CS-1004 Object Oriented programming
Week 2

Instructor:
ATIYA
LECTURER
(COMPUTER SCIENCE DEPARTMENT)
NATIONAL UNIVERSITY- FAST (KHI CAMPUS)
EMAIL: ATIYA.JOKHIO@NU.EDU.PK
ROOM: (LECTURER ROOM#19, CS DEPT BASEMENT-2)

Summarize

A class :

- It’s a blue print .
- It’s a design or template.

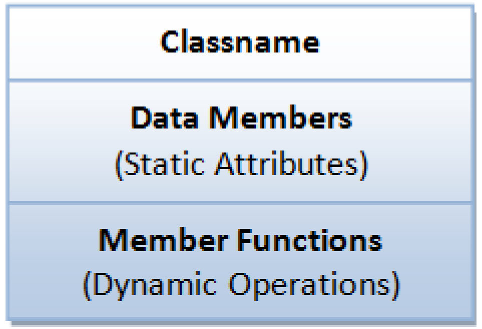
An Object:

- Its an instance of a class.
- Implementation of a class.

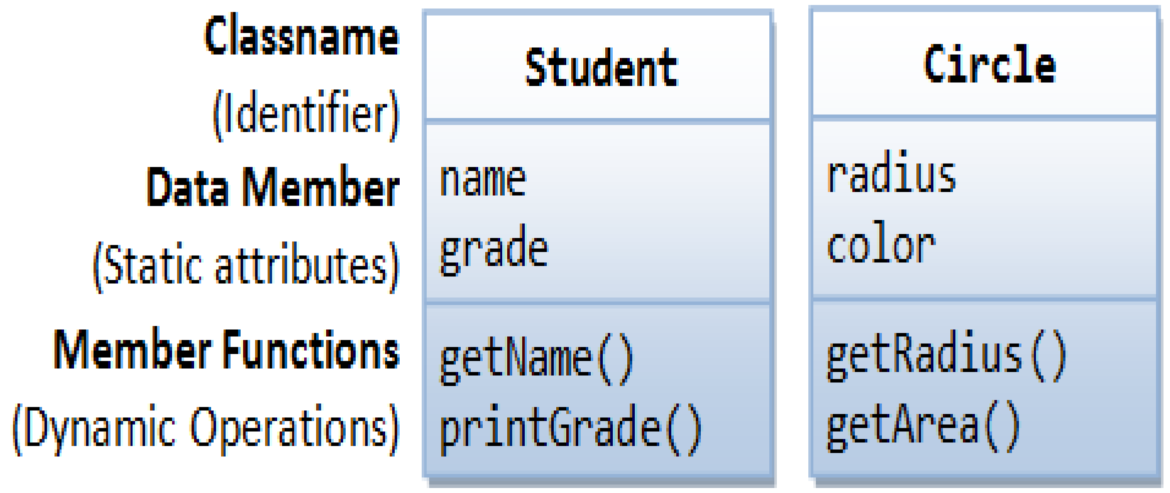
NOTE: Classes are invisible, object are visible

A Class is a 3-Compartment Box encapsulating Data and Functions

1.Classname (or identifier): identifies the class.
2.Data Members or ***Variables*** (or *attributes*, *states*, *fields*): contains the *attributes* of the class.
3.Member Functions (or *methods*, *behaviors*, *operations*): contains the *dynamic operations* of the class.



Example of classes

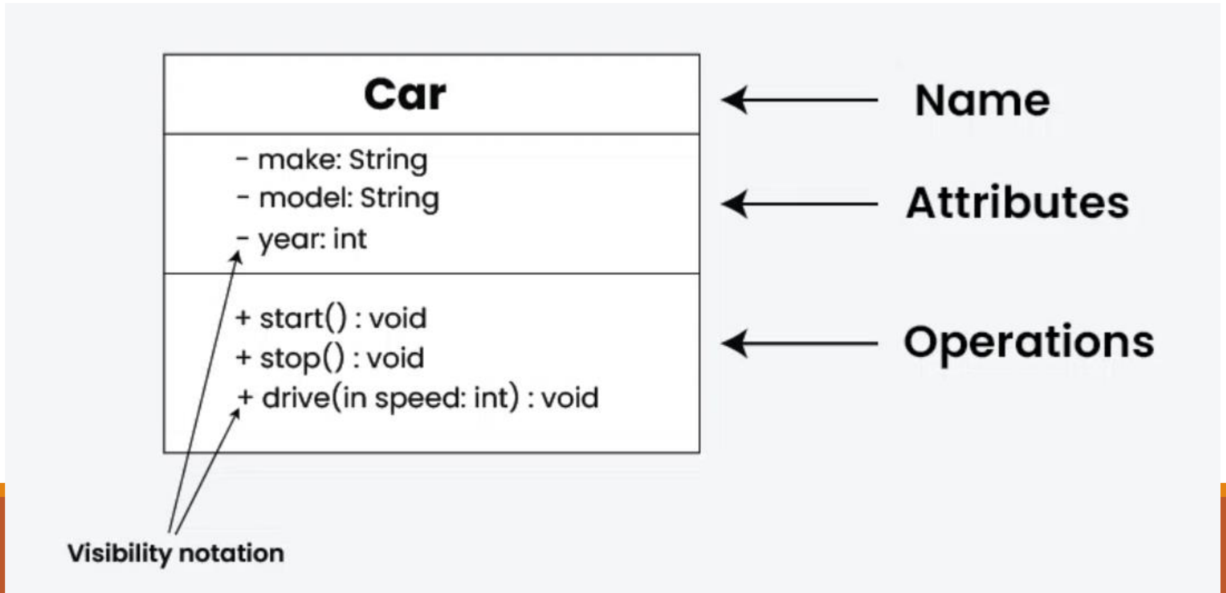


UML Class Notation

- Class diagrams are a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes within a system i.e. used to construct and visualize object-oriented systems.

- **What are some things that are not represented in a UML class diagram?**

 - The details of how the classes interact with each other algorithmic details; how a particular behavior is implemented



UML Class Notation

1. **Class Name:** The name of the class is typically written in the top compartment of the class box and is centered and bold.

2. **Attributes:** Attributes, also known as properties or fields, represent the data members of the class. They are listed in the second compartment of the class box and often include the visibility (e.g., public, private) and the data type of each attribute.

3. **Methods:** Methods, also known as functions or operations, represent the behavior or functionality of the class. They are listed in the third compartment of the class box and include the visibility (e.g., public, private), return type, and parameters of each method.

