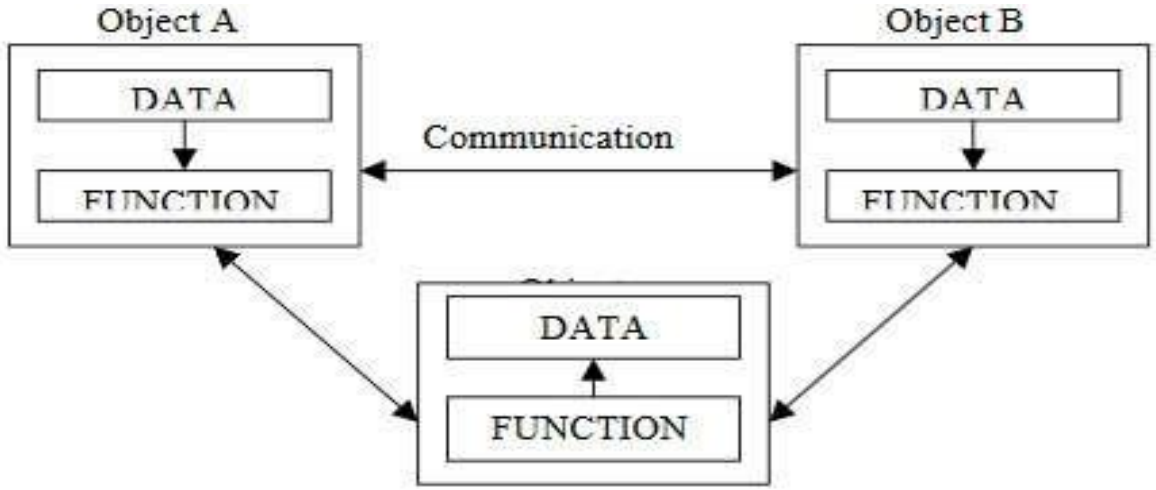
Object oriented programming Concepts

MCA -111

Object Oriented Programming Paradigm

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- 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



Basic Concepts of Object Oriented Programming

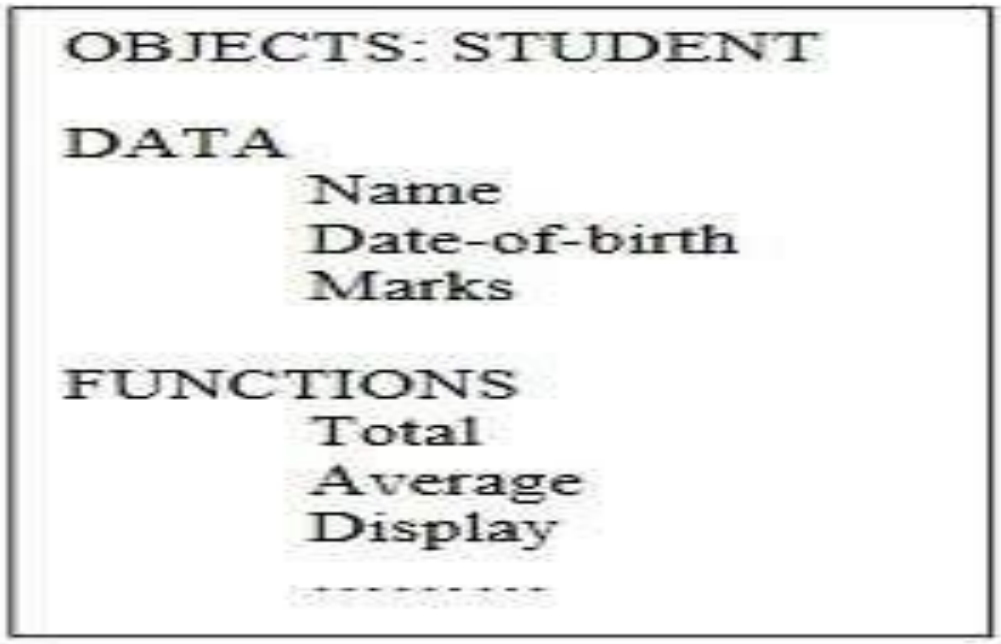
It is necessary to understand some of the concepts used extensively in object-oriented programming.These include:

- Objects
- Classes
- Data encapsulation
- Data abstraction
- Inheritance
- Polymorphism
- Message passing

Basic Concepts of OOP

➢ Objects :

