Class and Objects

Prof. Zankhana Dabhi
Department of Information Technology,
D. D. University, Nadiad.

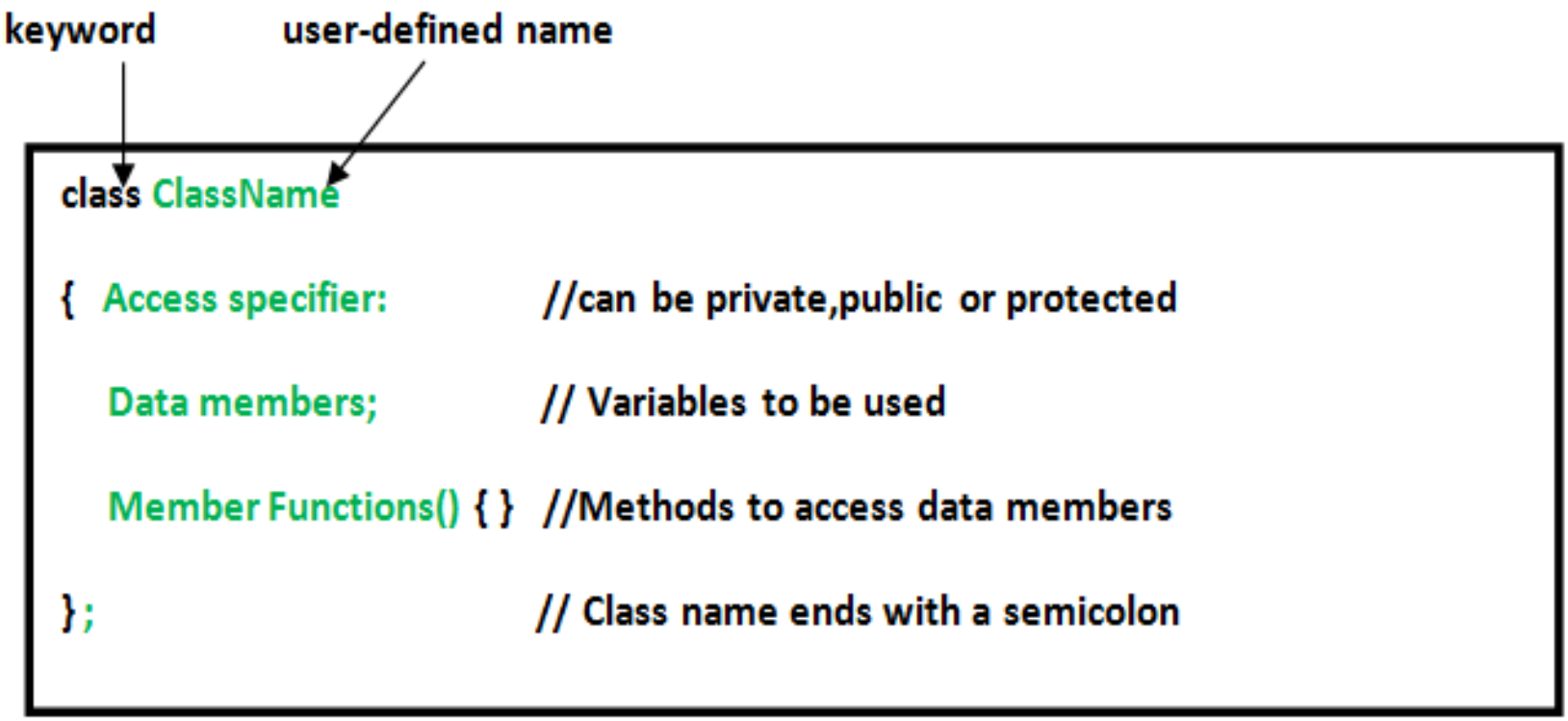
Class

- A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one unit.
- When you define a class, you define a blueprint for a data type.

