

Class and Object

A *class* is a prototype (template) from which objects are created

An *object* is a software bundle of related state and behavior

Student

has

Last name
First name
Age
List of courses

can

Pass an exam
Enroll to course

student1

Last name - Petrenko
First name - Ostap
Age - 19
List of courses – Java, MQC

student2

Last name - Romaniv
First name - Maryna
Age - 21
List of courses – Java, ATQC

Class Student

```
Int RollNo;  
String Name;  
  
SetRoll();  
DispRoll();  
SetName();  
DispName();
```

Object



class Box

```
{  
    private:  
        // data members  
    public:  
        // member functions  
};
```

← Class

Box b1

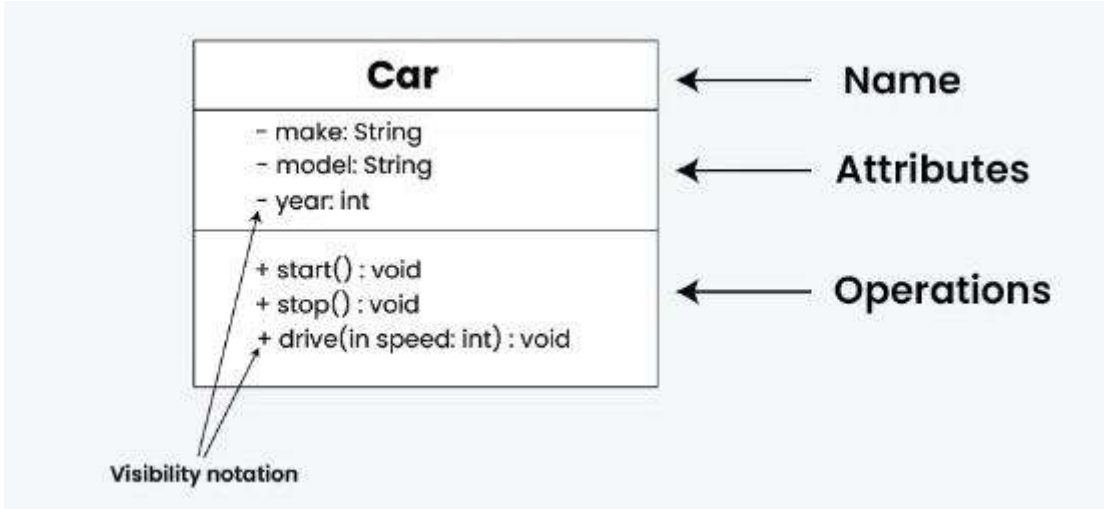
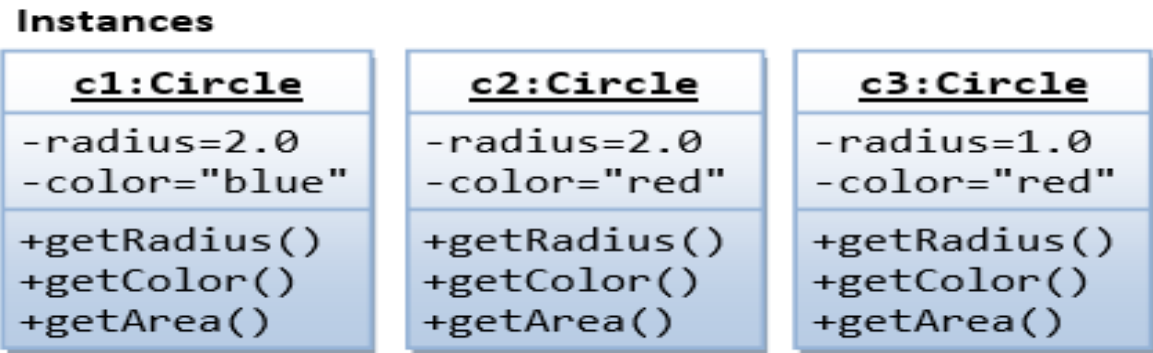
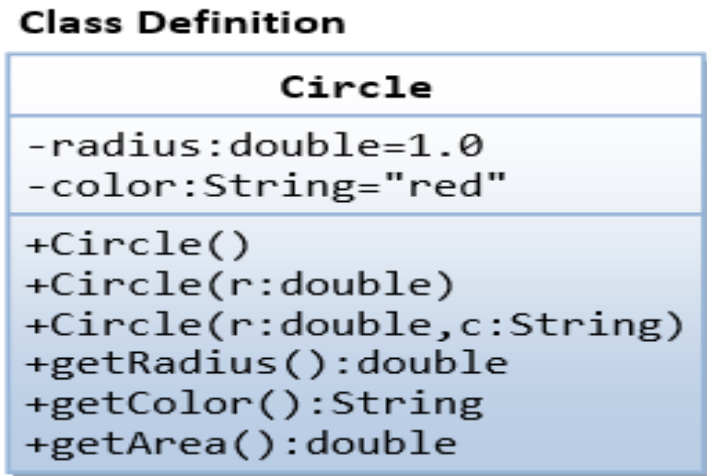
Box b2

Box b3

← Objects

Unified Modeling Language (UML) – Class Diagram

Class diagrams are a type of UML diagram used in software engineering to visually represent the structure and relationships of classes in a system. UML is a standardized modeling language that helps in designing and documenting software systems. They are an integral part of the software development process, helping in both the design and documentation phases.



