C++ Programming

Trainer : Akshita Chanchlani

Email: akshita.chanchlani@sunbeaminfo.com

Scope

- It decides area/region/boundry in which we can access the element.
- **Types of scope in C++:**
  1. **Block scope**
  2. **Function scope**
  3. **Prototype scope**
  4. **Class scope**
  5. **Namespace scope**
  6. **File scope**
  7. **Program scope**

Example Scope

```
int num6;           //Program Scope
static int num5;  //File Scope
namespace ntest
{  int num4;   //Namespace scope
   class Test
   {   int num3;   //Class Scope  };
}
void sum( int num1, int num2 ); //Prototype scope
int main( void )
{
   int num1 = 10; //Function Scope
   while( true )
   {  int temp = 0; }
   return 0;   //Block Scope
}
```

C++ friend Function and friend Classes

- It is a mechanism built in C++ programming to access private or protected data from non-member functions.
- This is done using a friend function or/and a friend class.
- If a function is defined as a friend function then, the private and protected data of a class can be accessed using the function.
- The complier knows a given function is a friend function by the use of the keyword **friend**.
- For accessing the data, the declaration of a friend function should be made inside the body of the class (can be anywhere inside class either in private or public section) starting with keyword friend.
- **Declaration of friend function in C++**
  - Syntax : class class_name { ... . friend return_type function_name(argument/s); ... . }
  - Now, you can define the friend function as a normal function to access the data of the class.
    No friend keyword is used in the definition.

    class className { ... . friend return_type functionName(argument/s); ... . }

    return_type functionName(argument/s) { ... . // Private and protected data of className can be accessed from // this function because it is a friend function of className. ... . }

Example Friend function

- If we want to access private members inside derived class
  - Either we should use member function(getter/setter).
  - Or we should declare a facilitator function as a friend function.
  - Or we should declare derived class as a friend inside base class.

We can declare global function (including main function) as well as member function as a friend inside class. We can write friend declaration statement inside any section(private/ protected/public) of the class. Consider this code snippet:	**class Test**
	{
	private: int number;
	public:
	Test(void)
	{ **this->number = 10** }
	friend void print(void);
	};
	void print(void)
	{ Test t; cout<<"Number : "<<t.number<<endl; }
	int main(void)
	{ print();
	return 0;
	}

friend Class in C++ Programming

- like a friend function, a class can also be made a friend of another class using keyword friend.

```
class B;
class A
{
        // class B is a friend class of class A
        friend class B;
         ... .. ...
}
class B
 {
... .. ...
}
```

In this example , all member functions of class B will be friend functions of class A. Thus, any member function of class B can access the private and protected data of class A.
But, member functions of class A cannot access the data of class B.

- When a class is made a friend class, all the member functions of that class becomes friend functions.

