Chapter 5 : Object Oriented System Design

Notes from System Programming sir,

Compiled by : Poshan Pandey

Object Oriented System Design.

-> focus on the objects handled by the system, rather than algorithms.

-> programs are designed and implemented as collections of objects, not as collections of procedures.

->

principles of object oriented programming:-

-> objects:-

- is basic unit of OOP.
- is a component of a program that knows how to perform certain actions and how to interact with other elements of the program.
- contains some data and defines a set of operations on that data that can be invoked by other parts of program.

e.g. consider symbol table as an object used by assembler. here, set of operations or methods are like Insert_symbol and Lookup_symbol. Its data would be contents of hash Table used to store symbols and their addresses.

-> ~~class:-~~

-

-> Class:-

- is a blueprint or template or set of instructions to build a specific type of object.
- defines the instance variables and methods of an object.
- an 'instance' is a specific object from specific class.
- many objects can be created from same class.

e.g. for an assembler to translate programs for different versions of machine, class could be opcode-table. from this class, object could be created to define instruction set for machine.

① Encapsulation :-

- means that the internal representation of an object is generally hidden from view outside of objects' definition.
- is the hiding of data implementation by restricting access to accessors and mutators.

② Abstraction :-

- is a model, a view or some other focused representation for an actual item.
- is the implementation of an object that contains same essential properties and actions we can find in the original object we are representing.

③ Inheritance :-

- is a way to reuse code of existing objects or to establish a subtype from an existing object.
- The relationship of classes through inheritance gives rise to a hierarchy.

<u>Subclass</u>

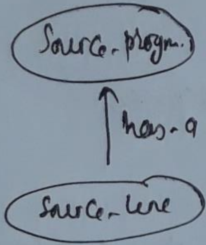
- is a modular, derivative class that inherits one or more properties from another class.

<u>Superclass</u>

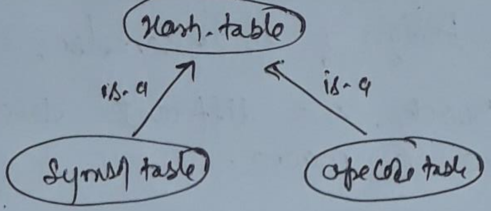
- establishes a common interface and foundation functionality, which specialized subclass can inherit modify and supplement.

④ Polymorphism :-

- means one name, many forms.
- manifests itself by having multiple methods all with same name, but slightly different functionality.



=>



1) has a relationship

2) is a relationship or inheritance

=> Hash-table is base class
=> other two are sub class.
=> if insert_item & search-item are methods of base class then other subclasses automatically contains definition of methods

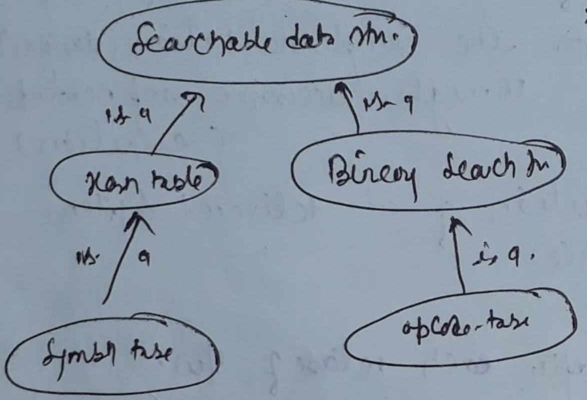


Fig 3) polymorphism.

- here,
-> Superclass searchable_data-structure defines two methods insert_item and search-for-item.
-> Hash-table and Binary-search-tree are subclasses. So inherits above methods.
-> implementation of the methods are different in those subclasses.
-> But, names of methods and way of invokation are same.
-> if search-for-item method is invoked as instance of symbol-table, it will result in a retrieval from hash table.
-> if same method is invoked on an instance of opcode-table, it will result in a binary-search-tree.
- This shows polymorphism.

Object Oriented Design of an Assembler:-

- according to Booch, two different development processes
  @ micro @ macro.