Visibility Notation: Visibility notations indicate the access level of attributes and methods.
Common visibility notations include:

- + for public (visible to all classes)
- - for private (visible only within the class)
- # for protected (visible to subclasses)

Class attributes

attributes (fields, instance variables)

visibility name : type = default_value

visibility:

+ public

# protected

 - private

/ derived

underline static attributes

attribute example: - balance : double = 0.00

Class operations / methods

operations / methods

visibility name (parameters) : return_ type

visibility:

+ public

# protected

- private

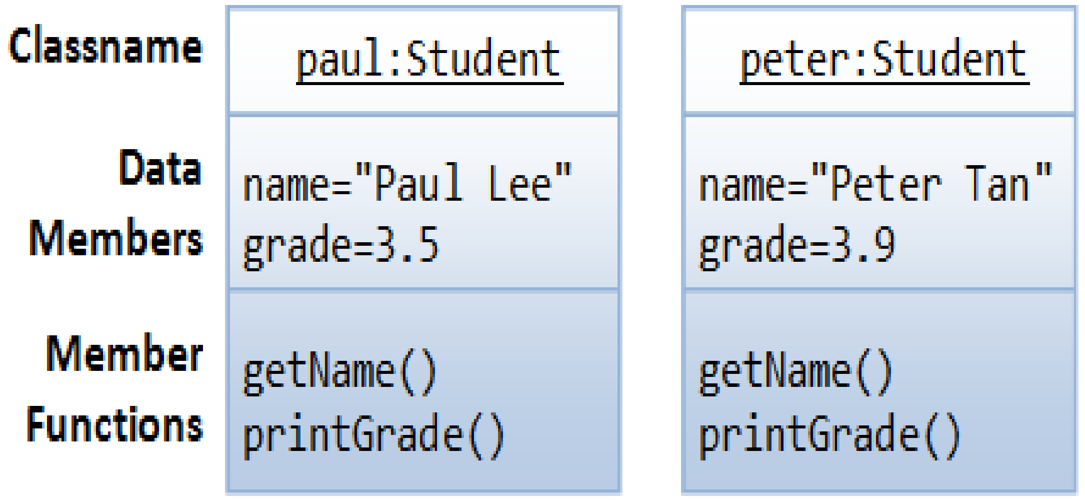
underline static methods

parameter types listed as (name: type)

omit return_type on constructors and when return type is void

method example: + distance(p1: Point, p2: Point): double

Example of objects



Two instances of the **Student** class

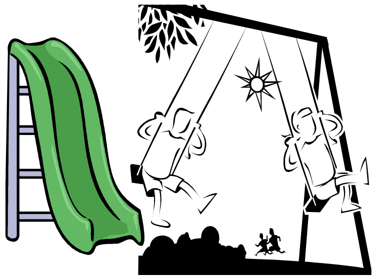
Each object of a class maintains its own copy of its attributes in memory

Class Activity

0/1- Library

2- Classroom

3- Parents


4- Playground

5- Pen

6- Book

7- Bag

8- Student

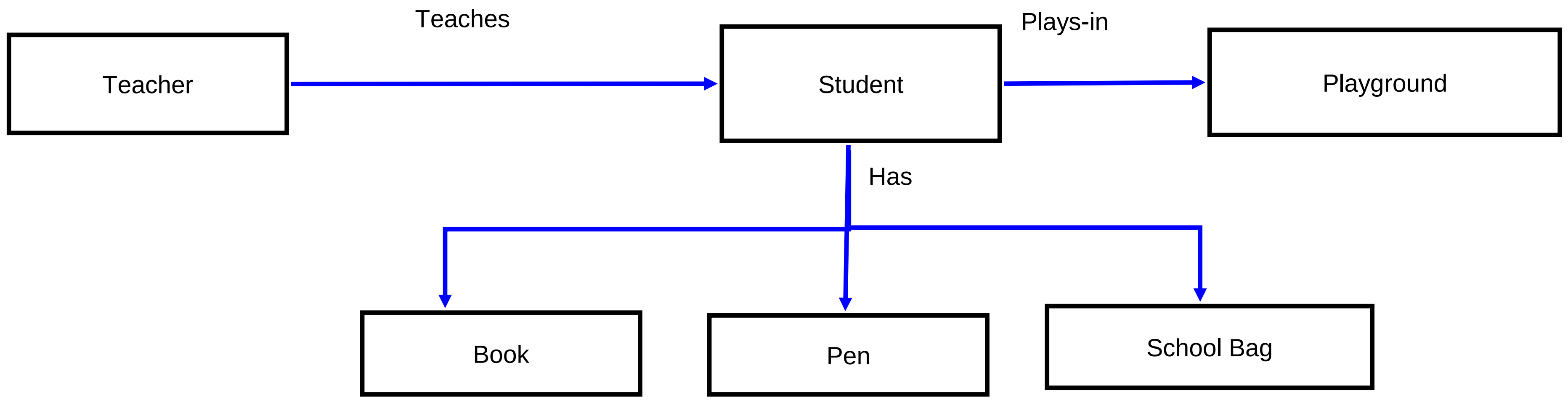
9 Teacher

Questions

1- Identify at least 3 behaviors and related data for an object.

2- Draw an object interaction model based on the objects .

Solution


Teaches
Plays-in
Teacher
Student
Playground
Has
Book
Pen
School Bag

Generalized Class

- The class that only exhibits the common features of its objects.

Examples:

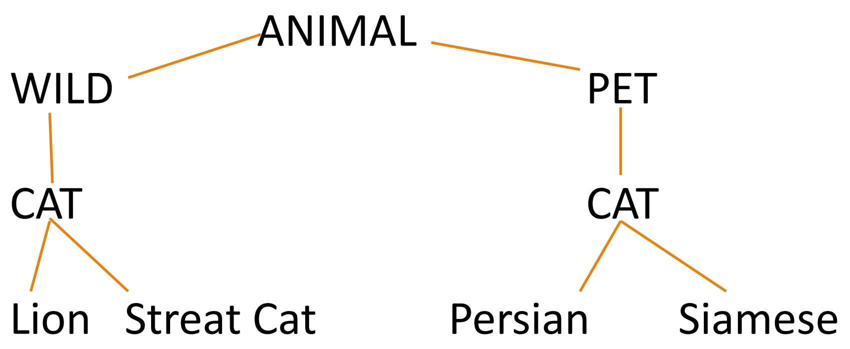
- ANIMAL
- BIRDS
- HUMAN
- No object of generalized class is found.