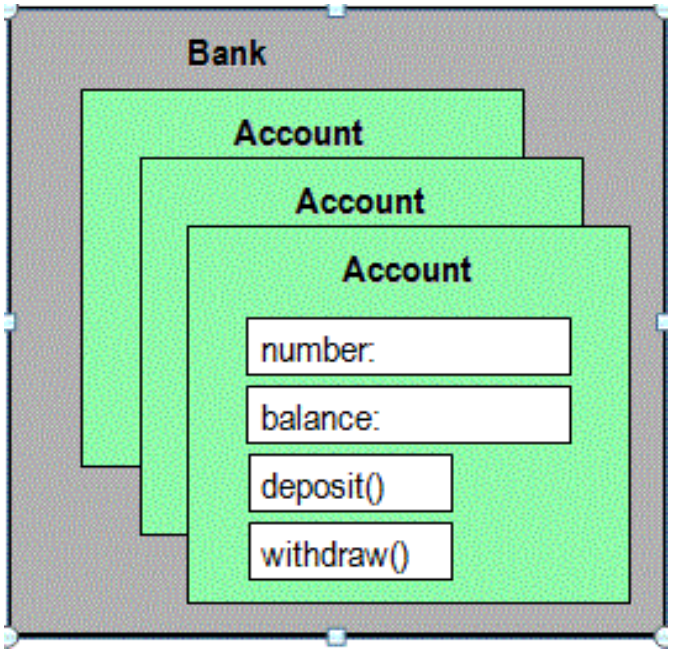
- Objects are the basic runtime entities in an object oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle



- Real-world objects have ***attributes*** and ***behaviors.***

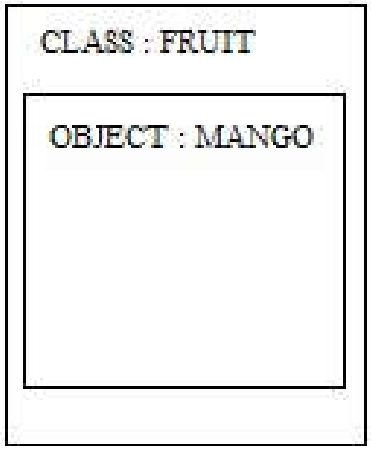
  Examples:

- Dog
  - Attributes: breed, color, hungry, tired, etc.
  - Behaviors: eating, sleeping, etc.
- Bank Account
  - Attributes: account number, owner, balance
  - Behaviors: withdraw, deposit



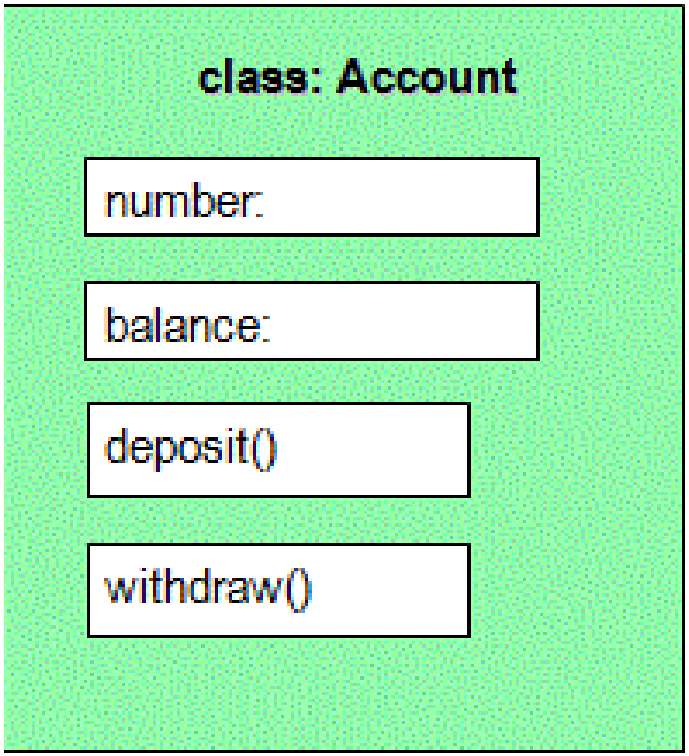
➢Class:

- Object contains data, and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of a class.
- A class can be thought of as a template used to create a set of objects.
- The objects are called ***instances*** of the class.



Examples:

- The "account" class describes the attributes and behaviors of bank accounts.

