



# Programming Language

Object Oriented Programming

*(Day 4 Part 1)*

**By: Bhaskar Ghosh**

*[bjghosh@gmail.com](mailto:bjghosh@gmail.com)*

Updated: 6 August 2024

# C++ Programming Language

## Object Oriented Programming (Day 4 Part 1)

- Introduction to OOPs
- C++ Class
- Classes and Objects
- Constructors and Destructors
  - Default Constructor, Parameterized Constructor, Default Copy Constructor, and Destructor
- Classes, Objects, and Memory
- Abstraction in C++ (Using Access Modifiers)
- this pointer
- static members in a class
- Friend functions and friend classes

# C++ Programming Language

## Introduction to OOPs (Day 3 Part 2)

- What are the main features of Object Oriented Programming?
- **Pillars of OOPS**
  - ☐ Abstraction
  - ☐ Encapsulation
  - ☐ Inheritance
  - ☐ Polymorphism
    - Compile-time polymorphism
    - Run-time polymorphism

# C++ Programming Language

## Intro to OOPs

- Abstraction
  - Data Abstraction is the property by virtue of which only the essential details are displayed to the user.
  - E.g., a man is driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know internal workings or details.
- Encapsulation
  - Wrapping up of data and operations under a single unit. It is the mechanism that binds together code and the data it manipulates

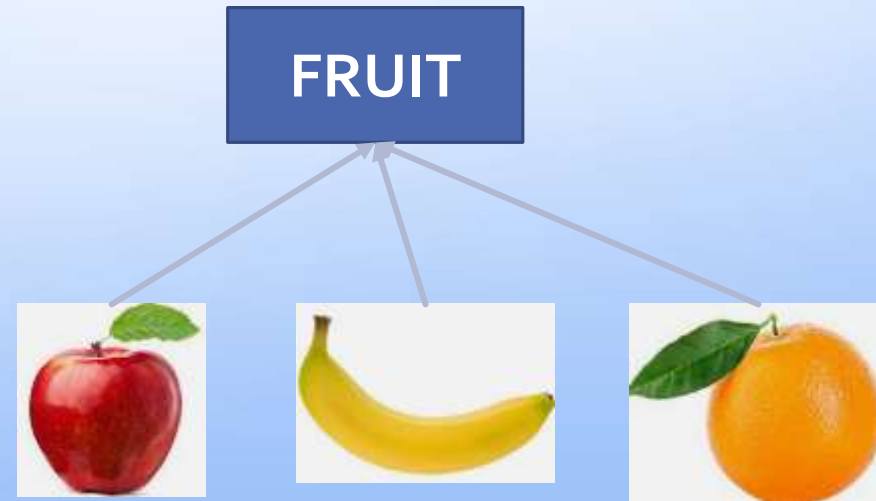
# C++ Programming Language

## Class

- A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity.
- Before we create an object, we first need to define the class.

Syntax to declare a class:

```
class <ClassName> {  
    field-declaration  
    method-definition  
}
```



# C++ Programming Language

## Object

- An object is a real-world entity.
  - An object is a runtime entity.
  - The object is an entity which has state and behavior.
  - The object is an instance of a class.
- 
- ☐ **State:** represents the data (value) of an object.
  - ☐ **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.

