

OBJECT ORIENTED PROGRAMMING USING C++ (OOPCF-015)

Prerequisite:

1. Problem-Solving Skills

- Ability to think logically and break down problems into smaller, manageable parts for coding solutions.

2. Familiarity with Basic Programming Concepts

- **Variables and Data Types:** Understanding how to declare variables and use different data types (e.g., integers, floats, characters).
- **Control Structures:** Basic knowledge of loops (for, while), conditional statements (if, else), and switch-case.
- **Arrays, Pointers, Structures:** Understanding how to declare and initialize it.
- **Functions:** Understanding how to define and call functions, pass arguments, and return values.

3. Mathematical Concepts

- Basic understanding of arithmetic operations, logic (AND, OR, NOT), and occasionally, basic algebra can be useful.

<http://pdvpmtasgaon.edu.in/uploads/dptcomputer/Let%20us%20c%20-%20yashwantkanetkar.pdf>

<https://base.org/quality-resources/problem-solving?srsltid=AfmBOor1ZHZZuMWsNfZ7Kyoi4TInrbsKcph5TNUxpLojIVsig42RstHb>

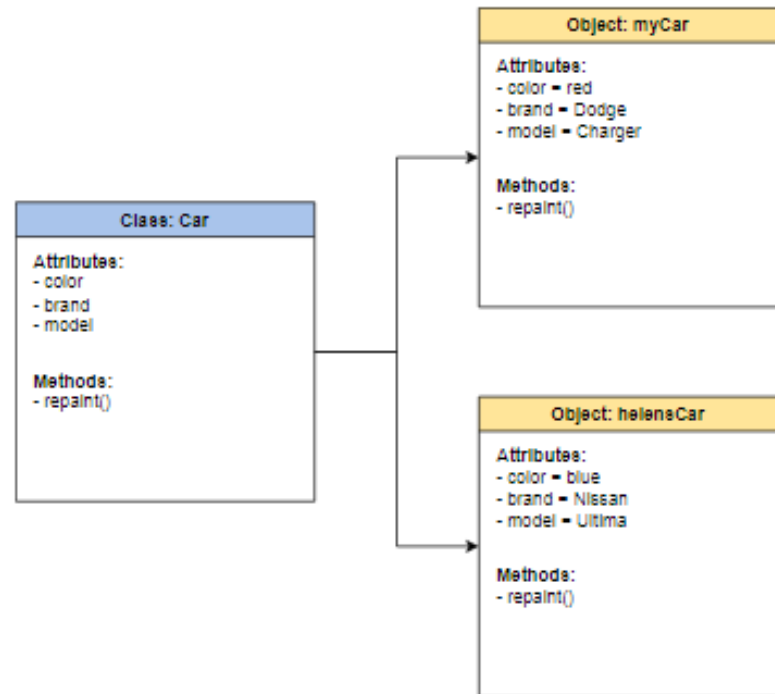
Introduction to Subject

OBJECT ORIENTED PROGRAMMING USING

C++ (OOPC)

What is OOP?

- It is a programming **paradigm** (beliefs, practices, and methodologies).
- Relies on the concept of **classes** and **objects**.
- It is used to structure a software program into simple, **reusable pieces of code** blueprints (usually called classes), which are used to create individual instances of **objects**.
- There are many object-oriented programming Languages, including JavaScript, C++, Java, and Python.



Class blueprint being used to create two Car type objects, myCar and helensCar

<https://www.educative.io/blog/object-oriented-programming>

- Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.
- OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well suited for software that is large, complex and actively updated or maintained.

Cont....



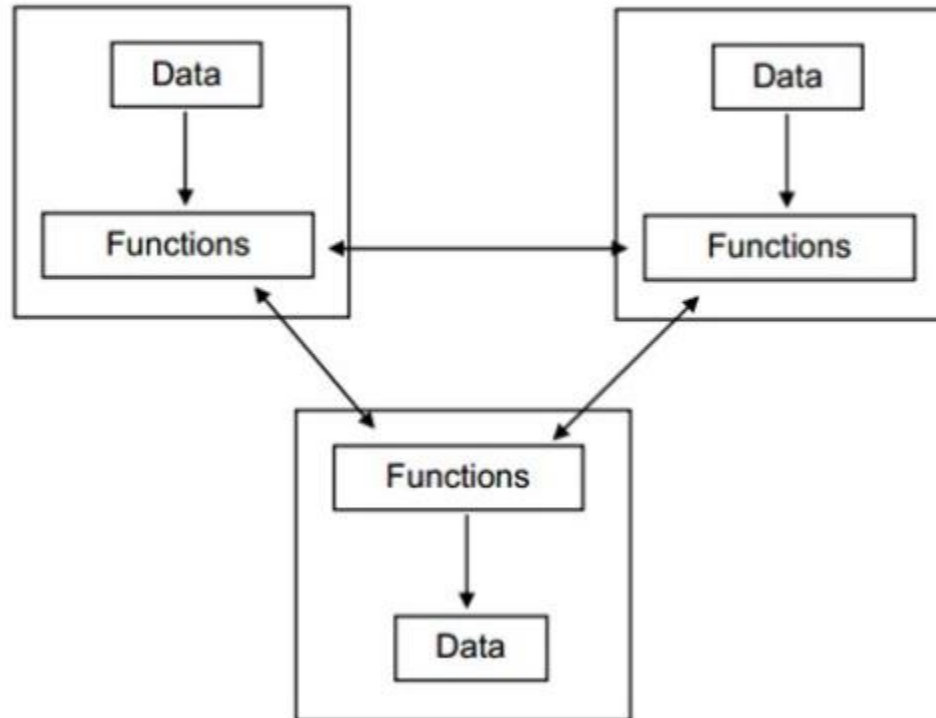
- Once an object is known, it is labeled with a class of objects that defines the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a *method*. Objects can communicate with well-defined interfaces called *messages*.
- It ties *data more closely* to the function that operate on it , and *protect* it from modification from outside function.

Object-oriented programming (OOP)

<https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP>

Cont....

Organization of Data and Functions in OOPs



Comparison of Top-Down vs Bottom-Up



1. Top-Down Approach:

Start by designing or considering the **high-level structure** of the program first, and then break down the components into smaller parts or functions.

- Start with the overall design.
- Break down tasks into smaller parts.
- Often used in structured programming, where planning and organization are key.

EX: Planning a Vacation First, you decide on the destination. (Big picture)

2. Bottom-Up Approach:

Starts with the **detailed components or small building blocks** and then works upwards to assemble them into a complete program.

- Start with small functions or modules.
- Build them incrementally into a complete system.
- Often used in object-oriented programming (especially in C++), as you define and test individual classes or objects first.

EX: Building a Car You start by gathering individual parts: engine, wheels, body frame, seats, etc. (Small components)

Module-I

OBJECT-ORIENTED FUNDAMENTALS

Content:



- **Need** of Object-Oriented Programming (OOP)
- **Features** of OOP
- **Introduction** to C++
- **Structure** of C++ program
- Built-in and user defined **data types**
- **Access specifiers**
- **Examples** illustrating
- Creation of **classes** and **objects**
- **Constructors and Destructors** - Default Arguments, Copy Constructors, Default constructors, Parameterized constructors, Destructor.

