Page: Date: / /

Composition

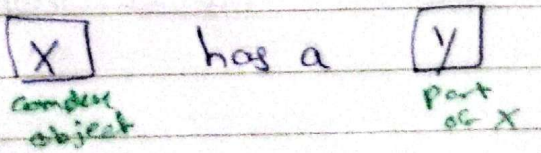
It is the process of building Complex objects from simpler objects.

- It is "Has a" relationship.
- It is the most strong relationship and means complete ownership.
- In this a class contains an object of another class and is responsible for its lifeline.
- If the Complex object is destroyed all the internal objects are also destroyed.

Symbol.

"—◆—" , it is represented by a filled diamond.

e.g



Yousaf

Day: Date: / /

Understanding through examples:

1) First way of Compositions (Member variable).

```
1   class Engine {
2
3   Public:
4           void start() { }
5
6   };
7
8   class Car {
9
10          Engine E; // Car "own a" or "has a" Engin
11
12  Public:
13          void startCar() {
14                  E.start();
15          };
16
17  };
```

In this E is an member object for Car.
If car is destroyed so is engine.

Day: Date:

2) 2nd Way of making ~~class~~ composition: (Refe-one)

```
class Engine {
public:
    void start() { }
};

Class Car {
~~public:~~
    Engine *E;
Public:
    voidstart Car() {
        E->start();
    }
    Car() : E(new Engine) { }

    Car() { // very important. (destructor).
        delete E;
    }
};
```

In this the pointer E is created by Car and also destroyed by Car.

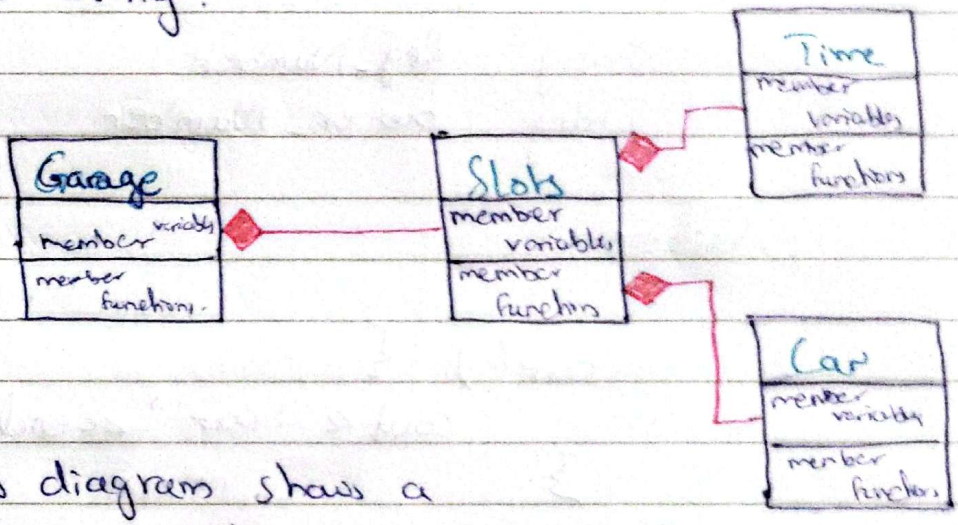
If the destructor didn't exist this won't be composition.

Yousuf

Day: Date: / /

3) Example to understand which objects are made first:

I'll be using:



This diagram shows a parking system.

Coding:

```
class Time {
        int  h, min, sec;

public:
        Time (int h=0, int m=0, int s=0): h(h),
          min (m), sec(s) { // constructor.
              cout << " time " << endl;
            }

};
```

Yousaf

Day: Date: / /

```
class Car {

    string   Reg_Number
    string   owner_Number

public:

    car() {
        cout << " car " <<endl;
    }

};

class Slots {

    Time     in;
    Car      C;
    double   rate;
    bool     isEmply;
    int      slot Number;

Public:
    slot () {
            cout << "slot" <<endl;
        }

};
```

Yousaf

Day: Date: / /

```
class Garage {

    slot normal [2]
    slot vip [2]
    int size;
public:

    Garage () {
        cout << " Garage " << endl;
    }

};


int main(){
    Garage g

}
```

Terminal next Page. →

Day: Date: / /

Terminal:

Time
Car
Slot
Time
Car
Slot
Time
Car
Slot
Time
Car
Slot
Garage.

