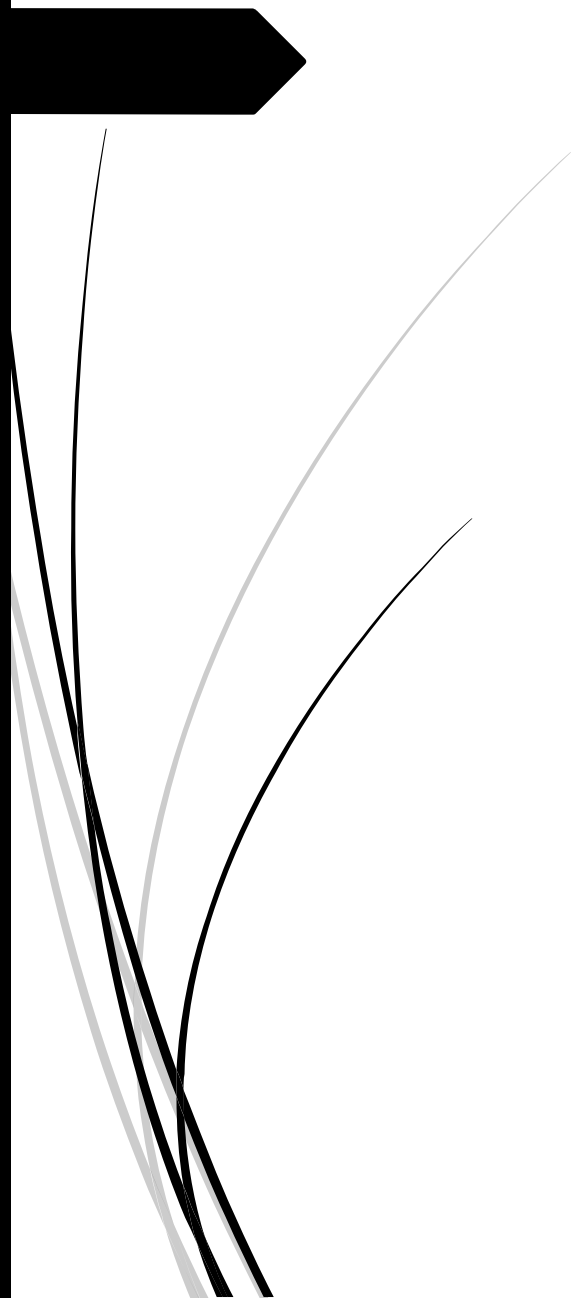
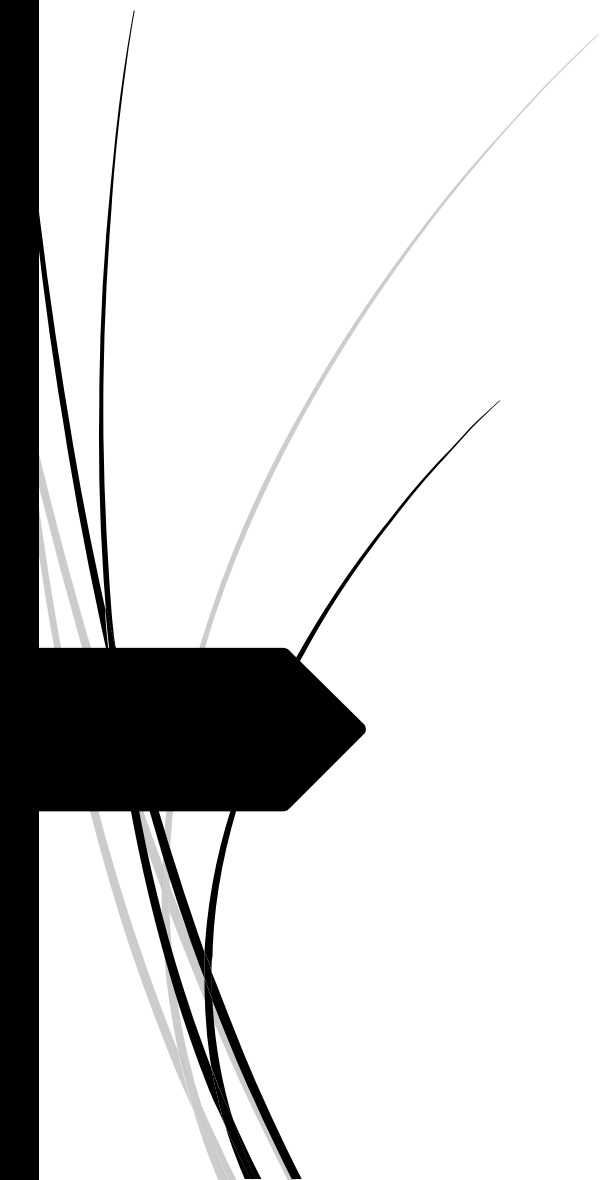
Concepts of Object Oriented Programming



- → Programming Paradigm
- → Procedure oriented programming paradigm
- → Object oriented programming paradigm
- → Advantages of OOPs
- → Basic concepts of OOPs



