10/12

Object Oriented Programming:

class: group of real world entities.

object: real world entities.

Properties/attributes

functions/member functions/methods

Syntax

```
class Student{
    string name;
    float cgpa;
    void getPercentage()
    {
        cout << (cgpa*10) << endl;
    }
};
```

(no parantheris)

(attributes)

(note)

(note)

We cannot directly access the attributes need the help of Access Modifiers:

- Private
- Public
- Protected

getters → get the value from private

setters → set value "

Encapsulation

is wrapping
methods) in a
data hiding us

constructor

→ calle

class Car
{ Sta
{ Publi
C

};

Car
retur
→ U
→
→ P

this

Encapsulation

is wrapping up of data & member functions (methods) in a single unit, also implements data hiding using access specifiers

constructor

→ called automatically.

```
class Car
{   string name;
public:
    Car(string nameCar)
    {
        name = namCar;
    }
};
Car C1("Maruthi");
return 0;
```

Car() → constructor

→ used for Initialisation
→ same name as class
→ Memory Allocation happened when constructor is called.

→ called Automatically when object creation happens

'this' — same attribute name

~~Name → = name~~

this → name = name;
*this.name = name.

Car(..) Parametrised
Car() non-"

Constructor Overloading

multiples constructors with same and different parameters

copy constructor

special constructor (default) used to copy properties of one object into another.

```
Car C1("Maruthi", "New");
Car C2(C1) //=> copy constructor
```

Inheritance

* When properties and member functions of base class are passed on to the derived class.

* base class / super / parent class

↓

* derived / sub / child class

* Used for code reusability

Polymorphism

└ multiple Forms.

- compile Time
- Run Time

```
class Student
{ public:
    int id;
    string name;
};
int main()
{ Student S1;
  S1.id = 101;
  S1.name = "
  cout <<
}
```

```
class Student
{ public:
    int id
    Stud
    }
};
int main(
{ Studen
} Cout
```

compile Time pol

class - blueprint

```
#include <iostream>
using namespace std;
class student
{   public:
        int id;
        string name;
};
int main()
{   Student S1;
    S1.id = 101;
    S1.name = "Daisy";
    cout << S1.id << " " << S1.name << endl;
}
```

constructor
- it is a special function that calls automatically when object created
- same name as class
- To exhibit garbage values in class hence constructor is required.

```
class Student
{ public:
      int id;

      Student() {
        id = 0;
      }
};
int main()
{ Student S1;
  cout << S1.id;
}
```

Constructor Overloading

Having multiple constructors in the same class with different parameters

EX: compile time polymorphism.

```
Student() // constructor
{
    cout << "constructor being invoked";
}
Student(int id, string name)
{
    this->id = id;
    this->name = name;
}
```

Destructor

*called automatically when object is destroyed

* We need a destructor to clean up resources that an object uses before the object is destroyed.

```
int* id;
Student() { id = new int(1); }
~Student()
{
    delete id;
    cout << "destructor";
}
```

Data + Me

Getter: used to

Setter: Used to

Inheritance

Whose a

and behaviour

SYNTAX :

Types

Encapsulation

Visit Encap

"Wrapping data and the functions that operate on that data into single unit as class and restricting direct access to the data"

Data + Methods = Encapsulation

Getter: used to read private data.
Setter: Used to modify private data safely

Inheritance

Where a new class acquires the properties and behaviour of an existing class.

SYNTAX : class Parent {

};
class child : Public Base
{

};

Types of Inheritance

1. Single Inheritance A → B
2. Multi Level Inheritance A → B → C
3. Multiple Inheritance A, B → C
4. Hierarchical Inheritance A → B, A → C
5. Hybrid Inheritance Combination of above:

*Base class constructor is always called before derived class constructors.
*Destructor is called in reverse order of constructor.
*When same base class appears more than once virtual is needed.

Polymorphism many + forms

⇒ One name many forms

Polymorphism allows the same function name to behave differently based on the situation

Compile Time polymorphism:

- Decisions happens at compile time
  - * Function Overloading
  - * Operator Overloading

Function Overloading:

Same func name with different Parameter.

Operator Overloading:

Syntax:

returnType Operator opsymbol () { ... }

Virtual functions

is a member function that you expect to be redefined in derived classes.

Parent class

virtual void show() {

}

Child class

redefine.

Abstraction — Hiding all unnecessary details and showing only the essential or important sensitive details.

Parts.

Access specifiers → I method to Implement Abstraction

→ Public
→ Protected
→ Private

} incorporates Abstraction

Abstraction	encapsulation
* data hiding shows important parts using Public one way to use Abstraction	* data hiding. * Member fu nc & data. grouping.

II method to implement Abstraction:

Abstract classes and Pure Virtual functions

Abstract classes

- Object ✗
- child class = blueprint
- atleast one pure virtual function.