Specialized Class

- The class that exhibits different or unique features (behaviors)

ANIMAL (Generalized)

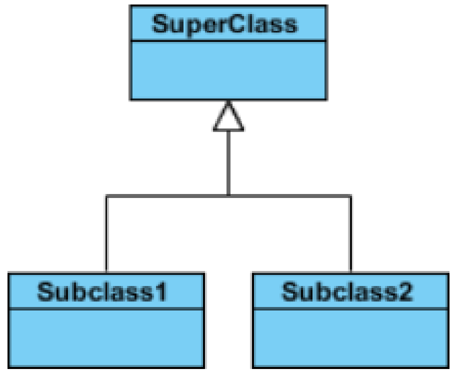
- Specialized:
  - Mammals
  - Cats
  - Dog



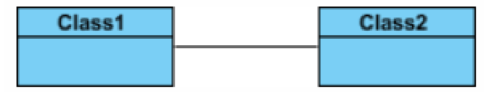
Relationship Type

-

Inheritance (or Generalization):
•Represents an "is-a" relationship.
•An abstract class name is shown in italics.
•SubClass1 and SubClass2 are specializations of Super Class.
•A solid line with a hollow arrowhead that point from the child to the parent class



Simple Association:
•A structural link between two peer classes.
•There is an association between Class1 and Class2
•A solid line connecting two classes



Type of classes

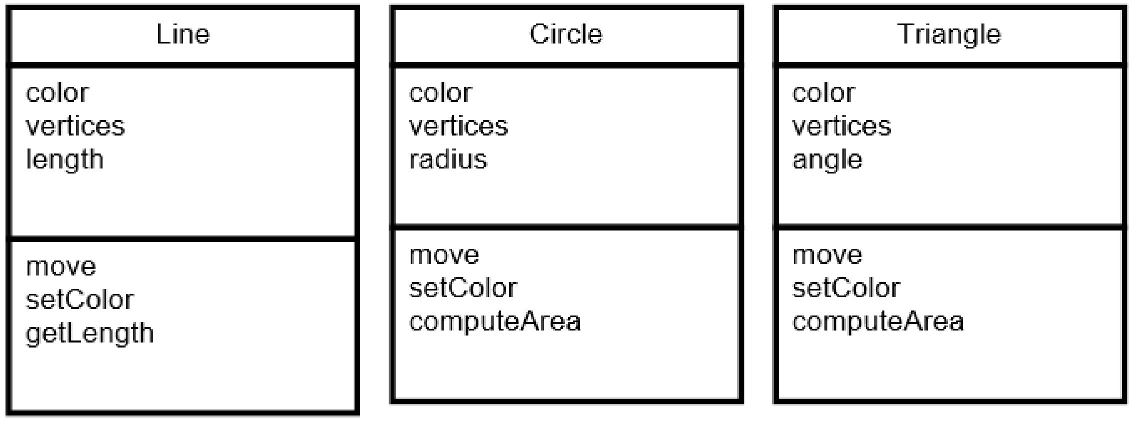
1- Abstract Class: The classes we make against abstract concepts are called abstract classes. Abstract Classes can not exist standalone.

2- Concrete Class: The entities that actually we see in our real world are called concrete objects and classes made against these objects are called concrete classes.

- Can be derived- can serve as a base class for other classes
- Implements abstract base class

3- Sub-type: Sub-typing means that derived class is behaviorally compatible with the base class. Also known as Extension. Type B is called a subtype of type A if an instance of B can be used in every situation where an instance of A is required. In a way, the subtype does everything the supertype does and then some more, so it can be used as substitute for the “parent” or “supertype”.

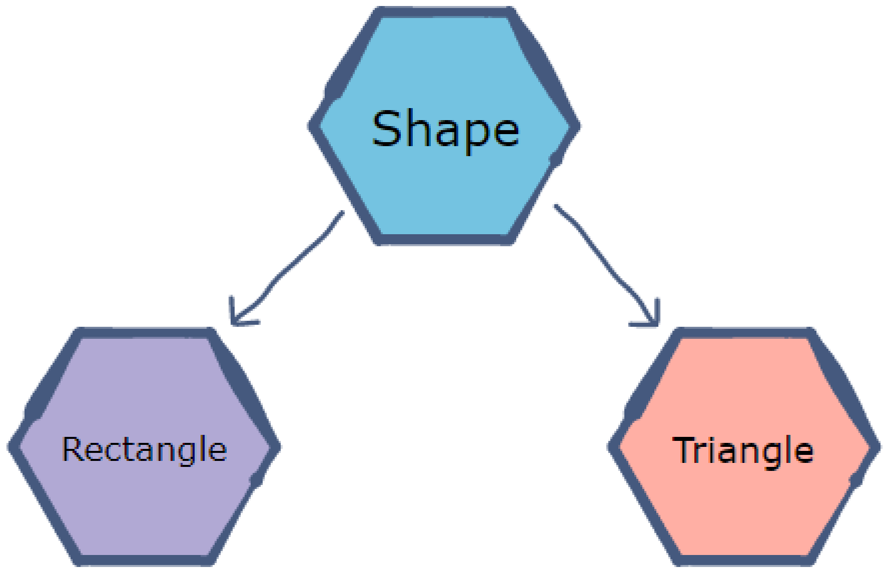
Concrete Class



Line is shape   Circle is a shape   Triangle is a shape

Abstract Class

- **Abstract class in C++** is a class that has at least one pure virtual function (i.e., a function that has no definition). The classes inheriting the abstract class must provide a definition for the pure virtual function; otherwise, the subclass would become an abstract class itself.
- The purpose of an **abstract class** (often referred to as an ABC) is to provide an appropriate base class from which other classes can inherit.
- For example, a Vehicle parent class with Truck and Motorbike inheriting from it is an abstraction that easily allows more vehicles to be added.
- Abstract classes are essential to providing an abstraction to the code to make it reusable and extendable.



Virtual member function

A virtual member function for which no implementation is given is called a *pure virtual function* .

```
Virtual void fun() = 0;
```

Sometimes implementation of all function cannot be provided in a base class because we don’t know the implementation. Such a class is called abstract class.

