Object Oriented Programming Using C++
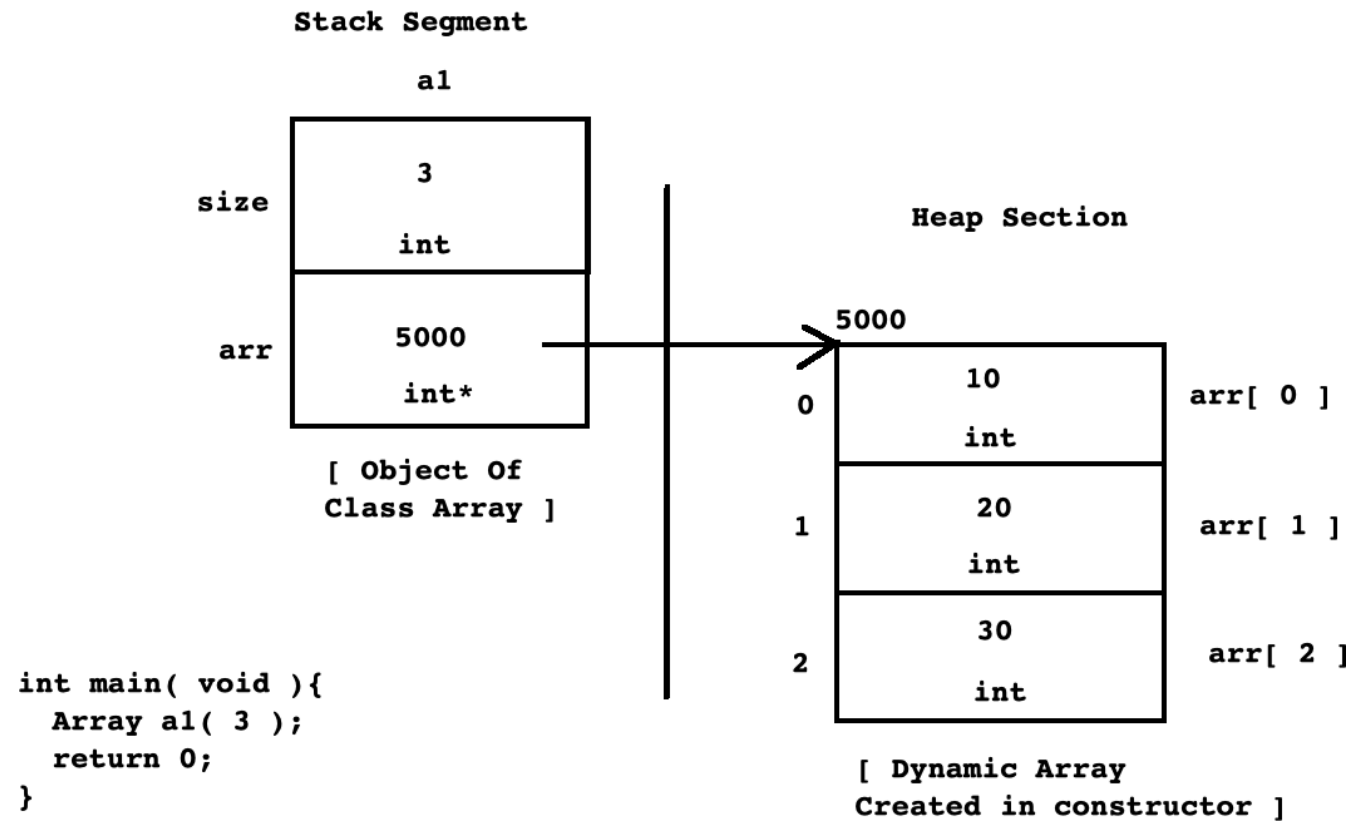
Ketan G Kore

ketan.kore@sunbeaminfo.com

Destructor

- Destructor is a member function of a class which is used to release resources hold by the object.

- If we do not define destructor inside class then compiler provide one destructor for that class by default. It is called default destructor.

- Default destructor do not perform any clean up operation on data member declared by the programmer.