Object

- An Object is an instance of a Class.
- When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Syntax of defining class



```
class student
{
	private:
	int roll_no;
	char name[20];
	public:
	void setdata(int r,char *n)
	{
		roll_no=r;
		strcpy(name,n);
	}
	void getdata()
	{
		cout<<"enter data:";
		cin>>roll_no>>name;
	}
	void showdata()
	{
		cout<<"roll no ="<<roll_no<<endl;
		cout<<"name is:"<<name<<endl;
	}
};
int main()
{
	student s1,s2;
	s1.setdata(10,"geeta");
	s2.getdata();
	s1.showdata();
	s2.showdata();
}
```

- Member functions defined outside class body

```
class student
{
        private:
        int roll_no;
        char name[20];
        public:
        void setdata(int,char*);
        void getdata();
        void showdata();
};
void student :: setdata(int r,char *n)
{
        roll_no=r;
        strcpy(name,n);
}
void student :: getdata()
{
        cout<<"enter data:";
        cin>>roll_no>>name;
}
void student :: showdata()
{
        cout<<"roll no ="<<roll_no<<endl;
        cout<<"name is:"<<name<<endl;
}
```

```
int main()
{
                student s1,s2;
                s1.setdata(10,"geeta");
                s2.getdata();
                s1.showdata();
                s2.showdata();
}
```

- Member functions defined as inline:

```
class student
{
        private:
        int roll_no;
        char name[20];
        public:
        void setdata(int,char*);
        void getdata();
        void showdata();
};
inline void student :: setdata(int r,char *n)
{
        roll_no=r;
        strcpy(name,n);
}
inline void student :: getdata()
{
        cout<<"enter data:";
        cin>>roll_no>>name;
}
void student :: showdata()
{
        cout<<"roll no ="<<roll_no<<endl;
        cout<<"name is:"<<name<<endl;
}
```

```
int main()
{
                student s1,s2;
                s1.setdata(10,"geeta");
                s2.getdata();
                s1.showdata();
                s2.showdata();
}
```

- Accessing member functions within class:

```
class Numberpairs
{
    int no1;
    int no2;
    public:
    void read()
    {
        cout<<"enter first no";
        cin>>no1;
        cout<<"enter second no";
        cin>>no2;
    }
    int max()
    {
        if(no1>no2)
            return no1;
        else
            return no2;
    }
    void show()
    {
        cout<<"maximum is="<<max();
    }
};
```

```
int main()
{
    Numberpairs n1;
    n1.read();
    n1.show();
}
```

- Passing object as an argument to function:

```
class dist
{
  private:
    int feet;
    float inches;
  public:
    void input()
    {
      cout<<"enter data";
      cin>>feet>>inches;
    }
    void display()
    {
      cout<<"distance is";
      cout<<feet<<"---"<<inches;
    }
    dist add(dist d)
    {
      dist temp;
      temp.feet=0;
      temp.inches = inches + d.inches;
      if(temp.inches >= 12.0)
      {
        temp.inches -= 12.0;
        temp.feet++;
      }
      temp.feet += feet + d.feet;
      return temp;
    }
};
```

```
int main()
{
  dist d1,d2,d3;
  d1.input();
  d2.input();
  d3=d1.add(d2);
  d3.display();
  return 0;
}
```

- Passing object as an argument to function:

```
class dist
{
 private:
  int feet;
  float inches;
 public:
  void input()
  {
   cout<<"enter data";
   cin>>feet>>inches;
  }
  void display()
  {
   cout<<"distance is";
   cout<<feet<<"---"<<inches;
  }
  void add(dist d1,dist d2)
  {
   feet=0;
   inches = d1.inches + d2.inches;
   if(inches >= 12.0)
   {
    inches -= 12.0;
    feet++;
   }
   feet += d1.feet + d2.feet;
   }
};
```