Need Of Object-Oriented Programming (OOP):



Why Do We Need Object-Oriented Programming?

- Object-Oriented Programming was developed because limitations were discovered in earlier approaches to programming.
- To appreciate what OOP does, we need to understand what these limitations are and how they arose from traditional programming languages.



Cont...



- Can represent real world entity.
- Open interface.
- Reusability and flexibility.
- Tolerate change in future.
- Productivity and decrease cost.
- Improve quality and cost.
- Manage time.

POP Vs OOP:

OOP

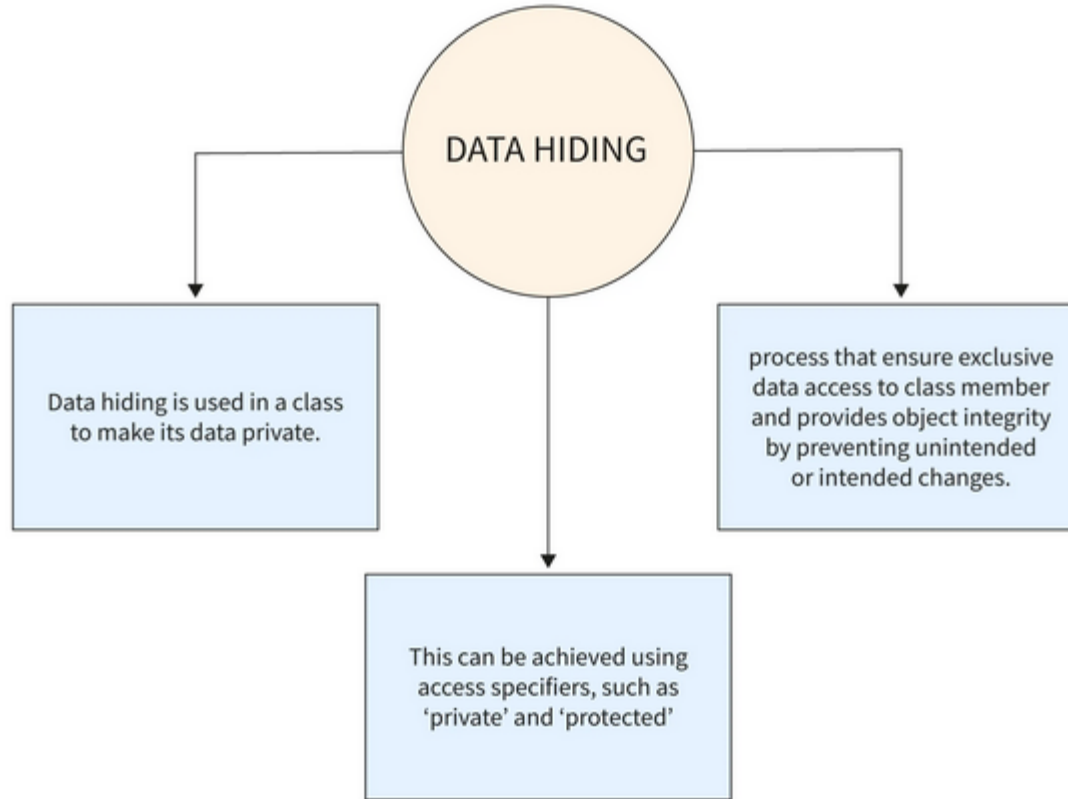
VS

POP

- Object-oriented.
- The program is divided into objects.
- Bottom-up approach.
- Inheritance property is used.
- It uses an access specifier.
- Encapsulation is used to hide the data.
- Concept of virtual function.
- C++, Python, Java.

- Structure oriented.
- Program is divided into functions.
- Top-down approach.
- Inheritance is not allowed.
- It doesn't use an access specifier.
- No data hiding is done.
- No virtual function.
- C, Pascal.

Data Hiding:



<https://www.scaler.com/topics/data-hiding-in-cpp/>

Example:



Bank Account

- Don't have direct access to the account **balance or transactions**.
- you interact with the bank through secure methods like **deposit, withdraw, and check balance**.

Why is this **Encapsulation**?

- The **balance** variable is **private**, so it **cannot** be modified directly.
- Only **public methods (deposit() and withdraw())** allow controlled modifications.
- The **data is secure** and maintains integrity by preventing invalid transactions.

This ensures that no one can **hack** or **tamper** with the account balance directly, just like in a real bank system!

Basic concepts of OOP:



- **Class:** A **class** is a **blueprint** for creating objects.(Car)
- **Object:** An **object** is an **instance of a class**.(Toyota)
- **Data Abstraction and Encapsulation:** Hiding data and restricting direct access.()
- **Inheritance:** **Inheritance** allows a class (**child class**) to **reuse properties and methods** of another class()
- **Polymorphism:** **Polymorphism** means "many forms." It allows **the same function to behave differently** based on the object calling it.(draw_shape)
- **Dynamic Binding:** **M**eans function calls are resolved at **runtime**, not compile time. This is possible using **virtual functions**.
- **Message passing:** **Message Passing** refers to communication between objects by sending and receiving **messages (function calls)**.

Benefits Of Object-Oriented Programming (OOP):

- **Code Reuse and Recycling**: Objects created for Object Oriented Programs can easily be reused in other programs.
- **Design Benefits**: Large programs are very difficult to write. Object Oriented Programs force designers to go through an extensive planning phase, which makes for better designs with less flaws. In addition, once a program reaches a certain size, Object Oriented Programs are actually *easier* to program than non-Object Oriented ones.
- **Software Maintenance**: Programs are not disposable. Legacy code must be dealt with on a daily basis, either to be improved upon (for a new version of an exist piece of software) or made to work with newer computers and software. An Object Oriented Program is much easier to modify and maintain than a non-Object Oriented Program. So although a lot of work is spent before the program is written, less work is needed to maintain it over time.
- **Simplicity**

Features Of OOP:

