Object Oriented programming (OOP) in C++

→ A real life entity like a car in my program

→ class is building block of OOP
↓
User defined data type

→ class → attributes
→ behaviors.

→ Employee employee1 (object of class Employee)

→ class Employee {

```
    String Name;
    String Company;
    int ~~Acage~~ Age;
}
```

} private default

→ access modifiers
- → private (hidden)
- → public (public)
- → protected (in between private & public)

→ class Employee {

```
public:
    String Name;
    String Company;
    int Age;
};
```

```
int main()
{ Employee employee1;
  employee1.Name = "Sabrina";
  employee1.Company = "YT-CodeBeauty";
  employee1.Age = 25;
}
```

→ Behaviour → Class Method → function inside a class.

```
→ class Employee {
  public:
      string Name;
      string Company;
      int Age;
      void IntroduceYourself() {
          cout << "Name - " << Name << endl;
          cout << "Company - " << Company << endl;
          cout << "Age - " << Age << endl;
  }
```

→ employee1.IntroduceYourself(); → invoking the function

→ there is a better approach of constructing our objects.

Constructor: is a special type of method that is invoked each time that an object of a class is created.

```
Employee employee1;
employee1.IntroduceYourself();

Employee employee2;
employee2.IntroduceYourself();
```

```
Name -
Company -
Age - -858993460
Name -
Company -
Age - -858993460
```

default constructor is a term to describe a constructor that is automatically generated by your compiler

Constructor
- → no return type
- → the same name as the class that it belongs to.
- → constructor must be public

```
class Employee {
public:
    string Name;
    string Company;
    int Age;
    Void IntroduceYourself() {
        cout << "Name - " << Name << endl;
        cout << "Company - " << Company << endl;
        cout << "Age - " << Age << endl;
    }
```

Parameters

```
    Employee (string name, string company, int age) {
        Name = name;
        Company = Company;
        Age = age;
    }

int main()
{
    Employee employee1 = ~~Employee();~~
                Employee ("Saldina", "YT-
                CodeBeauty"
    employee1.IntroduceYourself(); , 25);
```

```
Employee employee2 = Employee ("John", "Amazon"
                                , 35);
employee2.IntroduceYourself();

}
```

Four pillars for most important concepts of OPPS

1. encapsulation
2. abstraction
3. Inheritance
4. polymorphism

1. Encapsulation. The idea of encapsulation is idea of bundling or tying together data and methods that operate on that data so that they are grouped together within a class

→ A way for other classes to interact with the properties of my class well i provide very specific public functions that other class can invoke ~~and~~ and in that way interact with my data

Encapsulated properties → through its methods → methods are very often implemented as getters and setters

→

```
class Employee {
private:
    string Name;
    string Company;
    int Age;
public:
    void setName (string name) {
        Name = name;
    }
    string getName () {
        return Name;
    }
    void IntroduceYourself () {...}
    Employee (string name, string company, int age)
        {...}
};
```

setter

getter

→ void setAge (int age) {

if (age >= 18)

Age = age;

}

② Abstraction

abstraction means hiding complex things behind a procedure that makes those things look simple

→ abstract class

→ class AbstractEmployee {

virtual void AskForPromotion() = 0; → pure virtual method

};

→ contract

signing the contract

→ class Employee : AbstractEmployee {

}

→ Class Employee : AbstractEmployee {

private:

''

public:

''

void AskForPromotion() {

if (Age > 30)

std::cout << Name << "got promoted!" << std::endl;

else

std::cout << Name << ~~"got promoted!"~~ No promotion for you! << std::endl;

}

→ employee1. AskForPromotion();

employee2 - AskForPromotion();

③ Inheritance

Base class ⟶ super class or parent class

derived class ⟶ child class or subclass

→ class Developer : Employee {

public:

string FavProgrammingLanguage;

};

→ once you create your own constructor you lose default constructor

→ class Developer : Employee {
public:
string FavProgrammingLanguage;
Developer (string name, string company, int age, string favProgrammingLanguage)
: Employee (name, company, age)
{
FavProgrammingLanguage = favProgrammingLanguage;
}
};

→ class Developer : Employee {
public:
void FixBug() {
std::cout << ~~getName~~ Name (if protected) () << "fixed bugs using"
<< FavProgrammingLanguage << std::endl;
}
};

```
int main()
{
    Developer d = Developer ("Saldina", "YT-CodeBeauty"
                                        , 25, "C++");

    d.Fixbug();

}
```

