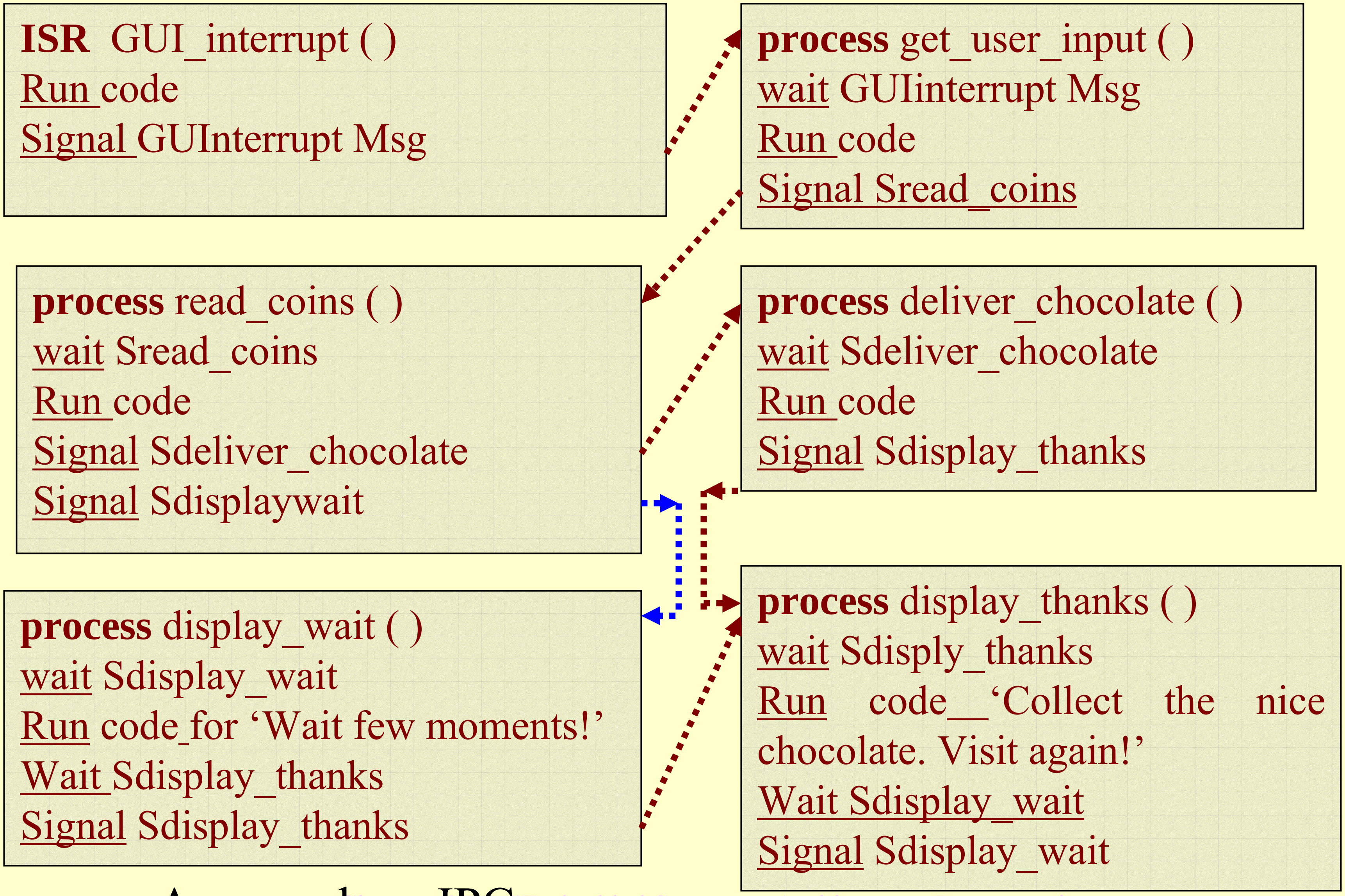
- There are several concurrent tasks (or processes or threads)
- Each task has the sequential codes in infinite loop.
- A task sends a message or signal (to Operating System) for another task.
- A task, which gets a message or signal (using a one-bit bit) from the Operating System , runs and remaining tasks remain in blocked state

Creation of Concurrent processes in ACVM

Concurrent Processes create

```
create process get_user_input ;
create process read_coins ( );
create process deliver_chocolate ( );
create process display_thanks ( );
create process display_wait ( );}
}
```