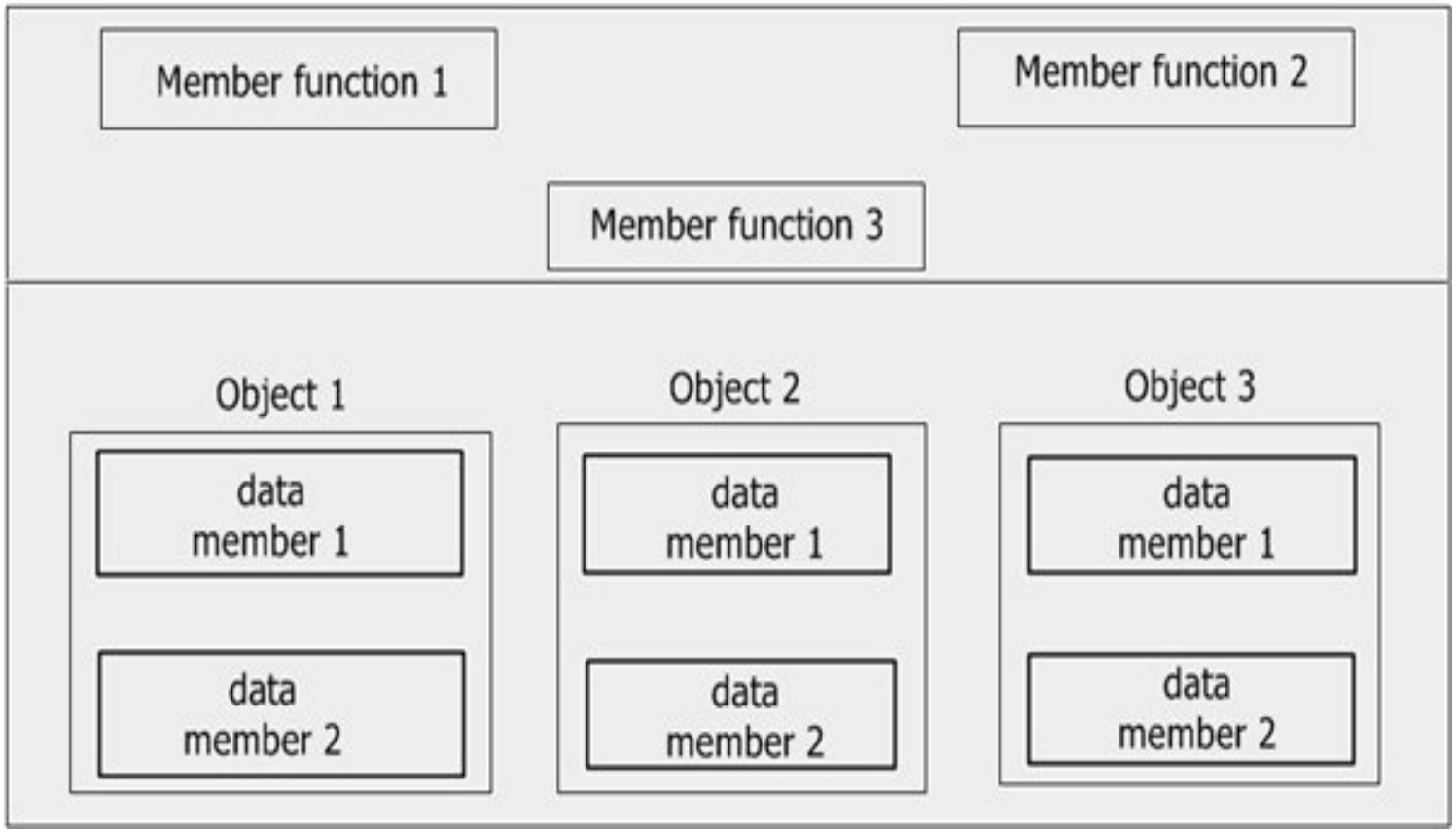
```
int main()
{
 dist d1,d2,d3;
 d1.input();
 d2.input();
 d3.add(d1,d2);
 d3.display();
 return 0;
}
```

Passing object by reference:

```
class Account
{
	int accno;
	float balance;
	public:
	void getdata()
	{
		cin>>accno>>balance;
	}
	void setdata(int a)
	{
		accno=a;
		balance=0;
	}
	void setdata(int a,float b)
	{
		accno=a;
		balance=b;
	}
	void display()
	{
		cout<<accno<<balance;
	}
	void moneytransfer(Account& acc,float amount)
	{
		balance=balance-amount;
		acc.balance=acc.balance+amount;
	}
};
```

```
int main()
{
		Account A1,A2;
		A1.getdata();
		A2.setdata(10);
		A1.moneytransfer(A2,2000);
		A2.display();
}
```

Classes, Objects and Memory



Constructors

- A constructor is a member function of the class.
- The name of the constructor is same as the name of the class.
- It has no return type, so can not use return type.
- It must be an instance member function, that is , it can never be static.

How to call Constructor?

- Constructor is implicitly invoked when an object is created.
- Constructor is used to solve the problem of initialization.

Why Constructor?

- Constructor constructs an object.
- Constructor is used to solve the problem of initialization.