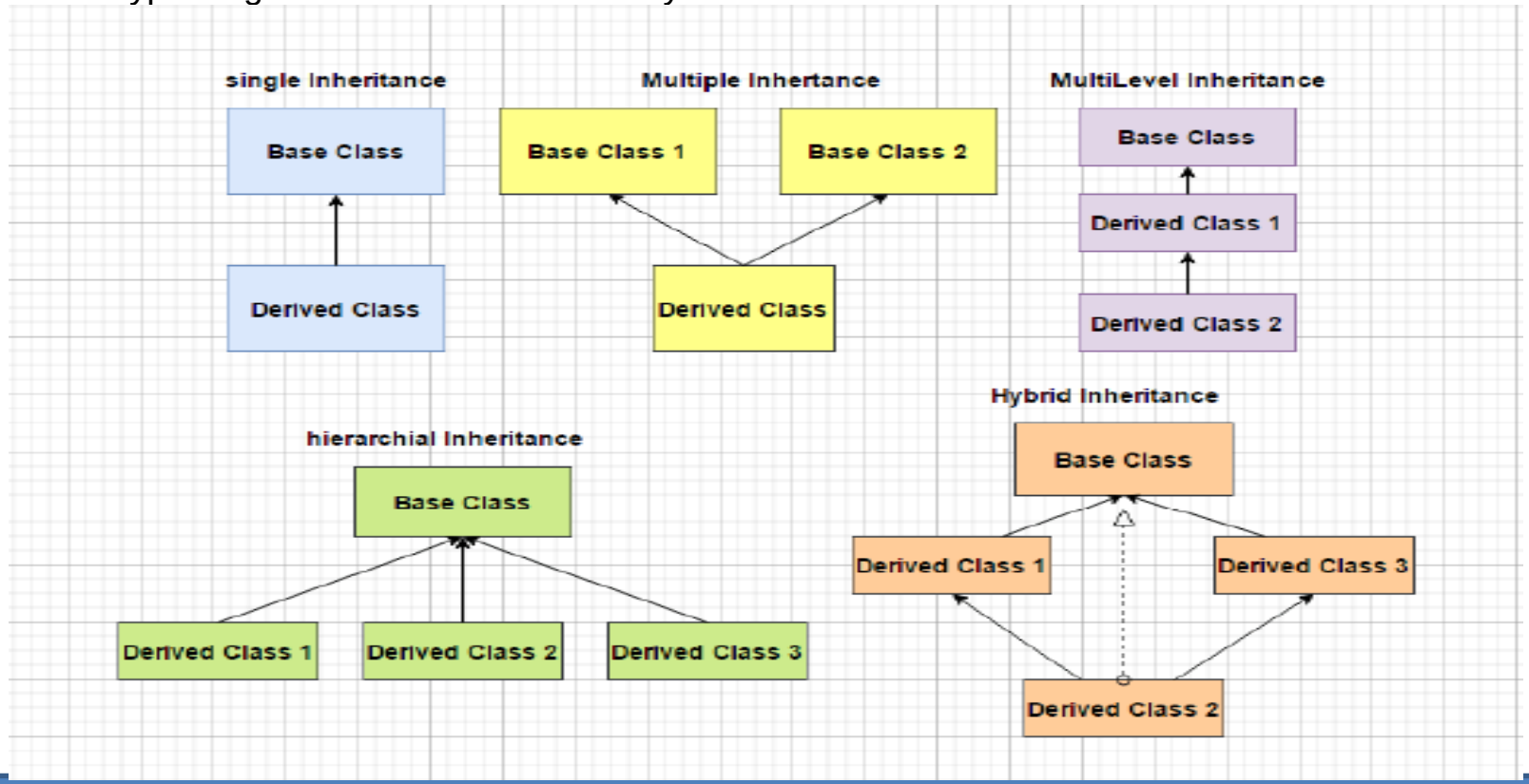
C++ Inheritance

- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
- It provides an opportunity to reuse the code functionality and fast implementation time.
- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.
- Syntax:
  - class derived-class: access-specifier base-class
- If "is-a" relationship exist between two types then we should use inheritance.
- Inheritance is also called as "Generalization".
- Example: Book is-a product
- During inheritance, members of base class inherit into derived class.

Types of Inheritance

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance

If we combine any two or more types together then it is called as hybrid inheritance.



Inheritance

- If we create object of derived class then non static data members declared in base class get space inside it.
- If we use private/protected/public keyword to control visibility of members of class then it is called access Specifier.
- If we use private/protected/public keyword to extend the class then it is called **mode of inheritance**.
- Default mode of inheritance is private.
  - Example: class Employee : person //is treated as class Employee : private Person
- Example: class Employee:public Person
- In all types of mode, private members inherit into derived class but we can not access it inside member function of derived class.
- If we want to access private members inside derived class then:
  - Either we should use member function(getter/setter).
  - or we should declare derived class as a friend inside base class.

Association

- If has-a relationship exist between two types then we should use association.
- Example : Car has-a engine (OR engine is part-of car)
- If object is part-of / component of another object then it is called association.
- If we declare object of a class as a data member inside another class then it represents association.
- Example Association:

```
class Engine
{ };
class Car
{           private:
   Engine e;   //Association
};
int main( void )
{  Car car;
   return 0;
}
```

Composition and aggregation are specialized form of association

Composition

- If dependency object do not exist without Dependant object then it represents composition.
- Composition represents tight coupling.
- Example: Human has-a heart.

```
class Heart
{ };
class Human
{  Heart hrt; //Association->Composition
};
int main( void )
{  Human h;
   return 0;
```

Aggregation

- If dependency object exist without Dependant object then it represents Aggregation.
- Aggregation represents loose coupling.

```
class Faculty
{ };
class Department
{
   Faculty f; //Association->Aggregation
};
int main( void )
{
   Department d;
```

Thank You