Basic Concepts of OOPs

Some key features of the Object Oriented programming

- Emphasis on data rather than procedure
- Programs are divided into entities known as objects
- Data Structures are designed such that they characterize objects
- Functions that operate on data of an object are tied together in data structures
- Data is hidden and cannot be accessed by external functions
- Objects communicate with each other through functions
- New data and functions can be easily added whenever necessary
- Follows bottom up design in program design

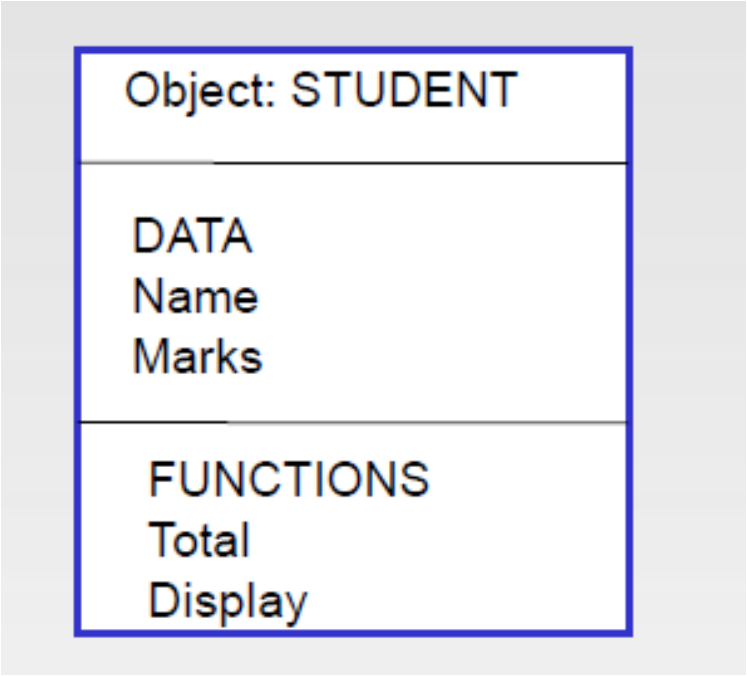
Basic Concepts of Object-Oriented Programming

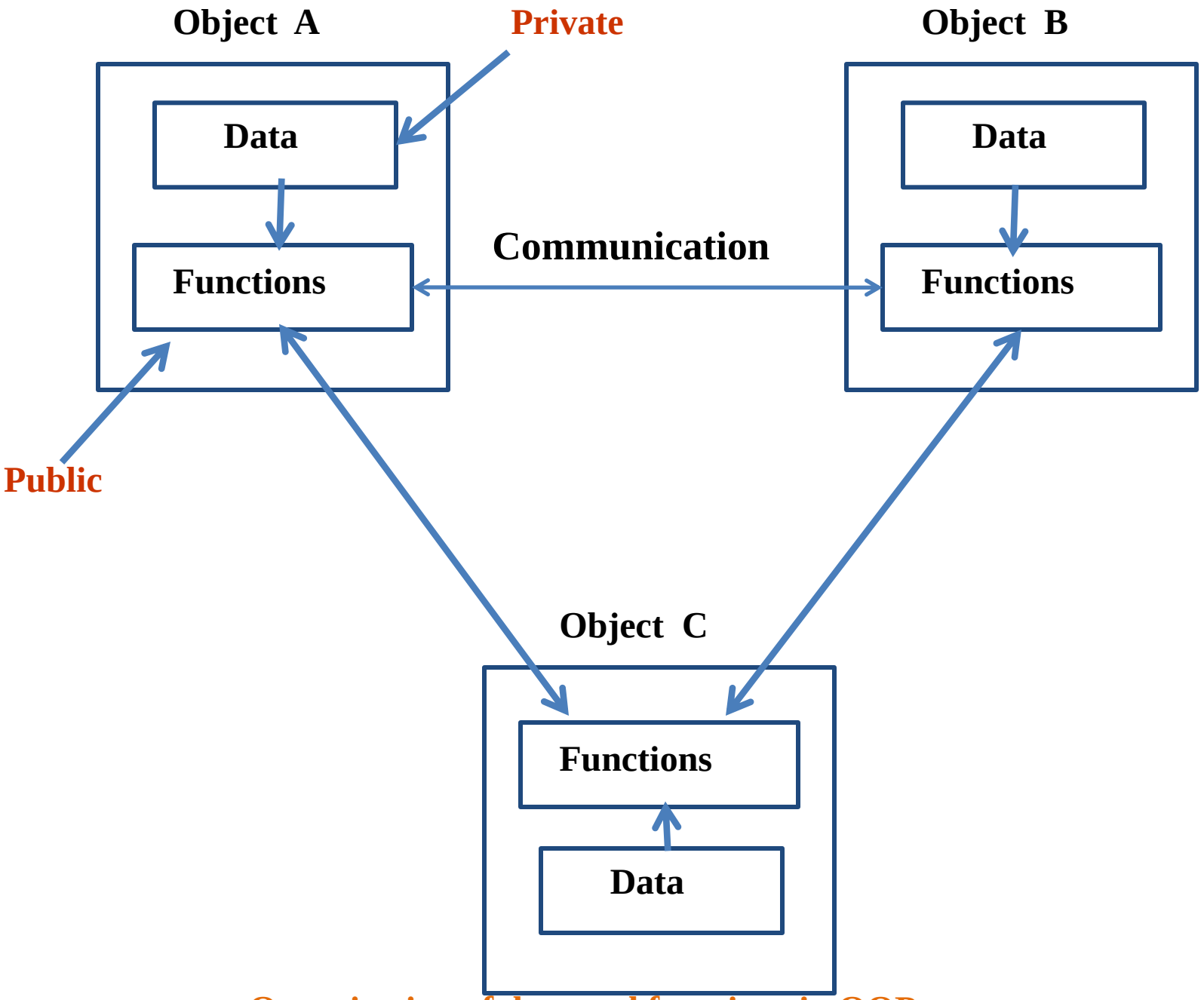
- Classes
- Objects
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

Objects

- Objects are the basic run-time entities in an object-oriented system.
- They may represent a person, a place, a bank account, a table of data or any item that the program must handle.
- Program objects should be chosen such that they match closely with the real-world objects.
- Objects take up space in the memory and have an associated address like structure in C.
- When a program is executed the objects interact by sending messages to one another.