Concurrent running of the processes in an ACVM



ISR GUI_interrupt ( )
Run code
Signal GUInterrupt Msg

process get_user_input ( )
wait GUIinterrupt Msg
Run code
Signal Sread_coins

process read_coins ( )
wait Sread_coins
Run code
Signal Sdeliver_chocolate
Signal Sdisplaywait

process deliver_chocolate ( )
wait Sdeliver_chocolate
Run code
Signal Sdisplay_thanks

process display_wait ( )
wait Sdisplay_wait
Run code for 'Wait few moments!'
Wait Sdisplay_thanks
Signal Sdisplay_thanks

process display_thanks ( )
wait Sdisply_thanks
Run code 'Collect the nice chocolate. Visit again!'
Wait Sdisplay_wait
Signal Sdisplay_wait

Arrows show IPCs

Concurrent running of the processes in an ACVM

- 1. Process get_user_input ( )─ waits for obtaining input for choice of chocolate from the child and signaling to process read_coins start.
- 2. Process read_coins ( ) ─ waits for signal get_user_input ( ) and start reading on signal from for reading the coins inserted in the ACVM for the cost of chocolate. Post a signal to process deliver chocolate to start and also post a signal to process display_wait ( ) to start

Concurrent running of the processes in an ACVM

- 3. Process deliver_chocolate ( ) ─ waits for signal from read_coins ( ) and start delivering the chocolate and post a signal to display_thanks ( ) to start.
- 4. Process display_wait ( ) ─ waits for signal from read_coins ( ) and start displaying ‘Wait few moments!’ and signal display_thanks.

Concurrent running of the processes in an ACVM

- 5. Process display_thanks ( ) ─ waits for signal from deliver_chocolate ( ) and for signal from display_wait ( ) and then start displaying ‘Collect the nice chocolate. Visit again!’

5. Object Oriented Programming Model

Object Oriented Programming Model

- (i) When there is a need for re-usability of defined object or set of objects that are common within a program or between many applications,

  (ii) when there is need for abstraction and (iii) when, by defining objects by inheritance and polymorphs, new objects can be created
- There is data encapsulation within an object.

Object Oriented Programming Model

- (ii) An object is characterised by its identity (a reference to it that holds its state and behavior), by its state (its data, property, fields and attributes) and by its behavior (operations, method or methods that can manipulate state of the object.
- (iii) Objects are created from the instances of a class. Defining the logically related group makes a class.
- Class defines the state and behavior. It has internal user-level fields for its state and behavior. It defines the methods of processing the fields

