Module -4

Supports for object Oriented programming- Inheritance - Dynamic Binding - Design issues for Object Oriented languages - Support for Object Oriented Programming in C++ - Implementation of Object oriented constructs. Exception handling - Basic concepts Design Issues.

Object Oriented Programming.

✓ Basic object oriented languages are SIMULA 67, Smalltalk 72, C++, CLOS, 88, Java, 95, Ada 95, Modula-3 etc. We are familiar with Java & C++.

✓ First object oriented progeamming language is SIMULA 67.

Basic oops concept include.

a) object b) class c) Inheritance d) Abstraction

e) Encapsulation f) Polymorphism → Basic concept.

a) Abstraction

Abstraction is a process by which the programmer can associate a name with potentially complicated program fragment, which can then be thought of in terms of its

Purposes or function, rather than in terms of implementation.

→ Two types of abstraction
→ they are data abstraction & control abstraction.
→ eg:- If we are assigning a value to a name it is data abstraction. and if we are writing & passing a function it is control abstraction.
→ Purpose of writing a function is majorly used for reusing the code / statement, just call that statement while its necessary.
→ Abstraction focus on the meaning; hiding the codes under a name. and suppress the irrelevant implementation details.
→ Assign names to complicated program fragment, which does some function.

Advantages of data abstraction :-

a) Reduces the conceptual load for the programmer.
it reduce the size of program.

b) It provides a method of fault containment by preventing programmer from using code in appropriate way.

c) Increases the independence among program components.

II) Elements of Object-Oriented Programming.

a) Class : Class is a user defined datatypes. It includes data members and member function.

b) Data items : They are objects.
Objects are the members of classes.
Objects store data in fields and behaviour in methods specified by their classes.

c) Encapsulation : Wrapping up of data & member function into an object is called encapsulation. Calling objects does not need to know the details how the service is accomplished.

d) Inheritance: Enables a new class to reuse the state & behaviours of old class.

e) Polymorphism: Enables one common interface for many implementations through dynamic method binding.

Common Object Oriented Languages

① ✓ First object oriented language - Simula 67

✓ C++ → Supports procedural & data oriented programming.

✓ Ada 95 - Supports " "

✓ CLOS - " functional programming.

✓ Scheme - " "

Dynamic binding.

Dynamic binding refers to the mechanism by which the implementation of a method called at runtime is determined based on the type of the object that is executed the method rather than the type of the reference variable. This allows for Polymorphism or the ability of an object to take a multiple forms and is a key feature of object Oriented programming like java.

eg:

```
public class A  → main class
  {
    draw()
    {
    }
  }

public class B extends A   ← Subclass.
  {
    draw()
  {
  }
```

→ Consider the situation

* There is a base class 'A' that defines a method draw() which draws some figure associated with the base class.

* A second class B is defined as the subclass of 'A'.

→ objects of this new class also need a draw method that is like that provided by A. but a slight difference. because the subclass objects.

→ So the subclass overrides the inherited draw method (same class's feature is copied to next class)

→ If the client of A and B has a variable that is reference to class A's object, that reference could also point to class B's object.

Design Issues of Object-Oriented Languages (feature):

✓ The Exclusivity of objects

→ The exclusive use of objects in purest model of object-oriented computation where all types are classes

→ it purest model.

→ The advantage of this choice is the elegance & pure uniformity of the language and its use.

→ disadvantage is that simple operations must be done through msg passing process, which often makes them slower than similar operations in the imperative model.

✓ Are Subclasses Subtypes?

eg: Subtype Small_Int is Integer range -100..100;

→ that means variable of Small_Int type

have all the operations of Integer variables but it can store only a subset of values possible in Integer.

∴ Subclass is the parent class & subtype is the derived class.

✓ Single & multiple Inheritence:

→ The purpose of multiple inheritence is to allow a new class to inherit from two or more classes.