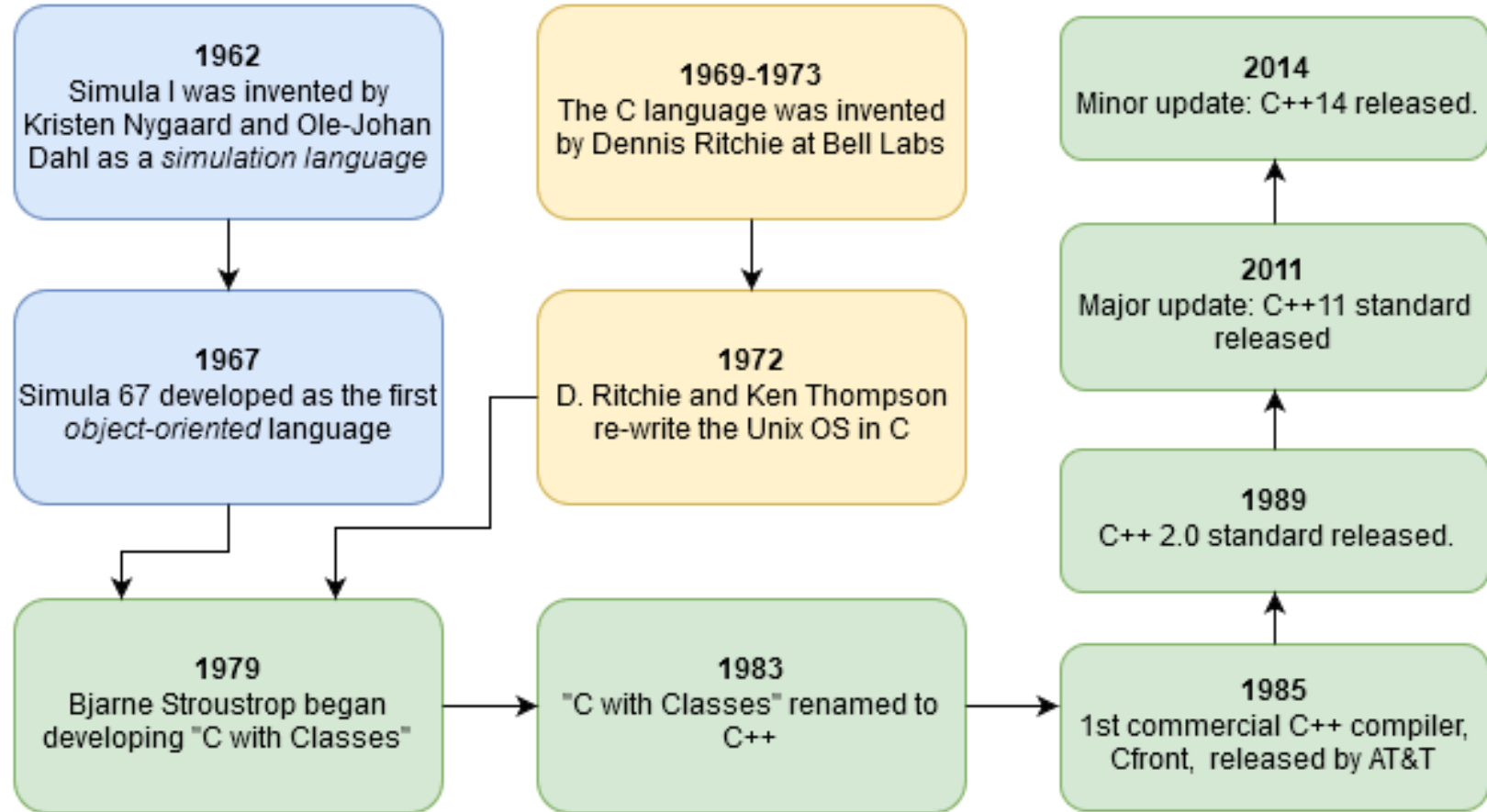
- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data Structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions.

- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows bottom up approach in program design.

Advantages of OOP?

- Encapsulation
- Code Reusability
- Modularity
- Polymorphism
- Improved Maintainability
- Scalability
- Real-World Representation
- Flexibility and Extensibility
- Data Abstraction

History of C++:



Introduction to C++:

- C++ is a cross-platform language that can be used to create high-performance applications.
- C++ was developed by Bjarne Stroustrup, as an extension to the C language.
- C++ gives programmers a high level of control over system resources and memory.
- C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.

Difference between C and C++

- C++ was developed as an extension of C, and both languages have almost the same syntax.
- The main difference between C and C++ is that C++ supports classes and objects, while C does not.

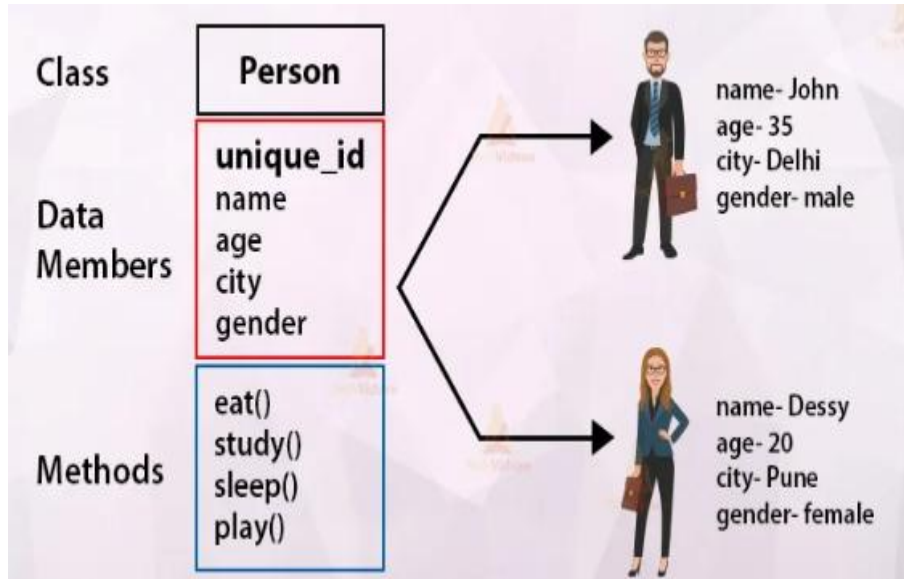
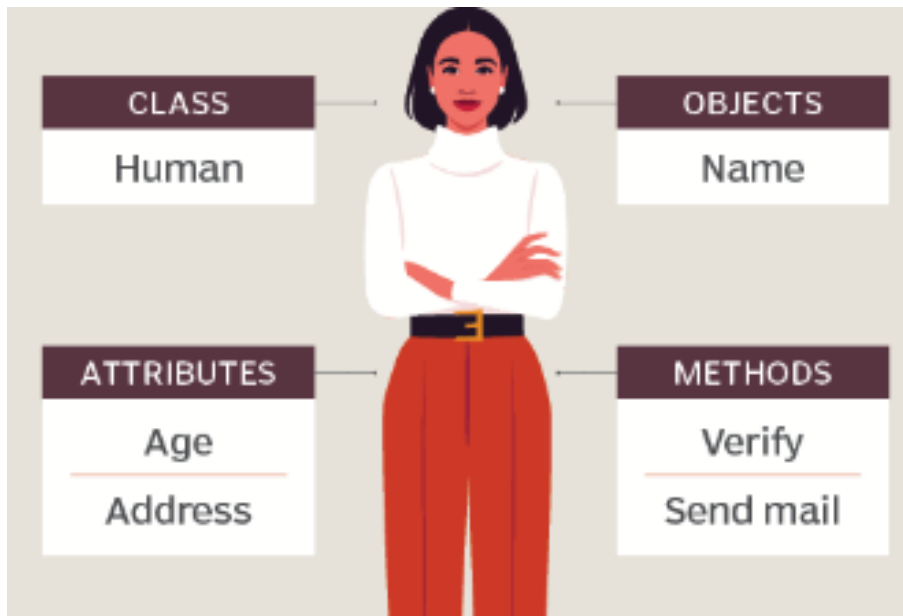
Class And Object:

• Class

- Class is a **Blueprint** for an Object.