Classes

```
class Car

{

        void accelerate()
        { \\ logic for acceleration }


        void brake()
        { \\ logic for brakes }

};
```

classes

```
class Car
{
        string model;
        int numOfDoors;
        string color;

        void accelerate()
        { \\ logic for acceleration }

        void brake()
        { \\ logic for brakes }

};
```

How to use the car?

```
int main()
{
        Car car;
        car.accelerate();
}
```

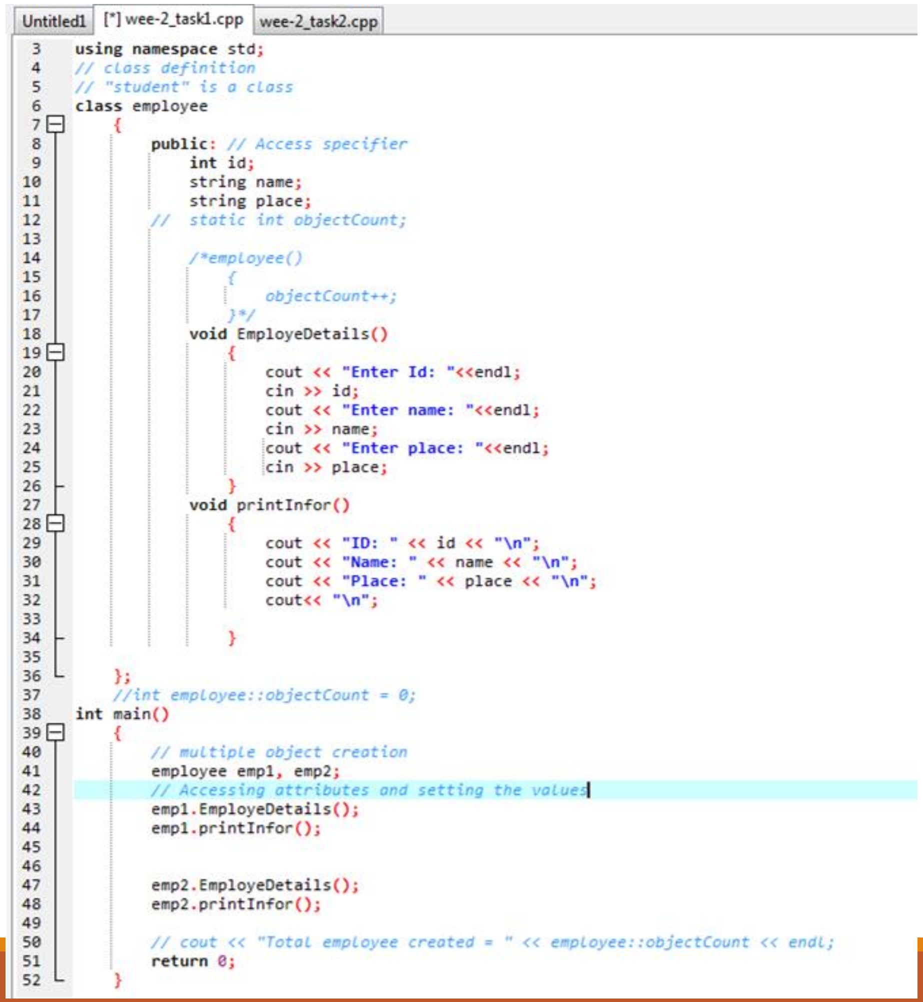
Object v/s Classes

BASIS FOR COMPARISON	OBJECT	CLASS
Definition	An instance of a class is known as Object.	A template or blueprint with which objects are created is known as Class.
Type of entity	Physical	Logical
Creation	Object is invoked by.	Class is declared by using class keyword.
Memory allocation	Creation of object consumes memory.	The formation of a class doesn't allocate memory.

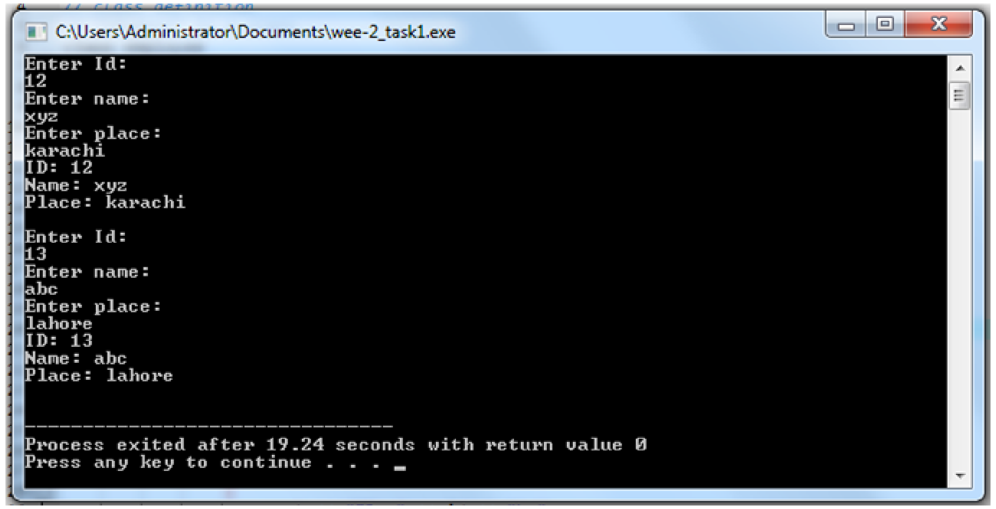
Multiple Objects

- You can create multiple objects of one class.
- The benefit of multiple instances of a single class is that each of them can hold different values to their variables, and the methods only affect that specific object, so each of them have the same skeleton, but are in independent states.
- An **example** of this is an enemy class in a game. If you have multiple enemies, you can create multiple instances of a class, where each enemy is an instance. They all may be in a different position, which could be a private variable of the class.

Introduction To Multiple Objects Into Real World



```cpp
using namespace std;
// class definition
// "student" is a class
class employee
    {
        public: // Access specifier
            int id;
            string name;
            string place;
        //  static int objectCount;

            /*employee()
                {
                    objectCount++;
                }*/
            void EmployeDetails()
                {
                    cout << "Enter Id: "<<endl;
                    cin >> id;
                    cout << "Enter name: "<<endl;
                    cin >> name;
                    cout << "Enter place: "<<endl;
                    cin >> place;
                }
            void printInfor()
                {
                    cout << "ID: " << id << "\n";
                    cout << "Name: " << name << "\n";
                    cout << "Place: " << place << "\n";
                    cout<< "\n";

                }

    };
    //int employee::objectCount = 0;
int main()
    {
        // multiple object creation
        employee emp1, emp2;
        // Accessing attributes and setting the values
        emp1.EmployeDetails();
        emp1.printInfor();


        emp2.EmployeDetails();
        emp2.printInfor();

        // cout << "Total employee created = " << employee::objectCount << endl;
        return 0;
    }
```



```
Enter Id:
12
Enter name:
xyz
Enter place:
karachi
ID: 12
Name: xyz
Place: karachi

Enter Id:
13
Enter name:
abc
Enter place:
lahore
ID: 13
Name: abc
Place: lahore


--------------------------------
Process exited after 19.24 seconds with return value 0
Press any key to continue . . . _
```

Introduction To Objects Into Real World

- **Object**
  - instance of class
- **State of an object**
  - Static and Dynamic
- **Behavior of an object**
  - Behavior is how an object acts & reacts, in terms of its state change and message passing.
- **Identity of an object**
  - **Employee** - Empid, Name, Gender, Mobile_No
- **Responsibility of an object**
  - Responsibility of an object is the role it serves within the system.

