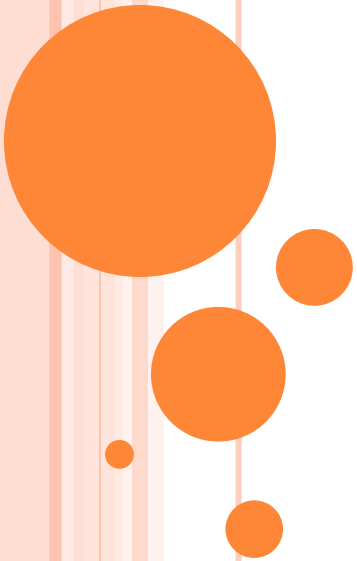
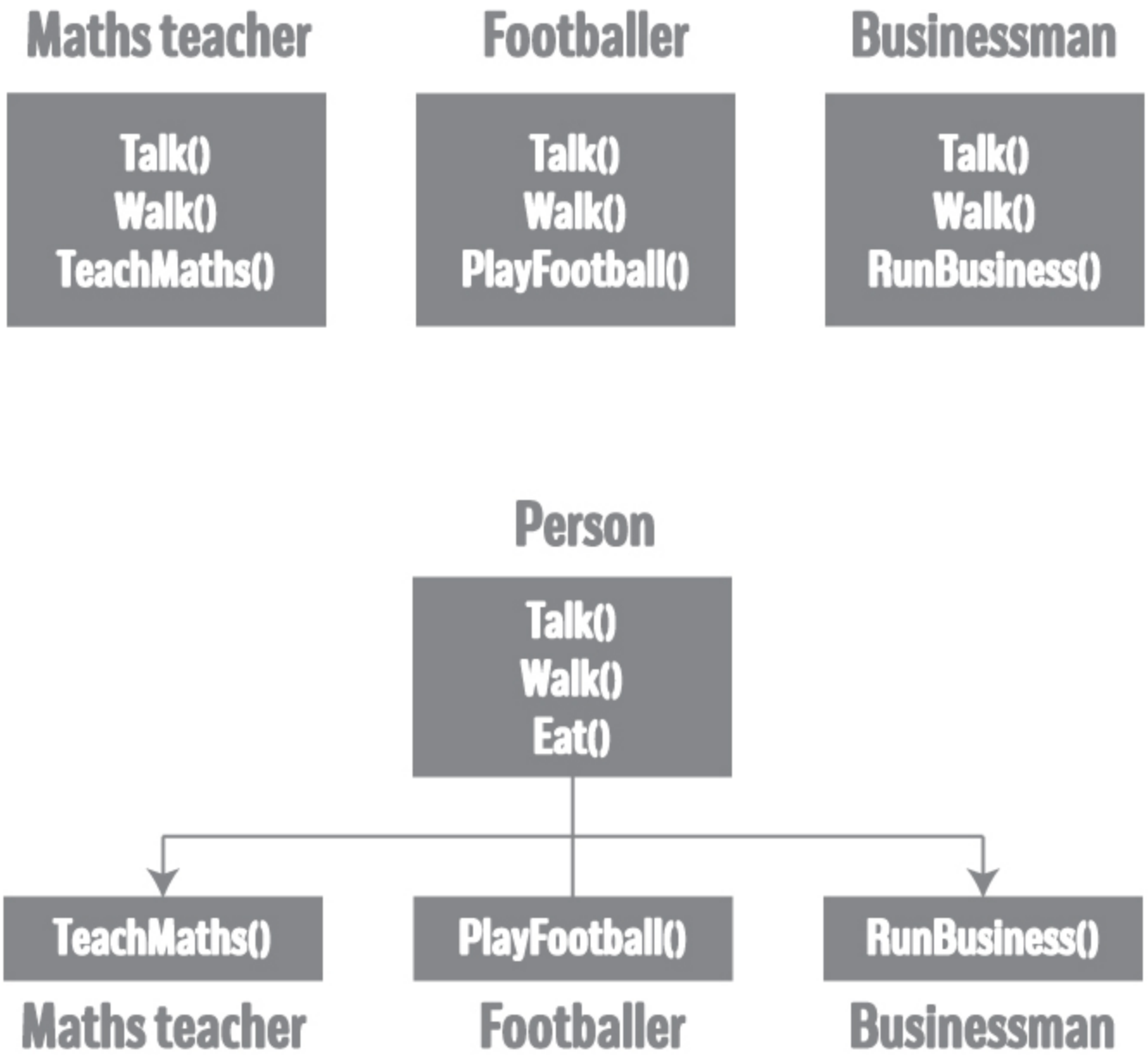


Inheritance
OOP/Computer Programming

By: Dr. Danish Shehzad


Maths teacher
Talk()
Walk()
TeachMaths()
Footballer
Talk()
Walk()
PlayFootball()
Businessman
Talk()
Walk()
RunBusiness()
Person
Talk()
Walk()
Eat()
TeachMaths()
PlayFootball()
RunBusiness()
Maths teacher
Footballer
Businessman

Introduction

- Probably most powerful feature of OOP
- New classes are created from existing classes