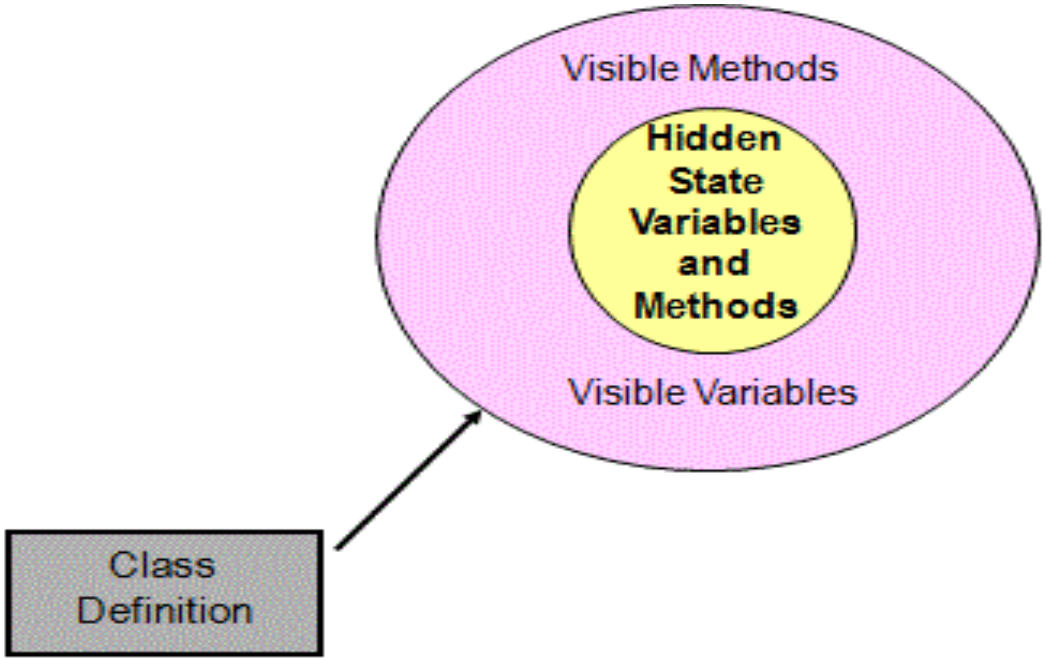


- A Class is a 3-Compartment Box encapsulating Data and Functions

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

➢Data 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, only those function which are wrapped in the can access it.
- These functions provide the interface between the object's data and the program.
- This insulation of the data from direct access by the program is called data hiding or information hiding.



```
#include <iostream.h>
#include<conio.h>
class Employee {
  private:
    // Private attribute
    int salary;
  public:
    // Setter
    void setSalary(int s) {
      salary = s;
    }
    // Getter
    int getSalary() {
      return salary;
    }
};
void main() {
  clrscr();
  Employee myObj;
  myObj.setSalary(5000);
  cout << myObj.getSalary();
  getch();
}
```

➢ Data Abstraction

- Abstraction refers to the act of representing essential features without including the background details or explanations.
- Since classes use the concept of data abstraction, they are known as **Abstract Data Types (ADT)**

```
#include <iostream.h>
#include<math.h>
int main()
{
 int n = 4;
   int power = 3;
   int result = pow(n,power);    // pow(n,power) is the  power function
   cout << "Cube of n is : " <<result<<endl;
   return 0;
}
```

/* The pow() function is present in the math.h header file in which all the implementation details of the pow() function is hidden.*/