- If a destructor excuted it will run complete opposite meaning like Garage = .
  Slot
  Car
  Time - - .

- Parts are created First.
- When we reach : This point
  Garage () { //constructor
  - -
  }
  The parts are made, They can also be made in first 2-3 lines • inside constructor but will explain in other examples.
- This point as you can see ~~Exit~~ is the member initializer point.

Yousaf

Day: Date: / /

4) Example to understand need of default constructor:

```
1  Class Point {
2
3      double x, y;
4
5  Public:                                   //Parameterized
6      Point(int x, int y) : x(x), y(y) {      Constructor
7          cout << "Point Const" << endl;
8      }
9      void display() {
10         cout << "(" << x << ", " << y << ")" << endl;
11     }
12     //consider getters and setters made.
13 };
14
15 Class Circle {
16
17     Point c;
18     double r;
19
20 Public:
21     Circle() {
22         r = 0;
23         cout << "circle";
24     }
25     Circle(double r, double x, double y=0) {
26         this->r = r;
27         c.setx(x).sety(y);
28         cout << "Circle Const" << endl;
29     }
30 };
```

Date: / /

```
30 int main() {
31
32         Circle c;
33         Circle C(5,1,3);
34
35 return 0;
36 }
```

error

Now error coming why?

in Point class no default constructor exist but when in line 21 or line 25 we reach This point a ~~circle stmt~~
circle(--) {--}
point needs to be created and without default constructor it is impossible.

How to fix the error:

1) Using a pointer and dynamic memory:
we can make point as a pointer
i.e Point * c.

Then in constructor we can write
new Point (x, y);
This will make point only when new Point() called.
a pointer, itself is NOT a datatype or object.

Yousaf

Date: / /

2) Member initializer list:

- We can use the member initializer list to fix the problem because we know that where error is coming there is member initializer list.

- In member initializer list you will make C using the point's constructor.

Circle (--) : C(x, y), r(r) { }

- we know that ~~Read~~ member are initialized at the point where the member initializer lists lie Always there, no where else, so we can initialize them using the constructor.

Yousaf

5) How does member initializer initialize variables:

```
class test {

    int * ptr;
    int size;
public:
    test (int s=0) :size (s) , ptr(new int(size)) { };

};
```

- This code will always give us error
- bcz ptr is initialized before size.
- It doesn't matter if we write size before in member initializer list the order that matters is;

```
int * ptr   | first
int size    | last
```

- It MATTERS in what order we declared the attribute.
- Member initializer ORDER doesn't matter.

Yousaf

Day: Very important. Date: / /

6) Using double pointer in composition to help if ~~the~~ independent class has constant attributes.

- Consider a place where the independent class has constant ~~para~~ attributes meaning they are immutable and ~~in the depen~~ the default constructor makes everything null initially and we cannot change it then.
- How would you handle that? In dependent class we normally use the default constructor but now we should be using the parameterized constructor for every object we are creating of the independent class so that the values are set accordingly. as they cannot be changed later.

Example:

- A flower class have 3 attribute:
  - Name of type string (unchangeable).
  - ~~Attri~~ Colour of type string (immutable).
  - Price (can be changed).

- Then make a bouquet class which can have many ~~class~~ flowers ~~class~~ ~~operator to have attributes~~.

Yousaf

Day: Date: / /

```
1  class Flower {
2
3  const string name;
4  const string color;
5        int price;
6
7  
8  public:
9
10       // make getters / setters.
11
12       void display () {
13             cout << color << " " << name << "\n" <<
14             "price: " << price << endl;
15       }
16
17 
18       Flower (string name, string color, int price): // Parameterized constructor
19             name(name), color(color), price(price) { }
20
21
22 };
23
24
25
26
27
28
29
```