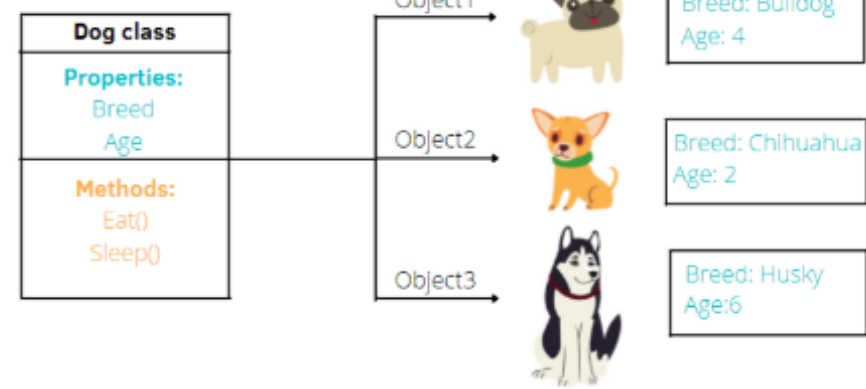
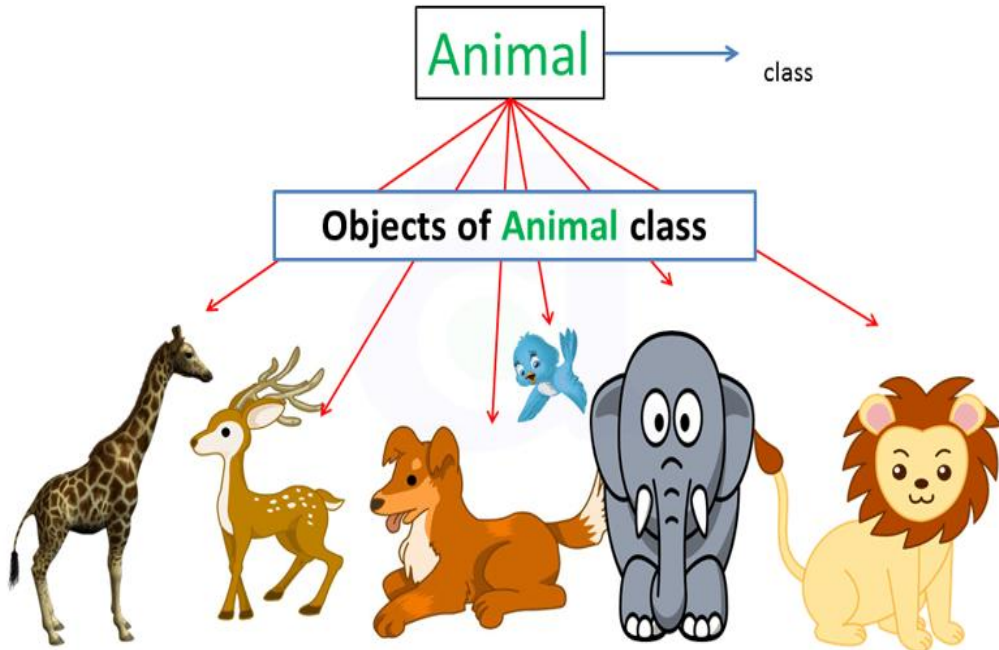
• Object

- Object is instance of class or it is created from class.



Object :

- Basic runtime **entities**
- Object takes up **space in memory**
- It have **state**(member data) and **behavior** (data function)
- Object are instantiated at runtime
- Program executes, object interact by sending **message** to one another
Eg: There are two objects “customer” and “account” then the customer object may send a message to account object requesting for the bank balance.
- Self-contained component that consists of **methods and properties to make a data useful**. It helps you to determines the behavior of the class.
- Each object contain data and code to manipulate data.



Use of Object :

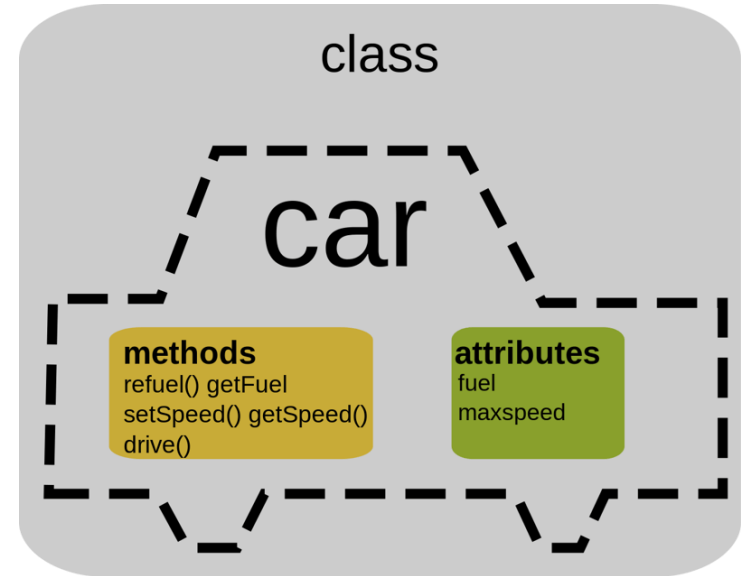
- It is used to **manipulate** data.
- It enables data members and member functions to perform the desired task.
- It helps you to know the type of message accepted and the type of returned responses.

Class :

- A class is an entity that **determines how an object will behave** and what the object will contain.
- In other words, it is a **blueprint** or a set of instruction to build a specific type of object.
- Once object has been defined you **can create any number of objects belong to that class**.
- It provides initial values for member variables and member functions or methods.
- Class is **collection of object**.