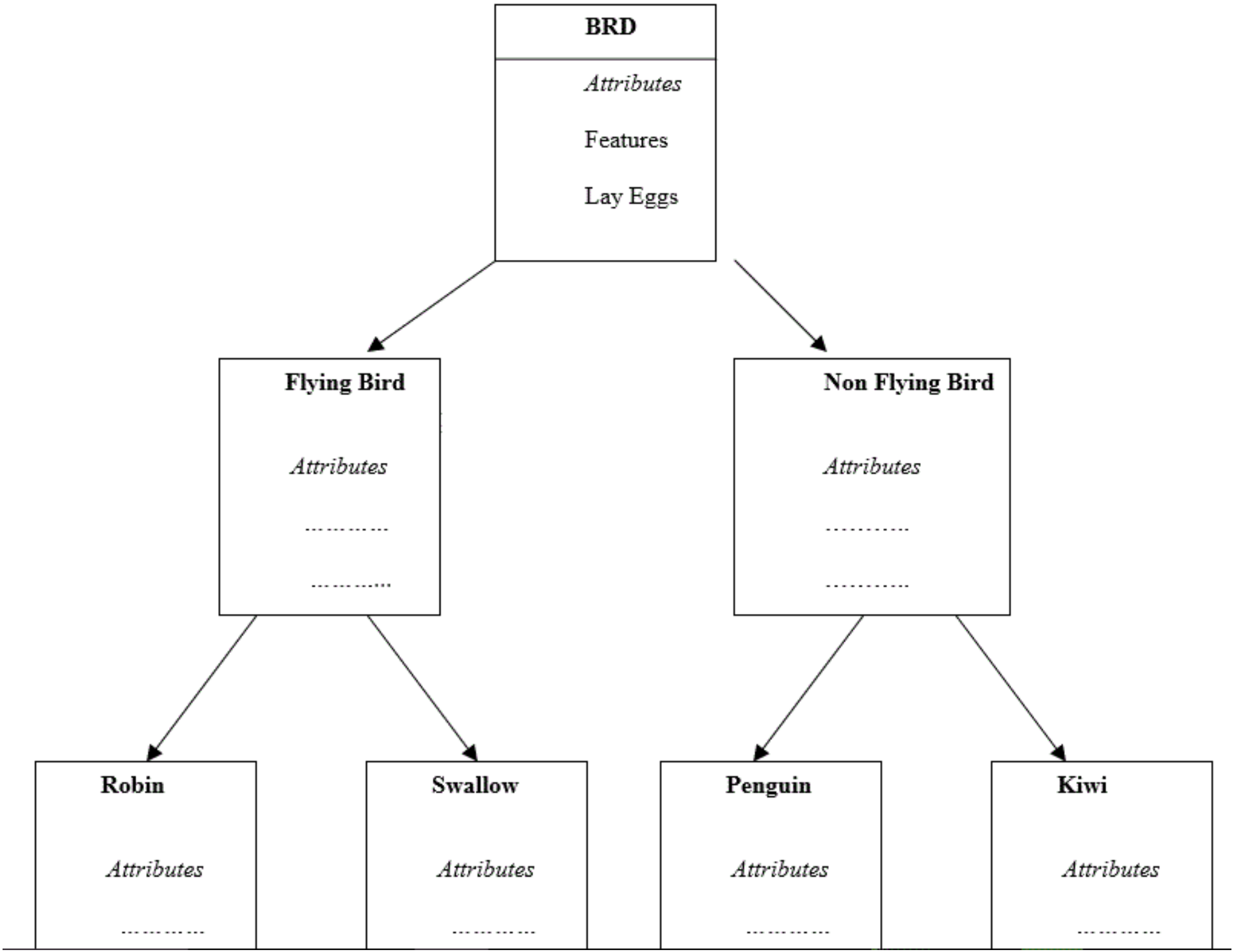
➢ Inheritance

- Inheritance is the process by which objects of one class acquire the properties and behaviors of objects of another class.

- In OOP, the concept of inheritance provides the idea of reusability. This means we can add additional features to an existing class without modifying it.

- The advantage of making a new class a child class (subclass) is that it will inherit attributes and methods of its parent class (also called the superclass).

- **Sub Class:** The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class:**The class whose properties are inherited by sub class is called Base Class or Super class.


BRD
Attributes
Features
Lay Eggs
Flying Bird
Attributes
Non Flying Bird
Attributes
Robin
Attributes
Swallow
Attributes
Penguin
Attributes
Kiwi
Attributes

Syntax:

