OOPS

Ref link:L

https://www.w3schools.com/cpp/cpp\_constructors.asp

C++ What is OOP?

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

## C++ What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

## class

Fruit

## objects

Apple

Banana

Mango

## Another example: class

Car

## objects

Volvo

Audi

Toyota

So,Appu this is important a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and functions from the class.

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and functions from the class.

## C++ Classes/Objects

C++ is an object-oriented programming language.

Everything in C++ is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an **object**. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

Attributes and methods are basically **variables** and **functions** that belongs to the class. These are often referred to as "class members".

A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

## Create a Class

Create a class called "MyClass":

class MyClass {       // The class  
  public:             // Access specifier  
    int myNum;        // Attribute (int variable)  
    string myString;  // Attribute (string variable)  
};

* The class keyword is used to create a class called MyClass.
* The public keyword is an **access specifier**, which specifies that members (attributes and methods) of the class are accessible from outside the class. You will learn more about [access specifiers](https://www.w3schools.com/cpp/cpp_access_specifiers.asp) later.
* Inside the class, there is an integer variable myNum and a string variable myString. When variables are declared within a class, they are called **attributes**.
* At last, end the class definition with a semicolon ;.

## Create an Object

In C++, an object is created from a class. We have already created the class named MyClass, so now we can use this to create objects.

To create an object of MyClass, specify the class name, followed by the object name.

To access the class attributes (myNum and myString), use the dot syntax (.) on the object:

### **Example**

Create an object called "myObj" and access the attributes:

class MyClass {       // The class  
  public:             // Access specifier  
    int myNum;        // Attribute (int variable)  
    string myString;  // Attribute (string variable)  
};  
  
int main() {  
  MyClass **myObj**;  // Create an object of MyClass  
  
  // Access attributes and set values  
  **myObj.myNum** = 15;   
  **myObj.myString** = "Some text";  
  
  // Print attribute values  
  cout << myObj.myNum << "\n";  
  cout << myObj.myString;  
  return 0;  
}

Appu first class ane datatypeto newclassname petti inside int string variables tisukuntam next oka objestnu ela create checi casll chestamante main loaa classnu object to tisukoni next aa class lo unna varablesnu dot to add chesukuntamokna pina chupinchinattu.

class MyClass idi calss name car anedi class

MyClass **myObj**;   idi object nano car anedi object

Colors avi attributes ante variables okna ..

Methods ante brake drive avi anni type of cars ku same kada

## Multiple Objects

## // Create a Car class with some attributes class Car {   public:     string brand;        string model;     int year; }; int main() {   // Create an object of Car   Car carObj1;   carObj1.brand = "BMW";   carObj1.model = "X5";   carObj1.year = 1999;   // Create another object of Car   Car carObj2;   carObj2.brand = "Ford";   carObj2.model = "Mustang";   carObj2.year = 1969;   // Print attribute values   cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";   cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";   return 0; } Class Methods

Methods are **functions** that belongs to the class.

There are two ways to define functions that belongs to a class:

* Inside class definition
* Outside class definition

In the following example, we define a function inside the class, and we name it "myMethod".

**Note:** You access methods just like you access attributes; by creating an object of the class and using the dot syntax (.):

### **Inside Example**

class MyClass {        // The class  
  public:              // Access specifier  
    void myMethod() {  // Method/function defined inside the class  
      cout << "Hello World!";  
    }  
};  
  
int main() {  
  MyClass myObj;     // Create an object of MyClass  
  myObj.myMethod();  // Call the method  
  return 0;  
}

outside Example:

To define a function outside the class definition, you have to declare it inside the class and then define it outside of the class. This is done by specifiying the name of the class, followed the scope resolution :: operator, followed by the name of the function:

### **Outside Example**

class MyClass {        // The class  
  public:              // Access specifier  
    void myMethod();   // Method/function declaration  
};  
  
// Method/function definition outside the class  
void **MyClass::myMethod()** {  
  cout << "Hello World!";  
}  
  
int main() {  
  MyClass myObj;     // Create an object of MyClass  
  myObj.myMethod();  // Call the method  
  return 0;  
}

## You can also add parameters: #include <iostream> using namespace std; class Car {   public:     int speed(int maxSpeed); }; int Car::speed(int maxSpeed) {   return maxSpeed; } int main() {   Car myObj; // Create an object of Car   cout << myObj.speed(200); // Call the method with an argument   return 0; }

## Constructors

A constructor in C++ is a **special method** that is automatically called when an object of a class is created.