Access Modifiers

- Access modifiers are used to control access to class members
- **public**, **private** & **protected** are three of the access modifiers available in C++
- In C++, class members are considered **private** when no access modifier is used

Getter/Setter Functions

- Getter functions (or accessor functions) are used to read value of a private member of some class
- Setter functions (or mutator functions) are used to modify the value of a private member of some class

Example – Getter Function

```
class BankAccount
{
    int PIN;          //private variable

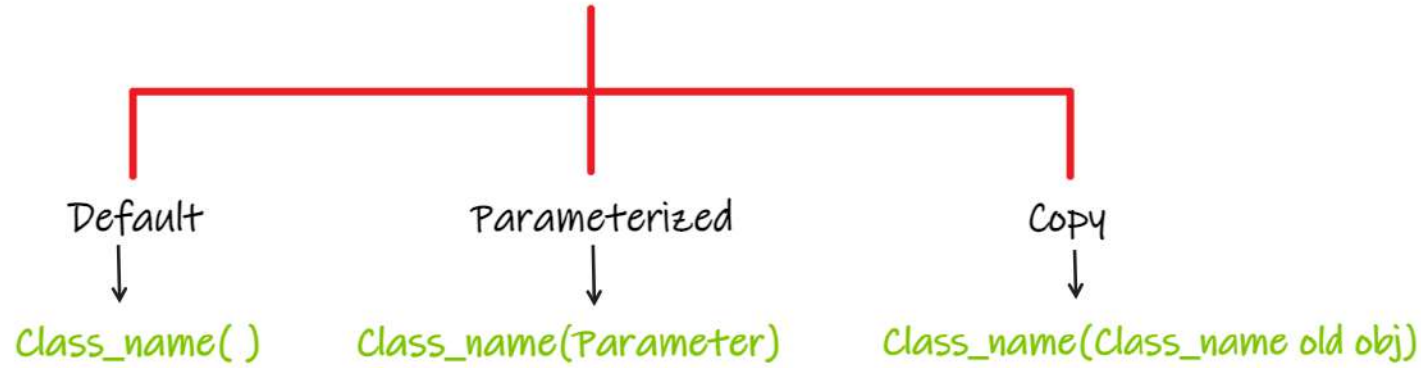
    int get_PIN() const
    {
        return PIN;
    }
}
```

Example – Setter Function

```
class BankAccount
{
    int accountNo;    //private variable

    void set_accountNo(int num)
    {
        accountNo = num;
    }
}
```

C++ Constructor



Constructor & Destructor in C++

```
class Cube
{
    int side;
public:
    Cube() // constructor
    {
        cout<<"Constructor Called";
    }
    ~Cube() // destructor
    {
        cout<<"Destructor Called";
    }
};
```

```

1  #include<iostream>
2  using namespace std;
3  class circle{
4      float radius;
5      public:
6      circle(){
7          radius=0;
8          cout<<"The default constructor is invoked"<<endl;
9      }
10     circle(float r=0):radius(r){
11         cout<<"The parametrized constructor is invoked"<<endl;
12     }
13     ~circle(){
14         cout<<"Destructor is invoked"<<endl;
15     }
16     float getRadius(){
17         return radius;
18     }
19     float Area(){
20         return 3.14*radius*radius;
21     }
22     float circumference(){
23         return 2*3.14*radius;
24     }
25 };

```

Diagram annotations for the class definition:

- Data Members:** points to `float radius;`
- Access Specifier:** points to `public:`
- Default Constructor:** points to the `circle()` constructor block.
- Parametrized Constructor:** points to the `circle(float r=0):radius(r)` constructor block.
- Destructor:** points to the `~circle()` destructor block.
- Getter function:** points to the `float getRadius()` method block.
- Class - Blueprint or template:** points to the entire `class circle{ ... };` definition.

```

27 int main(){
28     circle c1;
29     cout<<"Radius is : "<<c1.getRadius()<<endl;
30     cout<<"The area of the circle is : "<<c1.Area()<<endl;
31     circle c2(10);
32 }

```

Diagram annotations for the main function:

- object created by invoking default constructor:** points to `circle c1;`
- Accessing public methods of the object:** points to the `c1.getRadius()` and `c1.Area()` calls.
- object created using parametrized constructor:** points to `circle c2(10);`