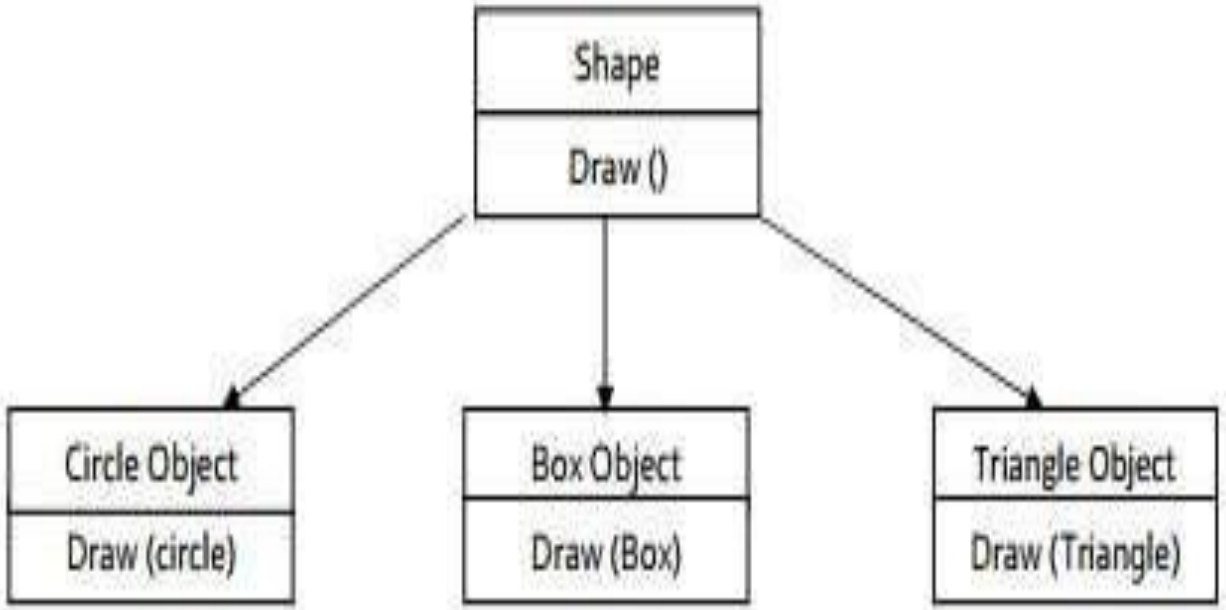
```
class subclass_name : access_mode base_class_name {
//body of subclass
};
```

```
#include <iostream.h>
//Base class
class Parent {
        public:
        int id_p;
};
// Sub class inheriting from Base Class(Parent)
class Child : public Parent {
        public:
        int id_c;
};
void main() {
Child obj1;
// An object of class child has all data members and member functions of class parent
obj1.id_c = 7;
obj1.id_p = 91;
cout << "Child id is " << obj1.id_c << endl;
cout << "Parent id is " << obj1.id_p << endl;
}
```

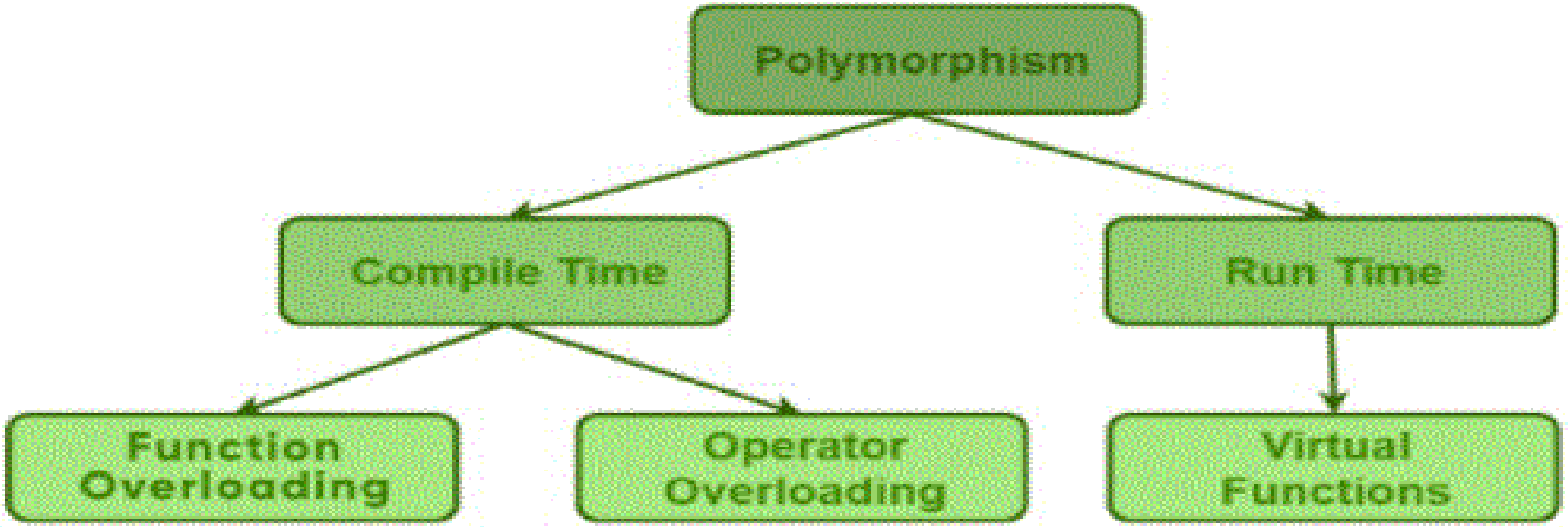
➢ Polymorphism

- **Polymorphism,** a Greek term means to ability to take more than one form.
- An operation may exhibits different behaviors in different instances. The behavior depends upon the type of data used in the operation.
- For example consider the operation of addition for two numbers; the operation will generate a sum. If the operands are string then the operation would produce a third string by concatenation.
- The process of making an operator to exhibit different behavior in different instances is known operator overloading.