# C++ Programming Language

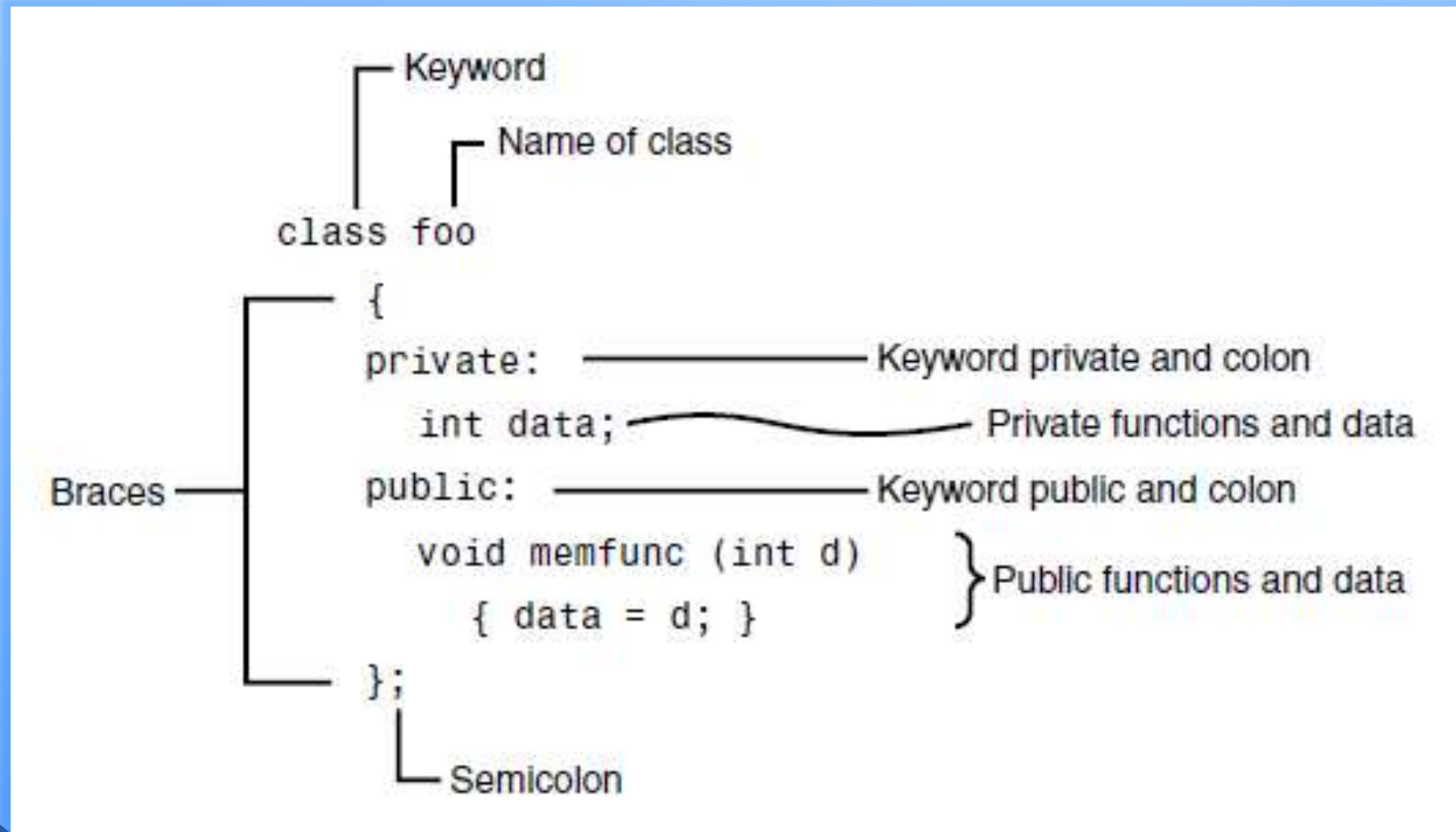
## Encapsulation

- Encapsulation in C++ is a mechanism of wrapping the data (data members) and code acting on the data (member functions) together as a single unit.
- In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the member functions of their current class.
- It is also known as **data hiding**. Implemented using access specifiers **private**, **public**, **protected**.

```
class Person{  
    private:  
        string name;  
        int age;  
  
    public:  
        void setName(string n)  
        { name = n; }  
  
        string getName()  
        { return name; }  
}
```

# C++ Programming Language

## Encapsulation and Data Hiding





# C++ Programming Language

## Creating and Using Objects

```
class Person{  
    private:  
        string name;  
        int age;  
  
    public:  
        void setName(string n)  
        { name = n; }  
  
        string getName()  
        { return name; }  
  
        void setAge(int a)  
        { age = a; }  
  
        int getAge()  
        { return age; }  
  
};
```

```
int main() {  
    Person p;  
    p.setName("James");  
    p.setAge(20);  
    cout <<  
        "Name : " << p.getName() <<  
        " Age : " << p.getAge();  
}
```

- Usually, Functions are public, Data is private.
- But, in some circumstances, we may need to use private functions and public data.