To create a constructor, use the same name as the class, followed by parentheses ():

## Appu constructor ante powerful onera manam mamuluga class create chesedi functions performance kosame kada so aa function speciallga construction ga chesre ante aa class name iste .. object create chesinappudu automatic ga aa function call avuddi okana….

## class MyClass {     // The class   public:           // Access specifier     MyClass() {     // Constructor       cout << "Hello World!";     } }; int main() {   MyClass myObj;    // Create an object of MyClass (this will call the constructor)   return 0; }

pina class MyClass idi ante class

MyClass() idi special function ..same class name so constructor ade special fun antam..

MyClass myObj;     .. so ikkada myobj create chesina ventane automatic ga function call avuddi adene myclass anedi okna..

## Constructor Parameters

Constructors can also take parameters (just like regular functions), which can be useful for setting initial values for attributes.

The following class have brand, model and year attributes, and a constructor with different parameters. Inside the constructor we set the attributes equal to the constructor parameters (brand=x, etc). When we call the constructor (by creating an object of the class), we pass parameters to the constructor, which will set the value of the corresponding attributes to the same:

### **Example**

class Car {        // The class  
  public:          // Access specifier  
    string brand;  // Attribute  
    string model;  // Attribute  
    int year;      // Attribute  
    Car(string x, string y, int z) { // Constructor with parameters  
      brand = x;  
      model = y;  
      year = z;  
    }  
};  
  
int main() {  
  // Create Car objects and call the constructor with different values  
  Car carObj1("BMW", "X5", 1999);  
  Car carObj2("Ford", "Mustang", 1969);  
  
  // Print values  
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";  
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";  
  return 0;  
}

[Try it Yourself »](https://www.w3schools.com/cpp/trycpp.asp?filename=demo_constructor_param)

Just like functions, constructors can also be defined outside the class. First, declare the constructor inside the class, and then define it outside of the class by specifying the name of the class, followed by the scope resolution :: operator, followed by the name of the constructor (which is the same as the class):

### **Example**

class Car {        // The class  
  public:          // Access specifier  
    string brand;  // Attribute  
    string model;  // Attribute  
    int year;      // Attribute  
    Car(string x, string y, int z); // Constructor declaration  
};  
  
// Constructor definition outside the class  
Car::Car(string x, string y, int z) {  
  brand = x;  
  model = y;  
  year = z;  
}  
  
int main() {  
  // Create Car objects and call the constructor with different values  
  Car carObj1("BMW", "X5", 1999);  
  Car carObj2("Ford", "Mustang", 1969);  
  
  // Print values  
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";  
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";  
  return 0;  
}

## Access Specifiers

## Access Specifiers

By now, you are quite familiar with the public keyword that appears in all of our class examples:

### **Example**

class MyClass {  // The class  
  **public:**        // Access specifier  
    // class members goes here  
};

[Try it Yourself »](https://www.w3schools.com/cpp/trycpp.asp?filename=demo_access_public)

The public keyword is an **access specifier.** Access specifiers define how the members (attributes and methods) of a class can be accessed. In the example above, the members are public - which means that they can be accessed and modified from outside the code.

However, what if we want members to be private and hidden from the outside world?

In C++, there are three access specifiers:

* public - members are accessible from outside the class
* private - members cannot be accessed (or viewed) from outside the class
* protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes. You will learn more about [Inheritance](https://www.w3schools.com/cpp/cpp_inheritance.asp) later.