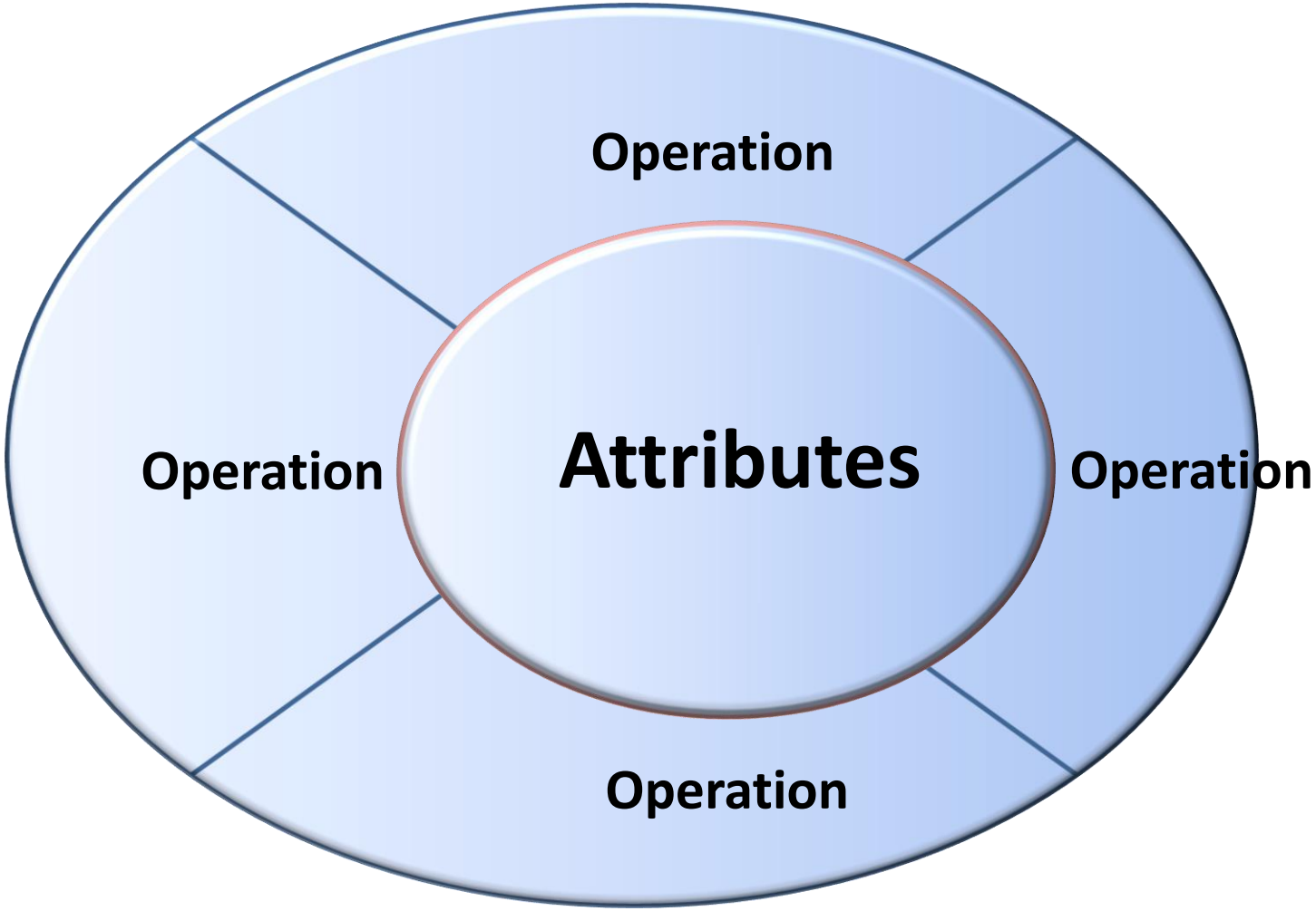
Way to present an object



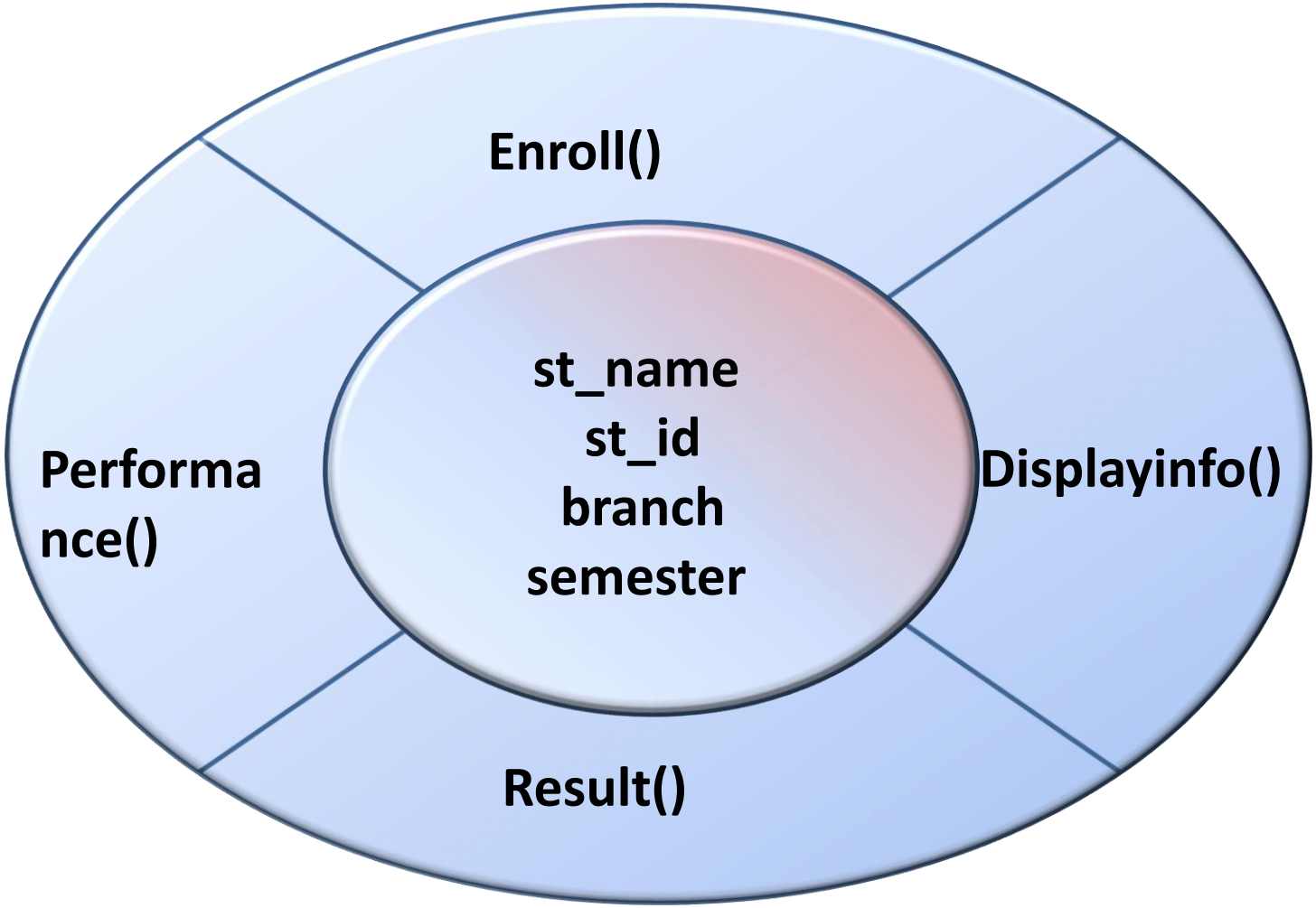


Organization of data and functions in OOP

Object

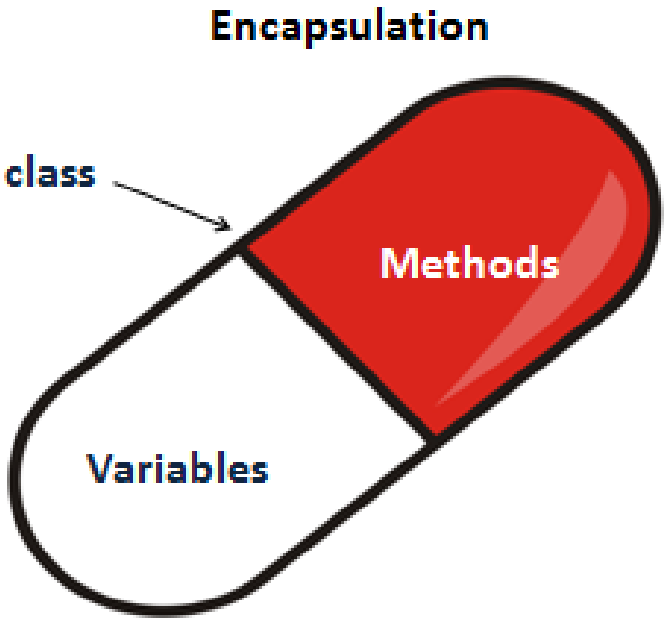


Example: StudentObject



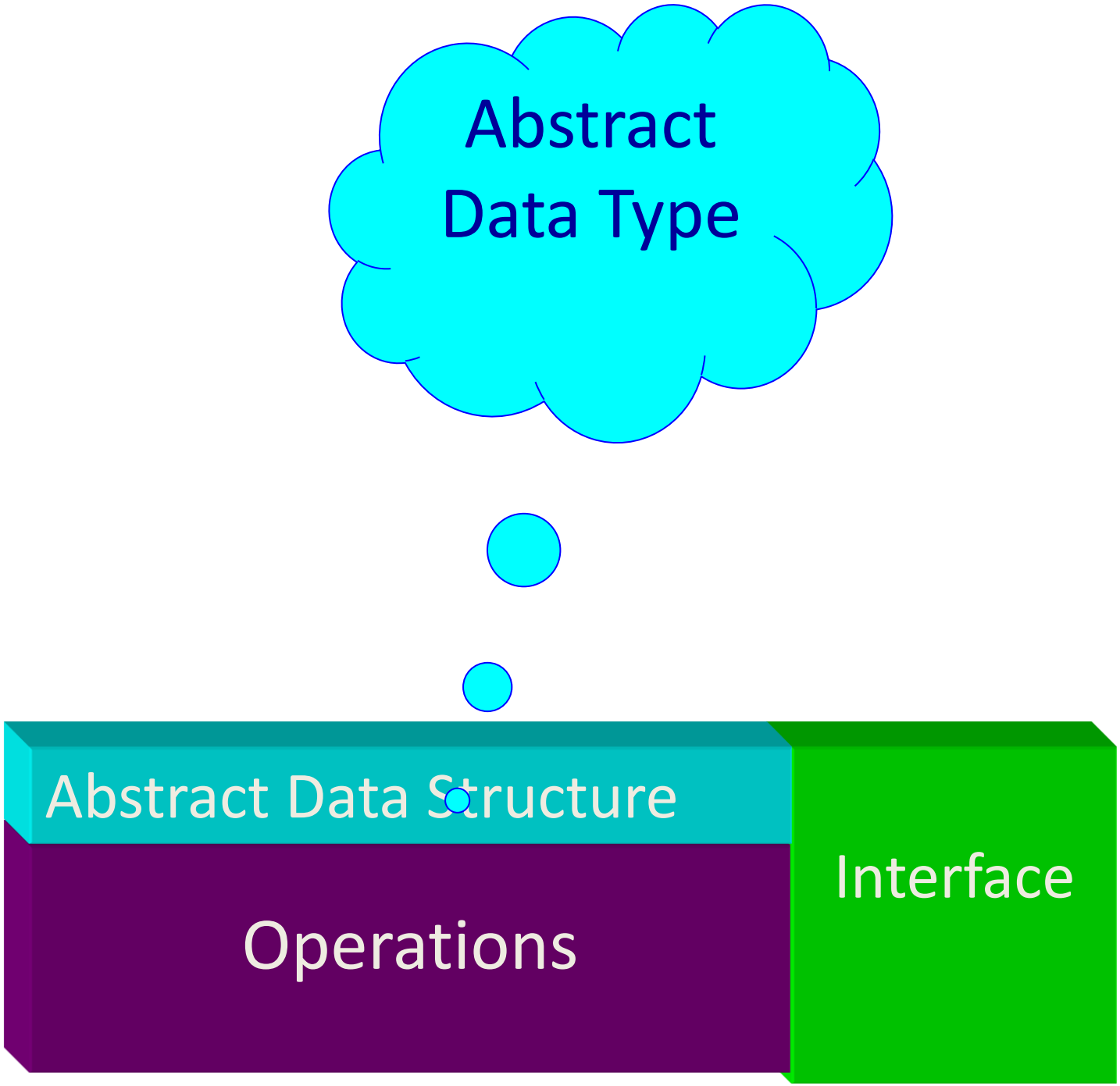
Encapsulation

- The wrapping up of data and functions into a single unit is known as Encapsulation.
- The data is not accessible to the outside world and only those functions which are wrapped in the class can access it.
- This insulation of the data from direct access by the program is called data hiding or information hiding.