- **Destructor calling sequence is exactly opposite of constructor calling sequence.**

- **We can not declare destructor static, constant or volatile but we can declare destructor inline and virtual.**

syntax:-

```
~Complex()
{
cout<<;
}
```

Destructor

Stack Segment

a1

size | 3
int

arr | 5000
int*

[Object Of
Class Array]

```
int main( void ){
  Array a1( 3 );
  return 0;
}
```

Heap Section

5000

0 | 10
int | arr[0]

1 | 20
int | arr[1]

2 | 30
int | arr[2]

[Dynamic Array
Created in constructor]

Destructor

- Due to following reasons, destructor is considered as special member function of the class:

    1. Its name is same as class name and always precedes with ~ operator.

    2. It doesn't have a return type.

    3. It doesn't have any parameters.

    4. We can call destructor on object/pointer/reference explicitly. But it is designed to call implicitly.

- We can overload constructor but we can not overload destructor.

    ➢ To overload any function either number of parameters must be different or if number of parameters are same then type of parameters must be different or order of type of parameters must be different.

    ➢ Since destructor do not take any parameter, we can not overload destructor.

Destructor

- If constructor and destructor are public then we can create object of the class inside member function as well as non member function.

- If constructor or destructor is private then we can create object of the class inside member function only.

```
~Array( void ){
    if( this->ptr != NULL ){
        delete[] this->ptr;
        this->ptr = NULL;
    }
}
```

Static Data Member

- Member function(static/non static) do not get space inside object. Hence size of object depends on size of all the data members declared inside class.

```cpp
class Test{
private:
    int num1;
    int num2;
    int num3;
public:
    Test( void ){
        this->num1 = 0;
        this->num2 = 0;
        num3 = 500;
    }
    Test( int num1, int num2 ){
        this->num1 = num1;
        this->num2 = num2;
        num3 = 500;
    }
};
int main( void ){
    Test t1( 10, 20 );
    Test t2( 30, 40 );
    Test t3( 50, 60 );
    return 0;
```
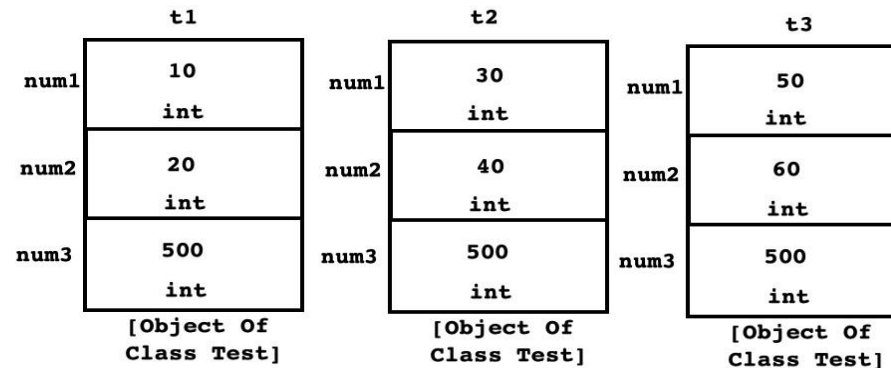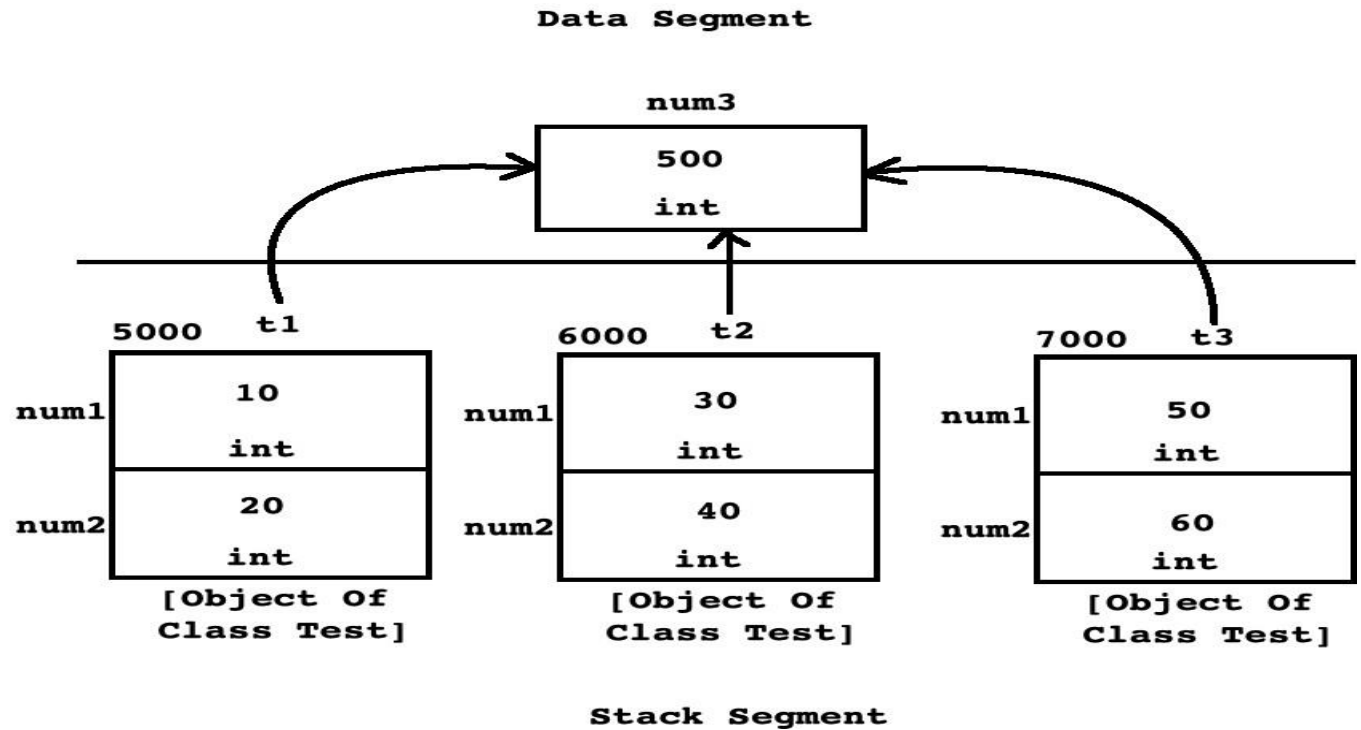
Static Data Member