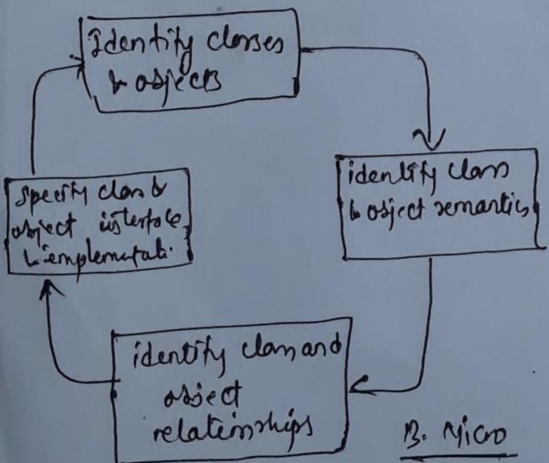
- Booch's macro process represents overall activities of development on a long range scale.
  -> Establish the requirements for the s/w (conceptualisation)
  -> Develop an overall model of system's behaviour (analysis)
  -> create an architecture for the implementation (design)
  -> Develop the implementation through successive refinements (evolution)
  -> Manage the continued evolution of a delivered system (maintenance)

- This macro process repeats itself after each relase of s/w.
- similar to waterfall model.
-

- Booch's micro process represents daily activitees of system developer.

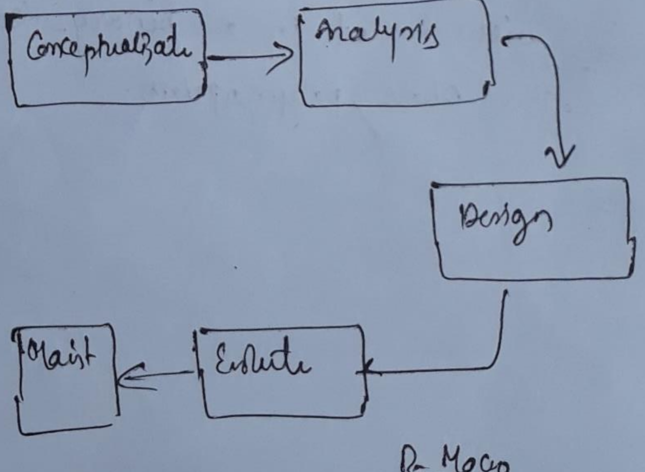
* These activities may be repeated as needed with increasing level of details.

  -> Identify the classes and objects of the system.
  -> Establish the behaviour and other attributes of the classes and objects.
  -> analyze the relationship among the classes and objects.
  -> specify the implementation of classes and objects.



Identify classes & objects
identify class & object semantics
identify class and object relationships
Specify class & object interface & implementati

B. Micro



B. Macro

Object Oriented Design of an Assembler:-

- according to Booch, two different development processes
  @ micro ⓑ macro.

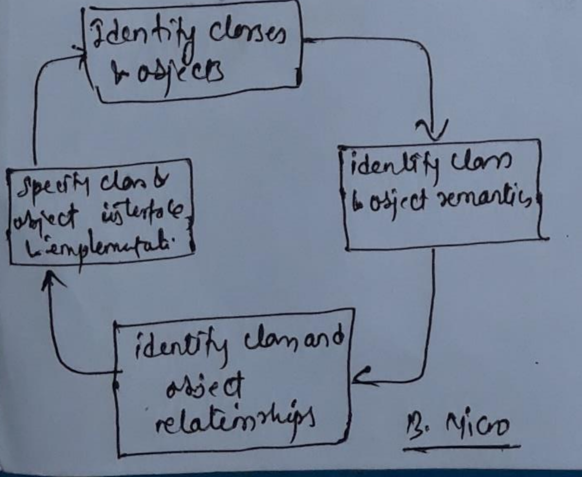
- Booch's Macro process represents overall activites of development on a long range scale.
  - → Establish the requirements for the s/w (conceptualisation)
  - → Develop an overall model of system's behaviour (analysis)
  - → create an architecture for the implementation (design)
  - → Develop the implementation through successive refinements (evolution)
  - → Manage the continued evolution of a delivered system (maintenance)

- This macro process repeats itself after each relase of s/w.
- similar to waterfall model.
-

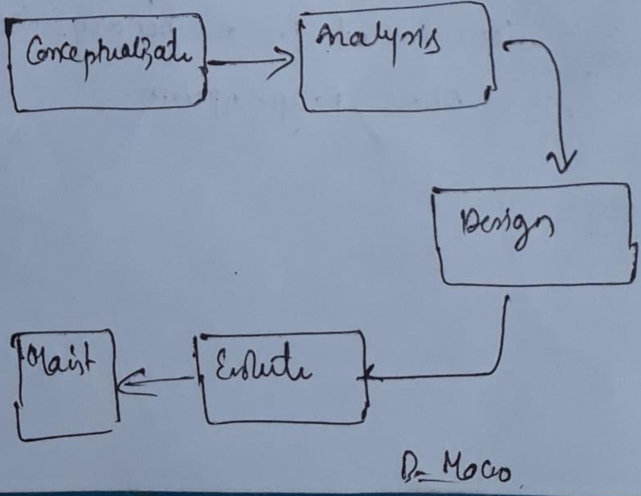
- Booch's micro process represents daily activitees of system developer.