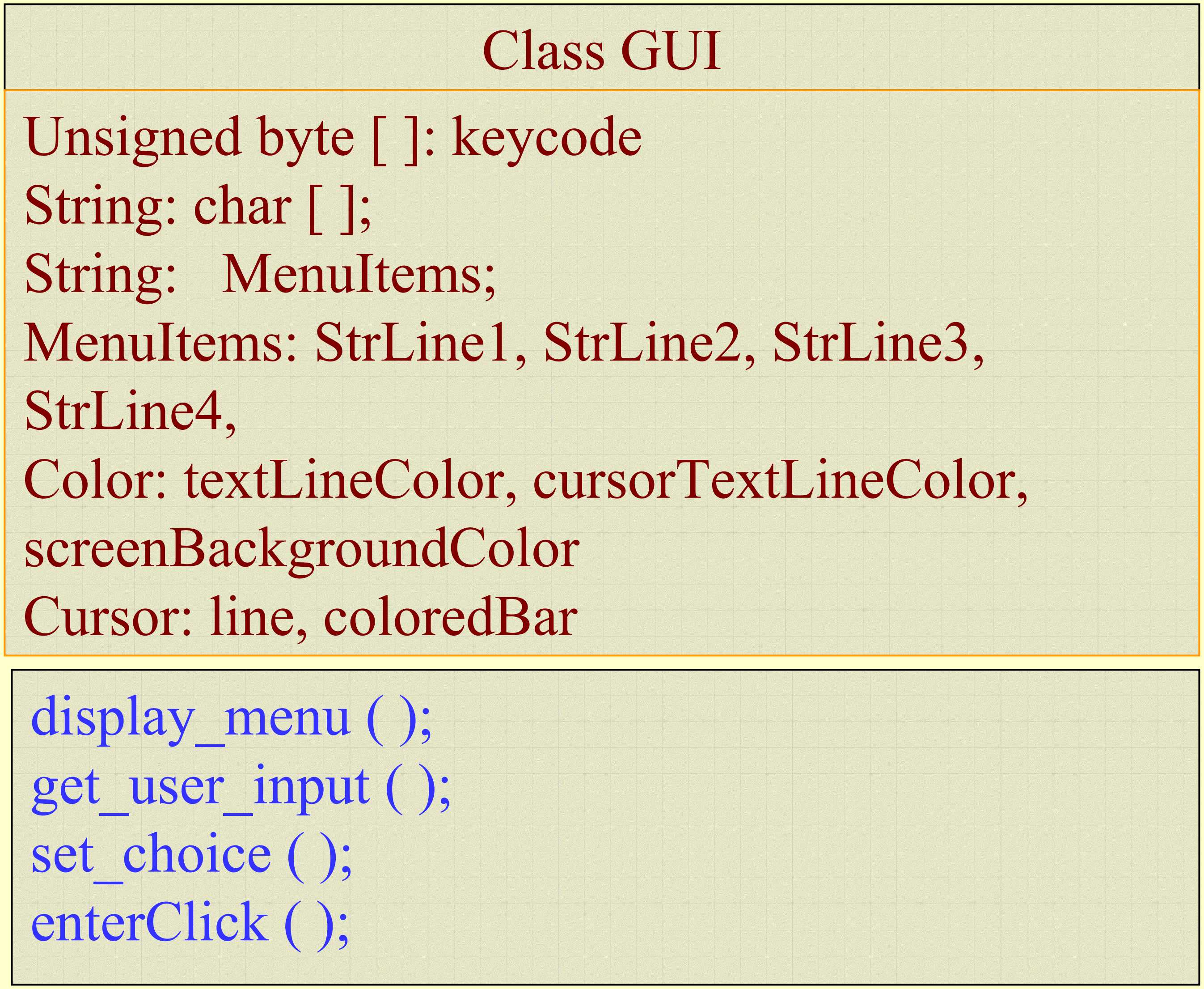
Object Oriented Programming Model

- A class can thus create many objects by copying the group and making it functional. Each object is_ functional. Each object can interact with other objects to process the states as per the defined behavior.
- A set of classes and their objects then gives the application program

Example of the ACVM classes and objects

1. Class GUI for graphic-user interaction. It has two methods display_menu ( ) and get_user_input ( ) and for obtaining input for the choice of chocolate from the child. It has method set_choice ( ) to set the choice selected

GUI

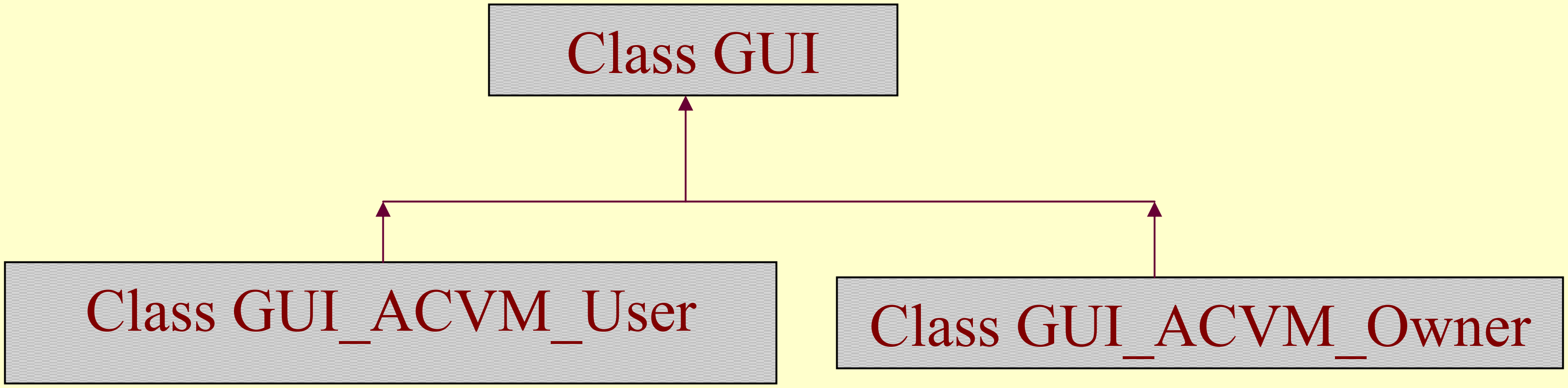


Class GUI

Unsigned byte [ ]: keycode
String: char [ ];
String: MenuItems;
MenuItems: StrLine1, StrLine2, StrLine3, StrLine4,
Color: textLineColor, cursorTextLineColor, screenBackgroundColor
Cursor: line, coloredBar

display_menu ( );
get_user_input ( );
set_choice ( );
enterClick ( );