Class Methods

- Methods are **functions** that belongs to the class.
- There are two ways to define functions that belongs to a class:
  - Inside class definition
  - Outside class definition

Defining Methods Separately

For methods that are declared but not defined in the class we need to provide a separate definition

To define the method, you define it as any other function, except that the name of the function is *ClassName*::*FuncName*

- :: is the scope resolution operator, it allows us to refer to parts of a class or structure

Inside Example

```
class MyClass {         // The class
  public:               // Access specifier
    void myMethod() {  // Method/function defined inside the class
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

Outside Example

```
class MyClass {        // The class
  public:              // Access specifier
    void myMethod();   // Method/function declaration
};

// Method/function definition outside the class
void MyClass::myMethod() {
  cout << "Hello World!";
}

int main() {
  MyClass myObj;     // Create an object of MyClass
  myObj.myMethod();  // Call the method
  return 0;
}
```

To define a function outside the class definition, you have to declare it inside the class and then define it outside of the class. This is done by specifiying the name of the class, followed the scope resolution :: operator, followed by the name of the function:

Class Outside Example-Parameter

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

class Car {
  public:
    int speed(int maxSpeed);
};

int Car::speed(int maxSpeed) {
  return maxSpeed;
}

int main() {
  Car myObj; // Create an object of Car
  cout << myObj.speed(200); // Call the method with an argument
  return 0;
}
```

Case-Study

1. Write the definition for a class called **Rectangle** that has floating point data members length and width. The class has the following member functions:
   void setlength(float) to set the length data member
   void setwidth(float) to set the width data member
   float area() to calculate and return the area of the rectangle
   void show() to display the length and width of the rectangle

Exercise

1. Write the definitions for each of the above member functions.
2. Write main function to create two rectangle objects. Set the length and width of the first rectangle to 5 and 2.5. Set the length and width of the second rectangle to 5 and 18.9. Display each rectangle and its area. check whether the two Rectangles have the same area and print a message indicating the result

Sructure

structure is another user defined data type available in C/C++ that allows to combine data items of different kinds.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −

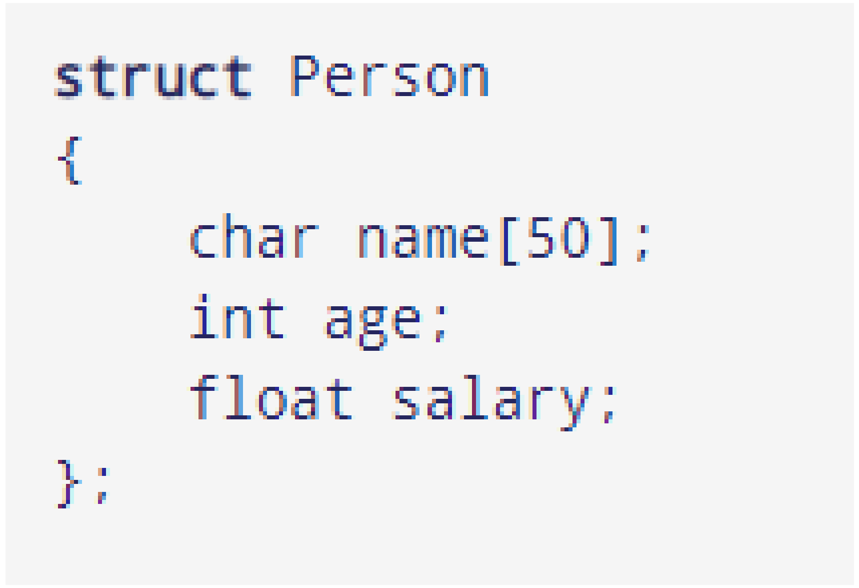
- Title
- Author
- Subject
- Book ID

Syntax of structure

The struct keyword defines a structure type followed by an identifier (name of the structure).

Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure.

- For example:



```
struct Person
{
    char name[50];
    int age;
    float salary;
};
```

C Structure vs C++ Structure

1. Member functions inside structure:

Structures in C cannot have member functions inside structure but Structures in C++ can have member functions along with data members.

```
struct person_str {
string name; int age; //constructor
person_str() {
name = "default";
age = 77;}
```

C Structure vs C++ Structure

2. Direct Initialization:

We cannot directly initialize structure data members in C but we can do it in C++

```
#include <stdio.h>

struct Record {
		int x = 7;
};

// Driver Program
int main()
{
		struct Record s;
		printf("%d", s.x);
		return 0;
}
```

```
/* Output : Compiler Error
6:8: error: expected ':', ',', ';', '}' or
'__attribute__' before '=' token
int x = 7;
                                         ^
In function 'main': */
```

C Structure vs C++ Structure

3. Using struct keyword:

In C, we need to use struct to declare a struct variable. In C++, struct is not necessary. For example, let there be a structure for Record. In C, we must use “struct Record” for Record variables. In C++, we need not use struct and using ‘Record‘ only would work.

```
struct Record {
    int x = 7;
};

// Driver Program
int main()
{
    struct Record s;
    printf("%d", s.x);
    return 0;
}
```

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

struct Record {
    int x = 7;
};

// Driver Program
int main()
{
    Record s;
    cout << s.x << endl;
    return 0;
}
```

C Structure vs C++ Structure

4. Static Members: C structures cannot have static members but is allowed in C++.

5. Access Modifiers: C structures do not have access modifiers as these modifiers are not supported by the language. C++ structures can have this concept as it is inbuilt in the language.

6. Constructor creation in structure: Structures in C cannot have constructor inside structure but Structures in C++ can have Constructor creation.

Structure vs Classes

In C++, a structure is the same as a class except for a few differences.

The most important of them is security.

A Structure is not secure and cannot hide its implementation details from the end user while a class is secure and can hide its programming and designing details.

- **Following are the points that expound on this difference:**
- What if we forget to put an access modifier before the first field?

1) Data Members of a class are private by default and members of a struct are public by default.