# C++ Programming Language

## Constructors

- It is a special type of member function which is used to initialize the object.
- A constructor is a member function that is executed automatically whenever an object is created.

- Rules:**
- Constructor name must be the same as its class name.
  - A Constructor must have no return type, not even void.

- Note:**
- If there is no constructor in a class, compiler automatically creates a default constructor.
  - One of the most common tasks a constructor carries out is initializing data members.

# C++ Programming Language

## Types of Constructors

- Default Constructor  
(No-arg Constructor)
  - A constructor without any parameter
- Parameterized Constructor
  - A constructor with a specific number of parameters.
- Copy Constructor

```
class Person{    ...
public:
    //creating a default constructor
    Person() : name(""), age(0) {}

    //creating a parameterized constructor
    Person(string n, int a) :
        name(n), age(a) {}

    //creating a custom copy constructor
    Person( const Person& ) {...}
    ...
}

int main(){
    Person p1;
    Person p2 ("James", 20);
    Person p3 = p1;
}
```

# C++ Programming Language

## Destructors

- It is a special type of member function which is called automatically whenever an object is destroyed.

### Rules:

- Destructor name must be the same as its class name, but is preceded by a tilde (~).
- Like constructors, destructors do not have a return value.
- They also take no arguments.

### Note:

- If there is no destructor in a class, compiler automatically creates the destructor.
- The most common use of destructors is to deallocate memory that was allocated for the object by the constructor.