Inherited classes from GUI



Class GUI

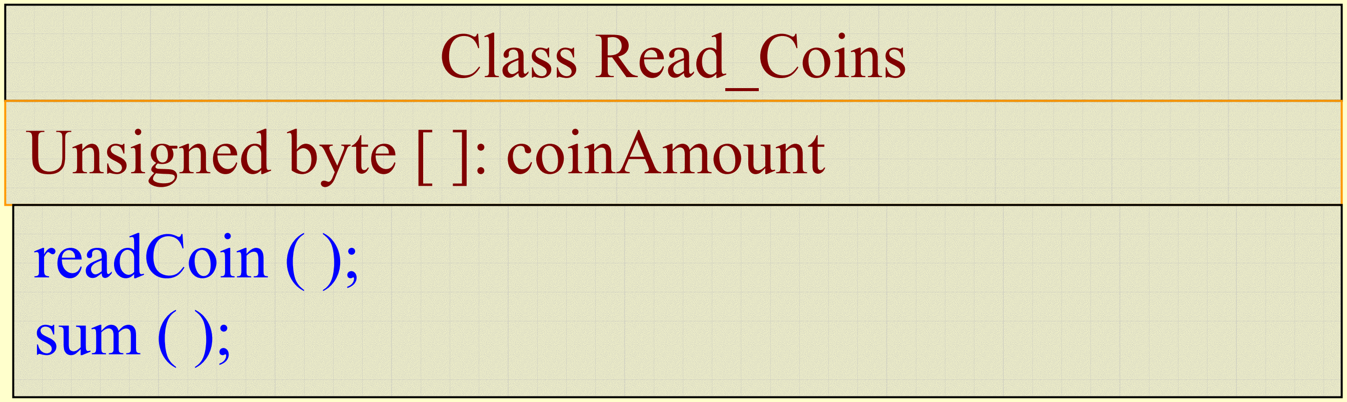
Class GUI_ACVM_User

Class GUI_ACVM_Owner

Class Read_Coins ( )

2. Class Read_Coins ( ) for reading the coins inserted. It has a method read ( ), read ( ) and read to read one, two and five rupee coins from three ports and a method sum ( ) for summing the total coins

Class Read_Coins

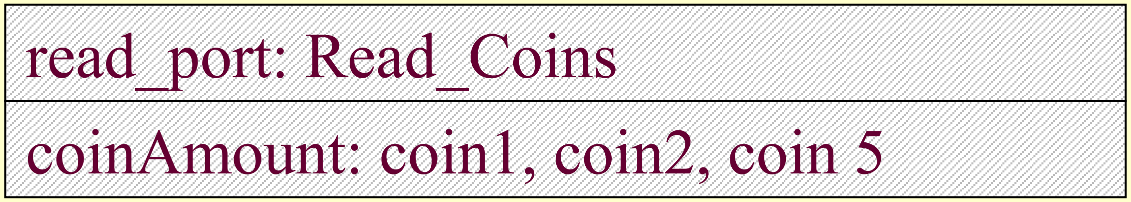


Class Read_Coins

Unsigned byte [ ]: coinAmount

readCoin ( );
sum ( );