Data Abstraction

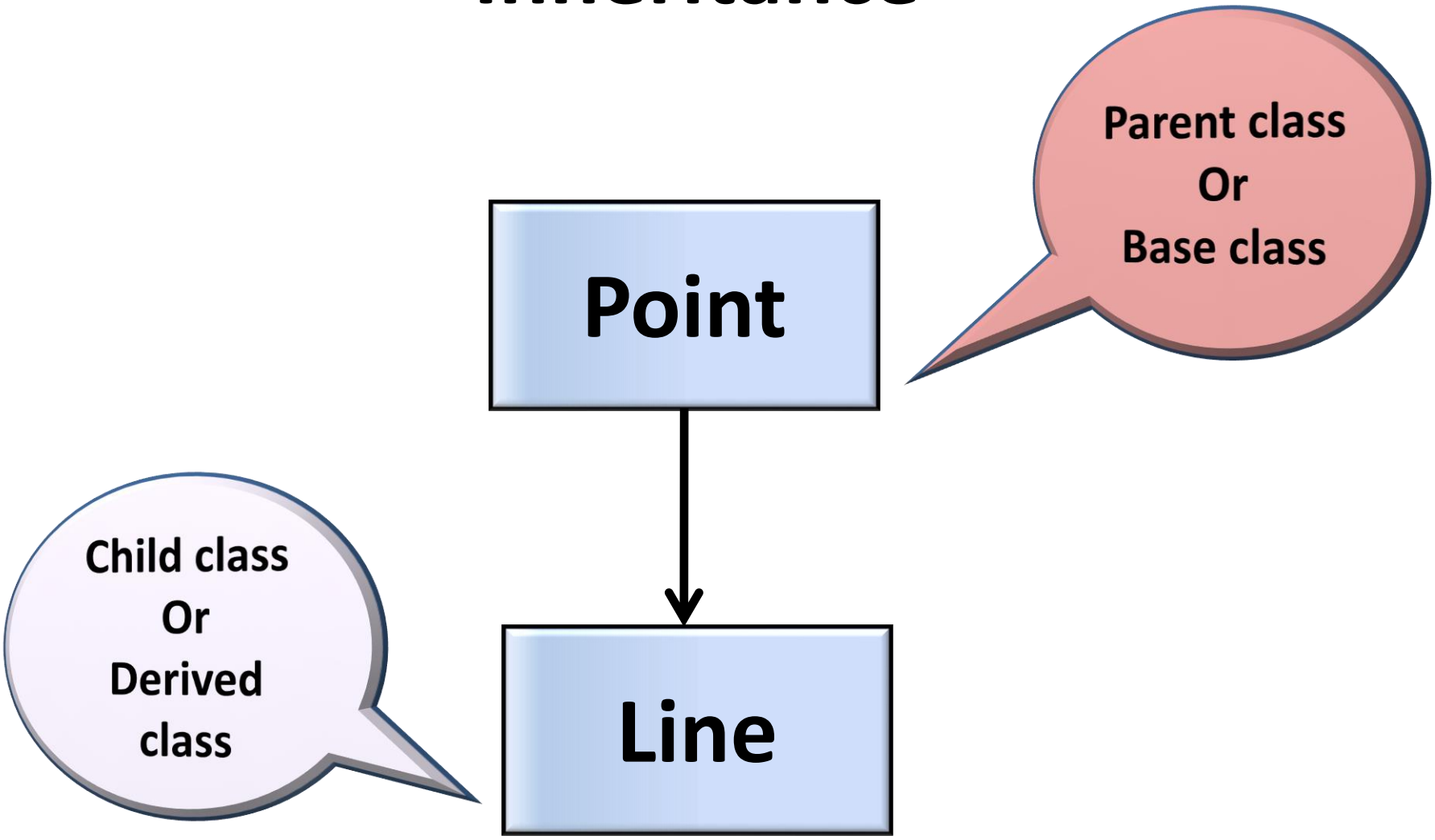
- Abstraction refers to the act of representing essential features without including the background details or explanations.
- Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost, and functions to operate on these data.
- Since the classes use the concept of data abstraction, they are known as Abstract Data Types.

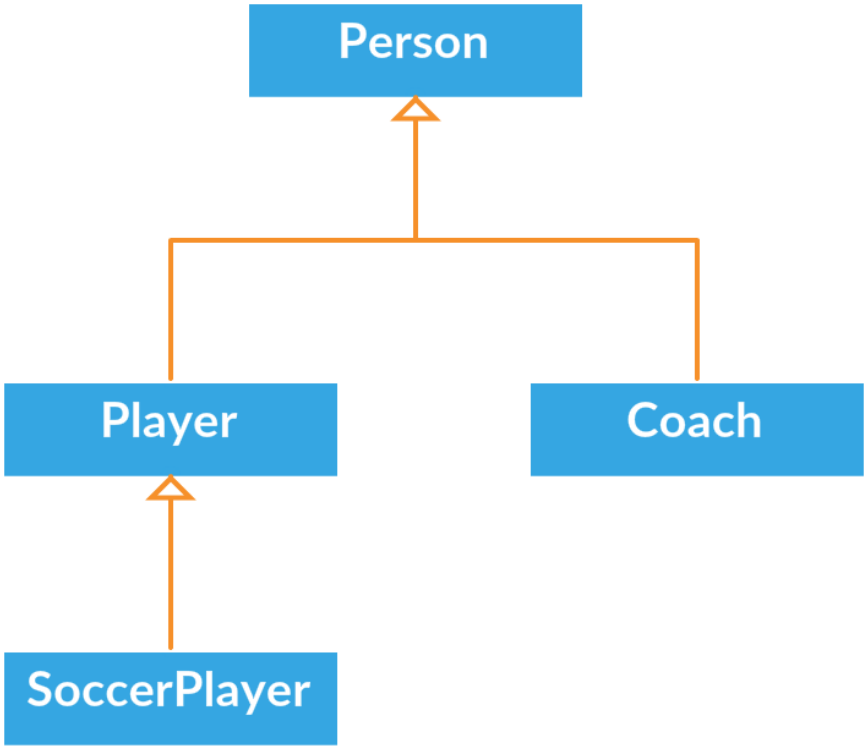

Abstract
Data Type
Abstract Data Structure
Operations
Interface

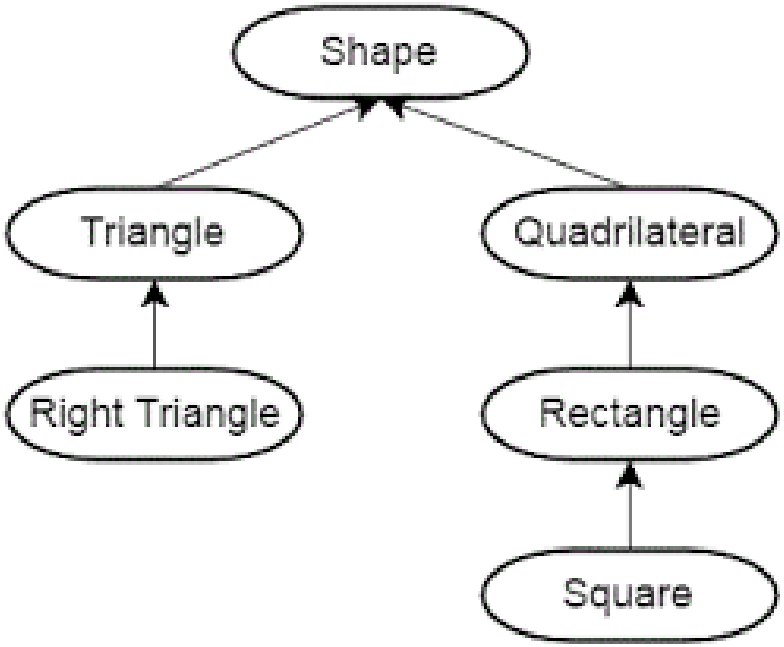
Inheritance

- Inheritance is the process by which objects of one class acquire the properties of objects of another class.
- The principle behind this sort of division is that each derived class shares common characteristics with the class from which it is derived.
- In OOP the concept of inheritance provides the idea of reusability.
- This means that we can add additional features to an existing class without modifying it.
- The real power of inheritance mechanism is that it allows the programmer to reuse a class in such a way that it does not introduce any undesirable side-effects into the rest of the classes.