* These activities may be repeated as needed, with increasing level of details:

  - → Identify the classes and objects of the system.
  - → Establish the behaviour and other attributes of the classes and objects.
  - → analyze the relationship among the classes and objects.
  - → specify the implementation of classes and objects.


Identify classes & objects
identify class & object semantics
identify class and object relationships
specify class & object interface & implementation

B. Micro



D. Macro

ii) Methods:.

ⓐ Enter - Enter a label and location counter value into table.
- return error if label is already defined.

ⓑ Search - Search table for specified label.
- return location counter value of label or error if label is not defined.

4) opcode-table

i) Contents:.

- Mnemonic instructions
- includes machine instruction format and opcode.
-

ii) Methods:.

ⓐ Search - Search table for specified Mnemonic instruction
- return information about instruction format and operands required.
- return error if mnemonic instruction not defined.

5) object-Program

i) Contents:.

- object program after assembly.
- includes machine language translation of instruction and data definition from obj. prog.
- includes program length.

ii) Methods:.

ⓐ Enter-text - Enter machine language translation of an instruction or data definition into object program.

ⓑ Complete - Enter prog length and complete generation of external obj. prog. file.

6) Assembly_Listing:

i) Contents

- Listing of lines of src. prog and corresponding machine language translation.
- includes errors for each line & summary of errors in program.

ii) Methods:

ⓐ Enter_line - Enter source line; the corresponding machine language translation and description of errors detected for the line into assembly listing.

ⓑ Complete - enter summary of errors detected and complete the generation of external assembly listing file.

object diagram:

- indicates the methods that are invoked by each object.
- e.g. source_program object invokes methods Create, Assign_location and Translate on the source_line objects.

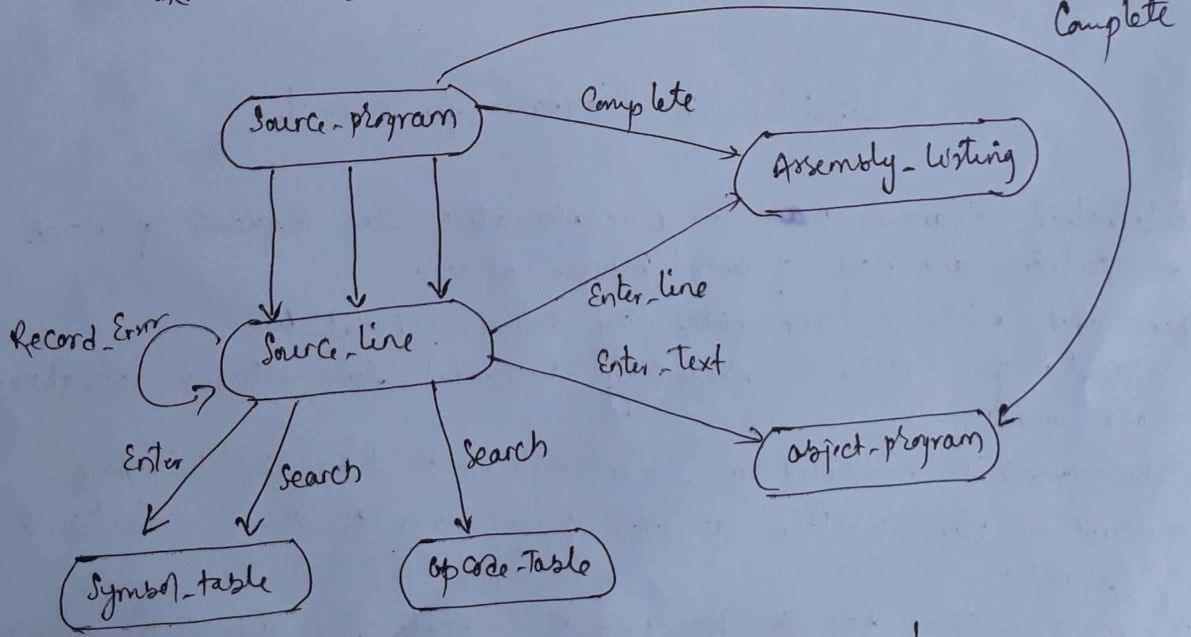


fig. object diagram of Assembler.