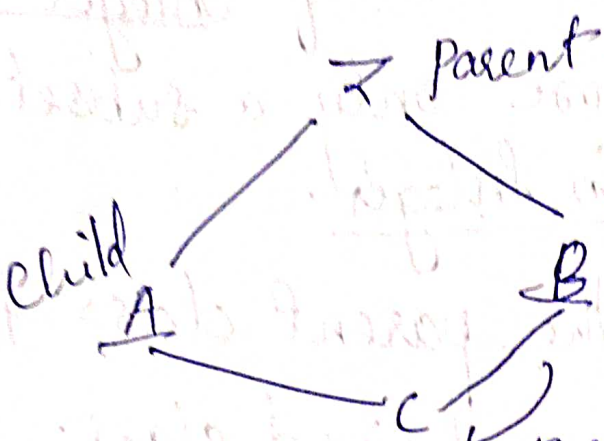
→ multiple inheritence is sometimes highly useful.

→ Why language designer not include?

→ Reason is that

a) Complexity b) efficiency.

Another issue is if both A and B are derived from a common parent 'Z'. and C has both A and B as parent class. This situation is called diamond or shared inheritance.



Subclass (parent is A and B).

Single & multiple inheritence

4) Allocation & Deallocation of objects

→ There are two design questions concerning the allocation and dealloc of objects.

→ First is the place where objects are allocated.

→ If they behave like an abstract data type, then they can allocate from anywhere.

class is declared abstra that provides common defn to base class.

→ This means that allocated from the runtime stack for this we use a keyword (new).

→ If they are all heap dynamic, there is an advantage of having uniform method of creation and access through pointer or reference variable.

* If the objects are stack dynamic there is a problem with regard to subtypes.

ie If class B is a ~~typ~~ child of class A then 'B' is the subtype of 'A'.

→ then the object of B type can be assigned to variable 'A' type.

eg: $a_1 = b_1$ // legal statement

If a_1 and b_1 are references to heap dynamic objects there is no problem.

→ However if a_1 and b_1 are stack dynamic then they are value variables.

Then the values are copied to the target object.

→ Those case where objects are allocated from the heap, and check whether deallocation is implicity, explicit or both.

- If deallocation is implicit, implicit method of storage reclamation is required.
- If deallocation is explicit that raises the issue of whether dangling pointers or references can be created. arise during object destruction

5) Dynamic & Static binding.

6) Nested classes.

7) Initialisation of objects.

→ The initialisation issue is whether how the objects are initialised to values when they are created.

Supports for object Oriented programming in C++.

✓ General Characteristics.

→ Supports methods as well as functions.
→ Thats why it is a hybrid language
→ which supports procedural & object oriented programming.
→ The object of C++ can be static, stack dynamic or heap dynamic.

✓ Inheritence.

If a class has parent, & the inherited datamember must be initialised when the subclass is created. To do this the parent constructor is implicitly called.

→ ie Subclass (Subclass parameters): parent_class(Super class parameters)

{

}

- Dynamic binding
- Evaluation.
  - → comparison b/w languages.

Implementation of Object Oriented Constructs

→ There are at least two parts of language support for object oriented programming.

ie a) Storage structures for instance variables.

b) dynamic binding of messages to methods.

1) Instance Data Storage.

→ In C++, classes are defined as extension of C's record structures - Structs.

→ Usually a storage structure for the instance variables of class instances ie a record

→ This form of structure is called class instance record (CIR)

→ CIR is static

→ it is built at compile time and used as a template for the creation of data of class instances.

→ Every class has its own CIR.

→ When a derivation takes place, the CIR for the subclass is copy to that of the parent class.

6) Dynamic binding of method calls to methods

→ Method in a class that are statically bound need not be involved in the CIR for the class.

→ The drawback of this technique is that every instance would need to store pointers to all dynamically bound methods that could be called from the instance.

* The list of dynamically bound methods that can be called from an instance of a class is the same for all instances of that class.

→ Therefore, the list of such methods must be stored only once.