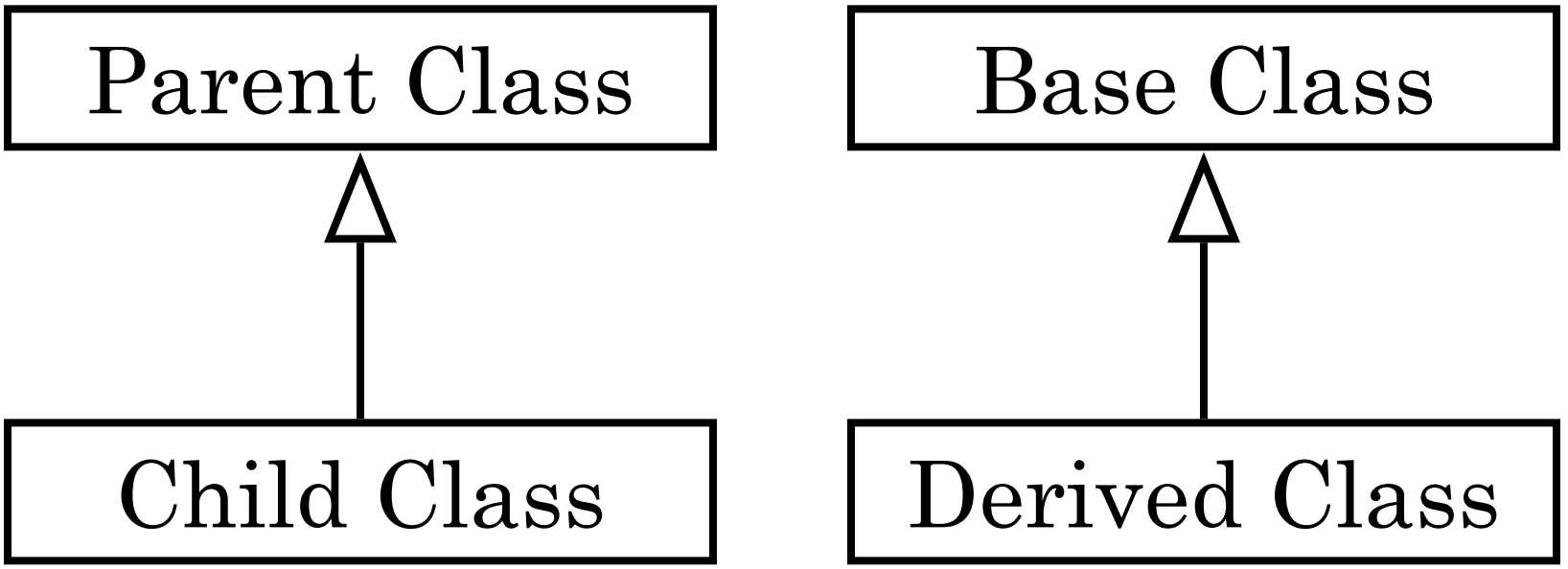
Advantages

- Reusability
- Saves time and effort
- Improves program structure and reliability

Inheritance in Classes

- If a class B inherits from class A, then B contains all the characteristics (information structure and behavior) of class A
- The parent class is called *base* class and the child class is called *derived* class
- Besides inherited characteristics, derived class may have its own unique characteristics

UML Notation



Introduction

- Existing classes are called base classes
- New classes are called derived classes

- Inheritance provides us a mechanism of software reusability which is one of the most important principles of software engineering