Inheritance




Person
Player
Coach
SoccerPlayer


Shape
Triangle
Quadrilateral
Right Triangle
Rectangle
Square

Polymorphism

- Polymorphism means the ***ability to take more than one form.***
- That is an operation may exhibit different behaviors in different instances. The behavior depends upon the types of data used in the operation.
- Example: Operation of addition.
- For two numbers the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.

Polymorphism

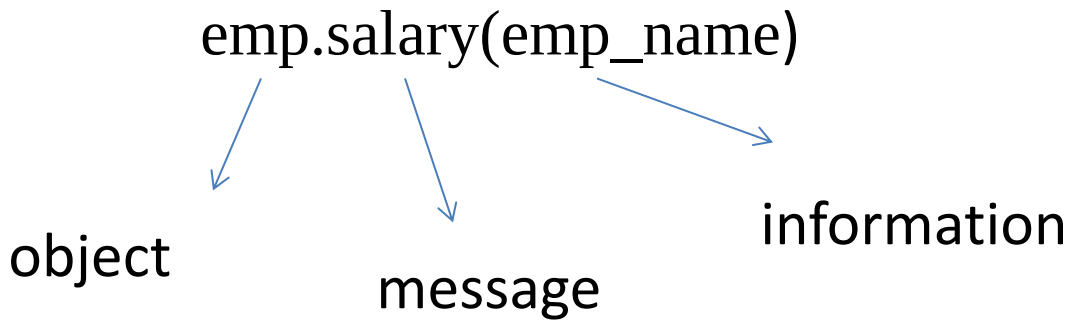
- The process of making an operator to exhibit different behaviors in different instances is known as **operator overloading.**
- Using a single function name to perform different types of tasks is known as **function overloading.**

Dynamic binding

- Binding refers to the linking of a procedure call to the code to be executed in response to the call.
- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run-time.
- It is associated with polymorphism and inheritance.

Message Communication

- An object-oriented program consist of a set of objects that communicate with each other.
- Object communicate with one another by sending and receiving information much the same way as people pass message to one another.
- Message passing involves specifying the name of the object, the name of the function (message) and the information to be sent.



Benefits of OOP

- Through inheritance we can eliminate the redundant code and extend the use of existing code.
- Data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple instances of an objects to co-exist without any interference.
- It is possible to map objects in the problem domain to those objects in the program.
- Easy to partition the work in a project based on objects.

Benefits of OOP

- Data-centered design approach enables us to capture more details of a model in implementable form.
- Object-oriented systems can be easily upgraded from small to large systems.
- Software complexity can be easily managed.

- Reusable software
- Easily modifying and extending implementations of the components without having to recode everything from the scratch
- Interacting easily with computational environment.
- Modeling the real world problem as close as possible to the users perspective.

Basic Concepts of Object-Oriented Programming

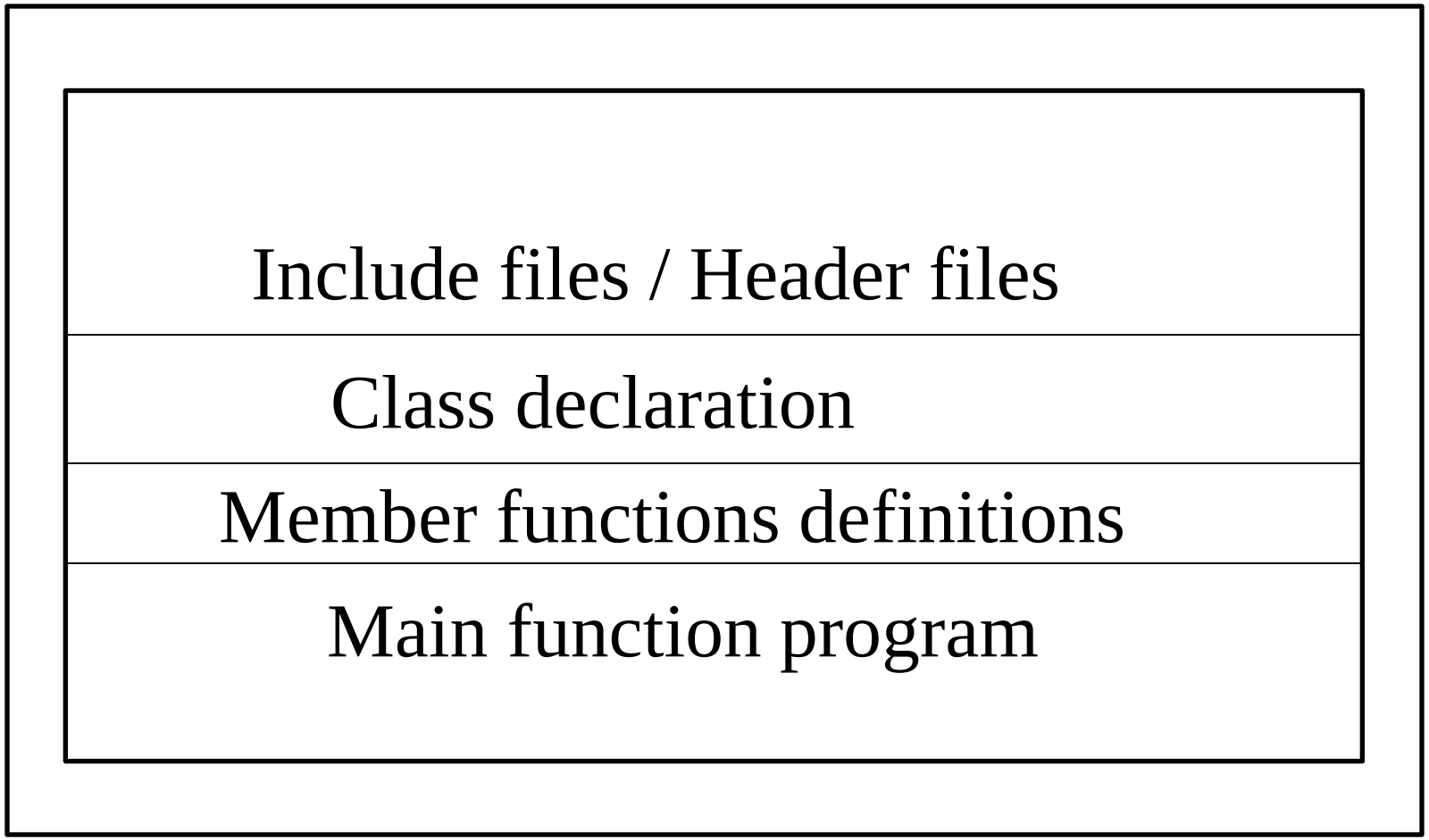
- Classes
- Objects
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

