Inheritance

- Inheritance is a way of re-using existing classes.
- This is done via a mechanism which allows a new class to be created out of existing classes.
- This new class is said to be derived from the existing base class.

An Account

```
class Account{
  private:
      float balance;
      float interestRate;
  public:
      float GetBalance();          // balance enquiry
      int   SetBalance(float newBalance);
                                   // new balance set
      float GetInterest_rate();
      int setInterestRate(float newInterestRate);
};
```

A savings Account

- Rather than create a new class, the concept of inheritance can be used.
- A class is declared which inherits all the data and methods of Account, and adds new data items and methods to facilitate the operation of an interest bearing account.

```
class InterestAccount : public Account {
private:
  float theAccumulatedInterest;
  static       float   theInterestRate;
public:
  InterestAccount();
  static void Prelude( const float );
  void CalcInterest();
  void AddInterest();
};
```

A savings Account

- The constructor for an object of type interest_account will use the constructor for account to set up the initial balance in the account.
- This does not have to be explicitly called if the constructor for the base class has no parameters.

```
InterestAccount::InterestAccount(){
  the_accumulated_interest = 0.0;
}
```

Order of Calling Constructors and Destructors

- The constructor in the base class is called first, followed by the constructor in the derived class.
- When the object's storage is released the destructor in the derived class will be called first, followed by the destructor in the base class.

Pointers to derived types

- Assume two classes B and D, and D is derived from B
- In this situation, a pointer of type *B may also point to an object of type D. I.e. a pointer to a base class may also be used as a pointer to any derived class.
- However, the opposite is not true, i.e. a pointer of type D * may not point to an object of type B
- Furthermore, although you can use a base pointer to point to a derive object, you can access only the members of the derived type that were imported from the base.
- i.e. you won't be able to access any members added by the derived class
- However, this can be changed by casting a base pointer into a derived pointer.

```
#include <iostream.h>
class base{
   int i;
public:
   void set_i((int num) {i=num;}
   int get_i(){return i;}
};

class derived: public base {
   int j;
public:
   void set_j(int num){j=num;}
   int get_j(){return j;}
};
```

```
void main()
{
   base *bp;
   derived d;

   bp = &d;  //base pointer points to derived object

   //access derived object using base pointer

   bp->set_i(10);
   cout << bp->get_i() << " ";

/* This won't work. You can't access elements of a
   derived class using a base class pointer */
   bp->set_j(88);       //error
   cout <<bp->get_j();  //error
}
```