Syntax:

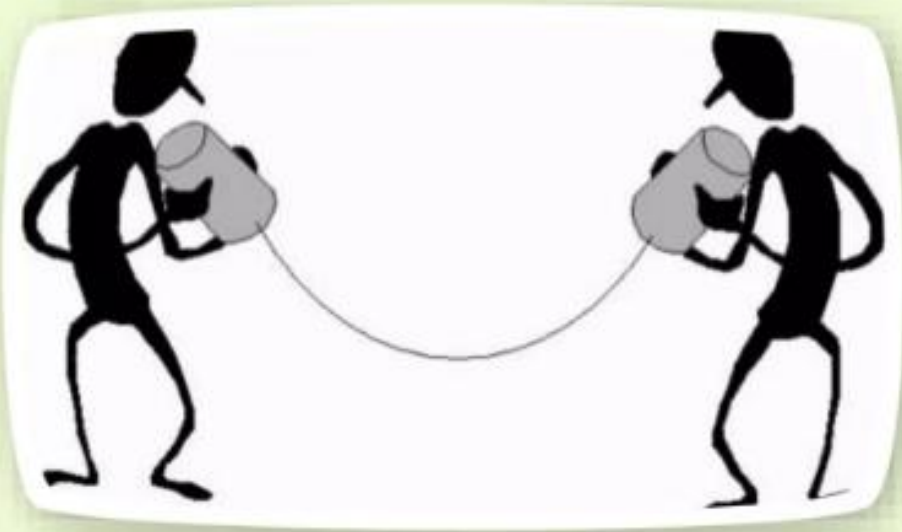
```
class ClassName {  
    access_specifier:  
    // Body of the class  
};
```



Uses of Class :

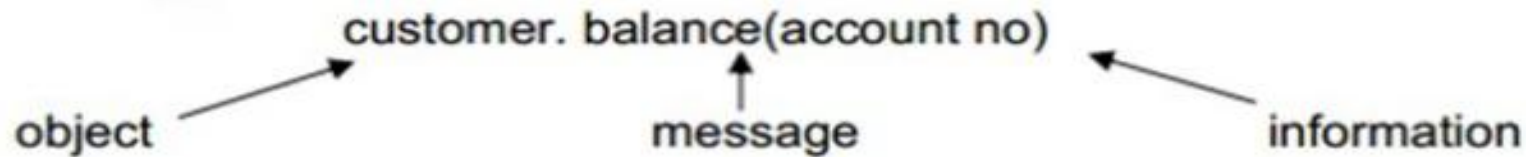
- Class is used to hold both data variables and member functions.
- You can use class to inherit the property of other class.
- It enables you to create user define objects.
- It can be used for a large amount of data and complex applications.

Message Passing



- Objects can communicate with each other by passing message same as people passing message with each other.
- Objects can send or receive message or information.

Ex-

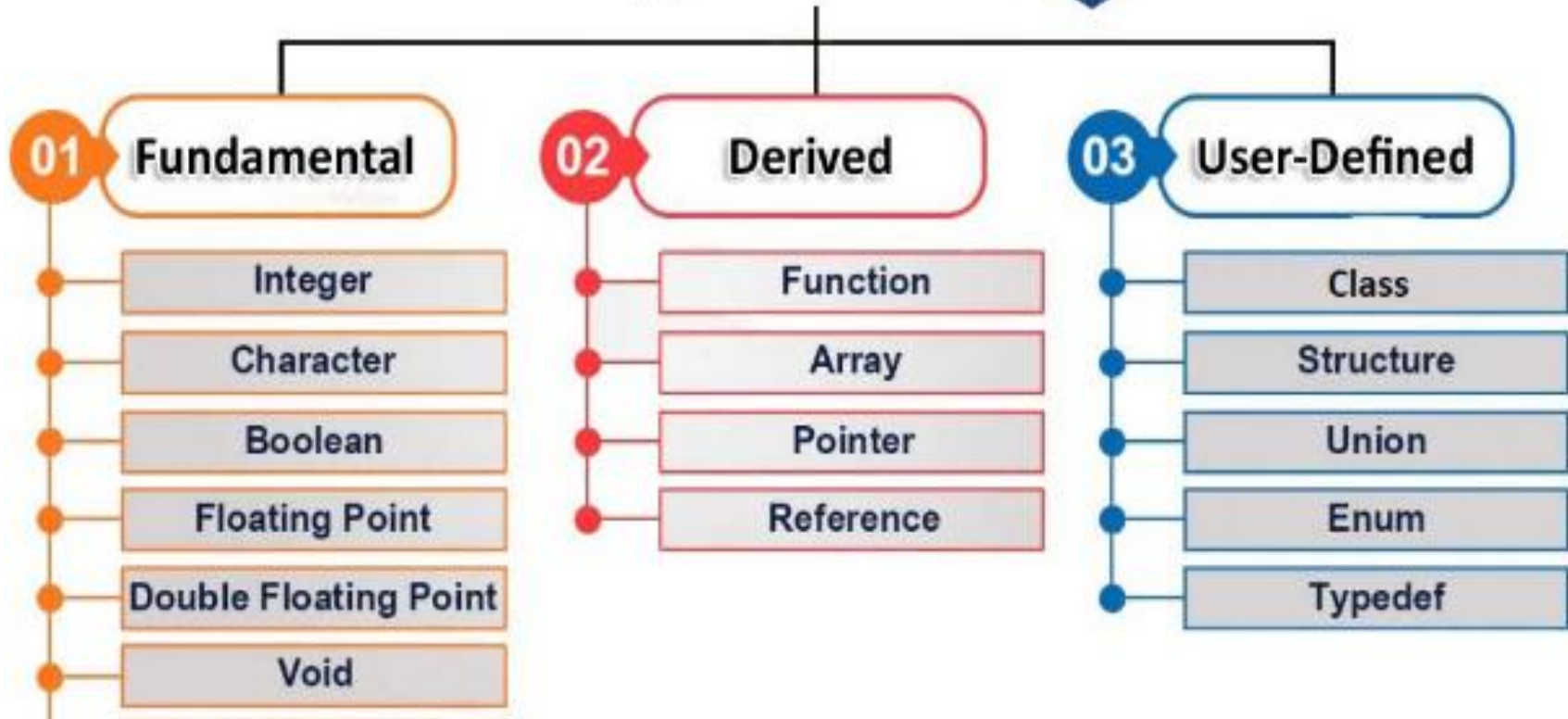


Key Differences :

- A class is a template for creating objects in program whereas the object is an instance of a class.
- You can declare class only once but you can create more than one object using a class.
- Classes doesn't have any values, whereas objects have its own values.
- A class does not allocate memory space on the other hand object allocates memory space.

Built-in and user defined data types

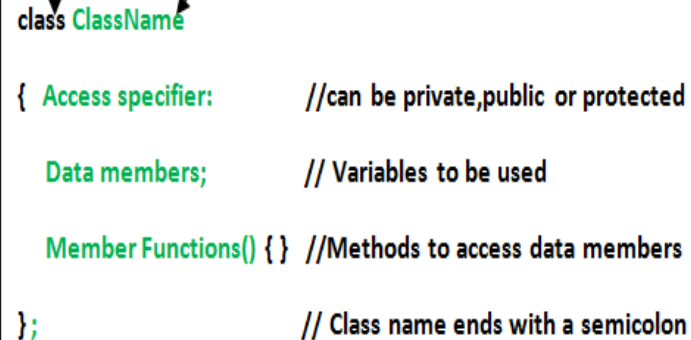
Data Type in C++



Creation of classes

```
class MyClass {  
    public:  
    int var;    // data member  
    void print() {    // member method  
        cout << "Hello";  
    }  
};
```

keyword user-defined name

A diagram showing the general syntax for creating a class. It is enclosed in a black rectangular box. The text inside the box is: 'class' (labeled as 'keyword' with an arrow), 'ClassName' (labeled as 'user-defined name' with an arrow), '{', 'Access specifier:' (with a comment '//can be private,public or protected'), 'Data members;' (with a comment '// Variables to be used'), 'Member Functions() {}' (with a comment '//Methods to access data members'), and '};' (with a comment '// Class name ends with a semicolon').

```
class ClassName  
{ Access specifier:    //can be private,public or protected  
  Data members;    // Variables to be used  
  Member Functions() {} //Methods to access data members  
};    // Class name ends with a semicolon
```