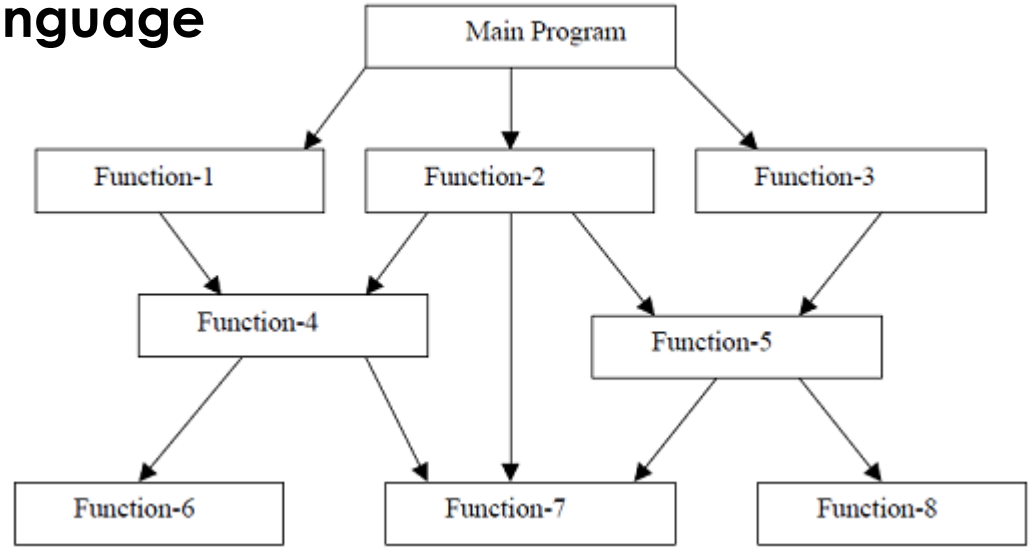
Programming paradigm

Programming paradigm

- Def : Programming paradigm are a way to classify programming languages according to the style of computer programming
- Some paradigm gives more important to the procedure, others gives more important to the data
- The term paradigm used for reduce the complexity of the programming language

Procedure oriented Programming paradigm

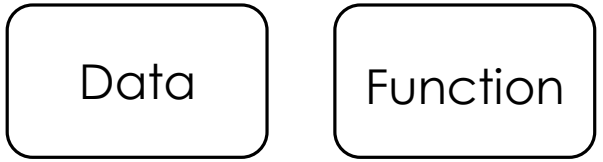
- It specifies a series of well structured steps and procedure to compose a program
- It contain a systematic order
- Also called top-down language



- Complex type programming paradigm
- For reduce complexity : use functions and modules
- Eg : C, Pascal, Basic, Fortran

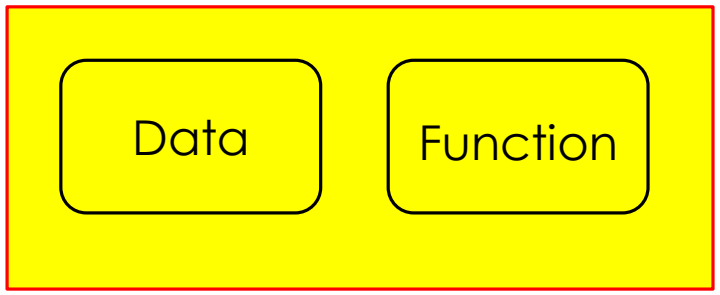
Reason for increase the complexity

- Give important to procedure than data
- to add a new data item, should rewrite the program
- Create a new data-type is difficult (no user defined datatype)
- Data and functions are treated as separately



Object oriented Programming paradigm

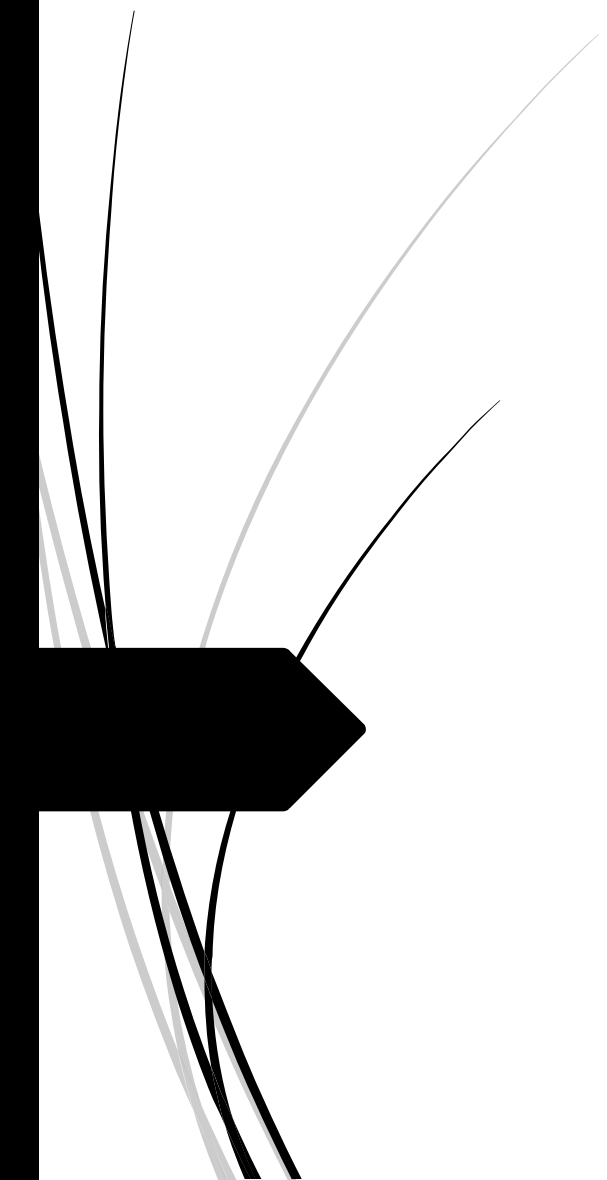
- It eliminate the drawback of POP Paradigm
- Bottom to up
- Data and functions are treated as a single unit



Object

Advantages of OOPs

- Modularity (divide Large program by module)
- Allow data abstraction (Hide / Protect data from user)
- Good for defining abstract data-types
- code re-usability
- Real world data entities can easily created
- Support to create new data-types (User defined)