Inheriting Data and Functions

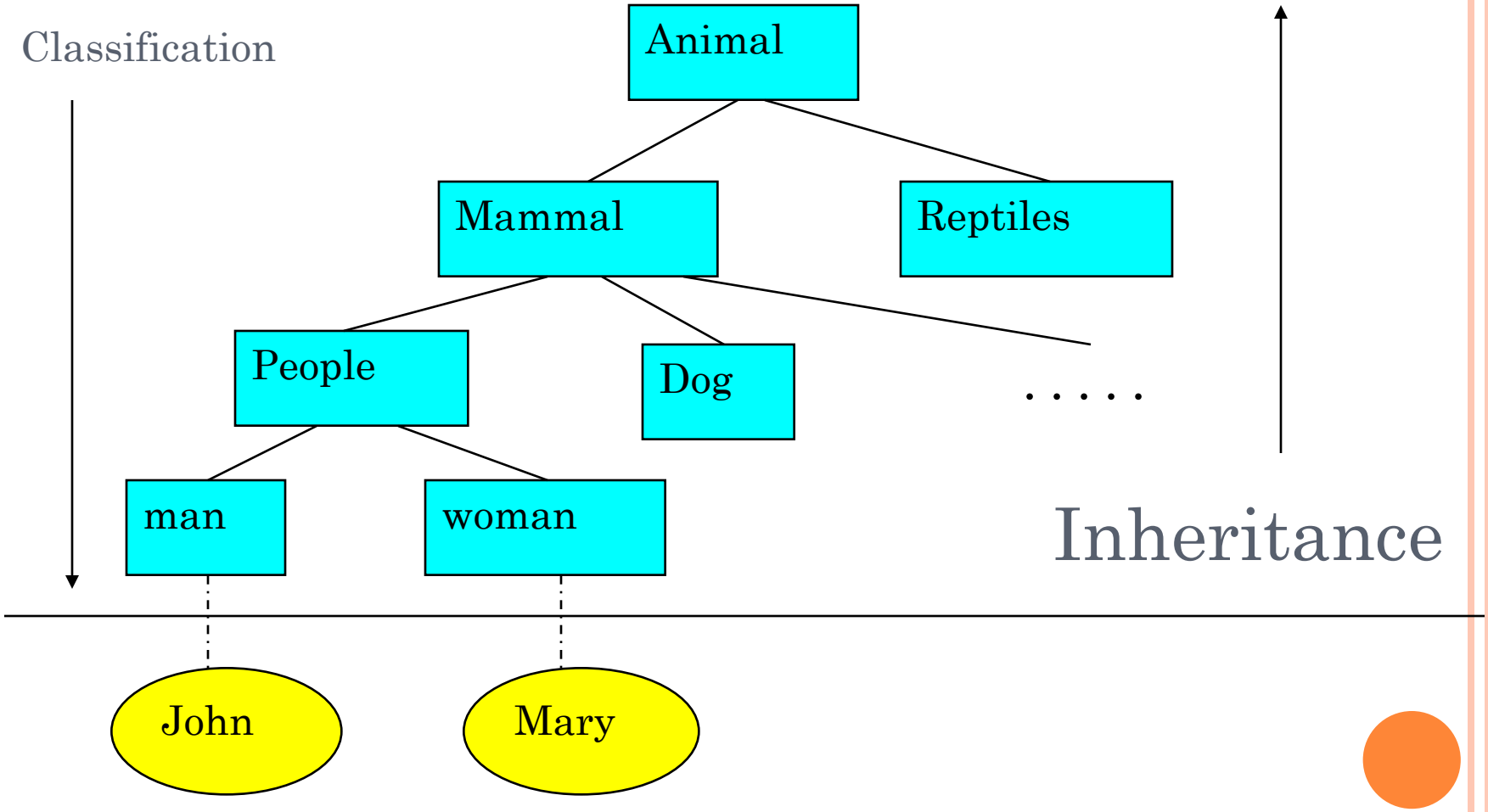
- All data members and member functions of base class are inherited to derived class

- Constructors, destructors and = operator are not inherited

Some definitions in class hierarchy

- Direct base class
  - Inherited explicitly (one level up hierarchy)
- Indirect base class
  - Inherited two or more levels up hierarchy
- Single inheritance
  - Inherits from one base class
- Multiple inheritance
  - Inheritance from multiple classes

Animals: Class's hierarchy



Inheritance Examples

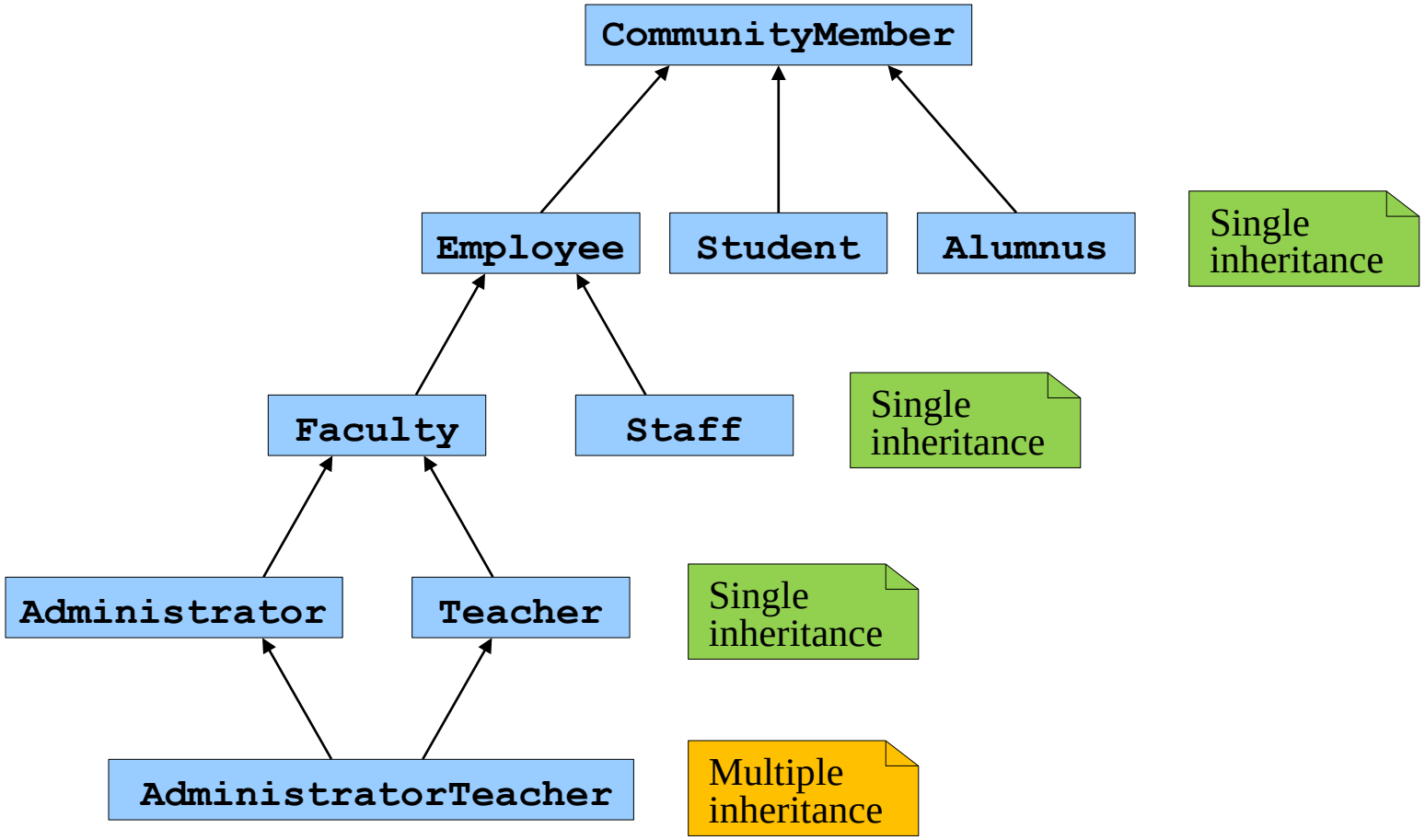
Base class	Derived classes
`Student`	`GraduateStudent` `UndergraduateStudent`