```
struct Robot {     OR     class Robot {
  float locX;                float locX;
```

Structure vs Classes

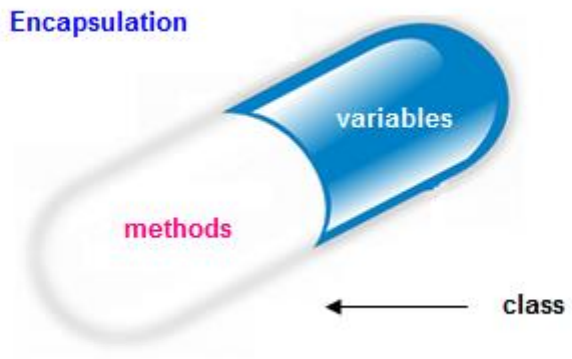
◦ **Class**	◦ **Structure**
Classes are of reference types.	Structs are of value types.
All the reference types are allocated on heap memory.	All the value types are allocated on stack memory.
Class has limitless features.	Struct has limited features.
Class is generally used in large programs.	Struct are used in small programs.
A Class can inherit from another class.	A Struct is not allowed to inherit from another struct or class.

Encapsulation (1st Principle of OOP)

- **Encapsulation** is a process of wrapping of data and methods in a single unit
- This is to prevent the access to the data directly, the access to them is provided through the functions of the class.
- The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class.

Real Life Example of Encapsulation

The common example of encapsulation is **Capsule**. In capsule all medicine are encapsulated inside capsule.



Real Life Example of Encapsulation

- A Phone stores phone numbers in digital format and knows how to convert it into human-readable characters

- We don’t know
  - How the data is stored
  - How it is converted to human-readable characters

ENCAPSULATION – ADVANTAGES

- Simplicity and clarity
- Low complexity
- Better understanding

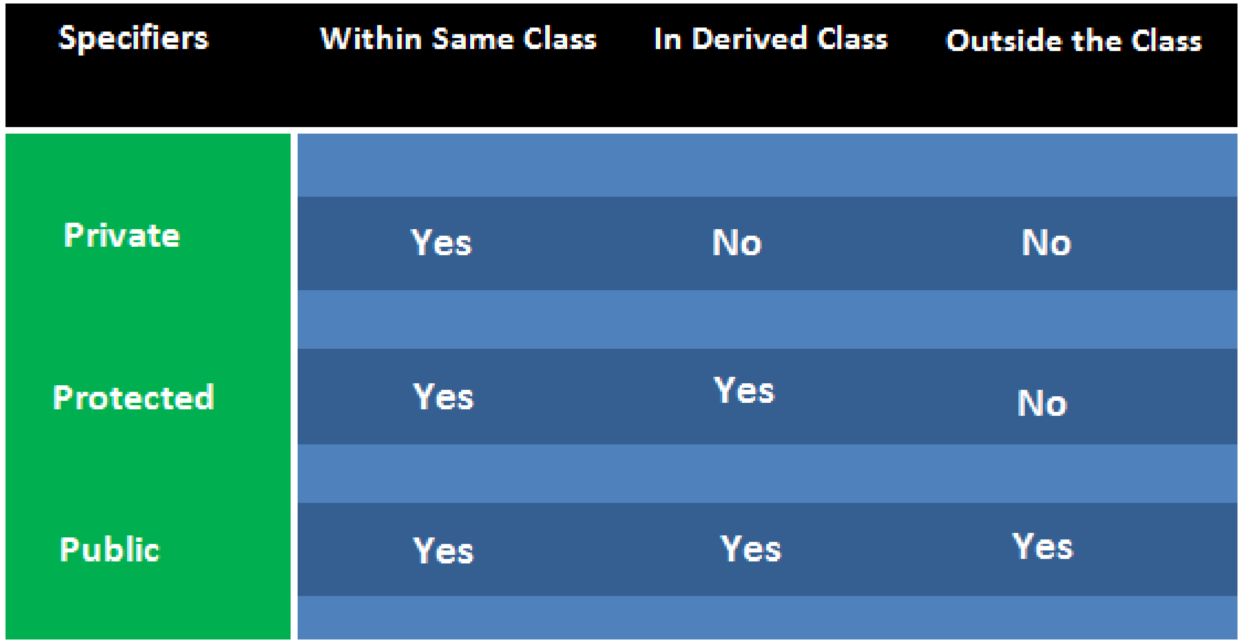
Example:

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


class Encapsulation
{
   private:
      // data hidden from outside world
      int num;


   public:
      // function to set value of
      // variable x
      void set(int a)
      {
         num =a;
      }
}// function to return value of
      // variable x
      int get()
      {
         return num;
};
// main function
int main()
{
   Encapsulation obj;
   obj.set(5);
   cout<<obj.get();
   return 0;
}
```

Access Specifiers

- Public
- Private
- Protected

✓In C++, class members are considered **private** when no access modifier is used



Specifiers	Within Same Class	In Derived Class	Outside the Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

public **Access Modifier**

- The **public** keyword is used to create public members (data and functions).
- The public members are accessible from any part of the program.