- **Instance Variable** : A data member of a class which get space inside object is called instance variable. In short, non static data member declared inside class is called instance variable.

- **Instance Method** : A member function of a class, which is designed to call on object is called instance method. In short, non static member function defined inside class is called instance method.

- **Concrete Method** : A member function of a class, which is having body is called concrete method.

- **Concrete Class** : A class from which we can create object is called concrete class. In other words, we can instantiate concrete class.

- **Revision of some oops concepts**:

  1. Data member declared inside class get space once per object according to their declaration inside class.

  2. If we call non static member function on object then non static member function get this pointer.

  3. With the help of this pointer, all the objects of same class share single copy of member function.

Static Data Member

- If we want to share value of any data member inside all the objects of same class then we should declare data member static.

```cpp
class Test{
private:
    int num1; //Instance Variable
    int num2; //Instance Variable
    static int num3; //Class level Variable
public:
    Test( void ){
        this->num1 = 0;
        this->num2 = 0;
    }
    Test( int num1, int num2 ){
        this->num1 = num1;
        this->num2 = num2;
    }
    //Test *const this = &t1
    void printRecord( void ){
        cout<<"Num1   :   "<<this->num1<<endl;
        cout<<"Num2   :   "<<this->num2<<endl;
        cout<<"Num3   :   "<<Test::num3<<endl;
    }
};
int Test::num3 = 500;    //Global Definition
int main( void ){
    Test t1( 10, 20 );
    t1.printRecord( );
    return 0;
}
```


Data Segment

num3

500
int

5000 t1

10
int
20
int

num1

num2

[Object Of Class Test]

6000 t2

30
int
40
int

num1

num2

[Object Of Class Test]

7000 t3

50
int
60
int

num1

num2

[Object Of Class Test]

Stack Segment

Static Data Member

- Static data member do not get space inside object. Rather all the objects of same class share single copy of it. Hence size of object is depends on size of all the non static data members declared inside class.

- A data member of a class which get space inside object is called instance variable.