`Shape`	`Circle`
	`Triangle`
	`Rectangle`

`Loan`	`CarLoan`
	`HomeImprovementLoan`
	`MortgageLoan`

`Employee`	`FacultyMember` `StaffMember`
`Account`	`CheckingAccount` `SavingsAccount`

Another example: University's community member's hierarchy



Inheritance in C++

- There are three types of inheritance in C++
  - Public
  - Private
  - Protected

Public Inheritance

- With public inheritance,
  - public and protected members of the base class become respectively public and protected members of the derived class.

Protected Inheritance

- Public and Protected members of the base class become Protected members of the derived class.

Private Inheritance

- With private inheritance, public and protected members of the base class become private members of the derived class.

```
class base
{
        public:
                int x;
        protected:
                int y;
        private:
                int z;
};

class publicDerived: public base
{
        // x is public
        // y is protected
        // z is not accessible from publicDerived
};
```

```
class protectedDerived: protected base
{
        // x is protected
        // y is protected
        // z is not accessible from protectedDerived
};

class privateDerived: private base
{
        // x is private
        // y is private
        // z is not accessible from privateDerived
}
```

Define a Class Hierarchy

- Syntax:

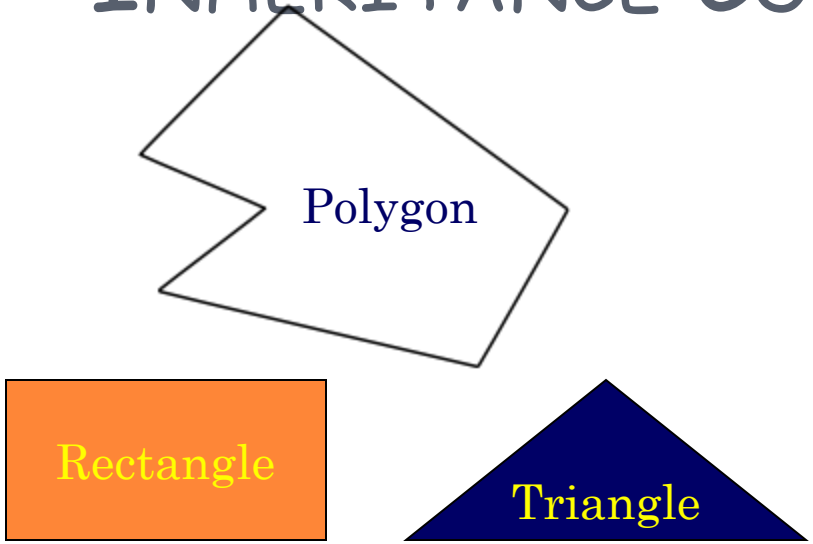
  class *DerivedClassName* : access-level *BaseClassName*

  where

  - access-level specifies the type of derivation
    - private by default, protected or public

- Any class can serve as a base class
  - Thus a derived class can also be a base class