In the following example, we demonstrate the differences between public and private members:

### **Example**

class MyClass {  
  **public:**    // Public access specifier  
    int x;   // Public attribute  
  **private:**   // Private access specifier  
    int y;   // Private attribute  
};  
  
int main() {  
  MyClass myObj;  
  myObj.x = 25;  // Allowed (public)  
  myObj.y = 50;  // Not allowed (private)  
  return 0;  
}

If you try to access a private member, an error occurs:

## error: y is private

## **Note:** By default, all members of a class are private if you don't specify an access specifier: class MyClass {   int x;   // Private attribute   int y;   // Private attribute };

## Appu access specifier declare cheyaledu so adi private ga tiskuntundi..by default private.

# **C++ Encapsulation**

## The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must declare class variables/attributes as private (cannot be accessed from outside the class). If you want others to read or modify the value of a private member, you can provide public **get** and **set** methods.

## Access Private Members

To access a private attribute, use public "get" and "set" methods:

## Access Private Members

To access a private attribute, use public "get" and "set" methods:

### **Example**

#include <iostream>  
using namespace std;  
  
class Employee {  
  private:  
    // Private attribute  
    int salary;  
  
  public:  
    // Setter  
    void setSalary(int s) {  
      salary = s;  
    }  
    // Getter  
    int getSalary() {  
      return salary;  
    }  
};  
  
int main() {  
  Employee myObj;  
  myObj.setSalary(50000);  
  cout << myObj.getSalary();  
  return 0;  
}

[Try it Yourself »](https://www.w3schools.com/cpp/trycpp.asp?filename=demo_encapsulation)

#### **Example explained**

The salary attribute is private, which have restricted access.

The public setSalary() method takes a parameter (s) and assigns it to the salary attribute (salary = s).

The public getSalary() method returns the value of the private salary attribute.

Inside main(), we create an object of the Employee class. Now we can use the setSalary() method to set the value of the private attribute to 50000. Then we call the getSalary() method on the object to return the value.

## Why Encapsulation?

* It is considered good practice to declare your class attributes as private (as often as you can). Encapsulation ensures better control of your data, because you (or others) can change one part of the code without affecting other parts
* Increased security of data

## Appu encapsulation lo private ane access specifier undi ante adi varable ite setter to call chesukopvali getter to assign chesukovachu .. alane setsalary ani main lo call cheyavachu … this all for security wise use avutundi.

# **C++ Inheritance**

## Inheritance

In C++, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

* **derived class** (child) - the class that inherits from another class
* **base class** (parent) - the class being inherited from

To inherit from a class, use the : symbol.

In the example below, the Car class (child) inherits the attributes and methods from the Vehicle class (parent):

### **Example**

// Base class  
class Vehicle {  
  public:  
    string brand = "Ford";  
    void honk() {  
      cout << "Tuut, tuut! \n" ;  
    }  
};  
  
// Derived class  
**class Car: public Vehicle** {  
  public:  
    string model = "Mustang";  
};  
  
int main() {  
  Car myCar;  
  myCar.honk();  
  cout << myCar.brand + " " + myCar.model;  
  return 0;  
}

#### **Why And When To Use "Inheritance"?**

- It is useful for code reusability: reuse attributes and methods of an existing class when you create a new class.

## Appu inheritance ante varasatvam…

## So parental class lo declare chesina methods anni child class (ade aa child class ku create chesina object dvara) objectlo call cheyavachu..

## Above dantlo vehicle main parental class ante base class … so ikkada car ane inko class child class or derived class .. so vehicle class lo unna honk ane method or function nu nuvvu car ane class object create chesukoni ante car my car ani object create chesukoni aa objectlo vehicle methodnu ( myCar.honk();call cheyavachu okna …

## Note:

## Base class leda parental class nu child class lo vadali ant : ee symbol vadali okne.. class**Car:**public**Vehicle**

## ila pina chupinchinattu : colon vadina child class lo matrame parental methods call cheyagalavu.

# **C++ Multilevel Inheritance**

A class can also be derived from one class, which is already derived from another class.

