**OOP:**

**Objects:** objects are real world entities

**Class:** these are the blueprint of the entities

Class contains properties and methods, poperties are attributes/variabkes, methods are funtions inside the class. Methods are also known as member function.

Access modifiers:

* **Private Member:** inside class, we write things in private: then it will not be accessible outside the class, default behaviour is private.

**Example:**

private:

methods/attribues

* **public Member:**

can be access in others inherited classes or from objects

* **protected Member:**

can be access only in other classes

**getter:**

these are the public function which are use to access the private member of class through objects, accessing indirectly through functions

**setter:**

these are the public function which are use to edit the private member of class through objects, accessing indirectly through functions

**Pillars or OOP:**

1. **encapsulation:**

wrapping up data and function inside single unit(class). Helps in data hiding

1. **Abstraction:**

Hiding unnecessary details and show necessary details to use

Access specifier are used to inplement

Can use abstract class.

1. **Inheritance:**

Members and methods of base class passed to the derived class.

Use for code reuseablility

After inheritance, 1st parent class constructor calls, then derived class constructor calls,

Derived class’s destrucor calls 1st then parent destructor

In case of parametrized constructor we do derived():Parent(){}

**Types of inheritance:**

* **Single level ineritance**
* **Multilevel** (parent to child then grand child)
* **Multiple inheritance** (class derived from two parents)

**Syntax:** Derived:public parent1.parent2{}

* Hierarical inheritance: (from one parent to multiple childs)
* Hybrid Inheritance: (combination of all of previous inheritance)

1. **Polymorphism:**

ability of objects to take multiple form, also known as constructor overloading

Types:

* **Compile time:**

we specify the return type and type of parameter so it decide at the complile time

**example** constructor overloading, function overloading

* **Run time:**

**Example:**

**function overriding**: parent and child class contain same name of function but different implementation

**Virtual function:**

We use keyword virtual to make a function virtual so derived class can be override and use

**Pure function:**

A constructor without body

**Constructor:**

These are special methods inside the class that inoked/call automatically at the time of object creation. It does not have return type.

Memory allocation happened when constructor called, only called at once.

It have same name like class.

**Types of constructor:**

1. Default constructor: (without parameters)
2. Parameterized constructor: (take parameter at object creation and use it when it invoke automatically)
3. Copy constructor:

Take data from other object and use that object’s data, use object as argument. We take reference of the object so if we will change data it will reflect in the original object

**Destructor:**

Call automatically and automatically deallocate the memory. Use to free-up space

~ClassName(){};

**Constructor overloading:**

Same type of constructor with some different datatype of number of arguents, when we give arguments it automatically detect which constructor should be called. This is an example of polymorphism.

**This:** it is a pointer points to current context of the object.

Like this->name=name; this->name is class’s attribute, name is parameter

**Shallow copy:**

it copies the value but points at same reference, so when we change data it reflects

**Deep copy:**

Normal copy like we copy the variable.

It copies all of the members of the object

**Abstract class:**

**Syntax:** abstract class{};

* These classes act as blueprint for other classes
* Can cannot make objects from abstract class
* Also known as interface
* Purpose is to become the blueprint for other classes

**Static:**

When we creat a variable and do increament it many times then it just do increment 1 seperately and do not increase more than 1.

But when use static word then the variable do not deleted from stack and to each updation it reflects in that variable.. work like reference to that variable

**Friend function:**

**Friend class:**