Types of Constructors

- Default constructor
- Parameterized constructor
- Copy constructor(will be discussed in chap:13 of virtual functions)

Default constructor

- Default constructor is the constructor which doesn't take any argument. It has no parameters.

```
#include <iostream>
using namespace std;
class construct {
public:
	int a, b;

	// Default Constructor
	construct()
	{
		a = 10;
		b = 20;
	}
};
int main()
{
	construct c;
	cout << "a: " << c.a << endl
		<< "b: " << c.b;
	return 1;
}
```

Parameterized constructor

Parameterized constructor is used to initialize an object when it is created

```
#include <iostream>
using namespace std;
class Point {
private:
	int x, y;
public:
	Point(int x1, int y1)
	{
		x = x1;
		y = y1;
	}
	int getX()
	{
		return x;
	}
	int getY()
	{
		return y;
	}
};
int main()
{
	Point p1(10, 15);
	cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
	return 0;
}
```

Constructor Overloading

- Overloaded constructors essentially have the same name (name of the class) and different number of arguments.

```
class construct
{
public:
    int a,b;
    construct()
    {
        a = 0;
        b=0;
    }
    construct(int a1)
    {
        a=a1;
        b=0;
    }
    construct(int a1, int b1)
    {
        a=a1;
        b=b1;
    }
    void disp()
    {
        cout<< a<< endl<<b<<endl;
    }
};
```

```
int main()
{
    construct  o;
    construct o1(5);        //  construct o1=5;
    construct  o2(10, 20);
    o.disp();
    o1.disp();
    o2.disp();
}
```

Find Output

```
#include<iostream>
using namespace std;
class Point {
    Point() { cout << "Constructor called"; }
};

int main()
{
   Point t1;
   return 0;
}
```

Find Output

```
#include<iostream>
using namespace std;
class Point {
    public:
        Point() { cout << "Constructor called"; }
};

int main()
{
  Point t1, *t2;
  return 0;
}
```

Find Output

```
#include <iostream>
using namespace std;
class Test
{
    public:
        Test() { cout << "Hello from Test() "; }
} a;

int main()
{
   cout << "Main Started ";
   return 0;
}
```

Find Output

```
#include <iostream>
using namespace std;
class Test
{
      private:
         int x;
      public:
         Test(int i)
         {
            x = i;
                cout << “Hello" << endl;
         }
};
 int main()
{
    Test t(20);
    t = 30;
    return 0;
}
```

Find output

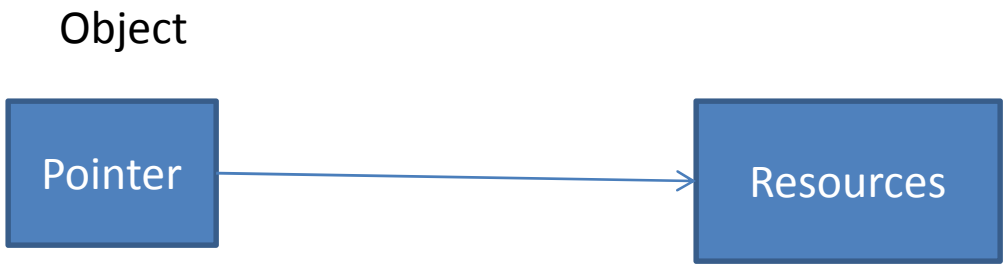
```
#include <iostream>
using namespace std;
class Point
{
        int x, y;
public:
        Point(int i = 0, int j = 0) { x = i; y = j; }
        int getX() { return x; }
        int getY() { return y; }
};

int main()
{
   Point p1;
   Point p2 = p1;
   cout << "x = " << p2.getX() << " y = " << p2.getY();
   return 0;
}
```

Destructor

- Destructor is a member function of the class.
- Its name is same as name of the class but preceded by tilde(~) symbol.
- Destructor has no return type.
- Destructor takes no argument.
- A destructor function is called automatically when the object goes out of scope.

Why Destructor?

- It should be defined to release resources allocated to an object.



Example

```
class test
{
    public:
    ~test()
     {
          cout<<“destructor called”;
     }
};
void fun()
{
    test t1;
}
int main()
{
    fun();
    getch();
}
```

Static class data

- When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.
- All static data is initialized to zero, if no other initialization is present.

Contd..