Basic Concepts Of OOPs

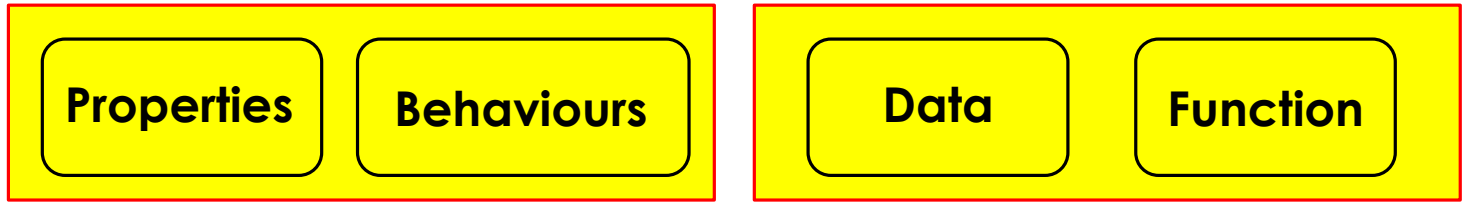
Basic concepts of OOPs

- Object
- Class
- Data Abstraction
- Data Encapsulation
- Modularity
- Inheritance
- Polymorphism

Object

- Anything around us can be treated as an object
- It have properties and behaviours

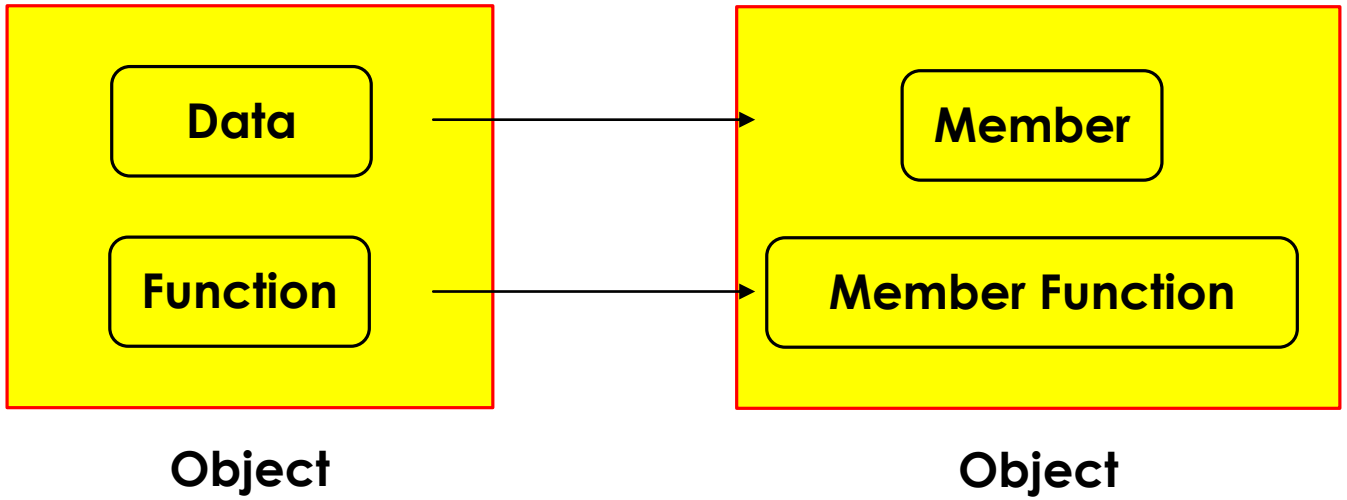
- The object combines data and faction as a single unit



Real-world Object

OOP Object

- The function inside the object is called member function
- And data is called member



- Object define inside the class

Classes

- Class is a user defined data type
- Class is collection of objects with similar attributes(properties)

- Class declare using the keyword "class"
- Class is a blueprint of objects

- Syntax

```
class class_name
{
    private:
        variable declarations;
        function declarations;
    public:
        variable declarations;
        function declarations;
};
```

```
class Student
{
    public:                     // Access specifier
        string name;            // Data Members

        void display()          // Member Functions()
            {
                cout << "Student Name is: " << name;
            }
};

int main()
{
    Student obj;                // Declare an object of class

    obj.name = "Appu";          // accessing data member
    obj.display();              // accessing member function
}
```

Data Abstraction

- It is an essential features of OOP
- Hiding details from out standers
- It gives the output without showing the details
  - Public : visible in main function (not hide)
  - Private : only visible inside the class (hide)
  - Protected : only visible in derived classes (hide)

```
class Student
{
    public:
        float mark;

        void display()
            {
                cout << "Student Mark is: " << mark;
            }
};
```

```
class Student
{
    private:
    float mark = 35;


    public:
        void display()
            {
                cout << "Student Mark is: " << mark;
            }
};
```

Data Encapsulation

- Binding of data and functions together
- Keeps both data and function safe from main program / outside interface
  - Private : only visible inside the class (hide)
  - Protected : only visible in derived classes (hide)