Inheritance Concept

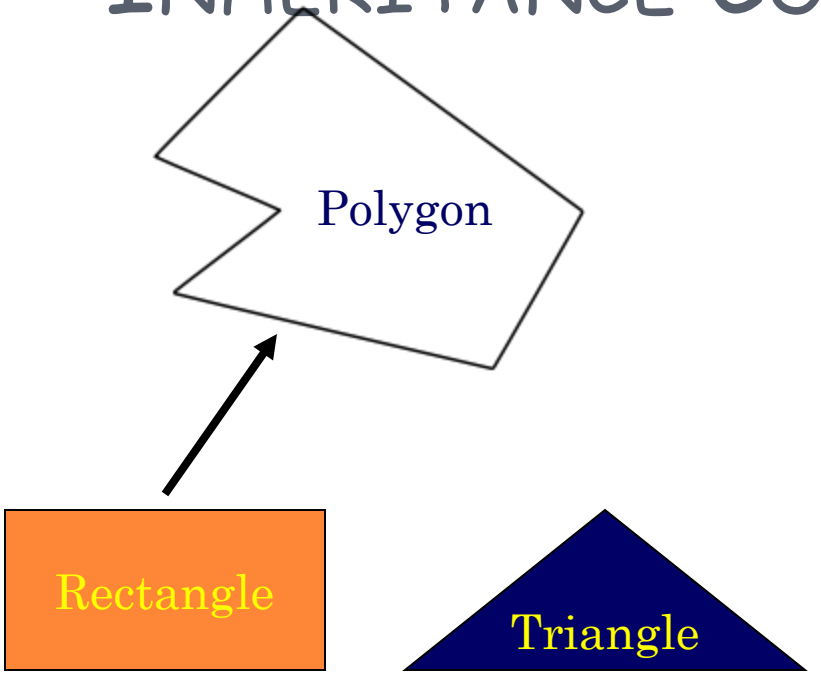



```
class Rectangle{
    private:
      int numVertices;
      float *xCoord, *yCoord;
    public:
      void set(float *x, float *y, int
nV);
      float area();
};
```

```
class Polygon{
   private:
     int numVertices;
     float *xCoord, *yCoord;
   public:
     void set(float *x, float *y, int
nV);
};
```

```
class Triangle{
   private:
     int numVertices;
     float *xCoord, *yCoord;
   public:
     void set(float *x, float *y, int
nV);
     float area();
};
```

Inheritance Concept

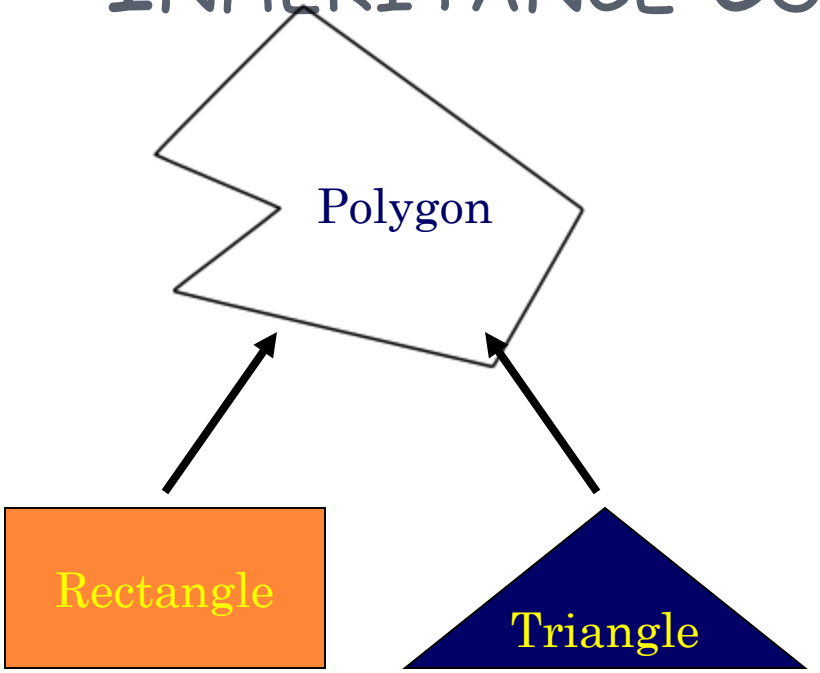



```
class Polygon{
    protected:
      int numVertices;
      float *xCoord, float *yCoord;
    public:
       void set(float *x, float *y, int
    nV);
};
```

```
class Rectangle : public
    Polygon{
    public:
       float area();
};
```

```
class Rectangle{
    protected:
      int numVertices;
      float *xCoord, float *yCoord;
    public:
       void set(float *x, float *y, int
    nV);
      public: float area();
};
```

INHERITANCE CONCEPT

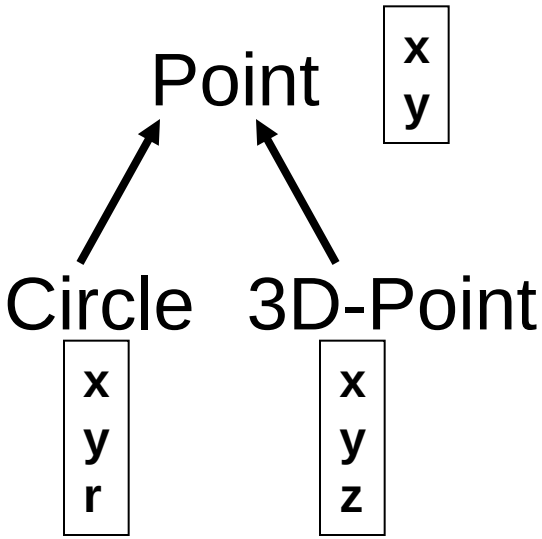



```
class Polygon{
    protected:
      int numVertices;
      float *xCoord, float *yCoord;
    public:
       void set(float *x, float *y, int
    nV);
};
```

```
class Triangle : public
    Polygon{
    public:
     float area();
};
```

```
class Triangle{
    protected:
      int numVertices;
      float *xCoord, float *yCoord;
    public:
       void set(float *x, float *y, int
    nV);
      public:  float area();
};
```