- Instance variable is designed to access using object or pointer/reference of object.

- A data member of a class which do not get space inside object is called class level variable.

- Class level variable is designed to access using class name and scope resolution operator.

- If we declare data member static then we must provide global definition for it. Otherwise linker will generate error.

- Non static data member get space once per object. But static data member get space once per class.

Static Data Member

- Only non static data member get space inside object. To initialize object we define constructor inside class. In other words, we should use constructor to initialize non static data member which get space inside object.

- Since static data member do not get space inside object, we should not initialize static data member inside constructor.

Static Member Function

- In C++, we can declare global function as well as member function static.

- If we want to access non static members of the class then member function should be non static and if we want to access static members of the class then member function should be static.

- Non static member function get this pointer. Hence inside non static member function, we can access static as well as non static members.

- Why static member function do not get this pointer?

  - If we call non static member function on object then non static member function get this pointer. In other words, non static member functions are designed to call on object.

  - Static member function is designed to call on class name.

  - **Since static member function is not designed to call on object, it doesn't get this pointer.**

Static Member Function

- Since static member function do not get this pointer, we can not access non static members inside static member function. In other words, static member function, can access only static members of the class directly.

- Using object, we can use non static members inside static member function.

- **Inside member function, if we are going to use this pointer then member function should be non static otherwise it should be static.**

- **We can not declare static member function:**
    1. **constant**
    2. **volatile**
    3. **Virtual**

Advantages of OOPS

1. To achieve simplicity

2. To achieve data hiding and data security.

3. To minimize the module dependency so that failure in single part should not stop complete system.

4. To achieve reusability so that we can reduce development time/cost/efforts.

5. To reduce maintenance of the system.

6. To fully utilize hardware resources.

7. To maintain state of object on secondary storage so that failure in system should not impact on data.

# Major and Minor pillars of oops

- 4 Major pillars
    1. Abstraction
    2. Encapsulation
    3. Modularity
    4. Hierarchy

- 3 Minor Pillars
    1. Typing
    2. Concurrency
    3. Persistence

Abstraction

- It is a major pillar of oops.
- **It is a process of getting essential things from object.**
- It describes outer behaviour of the object.
- Abstraction focuses on some essential characteristics of object relative to the perspective of viewer. In other words, abstraction changes from user to user.
- **Using abstraction, we can achieve simplicity.**
- Abstraction in C++

```
Complex c1;
c1.acceptRecord( );
c1.printRecord( );
```

Abstraction
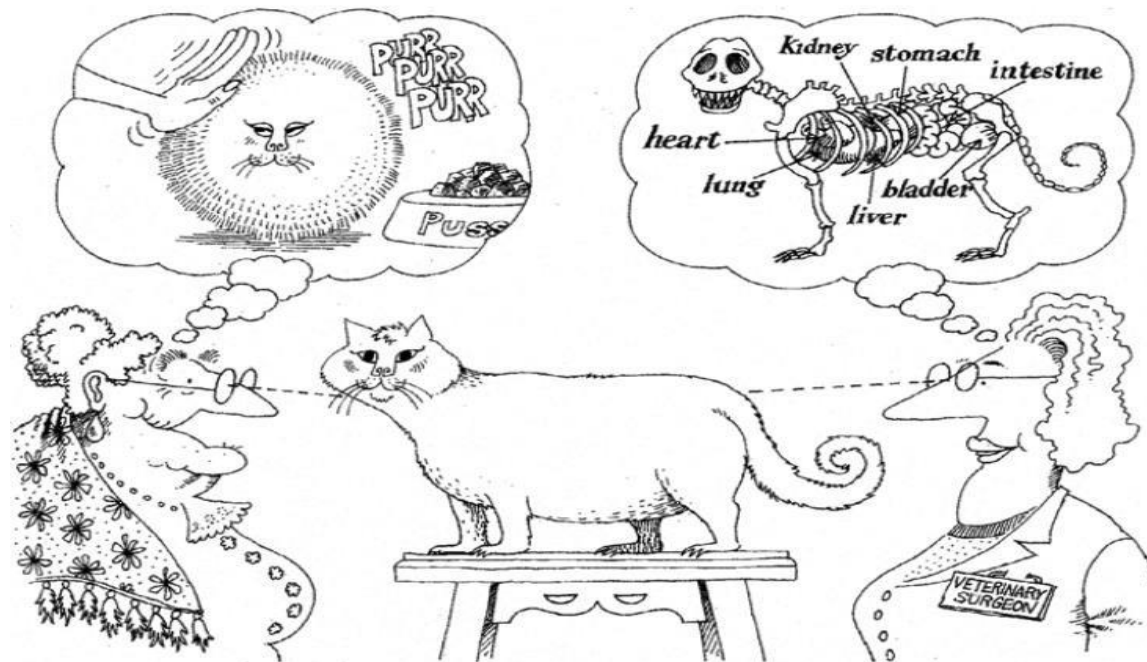
Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.
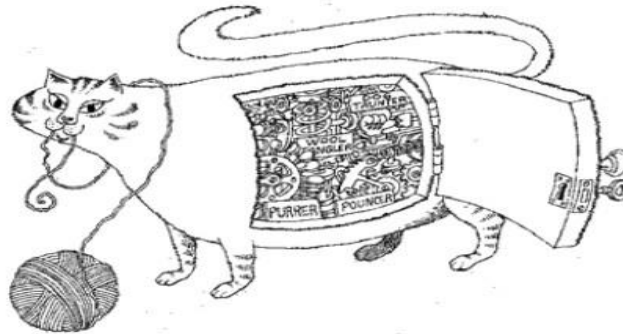
Encapsulation