Creation of classes

Keyword
↓

user defined
↓

```
Class student  
{  
public:
```

```
    int    Rollno.;  
    str    Name;  
    int    age;  
    int    showage()  
    {  
        cout<<" The age is:"<<age;  
    }  
};
```

Data(variables)
Data member

Member function

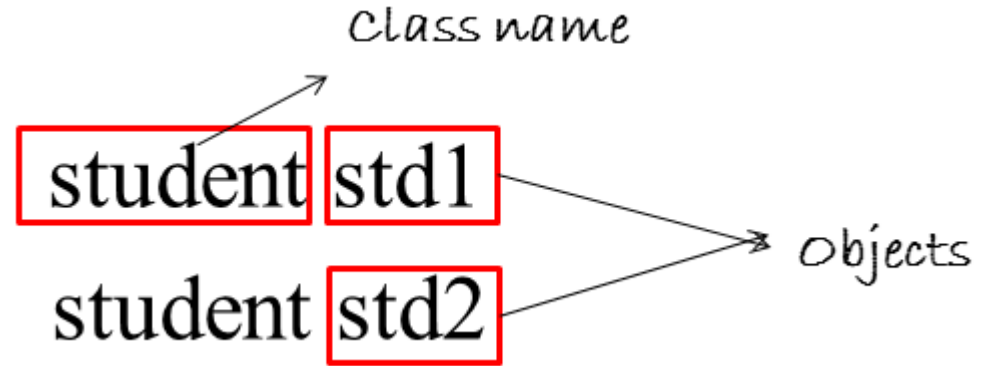
Creation of objects

Syntax

ClassName ObjectName;

Example

MyClass obj;



Accessing

obj.print()

Creation of objects

```
#include <iostream.h>
Class student
{
    public:
        int Rollno.;
        str Name;
        int age;
        int showage()
        {
            cout<<" The age is:"<<age;
        }
};
```

```
int main() {
    student std1;
    std1.Rollno . = 5;
    std1.Name = "Aman";
    std1.age=22;
    std.showage();
    return 0;
}
```

Object creation

Assigning values to variables using object std1

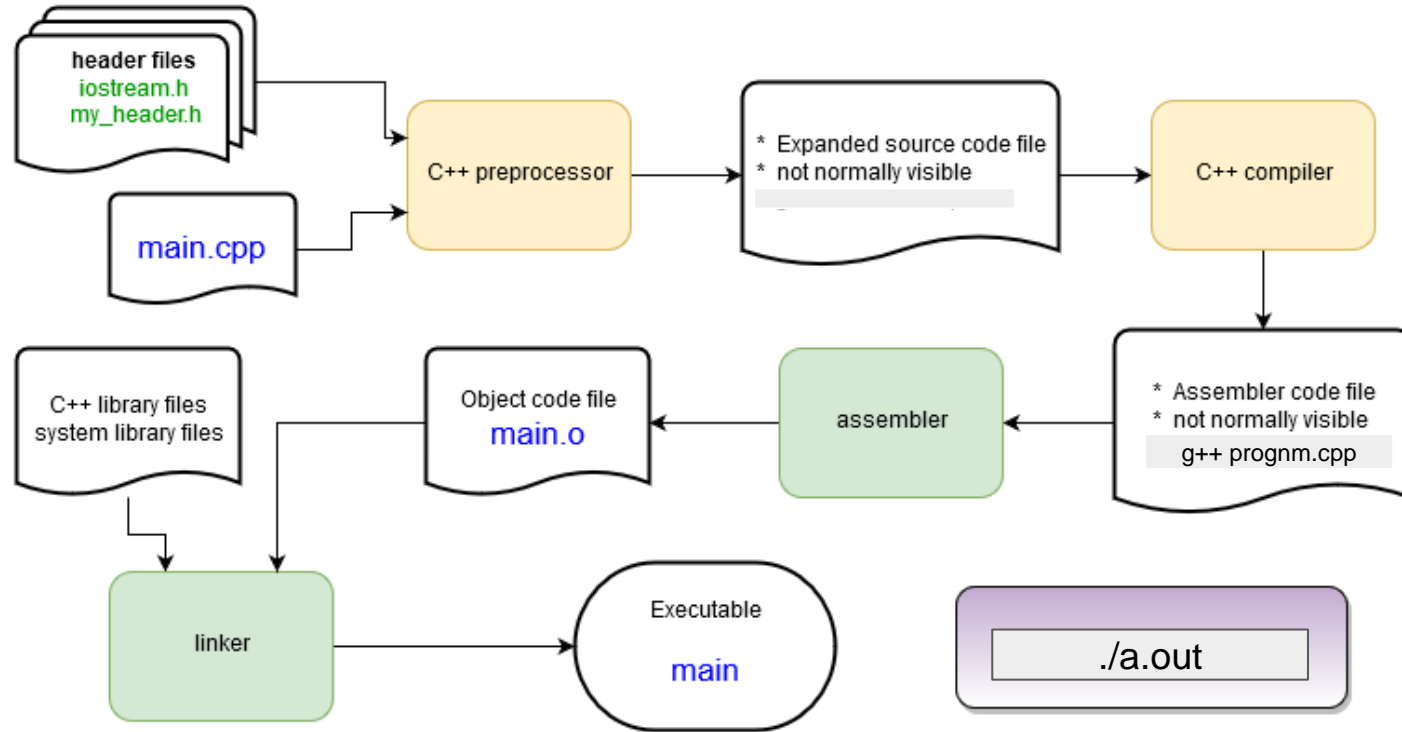
Assigning value to member function using object std1

Example

```
#include<iostream>
using namespace std;
class student{
    public:
    int marks=50;
    void display(){
        cout<<"your marks: "<<marks;
    }
};
int main(){
    student s;
    s.display();

    return 0;
}
```

Behind the Scenes: The Compilation Process



Hello, World! explained

```
main.cpp X
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      cout << "Hello world!" << endl;
8      return 0;
9  }
10
```

The *main* routine – the start of **every** C++ program! It returns an integer value to the operating system and (in this case) takes no arguments: `main()`

The **return** statement returns an integer value to the operating system after completion. 0 means “no error”. C++ programs **must** return an integer value.

```
main.cpp x
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      cout << "Hello world!" << endl;
8      return 0;
9  }
10
```

loads a *header* file containing function and class definitions
Loads a *namespace* called *std*.
Namespaces are used to separate sections of code for programmer convenience. To save typing we'll always use this line in this tutorial.