# C++ Programming Language

## Member Functions defined Outside the Class

- Member functions can also be defined outside the class.
- So, we can declare them inside the class, but define them outside, even in another file.
- We can use the ***scope resolution operator (::)*** to specify the class they belong to.

```
class Person{
    ...
public:
    Person();
    ~Person();
    void setName(string);
    ...
}

Person::Person() : name(""), age(0) {}

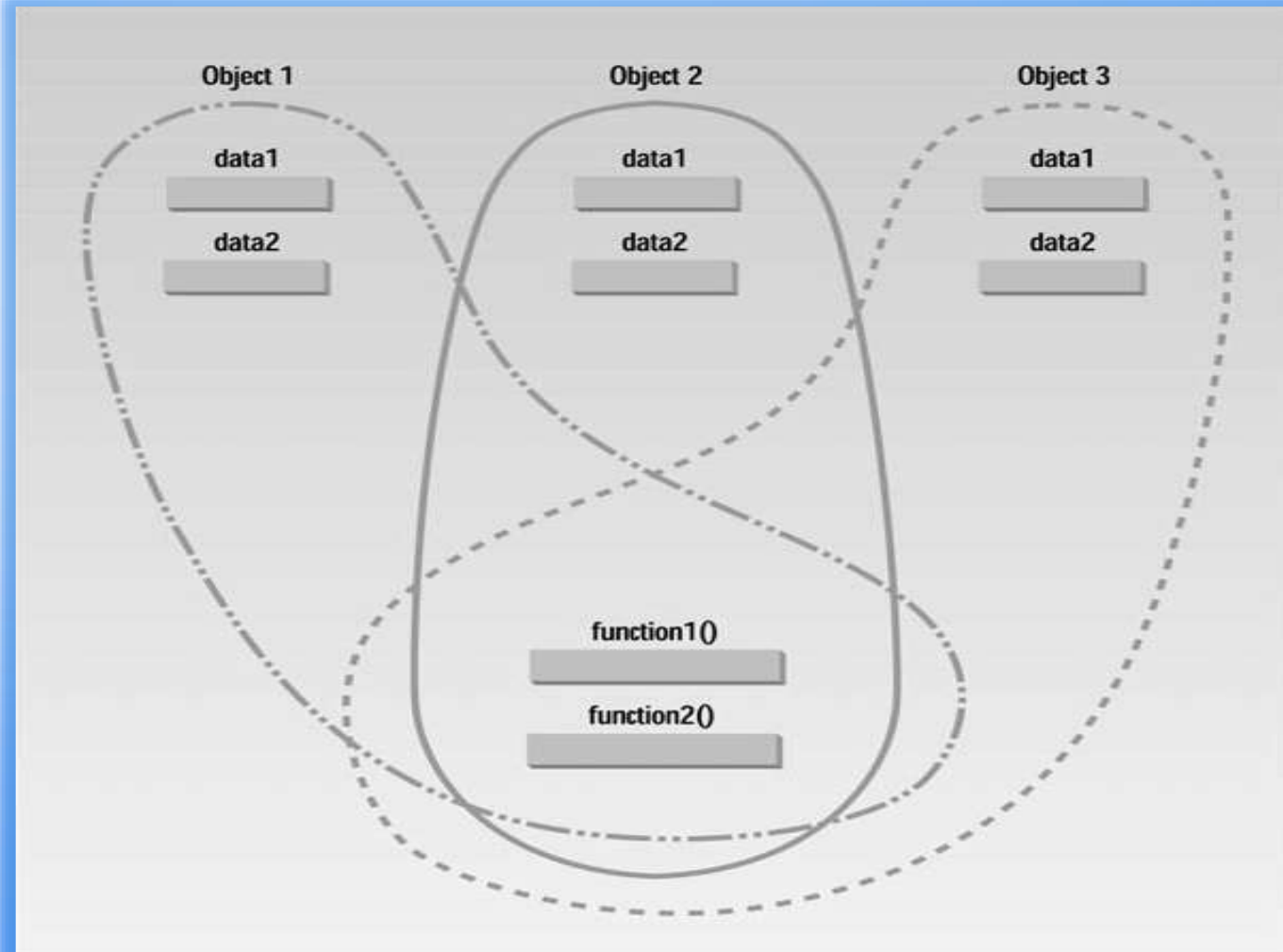
Person::~~Person() {}

void Person::setName(string n) { name = n; }
```

# C++ Programming Language

## Classes, Objects, and Memory

- Each object has its own separate data items.
- But, the member functions are created and placed in memory only once—when they are defined in the class definition.
- So, all the objects of a given class use the same member functions.



# C++ Programming Language

## Abstraction in C++ (Using Access Modifiers)

- There are 3 access modifiers in C++

**public**

- Used to create public members (data and functions)
- Accessible from any part of the program

**private**

- Used to create private members (data and functions)
- Can only be accessed from within the class
- Exception: friend classes and friend functions can access private members

**protected**

- Like private, but can be accessed from the derived class also

# C++ Programming Language

## this pointer

- "this" is a keyword representing the current class instance.
- The "this" pointer is automatically passed as a hidden argument in non-static member function calls.
- "this" is a **const** pointer.
  - We can change the value of the object it points to, but we cannot make it point to any other object.
- We can use arrow (->) operator to access other class members (data and functions) using "this" pointer from within non-static member functions.