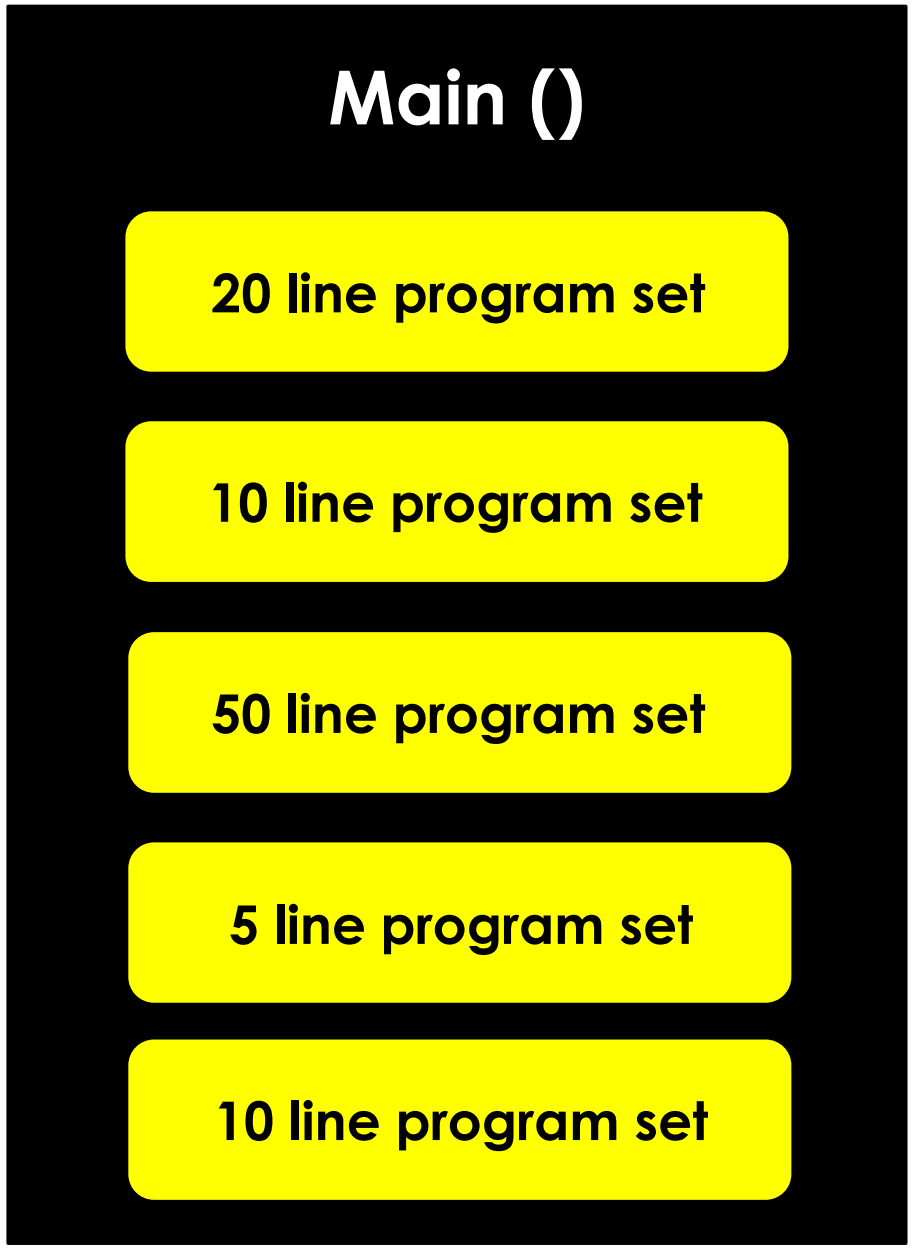
```
class Student
{
    protected:
        float mark = 0;
};
```

```
class Appu : public Student
{
    public :
        void english(float m)
            {
                mark = m;
            }

        void display()
            {
                cout << "English Mark is: " << mark;
            }
};
```

```
int main()
{
    Appu obj;

    obj.english(31);
    obj.display();
}
```