The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

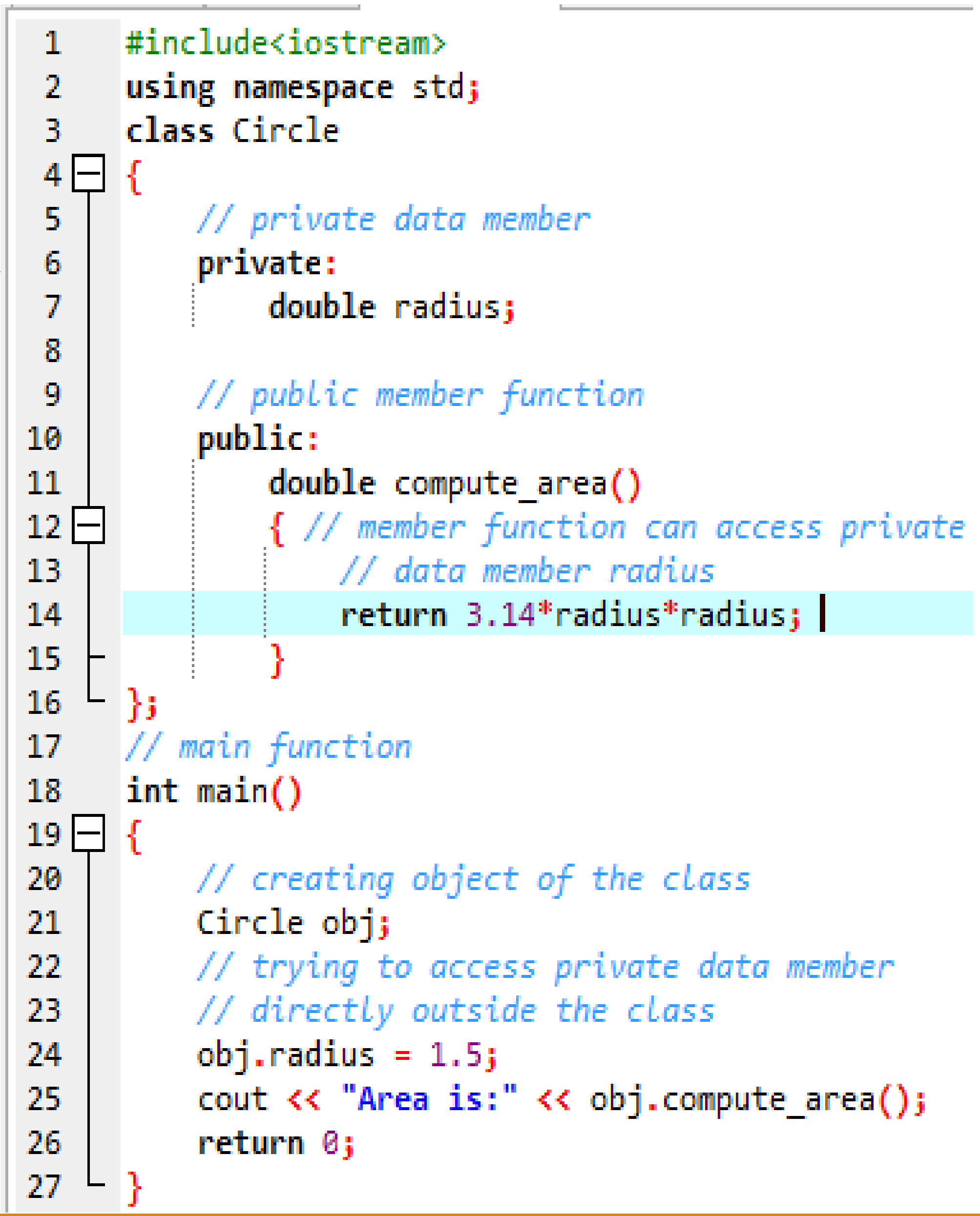
```
class MyClass {  // The class
  public:        // Access specifier
    // class members goes here
};
```

public Access Modifier

```
class Car{
public:
void accelerate()
{ \\ logic for acceleration }
 void brake()
 { \\ logic for brakes }
};
int main()
{
          Car car;
          car.accelerate();
}
```

private Access Modifier

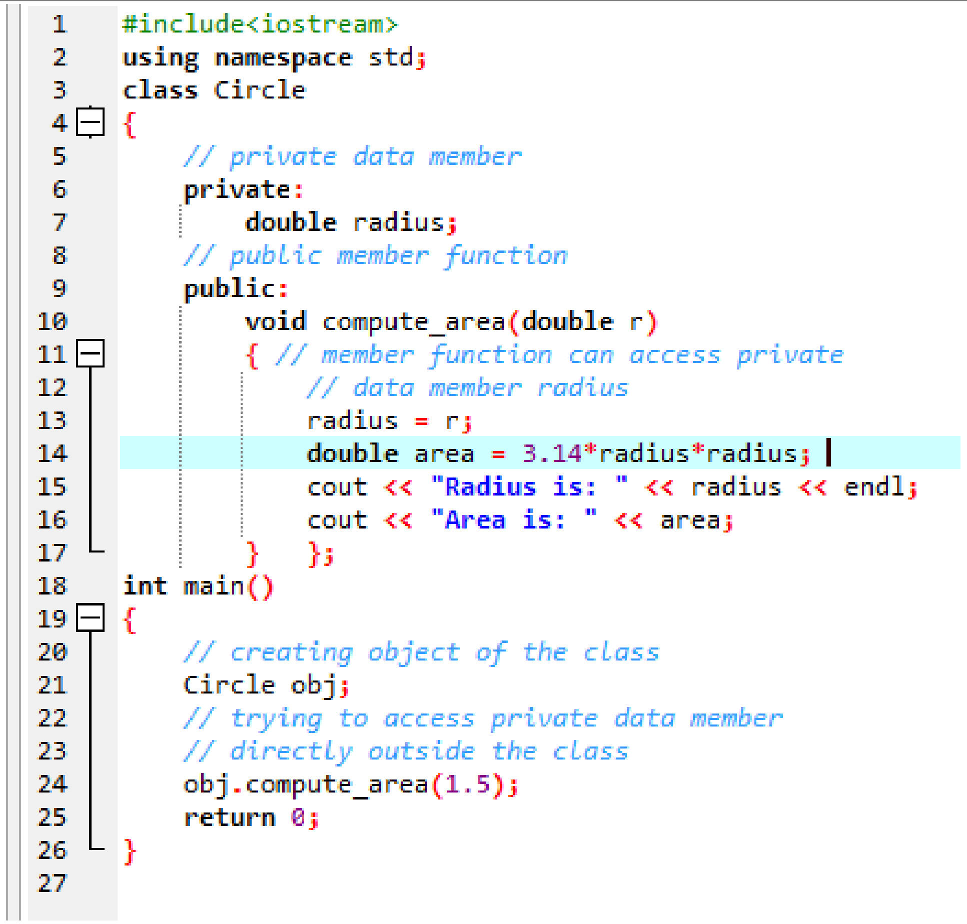
The class members declared as *private* can be accessed only by the functions inside the class. They are not allowed to be accessed directly by any object or function outside the class.