Programming elements

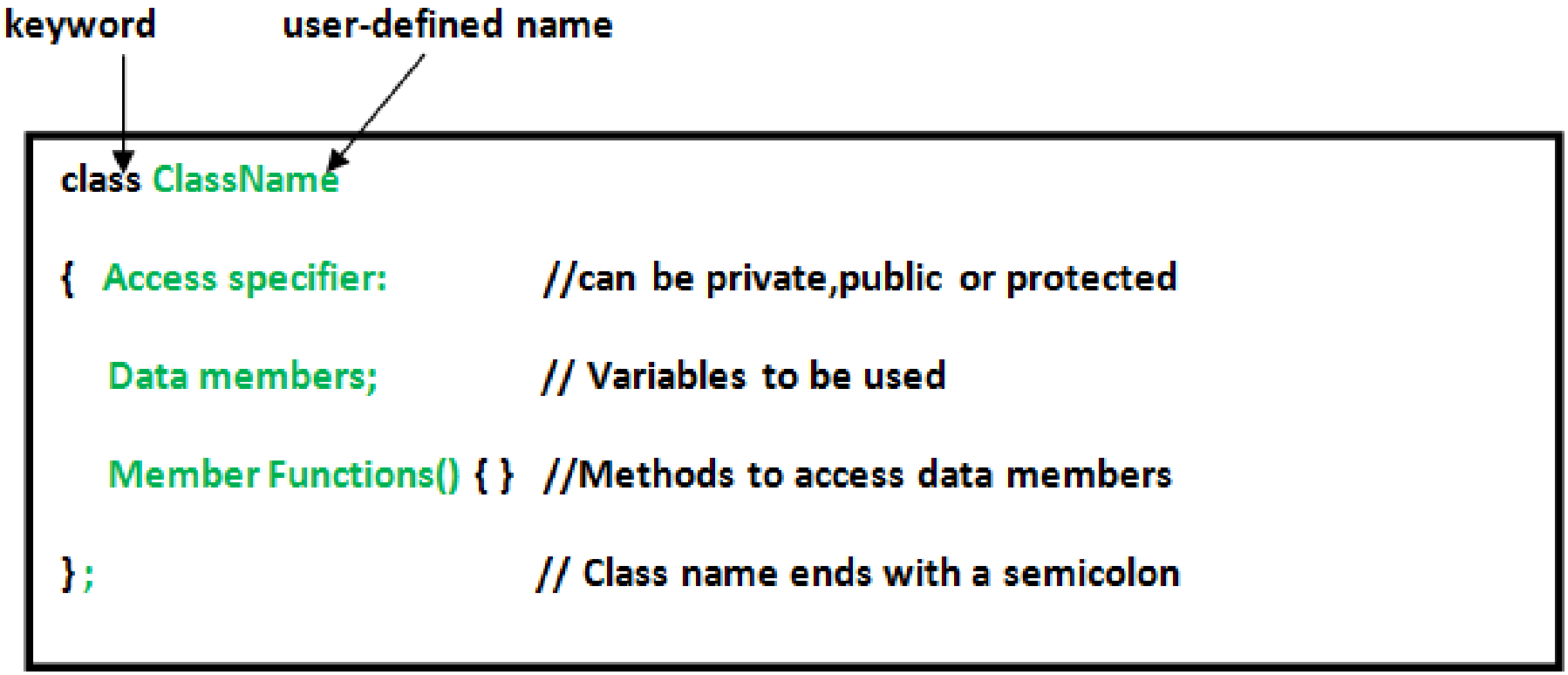
```
// Simple c++ program
#include<iostream>
using namespace std;
int main()
 {
cout<<“hello world.\n”;
return 0;
}
```

```
// Simple c++ program
#include<iostream.h>
using namespace std;
int main()
 {
    float n1,n2,sum ,avg;
    cout<<“enter two numbers:”;
    cin>>n1;
    cin>>n2;
    sum=n1+n2;
    avg=sum/2;
    cout<<“sum= “<<sum<<“\n”;
    cout<<“average=“<<avg<<“\n”;
    return 0;
}
```