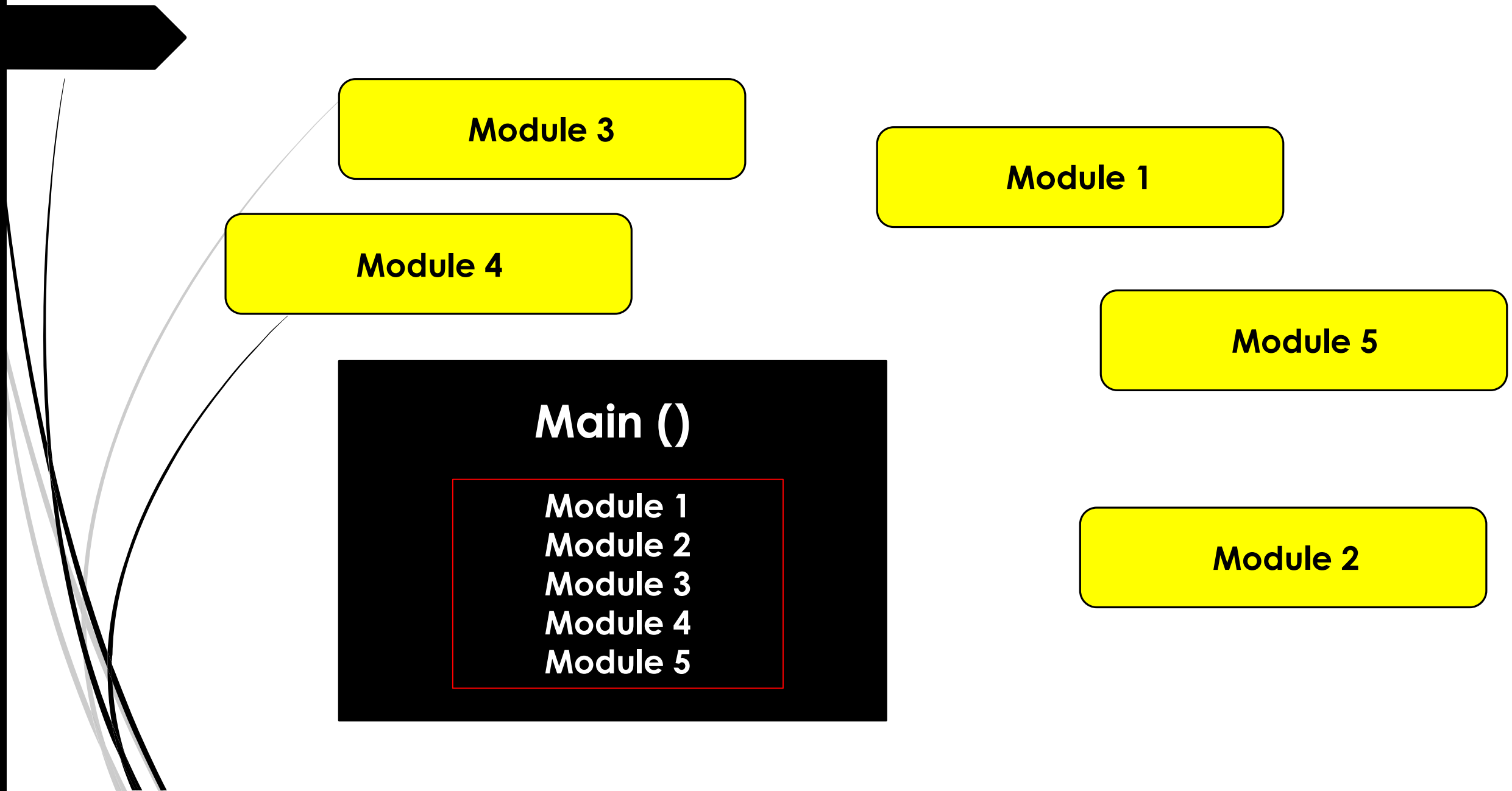
Modularity

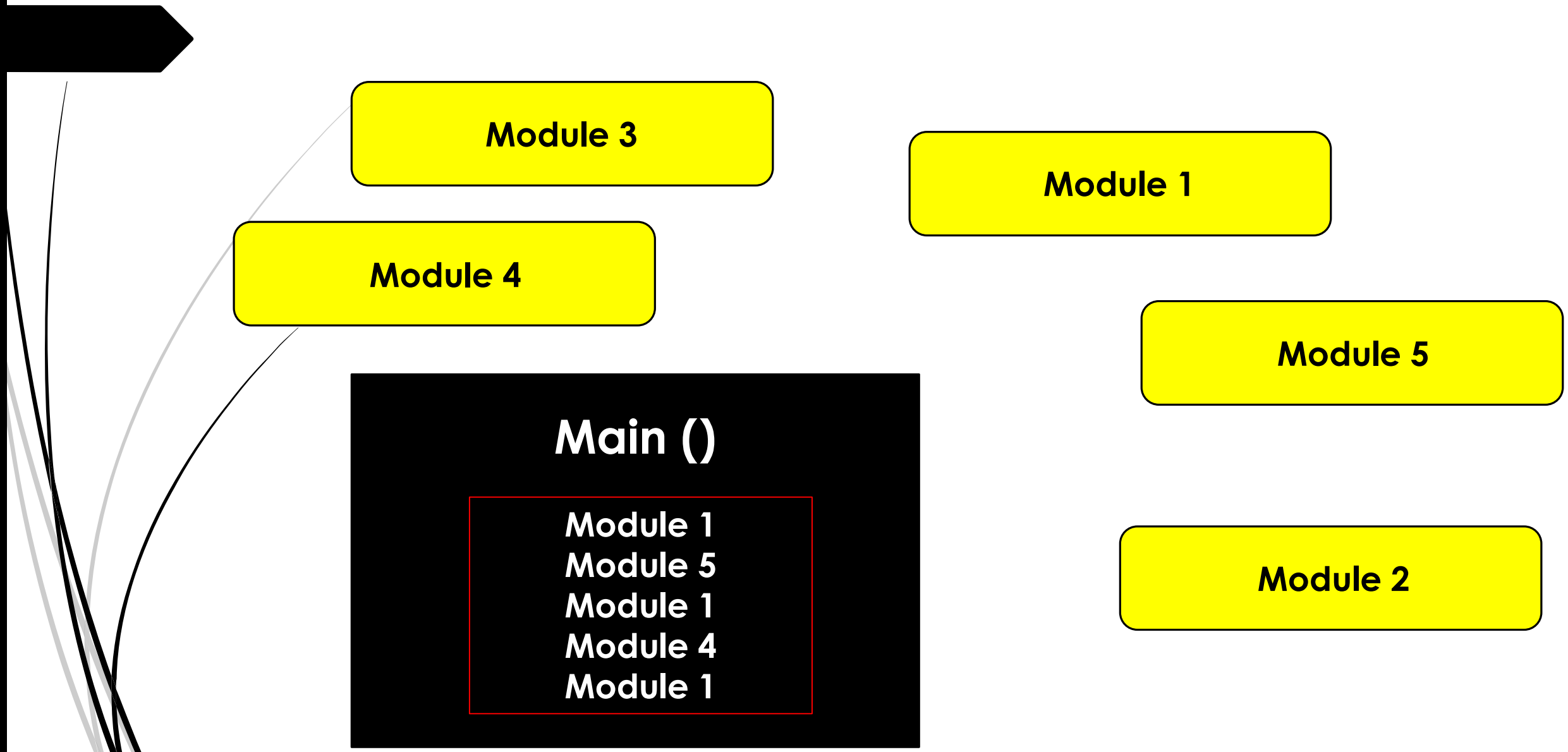
- All programs are written / divide into modules
- It reduce the complexity of program
- And each module can be used for another project (code re-usability)
- Each module execute when call the module using function name