Inheritance Concept

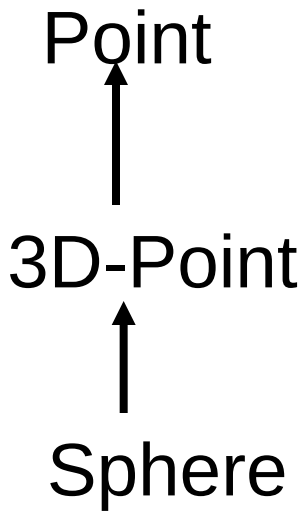



```
class Point{
    protected:
       int x, y;
    public:
       void set (int a, int b);
};
```

```
class Circle : public Point{
    private:
          double r;
};
```

```
class 3D-Point: public
    Point{
    private:
          int z;
};
```

Class Derivation




```
class Point{
    protected:
      int x, y;
    public:
      void set (int a, int b);
};
```

```
class 3D-Point : public
    Point{
    private:
        double z;
    … …
};
```

```
class Sphere : public 3D-
    Point{
    private:
      double r;
    … …
};
```

Point is the base class of 3D-Point, while 3D-Point is the base class of Sphe

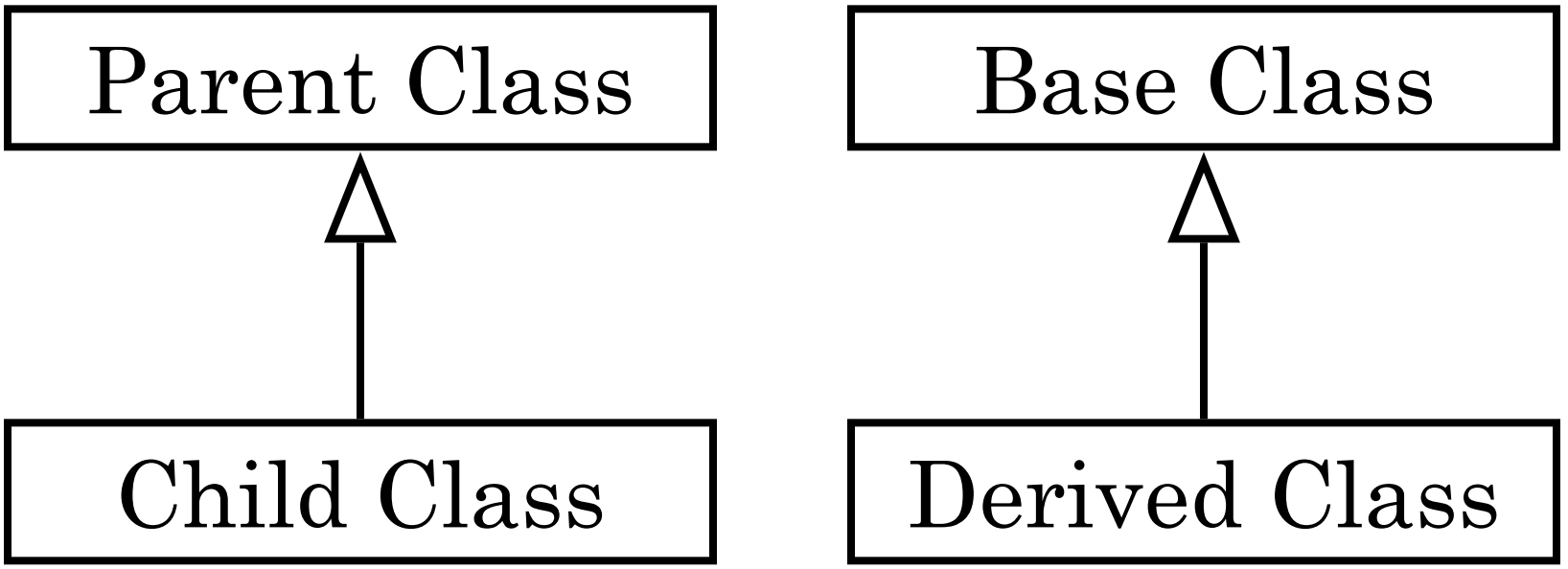
What to inherit?