Object read_port of the Class Read_Coins



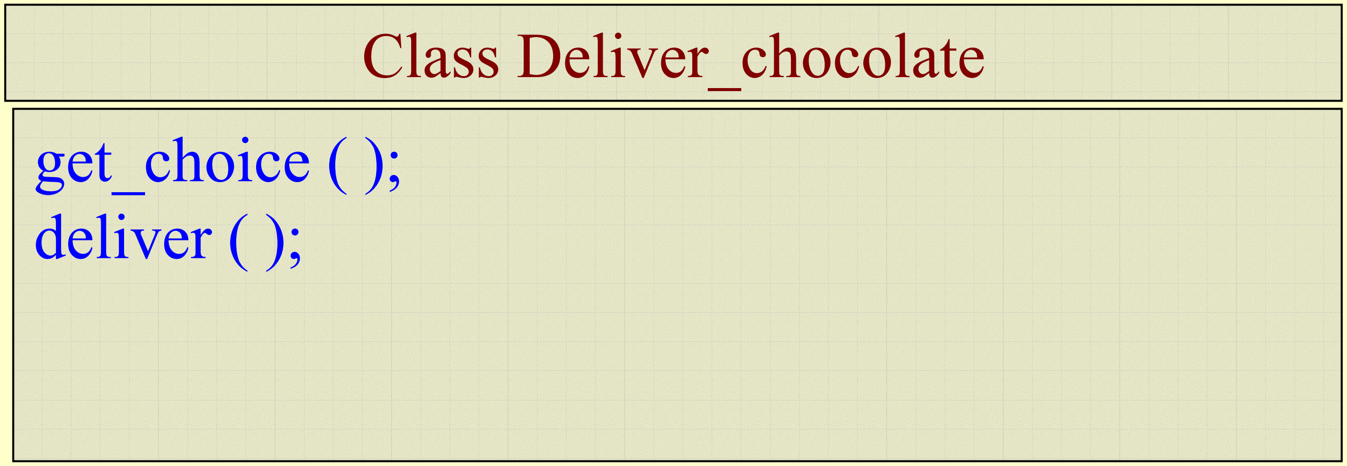
read_port: Read_Coins

coinAmount: coin1, coin2, coin 5

Class Deliver_chocolate

3. It has methods, get_choice ( ) to get the choice and deliver ( ) for delivering the chocolate.

Class Deliver_chocolate



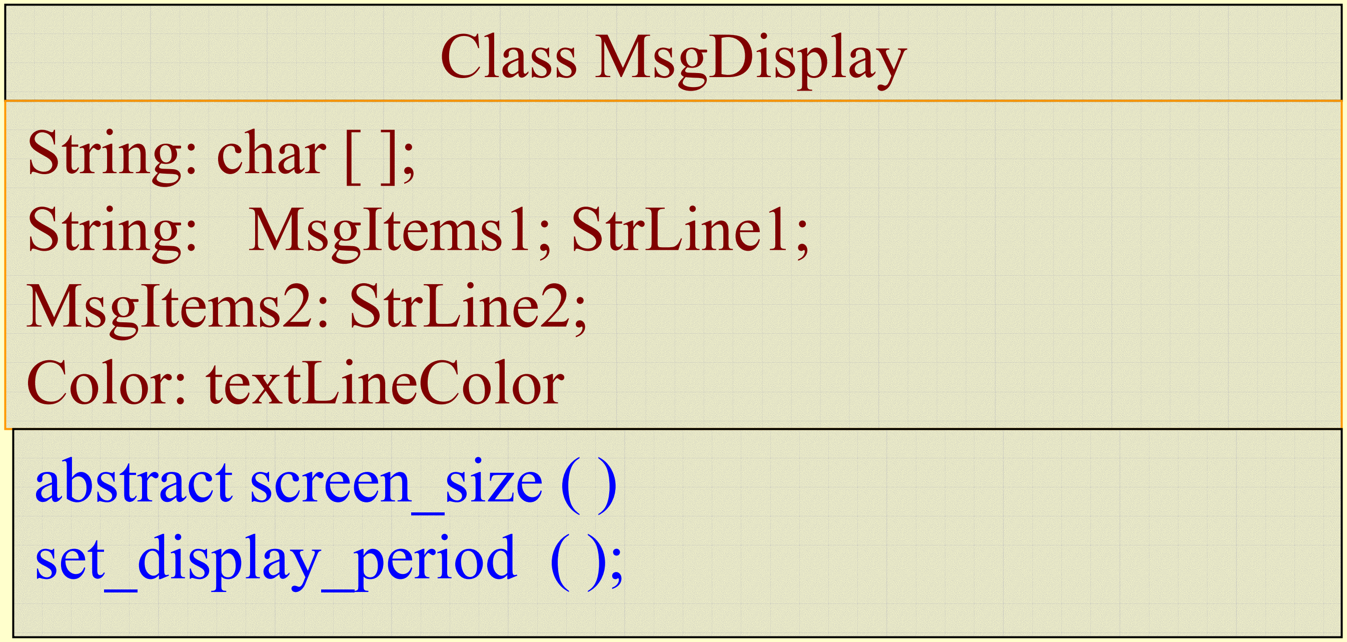
Class Deliver_chocolate

get_choice ( );
deliver ( );

Class MsgDisplay

4. It has methods display_wait ( ) and display_thanks ( ) for display wait message and thank message. Class GUI for graphic-user interaction. It has two methods display_menu ( ) and get_user_input ( ) and for obtaining input for the choice of chocolate from the child. It has method set_choice ( ) to set the choice selected