- It is a major pillar of oops.

- Definition:

    1. **Binding of data and code together is called encapsulation.**

    2. To achieve abstraction, we should provide some implementation. It is called encapsulation.

- Encapsulation represents, internal behaviour of the object.

- **Using encapsulation we can achieve data hiding.**

- Abstraction and encapsulation are complementary concepts: **Abstraction focuses on the observable behaviour** of an object, whereas **encapsulation focuses on the implementation** that gives rise to this behaviour.

Encapsulation

Encapsulation hides the details of the implementation of an object.

Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior.

Encapsulation

Encapsulation using C++.

```cpp
class Complex{
private:
    int real;
    int imag;
public:
    Complex( void ){
    }
    void acceptRecord( void ){
    }
    void printRecord( void ){
    }
};
```

Modularity

- It is a major pillar of oops.

- It is the process of developing complex system using small parts.

- **Using modularity, we can reduce module dependency.**

- We can implement modularity by creating library files.

    o   .lib/.a, .dll / .so files

    o   .jar/.war/.ear in java

Hierarchy

- It is a major pillar of oops.
- **Level / order / ranking of abstraction is called hierarchy**.
- Main purpose of hierarchy is **to achieve reusability**.
- Advantages of code reusability
    1. We can reduce development time.
    2. We can reduce development cost.
    3. We can reduce developers effort.
- Types of hierarchy:
    1. **Has-a** / Part-of => Association
    2. **Is-a** / Kind-of => Inheritance / Generalization
    3. **Use-a** => Dependency
    4. **Creates-a** => Instantiation

Typing

- It is a minor pillar of oops.

- Typing is also called as polymorphism.

- Polymorphism is a Greek word. Polymorphism = Poly(many) + morphism(forms).

- **An ability of object to take multiple forms is called polymorphism.**

- **Using polymorphism, we can reduce maintenance of the system.**

- Types of polymorphism:

  o **Compile time polymorphism**
    - It is also calling static polymorphism / **Early binding** / Weak Typing / False polymorphism.
    - We can achieve it using:
      1. **Function Overloading**
      2. **Operator Overloading**
      3. **Template**

  o **Run time polymorphism**
    - It is also calling dynamic polymorphism / **Late binding** / Strong Typing / True polymorphism.
    - We can achieve it using:
      1. **Function Overriding.**

Concurrency

- It is a minor pillar of oops.

- In context of operating system, it is called as multitasking.

- It is the process of executing multiple task simultaneously.

- Main purpose of concurrency is to utilise CPU efficiently.

- In C++, we can achieve concurrency using thread.

Persistence

- It is a minor pillar of oops.

- It is process of maintaining state of object on secondary storage.

- In C++, we can achieve Persistence using file and database.

Thank you