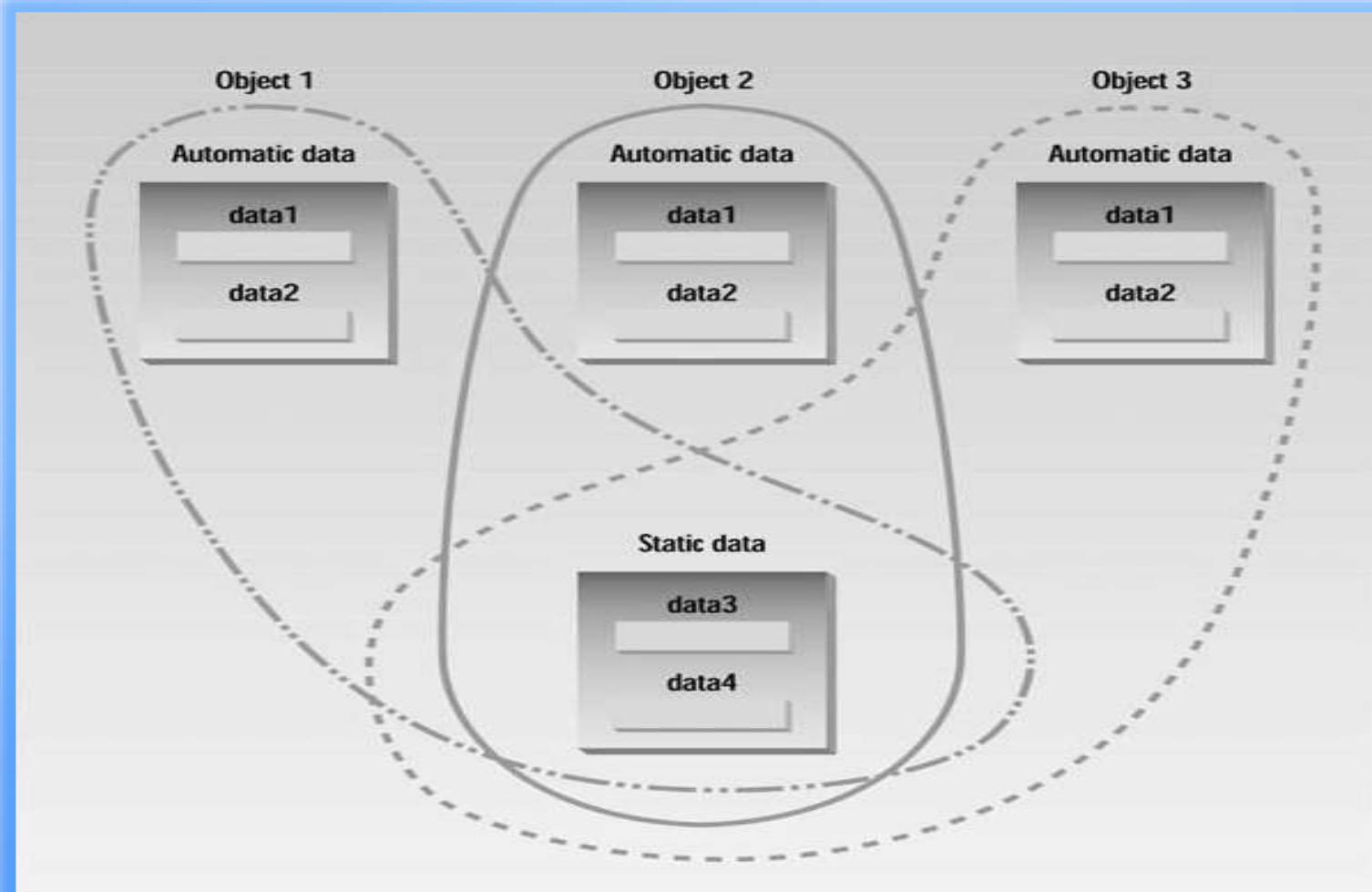
# C++ Programming Language

## static members in a class

- If a data item in a class is declared as static, only one such item is created for the entire class, no matter how many objects there are.
- A static data item is useful when all objects of the same class must share a common item of information.
- A member variable defined as static has characteristics similar to a normal static variable:
  - Visible only within the class, but its lifetime is the entire program.
  - Initialized before any object of the class is created.