Main()

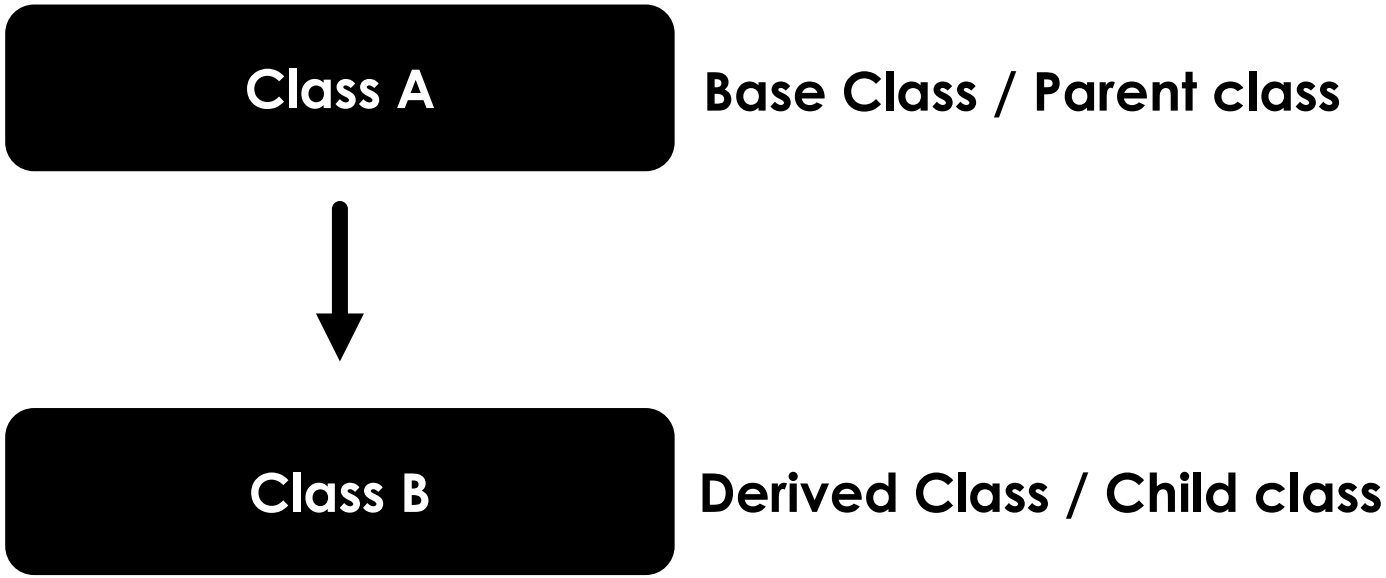
Total = 95 line program


Module 3
Module 1
Module 4
Module 5
Main ()
Module 1
Module 2
Module 3
Module 4
Module 5
Module 2