Saldina fixed bug using C++

private by default

→ class Developer : Employee {

}

}

→ class Developer : public Employee {

}

→ class Teacher : public Employee {

public:

String Subject;

void PrepareLesson() {

std::cout << Name << "is preparing" << Subject << "lesson" << std::endl;

}

Teacher (string name, string company, int age, string subject)

: Employee (name, company, age)

{

Subject = subject;

}

};

```
int main()
{
    Developer d = Developer("Saldira", "YT-CodeBeauty",
                                25, "C++");
    Teacher t = Teacher("Jack", "Coal School", 35,
                                "History");

    t.PrepareLesson();
    t.AskForPromotion();
}
```

```
Jack is preparing History lesson
Jack got promoted!
```

⑨ Polymorphism

→ poly + morph ⟶ many forms

→ describes the ability of an object or a method to have many forms

→ The most common use of polymorphism is when a parent class reference is used to refer to a child class object

→ Employee* e1 = &d;
Employee* e2 = &t;

e1 → work();
e2 → work();

→ work method virtual

Q. What are some other programming paradigms other than OOPs?

Programming paradigms — method of classification of programming languages based on their features.

There are mainly two types of Programming paradigms.

- Imperative PP
- Declarative PP

1. Imperative PP

How to execute program logic and defines control flow as statements that change a program state.

(a) Procedural ~~Prog~~ PP

specifies the steps a program must take to reach the desired state, usually read in order from top to bottom.

(b) OOP organizes programs as objects, that contain some data and have same behavior.

(c) Parallel Programming Paradigm breaks a task into subtasks and focuses on executing them simultaneously at the same time.

(2) Declarative PP what to execute and defines program logic but not a detailed control flow

(a) Logical PP is based on formal logic, which refers to a set of sentences expressing facts & rules about how to solve a problem

(b) Functional PP ~~is a progr~~ where programs are constructed by applying and composing functions

(c) Database PP. model is used to manage data and information structured as fields, records and files.

Q. what is meant by structured programming

method of programming which consists of a completely structured control flow.

structure → block → set of Rules
→ definitive control flow

such if else, while for
block structures, subroutines

→ Nearly all PP include structured programming

→ Therefore the class is basically a template or blueprint for objects. Also one can create as many objects as they want based on a class.

→ objects consume space and have some characteristic behaviour e.g. a specific car.

→ Encapsulation is the process of binding data members and methods of a program together to do a specific job, without revealing unnecessary details.

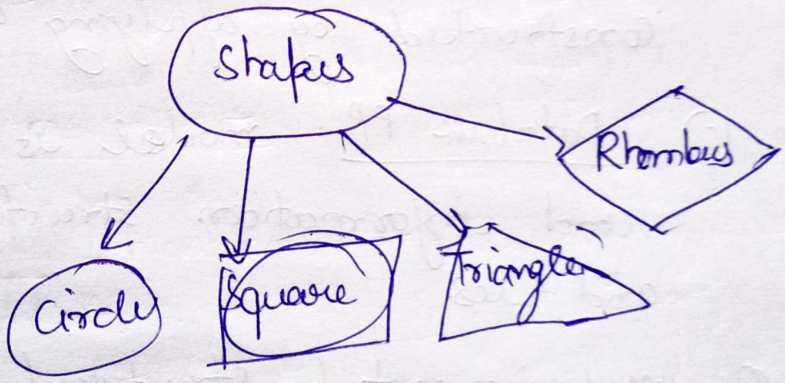
① Data hiding

② Data binding

→ Polymorphism

many

shapes



→ polymorphism refers to the process by which some code, data, method, or object behaves differently under different circumstances or contexts.

① Compile-time polymorphism (static or Early Binding)
② Run-time polymorphism (Dynamic or late Binding)

Q. How does C++ support polymorphism

Ans C++ is a OOP language.

Compile Time polymorphism

templates, function overloading and default arguments

Run time polymorphism

Virtual functions. takes the shape of functions based on type of object in reference and are resolved at runtime.

→ inheritance is the Mechanism by which an object or class is created using definition of another object or class.
↳ implementation simpler
→ code reuse.

Q. How much Memory does a class Occupy?
Ans Classes do not consume any memory. They are just a blueprint based on which objects are created. they actually initialize the class members and methods and therefore consume memory.

Q. Is it always necessary to create objects from class?

Ans No, An object is necessary to be created if the base class has non-static methods. But if the class has static methods, then objects don't need to be created. You can call the class method directly in this case, using the class name.