→ So the CIR for an instance needs only a single pointer to that list to enable it to find called methods

→ The storage structure for the list is often called virtual method table (vtable)

→ ~~Methods can be called~~
→ Methods call can be represented as offsets from the beginning of the vtable.

eg:

```
Public class A
{
    public int a,b;
    public void draw()
    {
        - - .
        - - .
    }
    Public int area()
    {
        - - -
        - -
    }
}
Public class B extends A
{
    Public int c, d;
    Public void draw()
    {
        - - .
        - -
    }
    Public void sift()
    {
        - -
    }
}
```

→ CAR for the A and B

→ The method pointer for the area method in B's vtable points to the code for A's area method.

→ Reason is 'B' does not override A's area method

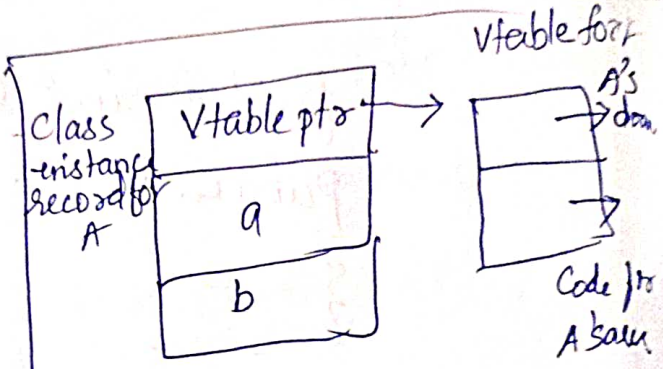
→ if B calls area() it is the area() method inherited from 'A'.

→ On the other hand, the pointer for draw() and sift in B's vtable point to B's draw & sift.

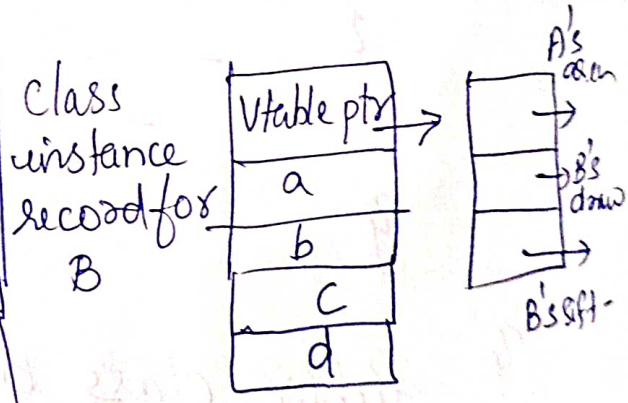
→ The draw method is overridden in B and sift is defined as an addition in B.

For multiple inheritance.

```
class A
{
  public:
   int a;
   virtual void fun()
    {
     - -
     - .
    }
  virtual void init()
    {
     - -
     - .
    }
};
```




```
class B
{
  Public:
   int b;
   virtual void sum()
    {
     - -
     - .
    }
};
```



```
class C : public A, private B
{ public:
  int c;
  virtual void func()
  { - -
    - .
  }
virtual void dud()
  {
    - -
    - .
  }
};
```

→ The class C inherits the variable a and the init method from the 'A' class.
→ It redefines the func() method
→ from B, 'C' inherits the variable b and sum method.
→ C defines its own variable 'c' and defines uninherited method dud:
→ There must be two vtable one for the A and C view and one for the B.
→ The first part of the CIR for C in this can be A and C view which begins with

v-table pointer for the methods of C
those are inherited from A

Exception handling.

→ Exceptions are events that can occur at time that cannot be predetermined.

→ In a language without exception handling

When an error occurs, control goes to the operating system, where a msg is displayed and pgm is terminated.

→ In a language with ~~out~~ exception handling programs are allowed to trap some exceptions, thereby providing the possibility of fixing the problems and continuing.

→ Exceptions are detectable by either h/w or s/w that may require special processing.

→ Special processing that may be required after detection of an exception is called exception handling.

→ Exception handling code unit is called exception handler.

Advantages of Built in Exception handling

- ✓ The code that detects error condition or unusual situations.
- ✓ It forces the programmer to consider