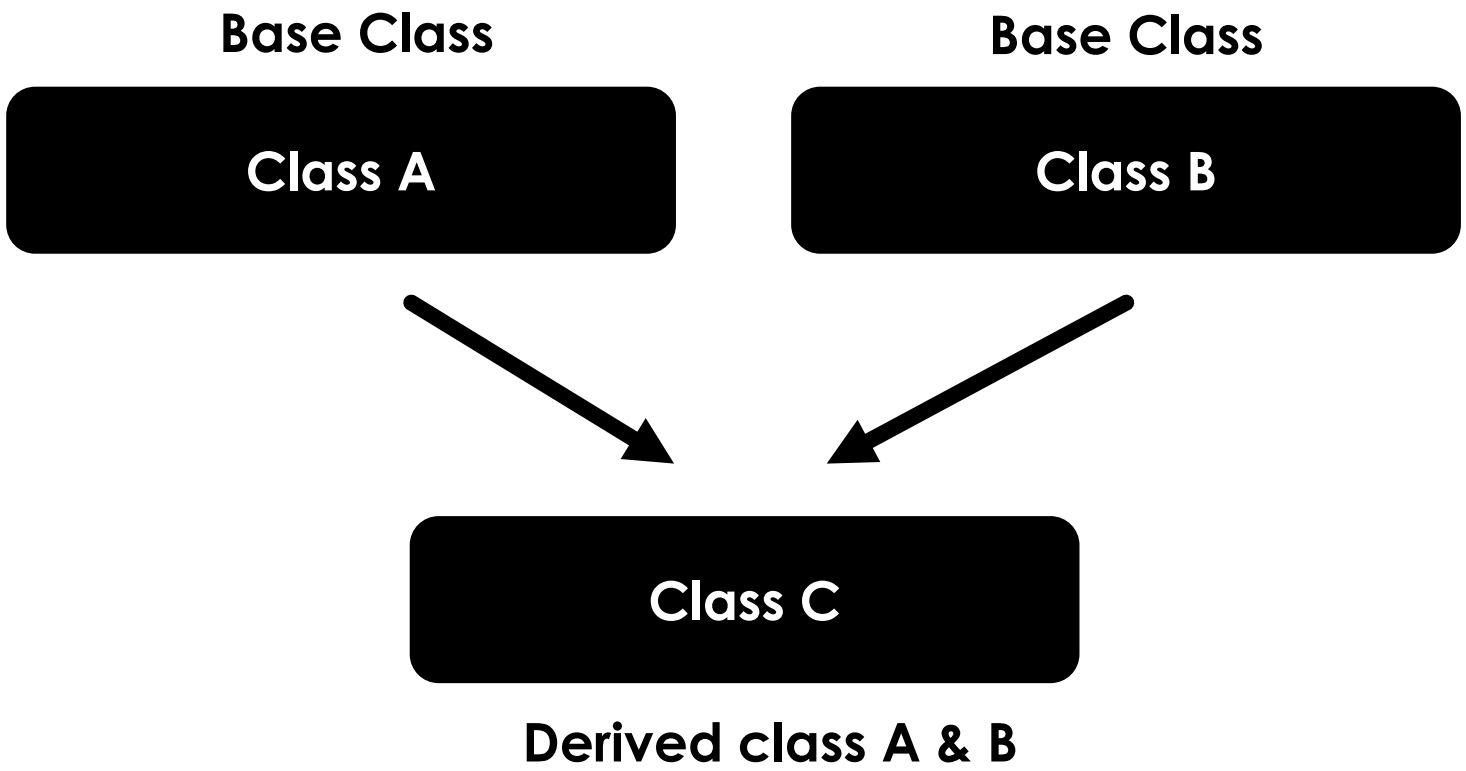

Module 3
Module 1
Module 4
Module 5
Main ()
Module 1
Module 5
Module 1
Module 4
Module 1
Module 2

- Inheritance
  - One class derived from another one
  - One class showing the property of another class
    - Single Inheritance
    - Multiple Inheritance
    - Hierarchical Inheritance
    - Multilevel Inheritance
    - Hybrid Inheritance