Example:
The message **draw()** sent to an object of type **Circle, Box** and an object of type **Triangle** will result in different behaviors for each object.



Types of Polymorphism



Method Overloading:

having multiple methods with same name but with different signature (number, type and order of parameters).

```
#include<iostream.h>
#include<conio.h>
class A  {
      public:
      void print(into a, into b)  {
       int c=a+b;
       cout<< “Here is int sum=“<<c;   }
       void print(double a,double b)   {
        double c=a+b;
        cout<< “Here is double sum=“<<c;   }
       void print(char* a, char* b)   {
        cout<< “Here is char sum=“<<a<<b;   }
};
void main() {
  A obja;
  obja.print(5,6);
  obja.print(50.30,100.20 );
  obja.print(“c++”, “program”);
  getch(); }
```

Method Overriding:

When a subclass contains a method with the same name and signature as in the super class then it is called as method overriding.

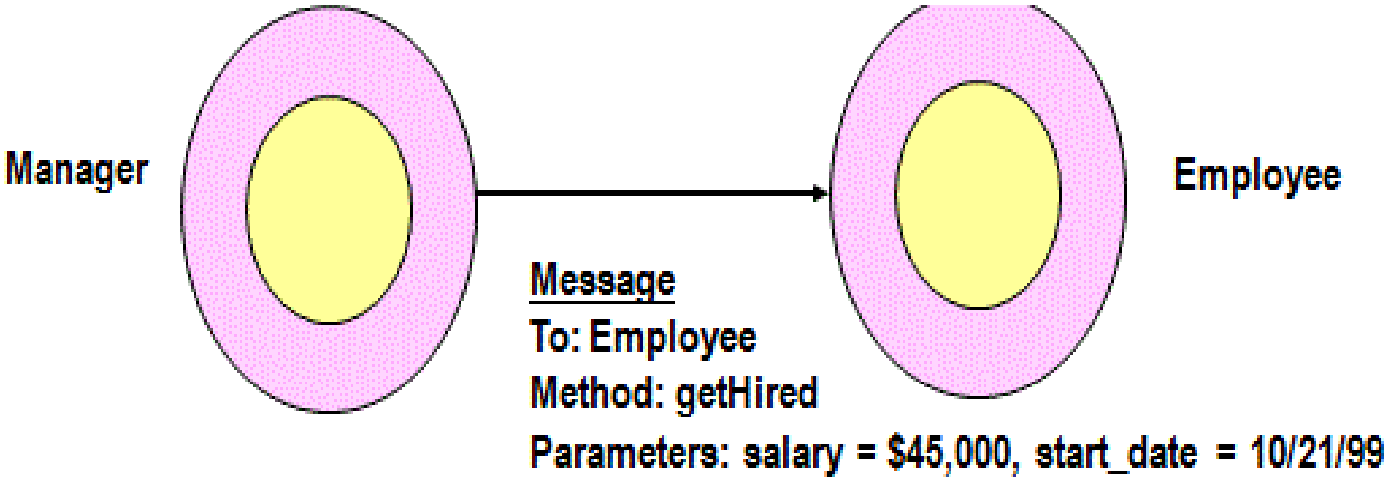
```
#include <iostream.h>
class Animal {
    public:
void eat(){
cout<<"Eating...";
    }
};
class Dog: public Animal
{
 public:
 void eat()
    {
       cout<<"Eating bread...";
    }
};
int main(void) {
   Dog d ;
   d.eat();
   return 0;
}
```

➢Message Passing

- An object-oriented program consists of a set of objects that communicate with each other.

- Message passing supports all possible interactions between two objects.

- Message passing is the mechanism that is used to invoke a method of the object.



Benefits of OOP

OOP offers several benefits to both the program designer & the user

- Through inheritance, we can eliminate redundant code and extend the use of existing class

- We can build programs from the standard working module the communicate with one another, rather than having to start writing code from scratch. This leads to saving of development time & higher productivity.

- The principle of 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 map objects in the problem domain to those in the program

.

Benefits of OOP

- It is easy to partition the work in a project based on objects.
- The 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.
- Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.
- Software complexity can be easily managed .