# C++ Programming Language

static members in a class



# C++ Programming Language

## Friend function

- In C++, global functions cannot access the private members of a class.
- But there is an exception:
  - A friend function in C++ is a non-member function that is granted access to the private and protected members of a class.
- We don't have to define the function inside the class.
  - Just declaring it as **friend** inside the class is enough.
  - The definition of a friend function will be outside the class, as a global function or as a member function of another class.
- A friend function in C++ cannot directly access the protected or private data members of the class.
  - It can access using an object of the class.

# C++ Programming Language

## Friend class

- Just like a friend function, a particular class can also have a friend class.
- A friend class shares the same privilege, i.e., it can access the private and protected members of the class whose friend it has been declared.
  - All the functions declared inside the friend class will also be able to access the private and protected members of the class.

```
class ClassB;  
  
class ClassA {  
    // ClassB is a friend class of ClassA  
    friend class ClassB;  
    ... ..  
}  
  
class ClassB {  
    ... ..  
}
```



# Programming Language

Inheritance (*Day 4 Part 2*)

**By: Bhaskar Ghosh**

*[bjghosh@gmail.com](mailto:bjghosh@gmail.com)*

Updated: 6 August 2024

# C++ Programming Language

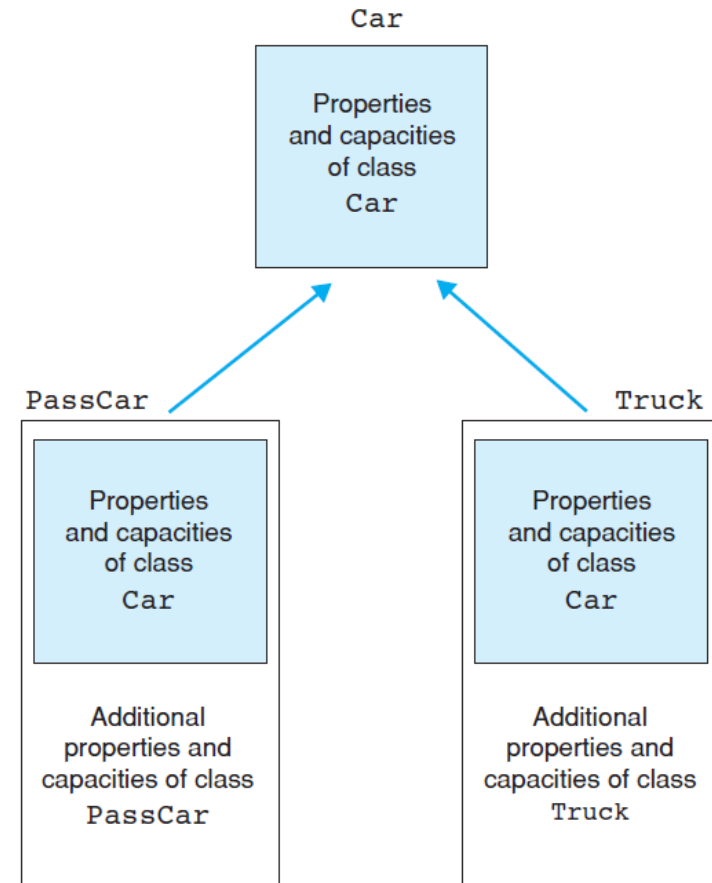
## OOPs – Inheritance (Day 4 Part 2)

- Inheritance in C++
- Is-A and Has-A Relationships
- Generalization in UML Class Diagrams
- Constructing and Destroying Derived Class Objects
- Type Conversions in Class Hierarchies
- Types of Inheritance in C++
  - Single, Multiple, Multi-level, Hierarchical
- Modes of Inheritance
  - Public, Protected, Private
- Function Overriding

# C++ Programming Language

## Inheritance

- Inheritance is the process of creating new classes, called derived classes, from existing or base classes.
- The derived class inherits all the capabilities of the base class but can add data and functions of its own.
- The base class does not change.



