Inheritance

- A mechanism for specialization
- A mechanism for reuse

- Fundamental to supporting polymorphism
class Account
{
protected:
  double fBalance;

public:
  Account( double aBalance );
  void deposit( double aAmount );
  void withdraw( double aAmount );
  double getBalance();
};

class BankAccount: public Account
{
private:
  double fInterestRate;

public:
  BankAccount( double aRate );
  void addInterest();
  void chargeFee( double aAmount );
};
Constructors and Inheritance

- Whenever an object of a derived class is instantiated, multiple constructors are called so that each class in the inheritance chain can initialize itself.

- The constructor for each class in the inheritance chain is called beginning with the base class at the top of the inheritance chain and ending with the most recent derived class.
Base Class_initializer

class BankAccount : public Account
{
public:
    BankAccount( double aRate ) :
        Account( 0.0 ),
        fInterestRate( aRate )
    {};
};
Facts About Base Class Initializer

- If a base class does not have a default constructor, the derived class must provide a base class initializer for it.

- Base class initializers frequently appear alongside member initializers, which use similar syntax.

- If more than one argument is required by a base class constructor, the arguments are separated by comma.

- Reference members need to be initialized using a member initializer.
Destructors and Inheritance

- Whenever an object of a derived class is destroyed, the destructor for each class in the inheritance chain, if defined, is called.

- The destructor for each class in the inheritance chain is called beginning with the most recent derived class and ending with the base class at the top of the inheritance chain.
Virtual Destructors

- When deleting an object using a base class pointer of reference, it is essential that the destructors for each of the classes in the inheritance chain get a chance to run:

```cpp
BackAccount *BAptr;
Account *Aptr;

BAptr = new BackAccount(2.25);
Aptr = BAptr;
...
delete Aptr;
```
Virtual Account Destructor

class Account
{
    public:
        virtual ~Account() { ... }
        ...
};

Not declaring a destructor virtual is a common source for memory leaks!
To give a member function from a base class new behavior in a derived class, one overrides it.

To allow a member function in a base class to be overridden, one must declare the member function virtual.

Note, in Java all member functions are virtual.
Virtual withdraw Method

class Account
{
    public:
    ...
    virtual void withdraw( double aAmount )
    {
        fBalance -= aAmount;
    }
    ...
};
Overriding the withdraw Method

class BackAccount : public Account
{
public:
...
    virtual void withdraw( double aAmount )
    {
        if ( !(fBalance - aAmount < 0.0) )
        Account::withdraw( aAmount );
    }
...}

never use ==
base call
BankAccount lBankAccount(2.25);
Account* Aptr = &lBankAccount;

Aptr->withdraw(50.0);

invokes BankAccount::withdraw
Facts About Virtual Members

- Constructors cannot be virtual.
- Declaring a member function virtual does not require that this function must be overridden in derived classes, except the member function is declared pure virtual.
- Once a member function has been declared virtual, it remains virtual.
- Parameter and result types must match to properly override a virtual member function.
- If one declares a non-virtual member function virtual in a derived class, the new member function hides the inherited member function.

- You can declare a private member function virtual to enable polymorphism within the scope of the declaring class.
Abstract Base Classes

- A class is abstract if it contains one or more pure virtual member functions.
  - An abstract class cannot be instantiated.
  - Derived classes must provide definitions for the pure virtual member functions, or declare them as pure virtual itself.
  - A pure virtual member function can be defined in the class that declares it pure virtual.
  - If one declares a pure virtual destructor, a definition for it must be given in the class that declares the destructor pure virtual.
  - Abstract classes required a virtual destructor.
Pure Virtual Member Function

class Account
{
    public:
        virtual double estimateReturn() = 0;
};
Abstract Classes as Interfaces

- An interface is a set of member functions.
- Interfaces never define data types.
- Interfaces never provide a default implementation of methods.
- A class implements an interface by defining all member functions declared by the interface.

- An abstract class can be viewed as an interface if it only contains public pure virtual member functions.
Interface TicTacToeView

class TicTacToeView
{
public:
  virtual ~TicTacToeView() {} // empty required destructor

  virtual void registerGame( TicTacToe* aGame ) = 0;
  virtual void printBoard() = 0;
  virtual void draw() = 0;
  virtual void declareWinner( int aPlayer ) = 0;
};
Access Level for Inheritance

- **public:**
  - Public members in the base class remain public.
  - Protected members in the base class remain protected.
  - Yields a “is a” relationship.

- **protected:**
  - Public and protected members in the base class are protected in the derived class.
  - Yields a “implemented in terms of” relationship.

- **private:**
  - Public and protected members in the base class become private in the derived class.
  - Yields a stricter “implemented in terms of” relationship.
### Multiple Inheritance

- C++ allows a class to inherit from several base classes at once.

- Multiple inheritance is a mechanism to compose orthogonal behavior.
class Account {
    { ... };
}

class BackAccount : public Account {
    { ... };
}

class WireAccount : public Account {
    { ... };
}

class CheckingAccount : public BackAccount,
                        public WireAccount {
    { ... };
}
Diamond Problem

class Account

class BankAccount

class WireAccount

class CheckingAccount

CheckingAccount inherits Account twice!
Virtual Base Classes

- A virtual base class allows multiple instances of a base to be included in object of a derived class.

- Members of virtual base classes are disambiguated by merging them into one representative.
Virtual Account Class

class Account
{ ... };

class BackAccount : virtual public Account
{ ... };

class WireAccount : virtual public Account
{ ... };

class CheckingAccount : public BackAccount, public WireAccount
{ ... };
How Virtual Base Classes Work

class Account

fBalance

class BankAccount

(Account::)fBalance

class CheckAccount

(Account::)fBalance

class WireAccount

fBalance