- *cout* is the *object* that writes to the stdout device, i.e. the console window.
- It is part of the C++ standard library.
- Without the “using namespace std;” line this would have been called as *std::cout*. It is defined in the *iostream* header file.
- << is the C++ *insertion operator*. It is used to pass characters from the right to the object on the left. *endl* is the C++ newline character.

Access specifiers



Access modifiers are used to implement an important aspect of Object-Oriented Programming known as **Data Hiding**.

Ex:ATM Machine

Types of access modifiers:

1. Public
2. Private
3. Protected

Access Modifiers \ Scope	Global	Derived class	Friend class	Within class
Private	✗	✗	✓	✓
Public	✓	✓	✓	✓
Protected	✗	✓	✓	✓

Note: If we do not specify any access modifiers for the members inside the class, then by default the access modifier for the members will be **Private**.

1. Public:



- All the class members declared under the public specifier will be available to everyone.
- Can be accessed by other classes and functions too.
- Public members of a class can be accessed from anywhere in the program using the direct member access operator (.)

Example:



```
#include<iostream>
using namespace std;
class student{
    public:
    int marks=50;
    void display(){
        cout<<"your marks: "<<marks;
    }
};
int main(){
    student s;
    s.display();

    return 0;
}
```

Practice Problem statements:



- Write a program in C++ to find the number is even or odd.
- Write a C++ program to implement a class called Circle that has private member variables for radius. Include member functions to calculate the circle's area and circumference.
- Write a C++ program to create a class called Rectangle that has public member variables for length and width. Implement member functions to calculate the rectangle's area and perimeter.
- Write a C++ program to create a class called Person that has public member variables for name, age and country. Implement member functions to display values.
- Write a C++ program to implement a class called Employee that has public member variables for name, employee ID, and salary. Include member functions to calculate salary based on employee performance.
- Write a program in C++ to find the first 10 natural numbers.
- Write a program in C++ to check whether a number is prime or not.

2. Private



- The class members declared as private can be accessed only by the member functions inside the class.
- They are not allowed to be accessed directly by any object or function outside the class.
- Only the member functions or the friend functions/ friend class are allowed to access the private data members of the class.

Example:



```
#include<iostream>
using namespace std;
class student{
    private:
        int marks;
    public:
        void setdata(int m){
            marks=m;
        }
        int setdata(){
            return marks;
        }
        void display(){
            cout<<"your marks: "<<marks;
        }
};
```

```
int main(){
    student s;
    s.setdata(85);
    cout<<"obtained marks: "<<s.setdata()<<endl;
    s.display();

    return 0;
}
```

Getter and Setter Methods

Define the class with private data members and public methods to get (read) and set (write) these data members.

```
// Getter methods
DataType getAge() {
    return age;
}

// Setter methods
void setAge(DataType newValue) {
    age = newValue;
}
```

Example:



```
#include <iostream>
using namespace std;
class student{
    private:
        int marks;
    public:
        int getmarks(){        //getter method
            cout<<"enter marks";
            cin>>marks;
            return marks;
        }
        int setmarks(){        //setter method
            return marks;
        }
        void display(){
            cout<<"student marks: "<<marks;
        }
};
```

```
int main() {
    student s;
    s.getmarks();
    s.setmarks();
    s.display();
    return 0;
}
```


3. Protected:

- The protected access modifier is similar to the private access modifier in the sense that it can't be accessed outside of its class
- Class members declared as Protected can be accessed by any subclass (derived class) of that class as well.
- Can access with the help of a friend class.

Derived class



Base Class: A base class is a class from which other classes are derived.

The Base class members and member functions are inherited to Object of the derived class.
also called parent class or superclass.

Derived Class: A class that is created from an existing class.

The derived class inherits all members and member functions of a base class.
also called a child class or subclass.

```
class BaseClass{  
    // members....  
    // member function  
}
```

```
class DerivedClass : public BaseClass{  
    // members....  
    // member function  
}
```

Example:



```
#include <iostream>
using namespace std;

class Base {
protected:
    int data;

public:

    int setData() {
        cout<<"enter data";
        cin>> data;
        return data;
    }
    int getdata(){
        return data;
    }
};
```

```
class Derived : public Base {
public:
    void displayData() {
        cout << "Data in Derived: " << data << endl;
        // Accessing protected data
    }
};

int main() {
    Derived obj;
    obj.setData();
    obj.displayData();

    return 0;
}
```

Example of protected modifier:



```
#include<iostream>
using namespace std;

class student{
    protected:
    int marks;
    public:
    int getmarks(){
        cout<<"enter marks";
        cin>>marks;
        return marks;
    }
    int setmarks(){
        return marks;
    }
    void display(){
        cout<<"your marks:"<<marks;
    }
};
```

PICT, Pune.

```
class firstyear:public student{

    public:
    void display(){
        cout<<"your derived marks:"<<marks;
    }
};

int main(){
    student s;
    s.getmarks();
    s.display();
    firstyear f;
    f.getmarks();
    f.display();
    return 0;
}
```

Mrs. Madhuri S. Patil

Base Class Modifier	Public Inheritance	Protected Inheritance	Private Inheritance
public members	Remain public	Become protected	Become private
protected members	Remain protected	Remain protected	Become private
private members	Not inherited	Not inherited	Not inherited

Constructors:

- It is a **special member function** of a class
- Automatically called when an object of the class is created.
- Constructors are not called explicitly, the compiler automatically invokes them.

Syntax

```
<class-name> (){  
...  
}
```

Characteristics/Features of Constructor

1. **Same Name as Class:** The constructor has the same name as the class.
1. **No Return Type:** It does not have a return type (not even `void`).
1. **Automatically Invoked:** It is called automatically when an object is created.
1. **Can Be Overloaded:** Multiple constructors with different parameters can exist.
1. **Memory allocation:** happens at time of constructor call.
1. **Call once:** Only calls once at time of object creation.

Types of Constructor:

1. **Default Constructor:** No parameters. They are used to create an object with default values.

1. **Parameterized Constructor:** Takes parameters. Used to create an object with specific initial values.

1. **Copy Constructor:** Takes a reference to another object of the same class. Used to create a copy of an object.