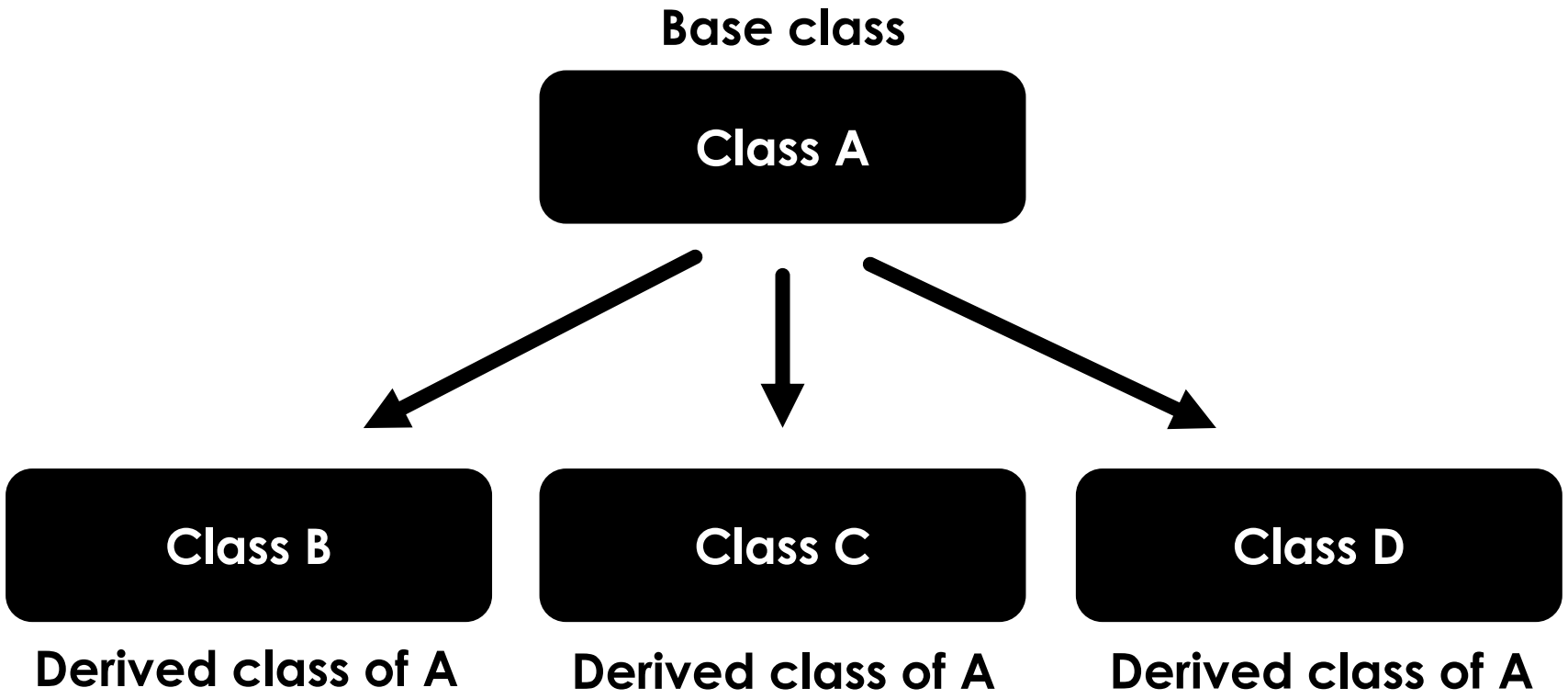
Single Inheritance



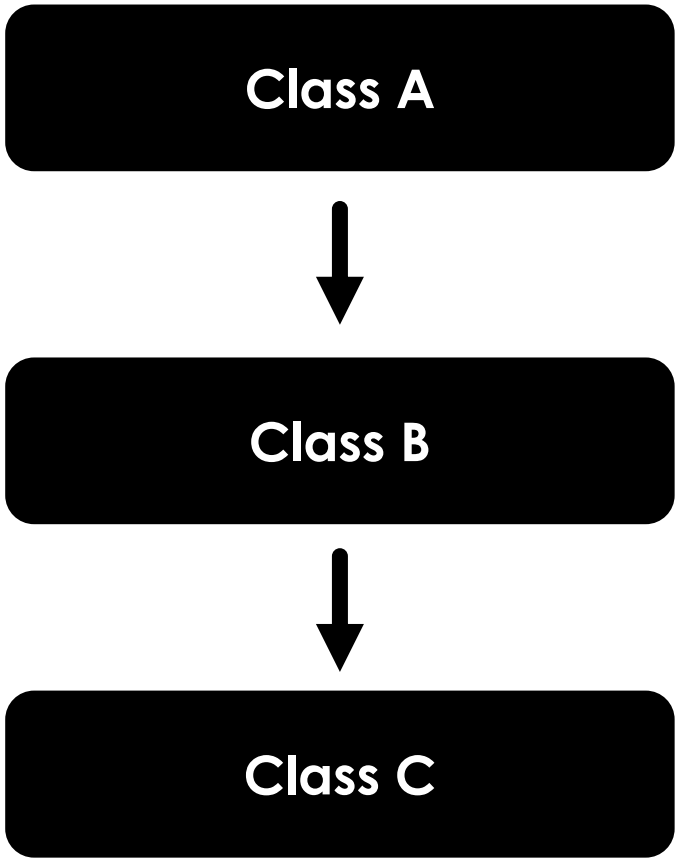
Multiple Inheritance



Hierarchical Inheritance



Multilevel Inheritance

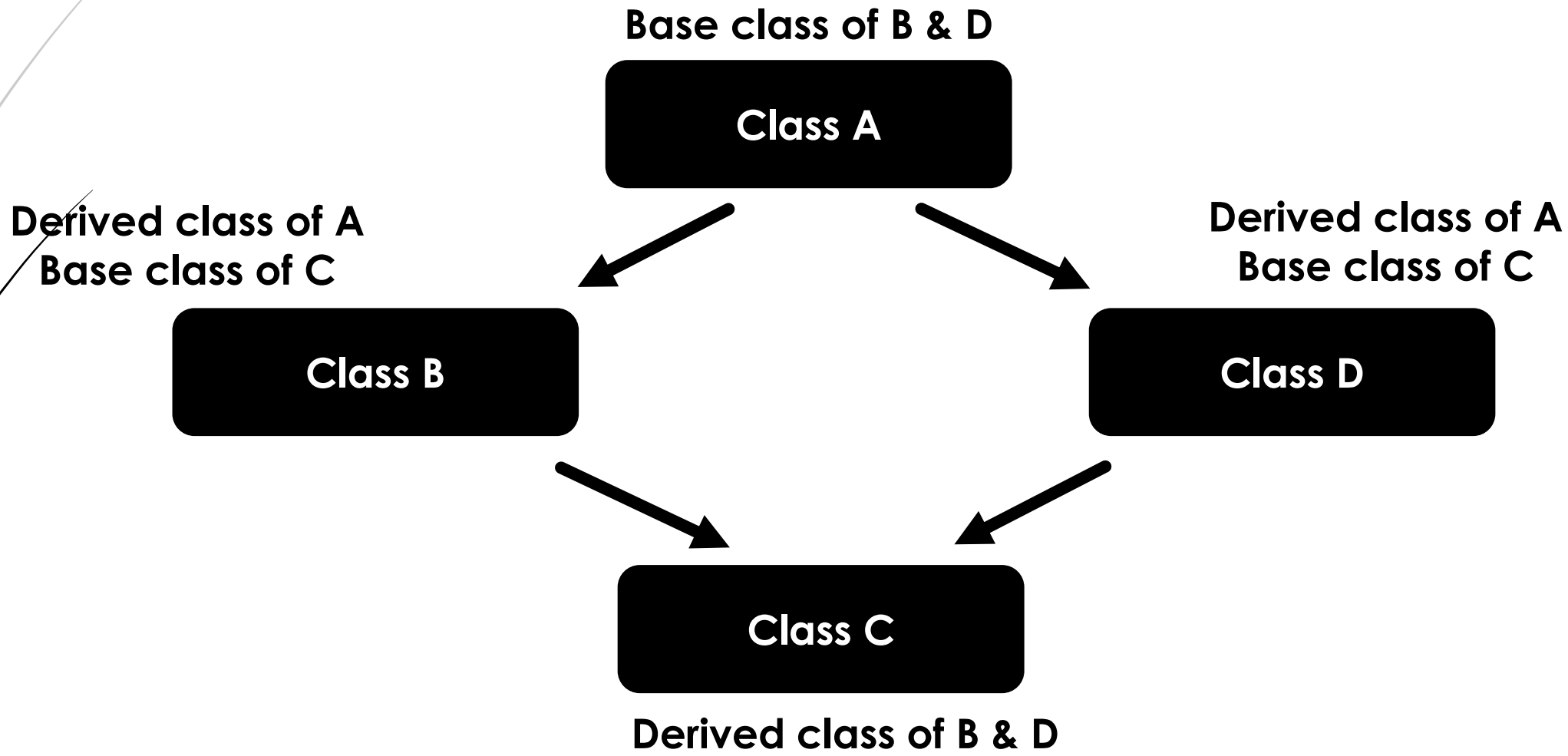


Base class of B

Derived class of A
Base class of C

Derived class of B

Hierarchical Inheritance



Polymorphism

- One function or operator or any other symbols can be used for different use called polymorphism
- Operator overloading
  - One operator is used for more than one operation is known as operator overloading
  - Ex : + (4 + 6) “hello” + “world”
- Function overloading
  - One Function name is used for more than one functions is known as function overloading

```
int main()
{
    int i = 4;
    int f = 6;

    display();
    display(f);
    display(i, f);
}
```

```
void display()
{
    cout<<"\ndisplay funtion call";
}
```

```
void display(int f)
{
    cout<<"\n\nf : "  <<f;
}
```

```
void display(int i, int f)
{
    cout<<"\n\nsum :"  << i + f ;
}
```