- object diagram may also indicate the class of each object.
- The invocation may be numbered to indicate the sequence in which they occur and the flows of info they cause.

Interaction diagram:

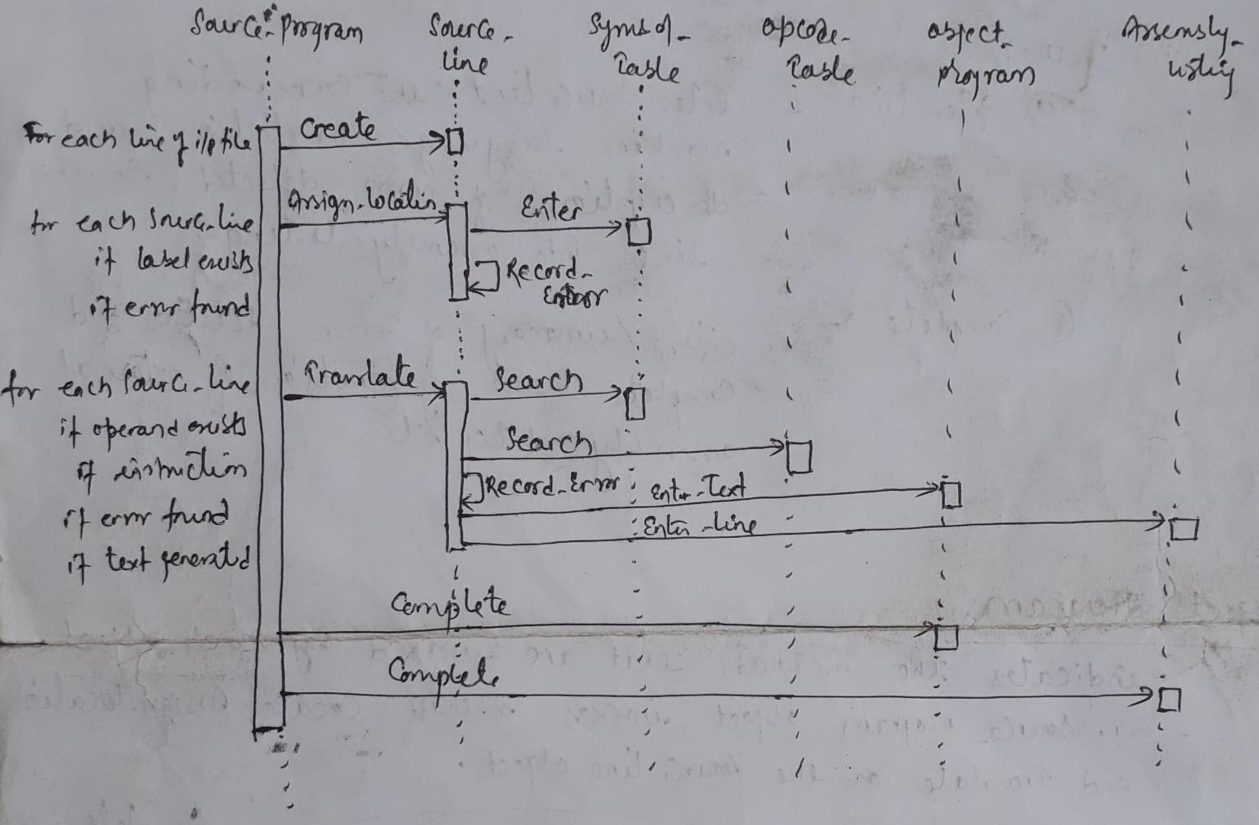


fig) interaction diagram for assembler.

- interaction digram makes easy to visualize the sequence of objects invocation and flow of control between objects.
- here, each object is represented by dashed vertical line.
- invocation of method is shown by horizontal line between one object and another.
- The sequence is indicated by their vertical position in diagram.
- a script is often written at L.H.S. of diagram to describe condition & iteration.
- A narrow vertical box can be used to indicate the time that flow of control is focused in each object.