Example:

```
#include <iostream>
using namespace std;
class student{
    public:
    int marks;
    student(){ //default constructor
        marks=75; //Initializes value to object
    }
    void display(){ //// Member function
        cout<<"student's
marks:"<<marks<<endl;
    }

};
```

```
int main() {
    student std; // Creating an object 'std'
of class 'student'
    std.display(); // // Calling member
function

    return 0;
}
```

Example:

```
#include <iostream>
using namespace std;
class student{
    public:
    int marks;
    student(int m){           //
Parameterized constructor
    marks=m; //Initializes value to object
    }
    void display(){ //// Member function
    cout<<"student's
marks:"<<marks<<endl;
    }

};
```

```
int main() {
    student std(85);           // passing 85
to the constructor
    std.display(); // // Calling member
function

    return 0;
}
```

Example:

```
#include <iostream>
using namespace std;
class student{
public:
    int marks;
    student(int m){                // Parameterized constructor
        marks=m; //Initializes value to object
    }
    student(const student &obj){    // Copy
        Constructor
        marks=obj.marks;
        cout<<"copy constructor marks:
"<<marks<<endl;
    }
    void display(){ /// Member function
        cout<<"student's marks:"<<marks<<endl;
    }
};
```

```
int main() {
    student std1(85);
    student std2=std1;    // Copy constructor is called
    std1.display(); // Calling member function
    return 0;
}
```

```
#include <iostream>
#include <string>
using namespace std;
```

```
class Student {
public:
    string name;

    // Constructor
    Student(string n) {
        name = n;
    }

    // Default Copy Constructor (Shallow, but safe)
    Student(const Student &s) {
        name = s.name; // Copies the string (deep copy)
        cout << "Shallow Copy Constructor Called" << endl;
    }

    void display() {
        cout << "Name: " << name << endl;
    }
};
```

```
int main() {
    Student s1("Alice");
    Student s2 = s1; // Copy
    Constructor is called

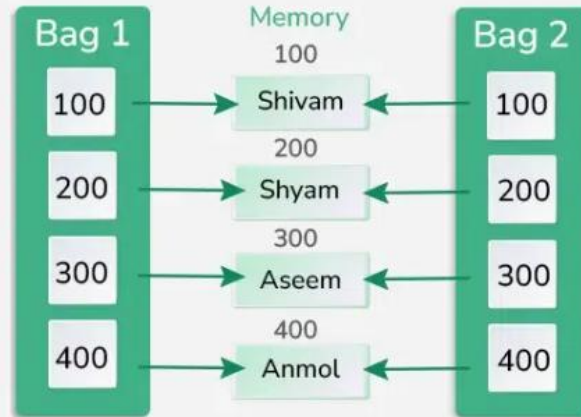
    s2.name[0] = 'M'; // Changes s2
    only, not s1

    s1.display();
    s2.display();

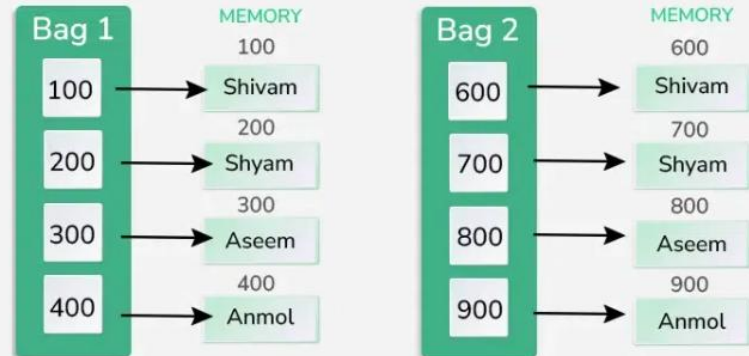
    return 0;
}
```

Without <code>const</code> (✗ Bad)	With <code>const</code> (✓ Good)
Allows modification of <code>s</code>	Prevents modification of <code>s</code>
Can lead to unexpected behavior	Ensures a true copy
Cannot pass <code>const</code> objects	Supports <code>const</code> objects

SHALLOW COPY



DEEP COPY



Destructors



Destructor is an **instance member** function that is invoked automatically whenever an object is going to be destroyed.

- class name preceded by a tilde (~) symbol.
- not possible to define more than one destructor.
- It cannot be declared static or const.
- Destructor neither requires any argument nor returns any value.
- It is automatically called when an object goes out of scope.
- release memory space occupied by the objects created by the constructor.
- objects are destroyed in the reverse

Syntax:

```
~ <class-name>() {  
  
    // some instructions  
  
}
```

Example:

```
#include <iostream>
using namespace std;
```

```
class student {
    public:
    int marks; // Data member

    // Default constructor
    student() {
        marks = 75; // Initializes marks
    }

    // Destructor
    ~student() {
        cout << "Destructor..!!" << endl;
    }
}
```

```
void display() {           //Member
    function
    cout << "Student's marks: " << marks <<
    endl;
    }
};

int main() {
    student std; // Creates an object
    std.display(); // Calls the 'display'
    function

    return 0; // The program ends here;
    destructor is called when 'std' goes out of scope
}
```


Dynamic Memory Allocation:



1. **Static Memory:** In static memory allocation, memory is allotted and deallocated by the compiler on its very own.
2. **Dynamic Memory:** In dynamic allocation, the allocation and deallocation of the memory happen at runtime.

The **new** operator in C++ is used to dynamically allocate a block of memory and store its address

The **delete** operator is used to deallocate the block of memory, which is dynamically allocated using the new operator.

Syntax:

```
data_type* ptr_var = new data_type;
```

```
delete ptr_var;
```

Syntax for array:

```
data_type* ptr_var = new data_type[size_of_the_array];
```

<https://www.scaler.com/topics/cpp/dynamic-memory-allocation-in-cpp/>

Smarts Pointers:

<https://www.codecademy.com/resources/docs/cpp/smart-pointers>

<https://www.scaler.com/topics/cpp/smart-pointers-in-cpp/>

Example:

```
#include <iostream>
```

```
int main() {  
    int *ptr=new int;  
    *ptr=10;  
    std::cout<<*ptr;  
    delete ptr;  
    return 0;  
}
```

Static Keyword in C++:

A [static variable](#) is a variable that is declared using the [keyword static](#). The space for the static variable is allocated only one time and this is used for the entirety of the program.

```
student MyStud
{
    static int a;
};
int MyStudt::a = 67;
```

<https://www.geeksforgeeks.org/static-keyword-cpp/>

Problem statements(H.W):

- Write a program that takes the following inputs from the user:

An integer

A float

A character

Print these inputs with the following format:

The integer should be printed in decimal, octal, and hexadecimal formats.

The float should be printed with 2 decimal places.

The character should be printed as a character and its ASCII value.

Problem statements(H.W):

- Create a class called Student that holds the following information about a student:

Name

Roll number

Marks of 3 subjects.

Create a member function that calculates the total marks and displays the student's details. Use a constructor to initialize the values and a destructor to display a message when the object is destroyed.

Problem statements(H.W):

- Design a class BankAccount with the following functionalities:

A constructor to initialize account number and balance for a new account.

A member function deposit() that accepts a deposit amount and adds it to the balance.

A member function withdraw() that accepts an amount and subtracts it from the balance if sufficient balance is available.

A destructor that displays a message showing the final balance when the object is destroyed.

Problem statements(H.W):

Write a C++ program to create a class Employee with:

- A default constructor that initializes empID to 0 and name to "Unknown".
- A parameterized constructor that takes empID and name as arguments.
- A function display() to print employee details.

Problem statements(H.W):

Create a class Shape with overloaded constructors:

- Shape(int) for square (side length).
- Shape(int, int) for rectangle (length & width).
- Shape(double) for circle (radius).

Write a function calculateArea() that prints the area of the shape.