# C++ Programming Language

## Inheritance – the Is-A Relationship

- Inheritance is an essential part of OOP, and permits code reusability.
- A Derived Class has “**is-a**” relationship with the Base Class.
- Example: Truck **is-a** Car
- This is different from has-a relationship, also known as containership.
  - A “**has-a**” relationship occurs between two classes when a member of one class has another class type.
  - Example: Person **has-a** PhoneNumber



# C++ Programming Language

## Inheritance

- Syntax:

```
class derived_class_name : access-specifier
base_class_name
{
    // body ....
};
```

- Example:

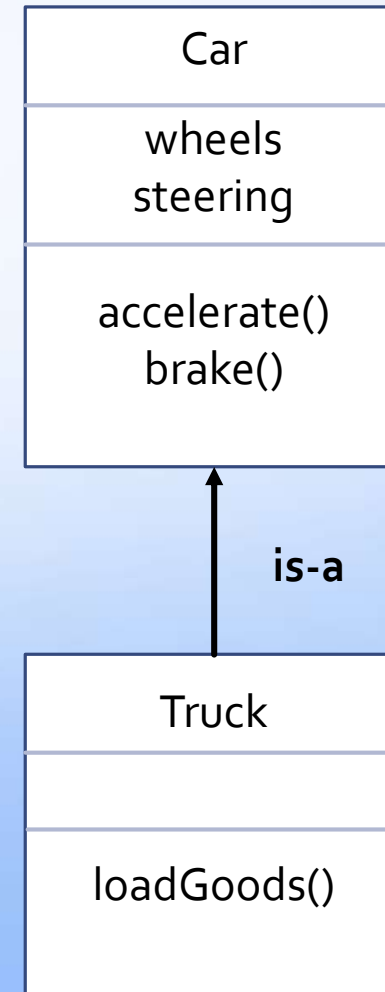
```
class Car {
    // accelerate() function
    // brake() function
};

class Truck : public Car {
    // loadGoods() function
};
```

# C++ Programming Language

## Generalization in UML Class Diagrams

- In the UML, inheritance is called generalization, because the parent class is a more general form of the child class.
- In UML class diagrams, generalization is indicated by a triangular arrowhead.
- The direction of the arrow emphasizes that the derived class refers to functions and data in the base class.



# C++ Programming Language

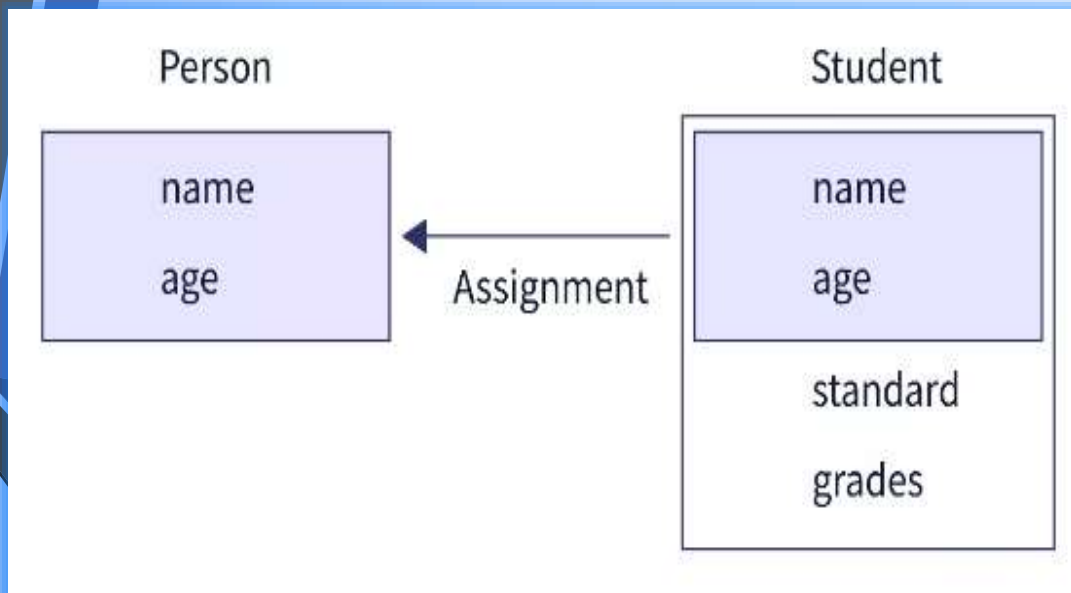
## Constructing and Destroying Derived Class Objects