In the following example, MyGrandChild is derived from class MyChild (which is derived from MyClass).

// Base class (parent)  
class MyClass {  
  public:  
    void myFunction() {  
      cout << "Some content in parent class." ;  
    }  
};  
  
// Derived class (child)  
class MyChild: public MyClass {  
};  
  
// Derived class (grandchild)  
class MyGrandChild: public MyChild {  
};  
  
int main() {  
  MyGrandChild myObj;  
  myObj.myFunction();  
  return 0;  
}

Appu pina subbarayudu myclass , yesho mychild , Shreyas my grandchild anni classes colon dwara connect ayyayi so ippudu Shreyas ane class subbarayudu ane class function nu use cheyavachu … endukante colon ane symbol to yesho yesho nundi shrey connection undi so u can use base class methods to grand child ade multilevel inheretence antaru.

Note : colon symbol dwara base class and derived class ade parent child connection declared alane dot(.) ane symbol dwara calling their functions ade methods okana..

# **C++ Multiple Inheritance**

A class can also be derived from more than one base class, using a **comma-separated list:**

### **Example**

### **Example**

// Base class  
class MyClass {  
  public:  
    void myFunction() {  
      cout << "Some content in parent class." ;  
    }  
};  
  
// Another base class  
class MyOtherClass {  
  public:  
    void myOtherFunction() {  
      cout << "Some content in another class." ;  
    }  
};  
  
// Derived class  
**class MyChildClass: public MyClass, public MyOtherClass** {  
};  
  
int main() {  
  MyChildClass myObj;  
  myObj.myFunction();  
  myObj.myOtherFunction();  
  return 0;  
}

Appu multiple inheretence ante .. iddaru fathers sonku properties ichiinattu , nuvvu declare chestunnappudu comma to declare chesuko colonto patu rendu base names commato (,) to declare appu ante u can call both base clases methods ade functions okna.. below unnay chudu..

**class MyChildClass: public MyClass, public MyOtherClass …idi child class declaration okna**

**ikkda myclass anedi oka base class alane my otherclass anedi inko base class.**

**Note: base class function call chestunnapudu dot vadite saripoddi..below laga object create chese call cheyali ..**

MyChildClass myObj;  
  myObj.myFunction();  
  myObj.myOtherFunction();

# **C++ Inheritance Access**

## Access Specifiers

You learned from the [Access Specifiers](https://www.w3schools.com/cpp/cpp_access_specifiers.asp) chapter that there are three specifiers available in C++. Until now, we have only used public (members of a class are accessible from outside the class) and private (members can only be accessed within the class). The third specifier, protected, is similar to private, but it can also be accessed in the **inherited** class:

### **Example**

// Base class  
class Employee {  
  **protected: // Protected access specifier**  
    int salary;  
};  
  
// Derived class  
class Programmer: public Employee {  
  public:  
    int bonus;  
    void setSalary(int s) {  
      salary = s;  
    }  
    int getSalary() {  
      return salary;  
    }  
};  
  
int main() {  
  Programmer myObj;  
  myObj.setSalary(50000);  
  myObj.bonus = 15000;  
  cout << "Salary: " << myObj.getSalary() << "\n";  
  cout << "Bonus: " << myObj.bonus << "\n";  
  return 0;  
}

Appu protect ani pettukunte matram private unna protect unna set and get use to access that variable ade attribute

So below

**protected: // Protected access specifier**  
    int salary;

dinni use cheyalante setSalary(int s) ani alane

getSalary() {  
      return salary ivi ila vadukovali

# **C++ Polymorphism**

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

Like we specified in the previous chapter; [**Inheritance**](https://www.w3schools.com/cpp/cpp_inheritance.asp) lets us inherit attributes and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a base class called Animal that has a method called animalSound(). Derived classes of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