- In principle, every member (but not private) of a base class is inherited by a derived class
  - just with different access permission

Task

1. Write a class employee that contains attributes of employee id and scale. The class contains member functions to input and output these attributes. Write a child class Manager that inherits Employee class. The child class has attributes of manager name and his department and member function to show all attributes(employee id, scale, name, department)

UML Notation



"IS A" Relationship

- IS A relationship is modeled with the help of public inheritance

- Syntax

```
class ChildClass
        : public BaseClass{
...
};
```

Accessing Members

- Public members of base class become public member of derived class
- Private members of base class are not accessible from outside of base class, even in the derived class (Information Hiding)

Example

```
class Person{
  char *name;
  int age;
  ...
public:
  const char *GetName() const;
  int GetAge() const;
  ...
};
```

Example

```
class Student: public Person{
  int semester;
  int rollNo;
  ...
public:
  int GetSemester() const;
  int GetRollNo() const;
  void Print() const;
  ...
};
```

Example

```
void Student::Print()
{
  cout << name << " is in" << "
  semester " << semester;
}
```

ERROR

Example

```
void Student::Print()
{
  cout << GetName()
          << " is in semester "
     << semester;
}
```

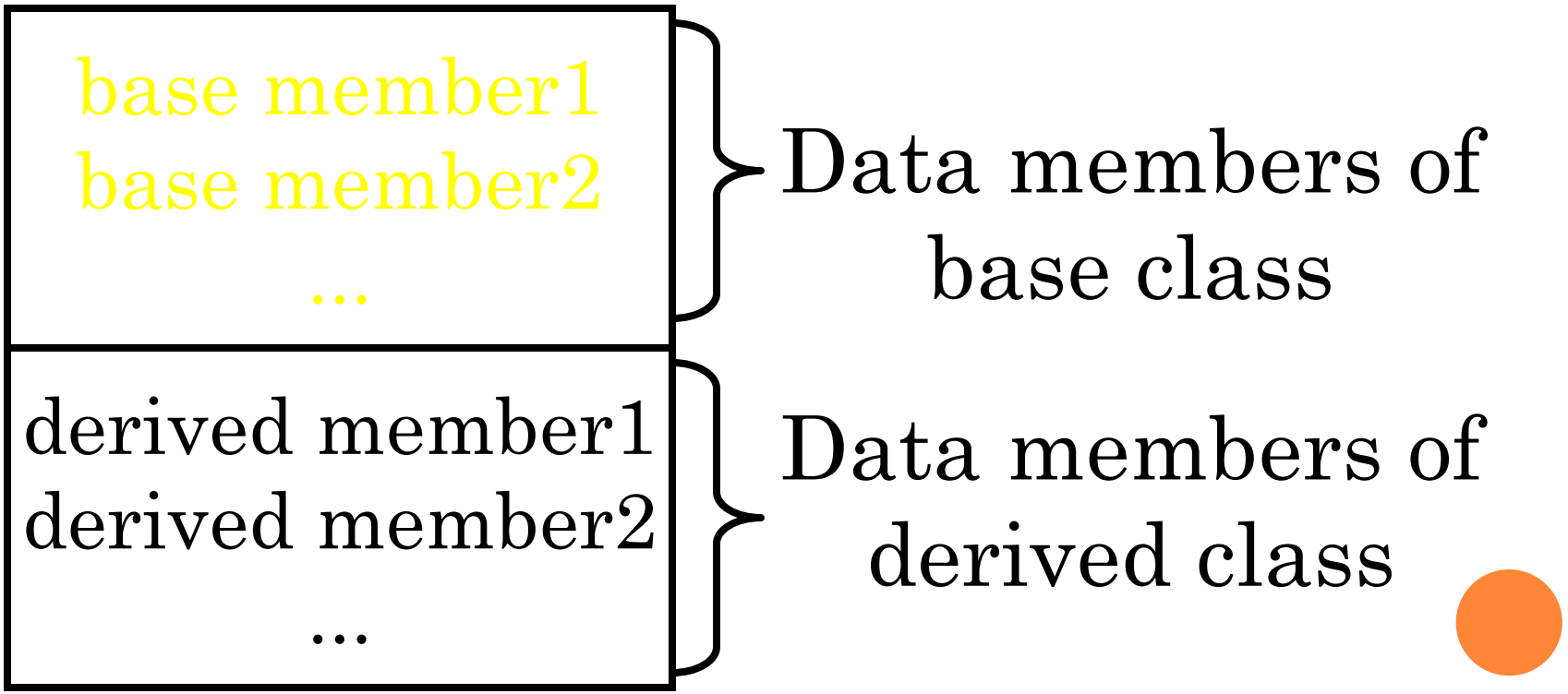
Example

```
int main(){
  Student stdt;

  stdt.semester = 0;//error
  stdt.name = NULL; //error
  cout << stdt.GetSemester();
  cout << stdt.GetName();
  return 0;
}
```