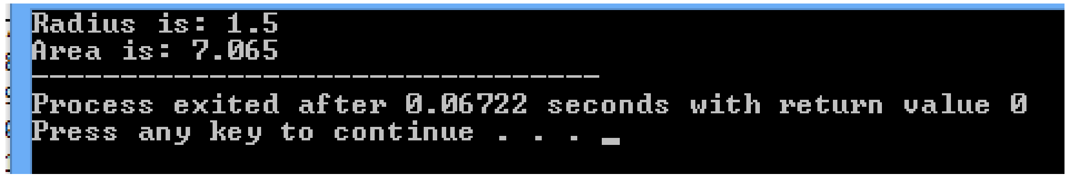
```cpp
#include<iostream>
using namespace std;
class Circle
{
    // private data member
    private:
        double radius;

    // public member function
    public:
        double compute_area()
        { // member function can access private
            // data member radius
            return 3.14*radius*radius;
        }
};
// main function
int main()
{
    // creating object of the class
    Circle obj;
    // trying to access private data member
    // directly outside the class
    obj.radius = 1.5;
    cout << "Area is:" << obj.compute_area();
    return 0;
}
```

The output of above program will be a compile time error because we are not allowed to access the private data members of a class directly outside the class.

private Modifier



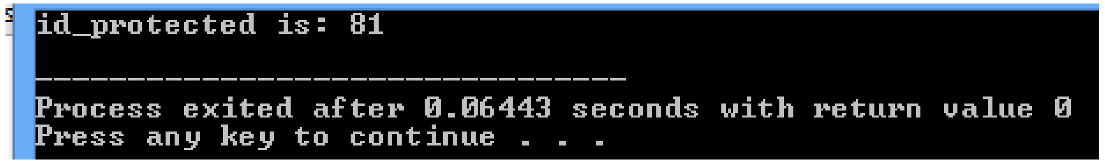
```
#include<iostream>
using namespace std;
class Circle
{
    // private data member
    private:
        double radius;
    // public member function
    public:
        void compute_area(double r)
        { // member function can access private
            // data member radius
            radius = r;
            double area = 3.14*radius*radius;
            cout << "Radius is: " << radius << endl;
            cout << "Area is: " << area;
        }   };
int main()
{
    // creating object of the class
    Circle obj;
    // trying to access private data member
    // directly outside the class
    obj.compute_area(1.5);
    return 0;
}
```



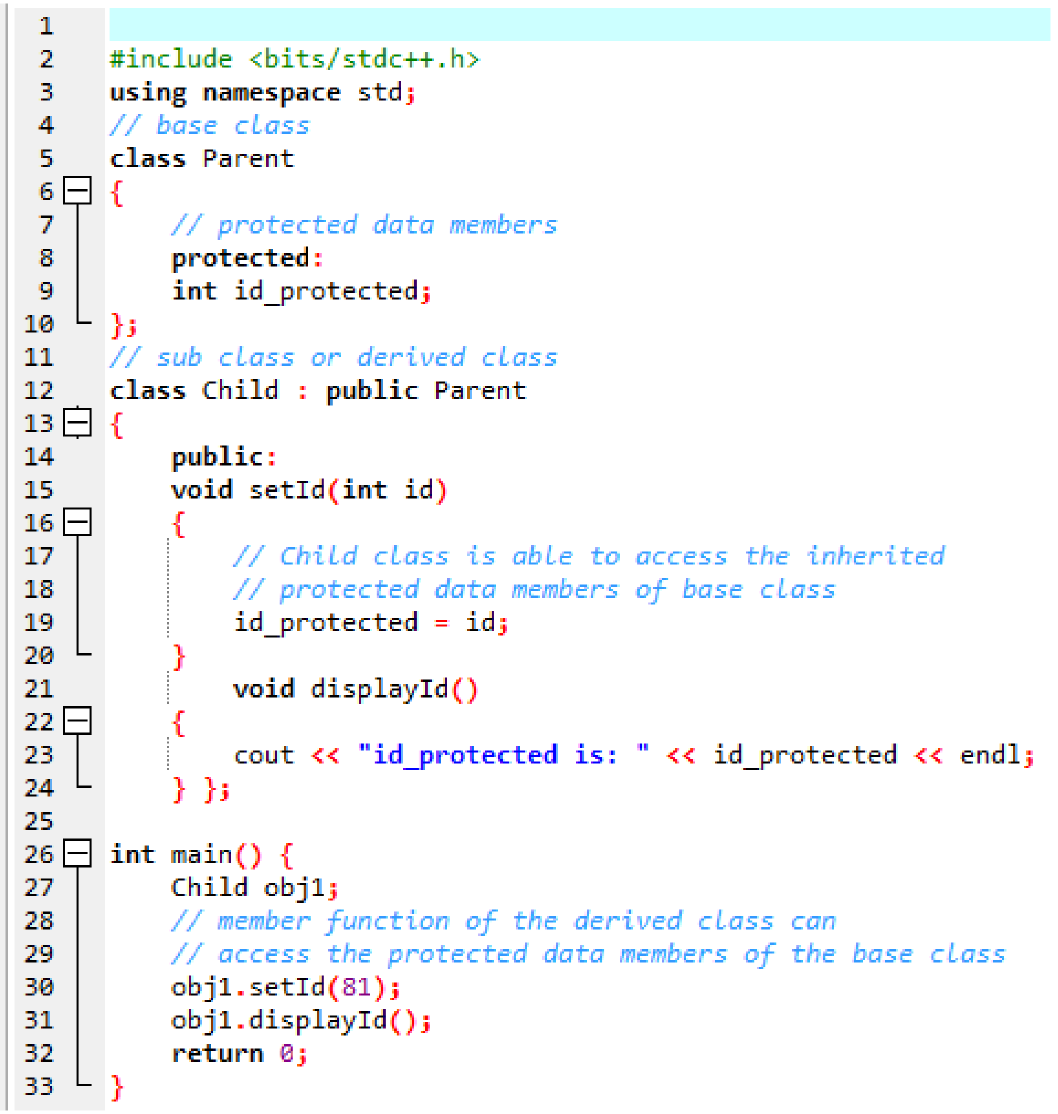
```
Radius is: 1.5
Area is: 7.065
--------------------------------
Process exited after 0.06722 seconds with return value 0
Press any key to continue . . .
```

protected-Modifier

Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any subclass(derived class) of that class.



```
id_protected is: 81

--------------------------------
Process exited after 0.06443 seconds with return value 0
Press any key to continue . . .
```



```

#include <bits/stdc++.h>
using namespace std;
// base class
class Parent
{
    // protected data members
    protected:
    int id_protected;
};
// sub class or derived class
class Child : public Parent
{
    public:
    void setId(int id)
    {
        // Child class is able to access the inherited
        // protected data members of base class
        id_protected = id;
    }
        void displayId()
    {
        cout << "id_protected is: " << id_protected << endl;
    } };

int main() {
    Child obj1;
    // member function of the derived class can
    // access the protected data members of the base class
    obj1.setId(81);
    obj1.displayId();
    return 0;
}
```