- The constructor of a derived class is required to create an object of the derived class type.
- As the derived class contains all the members of the base class, the base sub-object must be created and initialized first.
- So, the ***base class constructor is called first, then the derived class constructor.***
- When an object is destroyed, the ***destructor of the derived class is first called, followed by the destructor of the base class.***

# C++ Programming Language

## Type Conversions in Class Hierarchies

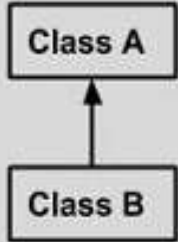
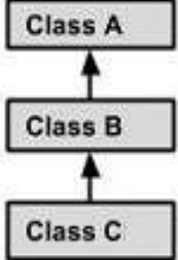
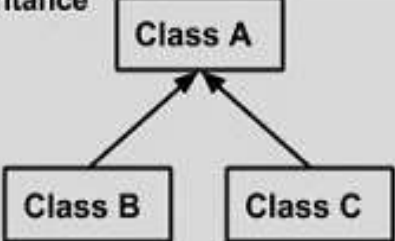
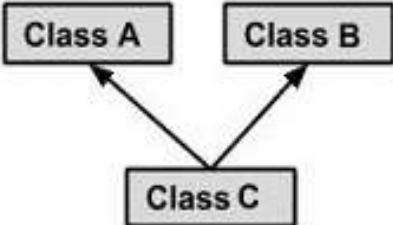
- If we assign any object which belongs to the derived class, to the base class object, the data members which are defined in the base class will be copied.
- The information which is extra in the derived class will be ignored during typecasting because the base class object is not capable of handling it.



```
int main() {  
    Student s1("XYZ", 19, "UnderGraduate");  
    Person p1;  
  
    //This p1 object can only access  
    // the public members of Person Class  
    p1 = s1;  
  
    // will call display method of Person  
    p1.display();  
}
```

# C++ Programming Language

## Types of Inheritance

<b>Single Inheritance</b>  <pre>graph BT; B[Class B] --&gt; A[Class A]</pre>	<code>class B : public A {...}</code>
<b>Multi Level Inheritance</b>  <pre>graph BT; C[Class C] --&gt; B[Class B]; B --&gt; A[Class A]</pre>	<code>class B : public A {...}</code> <code>class C : public B {...}</code>
<b>Hierarchical Inheritance</b>  <pre>graph BT; B[Class B] --&gt; A[Class A]; C[Class C] --&gt; A</pre>	<code>class B : public A {...}</code> <code>class C : public A {...}</code>
<b>Multiple Inheritance</b>  <pre>graph BT; C[Class C] --&gt; A[Class A]; C --&gt; B[Class B]</pre>	<code>class C : public A, public B {...}</code>

# C++ Programming Language

## Modes of Inheritance

- A child class can inherit a parent class in either public, protected or private mode.

Base class member access specifier	Mode of Inheritance		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible(Hidden)	Not accessible(Hidden)	Not accessible(Hidden)

# C++ Programming Language

## Function Overriding

- There are two options for the names of data members or methods in derived classes:
  1. The name does not occur in the base class → no redefinition.
  2. The name already exists in the base class → redefinition.
- In the second case, the member of the same name continues to exist unchanged in the base class.
- Name lookup rules for the compiler:
  - If a member is redefined in a derived class, then the compiler will call the derived class member, i.e. base class member with same signature will be hidden or overridden.
  - This situation is similar to the one seen for local and global variables.