Allocation in Memory

- The object of derived class is represented in memory as follows



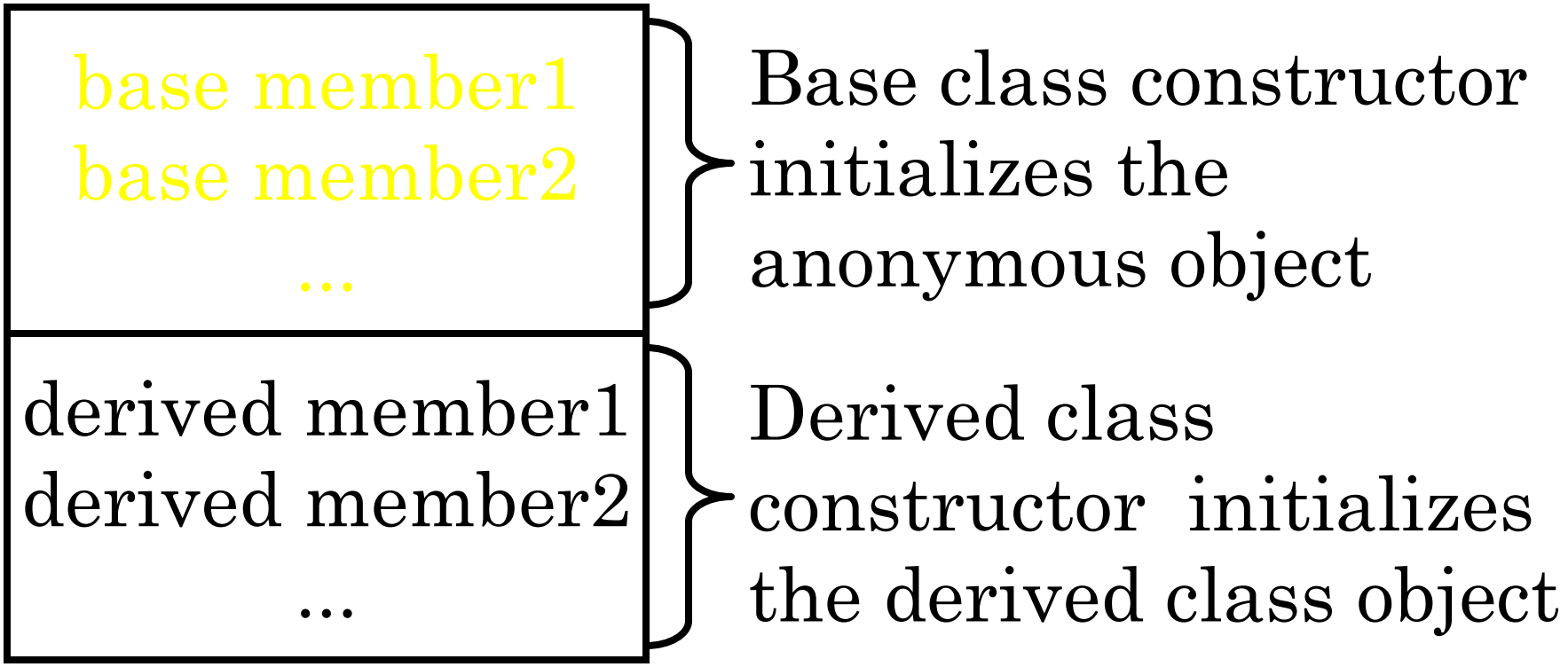
Allocation in Memory

- Every object of derived class has an anonymous object of base class

Constructors

- The anonymous object of base class must be initialized using constructor of base class
- When a derived class object is created the constructor of base class is executed before the constructor of derived class

Constructors



Example

```
class Parent{
public:
  Parent(){ cout <<
      "Parent Constructor...";}
};
class Child : public Parent{
public:
  Child(){    cout <<
      "Child Constructor...";}
};
```

Example

```
int main(){
  Child cobj;
  return 0;
}
```

Output:
Parent Constructor...
Child Constructor...

Constructor

- If default constructor of base class does not exist then the compiler will try to generate a default constructor for base class and execute it before executing constructor of derived class

Constructor

- If the user has given only an overloaded constructor for base class, the compiler will not generate default constructor for base class

Base Class Initializer

- C++ has provided a mechanism to explicitly call a constructor of base class from derived class

- The syntax is similar to member initializer and is referred as base-class initialization

Example

```
class Parent{
public:
  Parent(int i){…};
};
class Child : public Parent{
public:
  Child(int i): Parent(i)
  {…}
};
```

Example

```
class Parent{
public:
  Parent(){cout <<
       "Parent Constructor...";}
  ...
};
class Child : public Parent{
public:
  Child():Parent()
  {cout << "Child Constructor...";}
  ...
};
```

Base Class Initializer

- User can provide base class initializer and member initializer simultaneously

Example

```
class Parent{
public:
  Parent(){…}
};
class Child : public Parent{
  int member;
public:
  Child():member(0), Parent()
  {…}
};
```

Base Class Initializer

- The base class initializer can be written after member initializer for derived class
- The base class constructor is executed before the initialization of data members of derived class.

Example

```
class Parent{
public:
  Parent(){cout <<"Parent Constructor";}
  ~Parent(){cout<<"Parent Destructor";}
};

class Child : public Parent{
public:
  Child(){cout << "Child Constructor";}
  ~Child(){cout << "Child Destructo";}
};
```