Example:

// Base class  
class Animal {  
  public:  
    void animalSound() {  
      cout << "The animal makes a sound \n";  
    }  
};  
  
// Derived class  
class Pig : public Animal {  
  public:  
    void animalSound() {  
      cout << "The pig says: wee wee \n";  
    }  
};  
  
// Derived class  
class Dog : public Animal {  
  public:  
    void animalSound() {  
      cout << "The dog says: bow wow \n";  
    }  
};

Now we can create Pig and Dog objects and override the animalSound() method:

Example:

// Base class  
class Animal {  
  public:  
    void animalSound() {  
      cout << "The animal makes a sound \n";  
    }  
};  
  
// Derived class  
class Pig : public Animal {  
  public:  
    void animalSound() {  
      cout << "The pig says: wee wee \n";  
    }  
};  
  
// Derived class  
class Dog : public Animal {  
  public:  
    void animalSound() {  
      cout << "The dog says: bow wow \n";  
    }  
};  
  
int main() {  
  Animal myAnimal;  
  Pig myPig;  
  Dog myDog;  
  
  myAnimal.animalSound();  
  myPig.animalSound();  
  myDog.animalSound();  
  return 0;  
}

Output:

The animal makes a sound  
The pig says: wee wee  
The dog says: bow wow

#### **Why And When To Use "Inheritance" and "Polymorphism"?**

- It is useful for code reusability: reuse attributes and methods of an existing class when you create a new class.

Note:

Appu inheritance lo parental classes two tisukoni aa properties oka derived class ade child class lo comma dwara inherit chesi a parents iddarilo unde methods call chesaru..

Ade Polymorphysm lo oka parent two child classes tisukoni aa child classes lo aa parent method nu call chesadu ade override cheyadam ante oka child classlo call chesi alane inko child classlo same parent methodnu second time call cheste fist childku work avutundio second childku work avutundi .. okate action 3 times use cheste anni sarlu aa method override ayindi …

Above dantlo

1. class Animal
2. class Pig
3. class Dog  ee 3 vatillo same name toni animalsound method undi but 3 different implimentations … but nuvvu main function lo call cheste aa methods vastay ante okka animalsound ane method pig ku dog ku different actions display avutay ante ade method override avutu untundi okana..
4. ade one method(animalSound) with many forms (The animal makes a sound, The pig says: wee we, The dog says: bow wow veetini implemetations antam , pina programlo animalsound daggara{} ee bracelo rasaru ) okna ikkada actionsound anedi okate name but print ayyedi different okna( ante aa main functionlo manam animal.pig,dog ani seperatega call chesam anduku animal sounddi,pig sound di pigku dog sounddi dogku print avuddi) o0kana..

## C++ Files

## Appu files ante textfile mana system lo create chesedaniki .. okavela already file unte read chesedaniki … alane create read wriote chesedaniki 3 statemenmts unnay alane firstlo library add chesuko kinda code undi use chesuko..

## C++ Files

The fstream library allows us to work with files.

To use the fstream library, include both the standard <iostream> **AND** the <fstream> header file:

### **Example**

#include <iostream>  
#include <fstream>

There are three classes included in the fstream library, which are used to create, write or read files:

|  |  |
| --- | --- |
| **Class** | **Description** |
| ofstream | Creates and writes to files |
| ifstream | Reads from files |
| fstream | A combination of ofstream and ifstream: creates, reads, and writes to files |

## Create and Write To a File

To create a file, use either the ofstream or fstream class, and specify the name of the file.

To write to the file, use the insertion operator (<<).

### **Example**

#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main() {  
  // Create and open a text file  
  ofstream MyFile("filename.txt");  
  
  // Write to the file  
  MyFile << "Files can be tricky, but it is fun enough!";  
  
  // Close the file  
  MyFile.close();  
}

#### **Why do we close the file?**

It is considered good practice, and it can clean up unnecessary memory space.