- Declared inside the class body.
- They must be defined outside the class.
- Static member variable does not belong to any object, but to the whole class.
- They can also be used with class name.

Example

```
class test
{
	static int count;
	public:
	test()
	{
		count++;
	}
	int get_count()
	{
		return count;
	}
};
int test::count=0;
int main()
{
	test t1,t2,t3;
	cout<<t1.get_count();
	cout<<t2.get_count();
	cout<<t3.get_count();
}
```

Find output

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

class Player
{
private:
	int id;
	static int next_id;
public:
	int getID() { return id; }
	Player() { id = next_id++; }
};
int Player::next_id = 1;

int main()
{
	Player p1;
	Player p2;
	Player p3;
	cout << p1.getID() << " ";
	cout << p2.getID() << " ";
	cout << p3.getID();
	return 0;
}
```

Static Member Function

- They are qualified with the keyword static.
- They are also called as class member functions.
- They can be invoked with or without object.
- They can only access static members of the class.

Example

```
class Account
{
    float balance;
    static float roi;
    public:
    void setbalance(float b)
    {
            balance=b;
    }
};
float Account:: roi = 3.5;
int main()
{
    Account a1;
}
```

Example

```
class Account
{
    float balance;
    static float roi;
    public:
    void setbalance(float b)
    {
            balance=b;
    }
};
float Account:: roi = 3.5;
int main()
{
    Account :: roi;         //  error:  roi  is private
}
```

Example

```
class Account
{
    float balance;
    static float roi;
    public:
    void setbalance(float b)
    {
        balance=b;
    }
    void setRoi(float r)
    {
        roi=r;
    }
};
float Account:: roi = 3.5;
int main()
{
    Account a1;
    a1.setRoi(4.5);
}
```

Example

```
class Account
{
    float balance;
    static float roi;
    public:
    void setbalance(float b)
    {
        balance=b;
    }
    static void setRoi(float r)
    {
        roi=r;
    }
};
float Account:: roi = 3.5;
int main()
{
    Account :: setRoi(4.5);
}
```

Find output

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

class Test
{
      static int x;
      public:
         Test() { x++; }
         static int getX() {return x;}
};

int Test::x = 0;

int main()
{
   cout << Test::getX() << " ";
   Test t[5];
   cout << Test::getX();
}
```

Find output

```
#include <iostream>
class Test
{
public:
   void fun();
};
static void Test::fun()
{
   std::cout<<"fun() is staticn";
}
int main()
{
   Test::fun();
   return 0;
}
```

Const Member functions

- A const member function guarantees that it will never modify any of its class's member data.
- A function is made into a constant function by placing the keyword const after the declarator but before the function body.

Example

```
class aClass
{
    private:
    int alpha;
    public:
    void nonFunc()              //non-const member function
    { alpha = 99; }              //OK
    void conFunc() const        //const member function
    { alpha = 99; }             //ERROR: can't modify alpha
};
```

```
class Distance
{
        private:
        int feet;
        float inches;
        public:
        Distance() : feet(0), inches(0.0) { }
        Distance(int ft, float in) : feet(ft), inches(in)  { }
        void getdist()
        {
        cout << "\nEnter feet: "; cin >> feet;
        cout << "Enter inches: "; cin >> inches;
        }
        void showdist() const
        { cout << feet << "\'-" << inches << '\"'; }
        Distance add_dist(const Distance&) const;
};
Distance Distance::add_dist(const Distance& d2) const
{
        Distance temp;
        // feet = 0;                //ERROR: can't modify this
        // d2.feet = 0;             //ERROR: can't modify d2
        temp.inches = inches + d2.inches;
        if(temp.inches >= 12.0)
        {
        temp.inches -= 12.0;
        temp.feet = 1;
        }
        temp.feet += feet + d2.feet;
        return temp;
}
```

const Objects

```
class Distance
{
      private:
      int feet;
      float inches;
      public:
      Distance(int ft, float in) : feet(ft), inches(in)
      { }
      void getdist()
      {
      cout << "\nEnter feet: "; cin >> feet;
      cout << "Enter inches: "; cin >> inches;
      }
      void showdist() const
      { cout << feet << "\'-" << inches << '\"'; }
};
int main()
{
      const Distance football(300, 0);
      // football.getdist();              //ERROR: getdist() not const
      cout << "football = ";
      football.showdist();                //OK

}
```