Yousaf

Date: / /

```
30  class Bouquet {
31      int size;
32      Flower ** flowers;
33      int size;
34
35  public:
36
37      Bouquet (int size): size(size){ // parametrized constructor.
38
39          flowers = new Flower*[size]
40          // make all pointers initially NULL.
41      }
42
43
44      void operator () (Flower & F, int Index){
45          if (flowers[index] != Null) {
46
47              delete flowers[index];
48
49              flowers[index] = new Flower(F
50              .getName(), F.getColor(), F.
51              getPrice());
52
53
54          }
55          else {
56              flowers[index] = new Flower(F.getName(),
57              F.getColor(), F.getPrice());
58          }
59      }
```

very inefficient using parameterized constructor

Yousaf

Day: Date: / /

```
60
61      // consider the bouquet filled up or you
62      can make another variable to addup
63      check how much filled and display
64      till that.
65
66      void display() { // considering filled up.
67
68          for(int i=0; i<size; i++ ){
69              Flowers[i]->display();
70          }
71      }
72  // write destructor, which destroys the flowers*flowers
73      completely.
74  };
75
76  int main() {
77
78      Flower f1("rose","red", 3);
79      Flower f2("tulip","yellow",2);
80
81      Bouquet b1(2);
82
83      b1(f1, 0);
84      b1(f2, 1);
85
86      b1.display();
87
88  }
90.
```

Yousaf

Day: Date: / /

Terminal

```
red rose
Price: 3
yellow tulip
Price : 2
```

Explanation:-

line 3, 4:

Here we are making attributes of flower class, once its name and color has been decided it cannot change so that is why it is constant.

line 18: imp

Here we are initializing the variables, this is very important that we use Member Initializer as the variable are constants.

Line 32, 31:

Here we are ~~initializing~~ writing the member variables of Boquet class.

Yousaf

Day: Date: / /

Line 37 - 41:

The constructor for Bouquet

Here we are making an ARRAY OF Pointers. not an array of datatypes so the constructor is not called.

Line 44 - 59:

Now we are creating a new flower meaning now the constructor will be called.

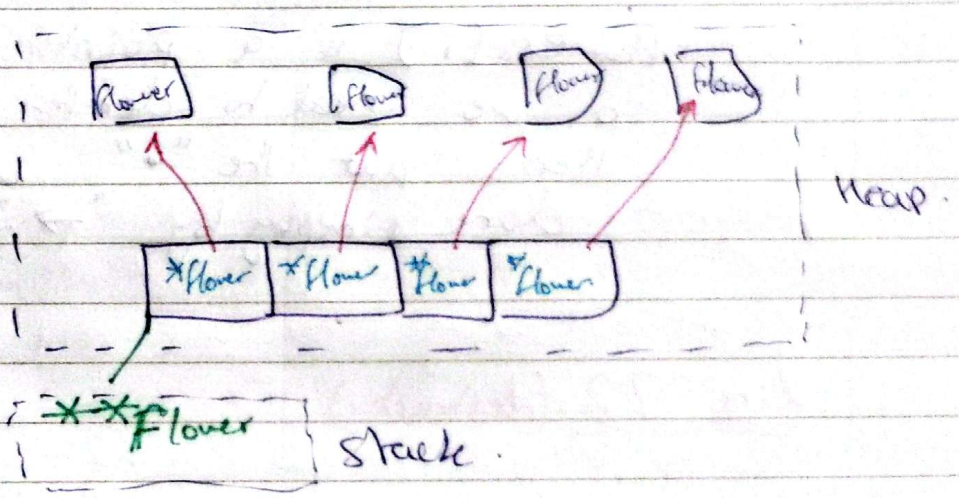
So here we are simply using the parameterised constructor to build the flower array.

Each pointer in the array points to a SINGLE Flower created and initialized in Heap.

- So here we are using a double pointer to make a 1D array of pointers and each pointer points to a single flower.

Yousaf

Day: Date: / /



but now we could have done:
flower[index] = &F;

but here Flowerfindex will not be created by the object and if we destroy it the one in main will be destroyed.

So this will be wrong and is another concept: Aggregation which will be discussed next.

Line 69:

In this line I have used
flower[i] → display();
why the "→"?

Yousaf

Date: / /

flowers[i] is a pointer so we need to dereference it and then put the "." which is done simply by "->".

Line 72: (destructor)

- In composition the object needs to destroy its components.
- so we will destroy the whole double pointer using for loops.
- This is necessary as if we leave it it will become "memory leak".

Yousaf