→ Constructor

Myclass myclassObject = new Myclass();

Constructor

Myclass *MyObjPtr = new Myclass();

Destructor

delete MyObjPtr;

→ types of Constructors in C++

Default Constructor which doesn't take any argument. it has no parameters.

Parameterized Constructor take some arguments.

Copy Constructor is a member function that initializes an object using another object of same class.

```
class ABC
{ int x;
  ABC (int y)
  {
   x = y;
  }
  ABC (ABC abc)
  {
    x = abc.x;
  }
}
```

→ A copy constructor will clone an object and its values, into another object, is provided that both the objects are of same class.

Destructor

Destructor free up the resources and memory occupied by an object. Destructor are automatically called when an object is being destroyed.

Q. Difference b/w a class and structure

Ans → structure is saved in the stack memory
→ class is saved in the heap memory
→ Data Abstraction cannot be achieved with the help of structure.
→ But with class, Abstraction is majorly used

Q. limitations of inheritance?

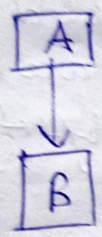
→ inheritance needs more time to process, as it needs to navigate through multiple classes for its implementation.

→ base class and child class are very tightly coupled together. — nested changes.

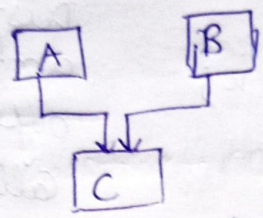
→ Complex for implementation → unexpected errors or incorrect effects.

Types of inheritance

- Single inheritance
- Multiple inheritance
- Multi-level inheritance
- Hierarchical inheritance
- Hybrid inheritance

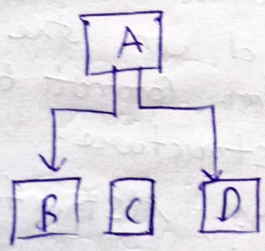


Single

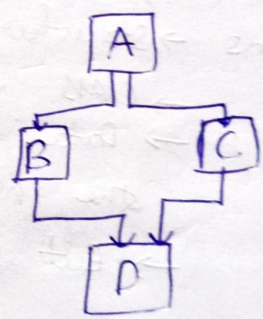


Multiple Inheritance

Multilevel



Hierarchical inheritance



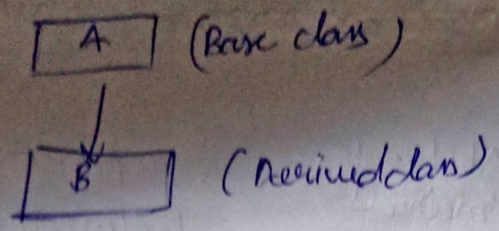
Hybrid

☆ Subclass ⟶ childclass

Q. Define a superclass?

Superclass is also a part of Inheritance. The superclass is an entity, which allows

subclasses or child classes to inherit from itself.



Q. <u>interface</u>

a special type of class, which contains methods, but not their definition.

To use an interface, you cannot create objects. Instead, you need to implement that interface and define the methods for their implementation.

→ static polymorphism → Method overloading or operator overloading

dynamic polymorphism → Method overriding

→ overloading (compile time polymorphism)

which an entity has multiple implementations with same name.

overriding (runtime polymorphism)

which an entity has the same name but its implementation changes during execution.

→ Data Abstraction is accomplished with the help of abstract methods or abstract classes.

Abstract Class → Abstract methods → implemented ✗ declared ✓

Subclass → define and implement them

inherits the abstract class

Q. Difference between abstract class and interface

Ans The main difference b/w two is that, when an interface is implemented, the subclass must define all its methods and provides its implementation. Whereas when an abstract class is inherited, the subclass does not need to provide the definition of its abstract method, until and unless the subclass is using it.

abstract class → abstract methods
abstract class → non-abstract methods

→ Access Modifiers → encapsulation

exception → special event
→ raised during the execution of a program at runtime, that brings the execution to a halt.

→ where user wants to do something for which program is not specified like undesirable input.

exception handling

exception handling is mechanism for identifying the undesirable states that the program can reach and specifying the desirable outcomes of such states.

Try-catch is the most common method used for handling exceptions in the program.

Garbage Collection

Garbage collection refers to the mechanism of handling the memory in the program. Through garbage collection, the unwanted memory is freed up by removed the objects that are no longer needed.

Note.

Java applications are based on OOPs models hence cannot be implemented without it.

C++ can be implemented without OOPs, as it also supports the C-like structural programming model.