## Read a File

To read from a file, use either the ifstream or fstream class, and the name of the file.

Note that we also use a while loop together with the getline() function (which belongs to the ifstream class) to read the file line by line, and to print the content of the file:

### **Example**

// Create a text string, which is used to output the text file  
string myText;  
  
// Read from the text file  
ifstream MyReadFile("filename.txt");  
  
// Use a while loop together with the getline() function to read the file line by line  
while (getline (MyReadFile, myText)) {  
  // Output the text from the file  
  cout << myText;  
}  
  
// Close the file  
MyReadFile.close();

# **C++ Exceptions**

## C++ Exceptions

When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an **exception** (throw an error).

## C++ try and catch

Exception handling in C++ consist of three keywords: try, throw and catch:

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The throw keyword throws an exception when a problem is detected, which lets us create a custom error.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

### **Example**

try {  
  // Block of code to try  
  throw exception; // Throw an exception when a problem arise  
}  
catch () {  
  // Block of code to handle errors  
}

Consider the following example:

### **Example**

try {  
  int age = 15;  
  if (age >= 18) {  
    cout << "Access granted - you are old enough.";  
  } else {  
    throw (age);  
  }  
}  
catch (int myNum) {  
  cout << "Access denied - You must be at least 18 years old.\n";  
  cout << "Age is: " << myNum;  
}

[Try it Yourself »](https://www.w3schools.com/cpp/trycpp.asp?filename=demo_try_catch)

#### **Example explained**

We use the try block to test some code: If the age variable is less than 18, we will throw an exception, and handle it in our catch block.

In the catch block, we catch the error and do something about it. The catch statement takes a **parameter**: in our example we use an int variable (myNum) (because we are throwing an exception of int type in the try block (age)), to output the value of age.

If no error occurs (e.g. if age is 20 instead of 15, meaning it will be be greater than 18), the catch block is skipped:

### **Example**

int age = 20;

[Try it Yourself »](https://www.w3schools.com/cpp/trycpp.asp?filename=demo_try_catch2)

You can also use the throw keyword to output a reference number, like a custom error number/code for organizing purposes:

### **Example**

try {  
  int age = 15;  
  if (age >= 18) {  
    cout << "Access granted - you are old enough.";  
  } else {  
    throw 505;  
  }  
}  
catch (int myNum) {  
  cout << "Access denied - You must be at least 18 years old.\n";  
  cout << "Error number: " << myNum;  
}

[Try it Yourself »](https://www.w3schools.com/cpp/trycpp.asp?filename=demo_try_catch3)

## Handle Any Type of Exceptions (...)

If you do not know the throw **type** used in the try block, you can use the "three dots" syntax (...) inside the catch block, which will handle any type of exception:

### **Example**

try {  
  int age = 15;  
  if (age >= 18) {  
    cout << "Access granted - you are old enough.";  
  } else {  
    throw 505;  
  }  
}  
catch (...) {  
  cout << "Access denied - You must be at least 18 years old.\n";  
}

## Appu Error handling ne throw an exception antamu… okna

## Ikkada try ante {} ee bracelo emunte adi try chestundi adi success aite ok ledante else ani manam comment raste adi tisukuntundi ..next catchlo manam try ku execution time em jarigindi ane situation ikkada catchlo catch avutundi .. catchlo parameter pass cheste trowlo vachindi tisukoni display chestundi.. ledu ante jast catchlo aa condition try di work ite ok ledante no ani catch lo undedi print avutundi … practice the above one

## Try …{} bracelo action checking..

## Throw lo man comment mana mem raste adi ..

## Catchlo try action butty answer vastundi .. trowlo unde comment catchku parameterga pass kuda chesi comment pettukovachu ..

## Okna

## Nuvvu try raseppudu try ani bracepettukuntavu aa brace madyalo aa code unchali okna..next catch ani manam edi kavalanna comment rasukoni pettavachu ledante trow lo vachina di pass chesi catch lo rayavachu….