Class MsgDisplay



Class MsgDisplay

String: char [ ];

String: MsgItems1; StrLine1;

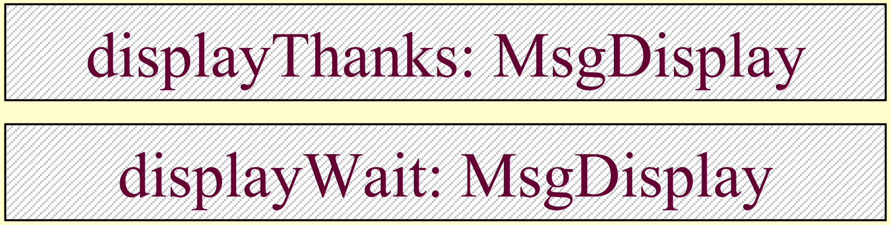
MsgItems2: StrLine2;

Color: textLineColor

abstract screen_size ( )

set_display_period ( );

Objects of the Class MsgDisplay



displayThanks: MsgDisplay

displayWait: MsgDisplay

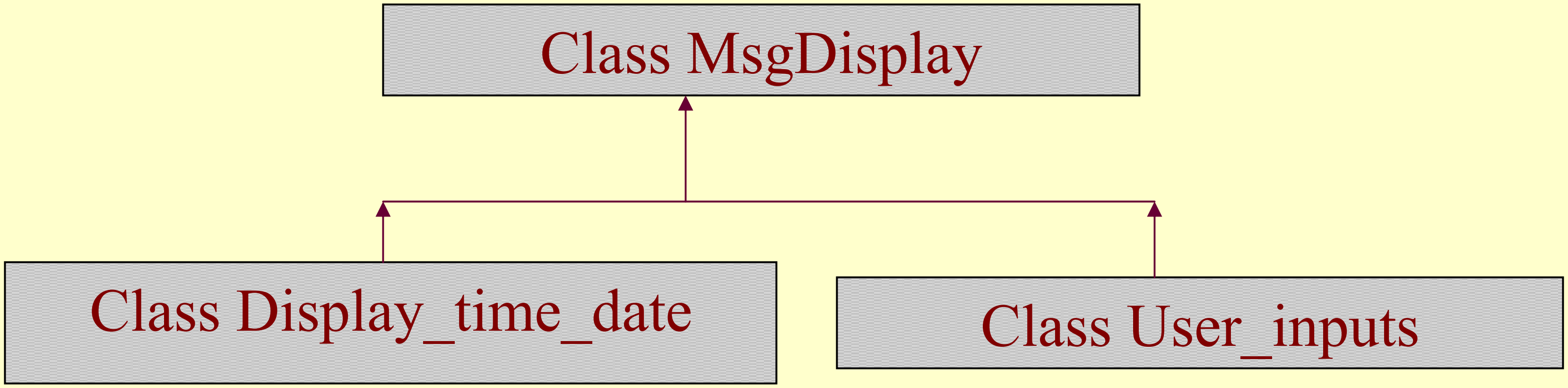
Class GUI and Class MsgDisplay

- Class GUI creates GUI objects as multiple instances of GUI
- Class MsgDisplay creates message display objects as multiple instance of wait and thanks messages.
- Class MsgDisplay can be interfaced to an interface screen_size ( ), which has an abstract method screen_size ( ).
- The abstract method screen_size ( ) is implemented in a MsgDisplay implementing class on extending

Extending class MsgDisplay

- New class Msg-Time_Display
- Extended class Msg-Time_Display inherits all attributes and methods of class MsgDisplay
- Extended class have another method display_time_date ( ) for displaying time and date also with each message.
- Extended class can interface to an interface set_display_period.
- Msg-Time_Display will now implement the method set_display_period ( ) to set display period of 1 or 2 minute for thanks and wait messages

Inherited classes from Class MsgDisplay



Objected oriented approach features

- Re-usability of the defined objects from GUI and set of objects that are common within a program or between the many applications are created.
- Also we have abstract methods, screen_size ( ) and set_display_period which are implemented in the interfacing classes.

Objected oriented approach features

- Inheritance in new objects, which are created by extending the class MsgDisplay.
- Encapsulation of methods and attributes in the class and objects

Summary

We learnt

- Sequential program modeling
- DFG and CDFG modeling
- State machine modeling
- Concurrent processes modeling
- Object oriented modeling

End of Lesson 1 of Chapter 6