Structure of C++ program

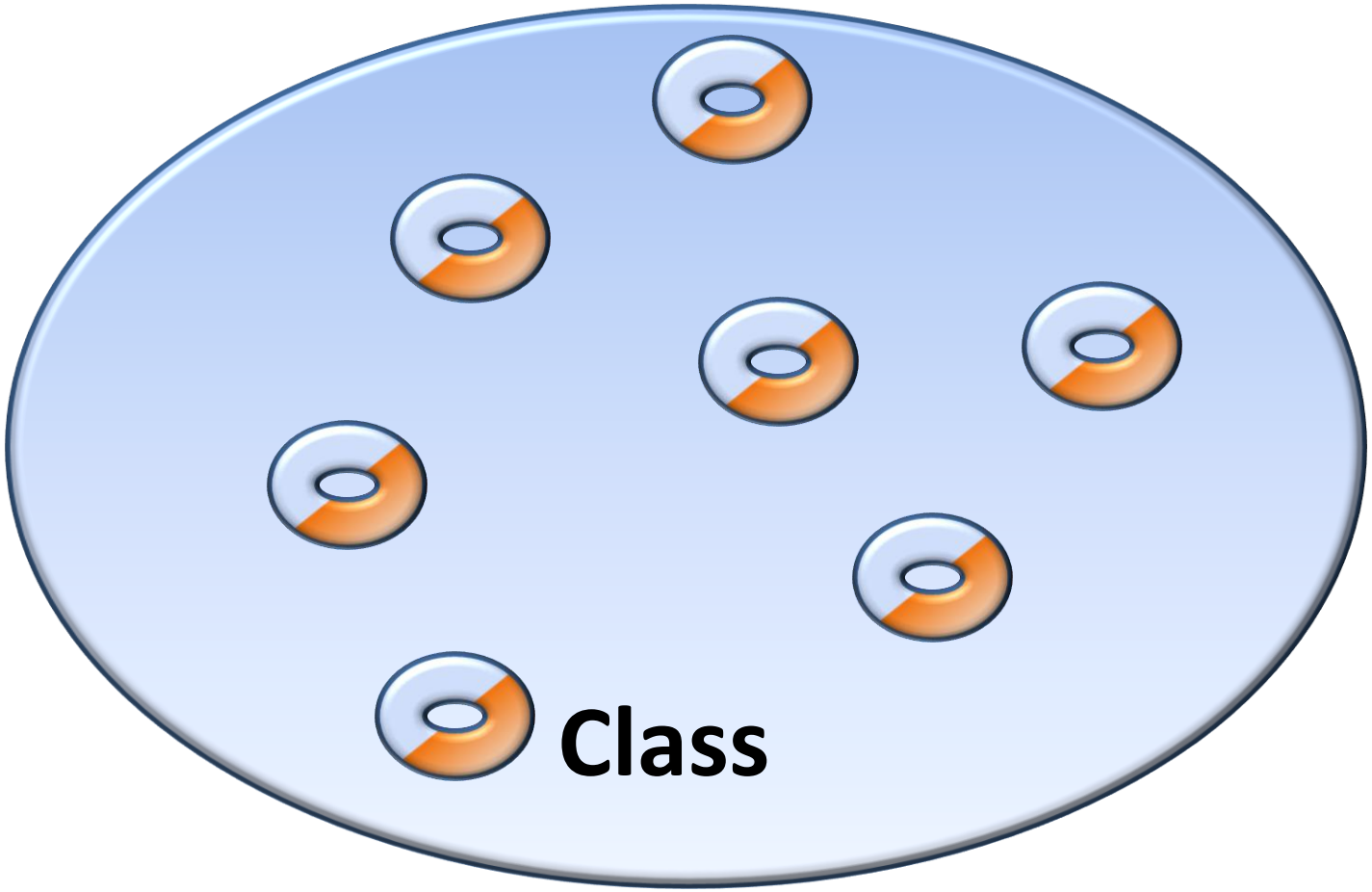


Structure of C++ Class



Class

- Class is a collection of **similar objects**.



classes

1. Classes are the basic language constructs of C++ for creating the user defined data types.
2. A class encloses both data and functions that operates on the data, in to a single unit.
3. The variables and functions enclosed in a class are called member data (data members) and member functions respectively.
4. Placing data and functions together in a single unit is the central idea of object oriented programming.

classes

```
#include<iostream>
using namespace std;
class student
{
    private :
        int id;
        char name[20];
    public :
        Void Getdata(void);
        Void display (void)
        {
            cout << id <<'\t' << name << endl;
        }
};
int main( )
{
```

Data Members

Member Functions

classes

4. Classes are syntactically an extension of structures, the difference is that, all the members of structures are public by default whereas, the members of classes are private by default.
5. Structures contains only data members whereas classes contains both data members and member functions.

classes

Difference between *class* and *struct*

By default, all data fields of a struct are public. However, by default, all data fields of a class are private.

The ***struct*** type defined in (a) can be replaced by the ***class*** defined in (b) by using ***public*** access modifier.

```
struct Student
{
  int id;
  char firstName[30];
  char mi;
  char lastName[30];
};
```

(a)

```
class Student
{
public:
  int id;
  char firstName[30];
  char mi;
  char lastName[30];
};
```

(b)

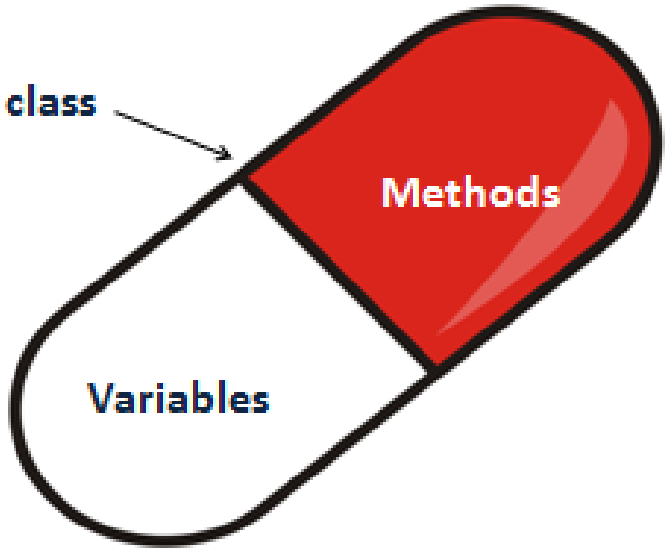
classes

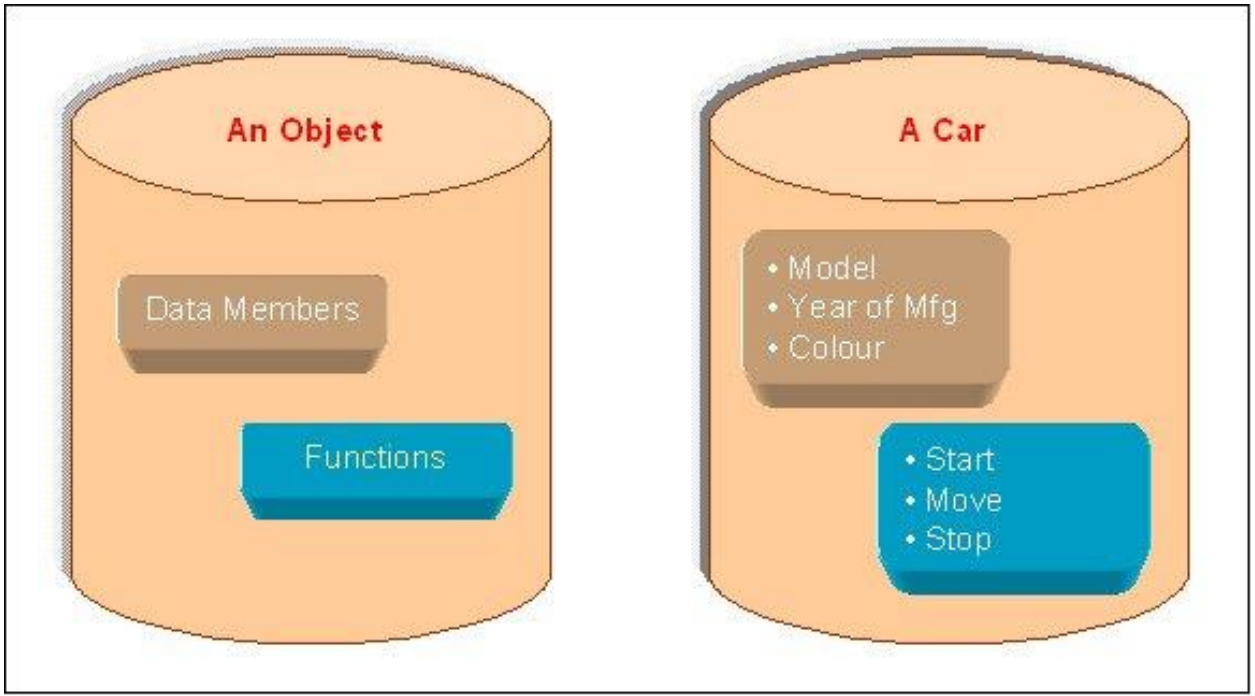
7. Thus a class is defined as an user defined abstract data type having collection of data members along with member functions.

8. This association of data and functions together as a single wrapper is called Encapsulation.

9. Classes without any data members and member functions are called as empty classes.

Encapsulation





Empty Class

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

class DivisionByZero{};
      // use it only
      // for handling exceptions
```

classes

10. Access Specifiers
    - Public
    - Private
    - Protected

11. Private members – Are accessible only to their own class's members.
12. Public – Public members are not only accessible to their own members, but also from outside the class.
13. Protected – Used during inheritance.
14. The members in a class without any access specifiers are Private by default.
15. A class which is totally private is hidden from the external world and will not serve any useful purpose.
16. It's a general practice to declare data members as private and member functions as public.

classes

17. The name of data and member functions of a class can be same as those in other classes, the members of different classes do not conflict each other.

```
class student
{
        char name[30];
        int regno;
        int age;
public:
        void getdata(void);
        void display(void);
};
```

```
class person
{
        char name[30];
        int age;
public:
        void getdata(void);
        void display(void);
};
```

classes

18. More than one class with same class name in a program is not permitted though the members differs.

```
class student
{
        char name[30];
        int regno;
        int age;
public:
        void getdata(void);
        void display(void);
};
```

```
class student
{
        int height;
        int weight;
public:
        void read(void);
        void print(void);
};
```

classes

19. A class can have multiple member functions(but not data members) with same name as long as they differ in terms of signature. This feature is known as Function or method overloading.

```
class student
{
        char name[30];
        int regno;
        int age;
Public: int age;   X
public:
        void getdata(void);
        int getdata(int);
        void display(void);
};
```

```
class student
{
        int height;
        int weight;
public:
        void read(void);
        void print(void);
};
```

classes

20. Like structures the data members of a class cannot be initialized during their declaration, but they can be initialized by its member functions.

```
class GeoObject
{
        float x, y=5; // Error: data members cannot be initialized here
public:
        void setorigin( )
        {
                x=y=0.0;   ✔
        }
};
```

A Simple Class

```
class item
{
    int number;             // variables declaration
    float cost;                     // private by default
Public:
    void getdata(int a, float b); // functions declaration
    void putdata(void);       // using prototype
};                                  // ends with semicolon
```

- **Attributes also called *data members***
  - ***Because they hold information.***

- **Functions that operate on these data are called *methods or member functions*.**

```
#include<iostream>
#include<string.h>
using namespace std;
class student
{
public:
        int rno;
        string name;
};
int main()
{
        student s1;
        student s2;
        s1.rno=1001;
        s1.name="SBR";
        s2.rno=1002;
        s2.name="SAS";
        cout<<s1.rno;
        cout<<s1.name;
        cout<<s2.rno;
        cout<<s2.name;
}
```

```
#include <iostream>
#include <cstring>

using namespace std;

int main ()
{
   char str1[10] = "Hello";
   char str2[10] = "World";
   char str3[10];
   int  len ;

   // copy str1 into str3
   strcpy( str3, str1);
   cout << "strcpy( str3, str1) : " << str3 << endl;

   // concatenates str1 and str2
   strcat( str1, str2);
   cout << "strcat( str1, str2): " << str1 << endl;

   // total lenghth of str1 after concatenation
   len = strlen(str1);
   cout << "strlen(str1) : " << len << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
```

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{
   string str1 = "Hello";
   string str2 = "World";
   string str3;
   int  len ;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total lenghth of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
str3 : Hello
str1 + str2 : HelloWorld
str3.size() :  10
```

```
// an example with class
#include<iostream.h>
using namespace std;
class person
{
    char name[30];
    int age;
public:
    void getdata(void);
    void display(void);
};
        int main()
        {
        person p;
        p.getdata();
        p.display();
        return 0;
        }
void person :: getdata(void)
{
        cout<<"enter name: ";
        cin>>name;
        cout<<"enter age: ";
        cin>>age;
}
void person :: display(void)
{
        cout<<"\n Name: "<<name;
        cout<<"\n Age: "<<age;
}
```

Output:

Enter name: raja

Enter age: 30

Name : raja

Age : 30

Class Definition - Access Control

- Information hiding
  - To prevent the internal representation from direct access from outside the class
- Access Specifiers
  - public
    - may be accessible from anywhere within a program
  - private
    - may be accessed only by the member functions, and friends of this class
  - protected
    - acts as public for derived classes
    - behaves as private for the rest of the program

```
class Cube
{
 public:
 int side;
 int getVolume()
 {
  return side*side*side;
 }
};
```

```
class Cube
{
 public:
 int side;
 int getVolume();
}

int Cube :: getVolume()     //
{
 return side*side*side;
}
```

```
int main()
{
 Cube C1;
 C1.side=4;   // setting side value
 cout<< "Volume of cube C1 ="<< C1.getVolume();
}
```

```
#include<iostream.h>
using namespace std;
class smallobj
{
    int data;
public:
    void setdata(int d);
    { data = d; }
    void showdata( );
    { cout<<data; }
};
```

```
int main( )
{
    smallobj s1,s2;
    s1.setdata(1066);
    s2.setdata(1776);

    s1.showdata();
    s2.showdata();
    return(0);
}
```

```
#include<iostream.h>
using namespace std;
class part
{
    int modelno;
    float cost;
public:
    void setpart(int mn,float c);
    { modelno = mn;
      cost = c; }
    void showpart( );
    { cout<<modelno;
     cout<<cost; }
};
```

```
int main( )
{
    part part1;
    part1.setpart(624,200);
    part2.setpart(177,300);

    part1.showpart();
    part2.showpart();
    return(0);
}
```

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

class distance
{
    int feet;
    float inches;
public:
    void setdist(int ft,float in);
    { feet = ft;
      inches = in; }

    void getdist( );
    { cin>>feet;
     cint>>inches; }

    void showdist()
    { cout<<feet<<inces; }
};
```

```
int main( )
{
    distance dist1,dist2;
    dist1.setdist(11,6.2);
    dist2.getdist( );

    cout<<dist1.showdist();
    cout<<dist2.showdist();
    return(0);
}
```

Another Example

```
#include <iostream.h>

class circle
{
    private:
       double radius;

    public:
       void store(double);
       double area(void);
       void display(void);

};
```

```
// member function definitions

void circle::store(double r)
{
    radius = r;
}
double circle::area(void)
{
    return 3.14*radius*radius;
}
void circle::display(void)
{
    cout << “r = “ << radius << endl;
}
```

```
int main(void) {
    circle c;   // an object of circle class
    c.store(5.0);
    cout << "The area of circle c is " << c.area() << endl;
    c.display();
}
```