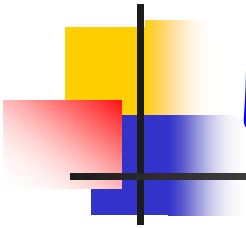


Lecture-4

- Inheritance – review.
- Polymorphism – review
  - Virtual functions
- Abstract classes
- Miscellaneous Topics
  - Function Overloading and Overriding
  - this keyword
  - static members
  - inline functions
  - Passing arguments by values and reference

Inheritance – base class & derived class

- Base class

```
class account
{
    int user_SSN;
    int accountNumber;
public:
    void deposit (int amount);
    void withdraw (int amount);
    void showAccountType( );
};
```

- Derived class or child class

```
class checkingAccount : public account // checkingAccount is
{                                       // derived from account
    int lastCheckCleared;              // not present in account
    void showAllChecksCleared( );      // not present in account
    void showAccountType( );           // defined in both classes
};
```

Inheritance – base class and derived classes

- Base Class

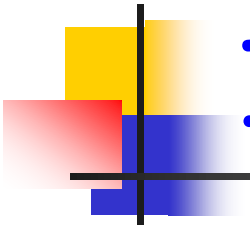
```
class account
{
private:
    int user_SSN;
    int accountNumber;
    int balance;
public:
    account ( ) { }
    account (int ssn, acctNum);
    ~account( ) { }
    void deposit (int amount)
    void withdraw (int amount);
    void showAccountType( );
};
```

- Derived (or child) class-1

```
class checkingAccount : public account
{
public:
    int lastCheckCleared;
    void showChecksCleared ( );
    void showAccountType( );
};
```

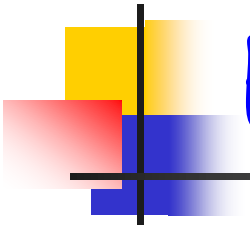
- Derived (or child) class-2

```
class IRA_account : public account
{
public:
    void buyFund (int fund_ID);
    void sellFund (int fund_ID);
    void showAccountType( );
};
```



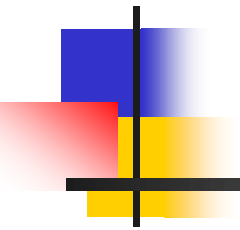
Inheritance – continued.

- Important points to note:
  - Derived classes have access to members of base classes in this example.
  - Derived classes can have their own members.
    - E.g. lastCheckCleared, showAllChecksCleared( ), buyFund( ), sellFund( ), etc.
    - Members of one derived class are not accessible to another

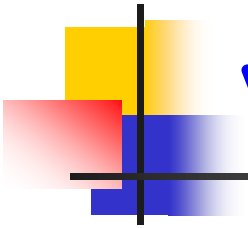


Examples

- Valid usage in an external function
  - account acct(123456, 5672);
  - checkingAccount ca;
  - acct.deposit (700);
  - acct.withdraw (300);
  - ca.deposit (1000);
  - ca.showAllChecksCleared( )
- Invalid usage in derived class
  - ca.user_SSN = 1234; // Can't access user_SSN
  - ca.accountNumber = 567;



Polymorphism & virtual functions



virtual functions

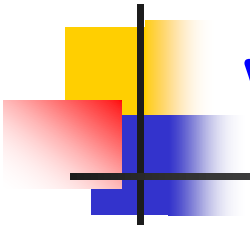
- Function "double showAccountType( )" is defined in both base and child classes.
  - Supposed to return different values
    - virtual void Account::showAccountType( )
      { cout << "Account" << endl; }
    - void CheckingAccount::showAccountType( )
      { cout << "Checking Account" << endl; }
    - void IRA_Account::showAccountType( )
      { cout << "IRA Account " << endl ; }

virtual functions … contd.

```
main( )
{
   Account *x = new CheckingAccount();
   x→showAccountType( );
   // What will this print?
}
```

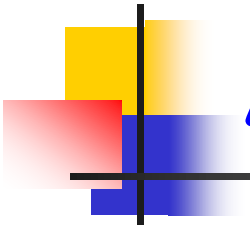
- This will print
  - Account, if the function is NOT virtual
  - Checking Account , if it is defined virtual

Why are virtual functions needed?



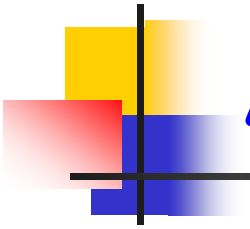
- Mainly to enforce class specific functional implementation.
- Should not call base class function from a child object.
- An account object may take different "forms" at different times
  - Checking account, IRA account, etc.
  - showAccountType( ) should use derived class specific function.

⇒ Polymorphism



Abstract classes

- Consider an object of Account.
- It makes sense to have
  - A specific type (e.g., checking) of account
  - Not just a generic account object.
- A user should be able to create
  - Specific object types.
  - NOT generic objects.
- An abstract class is the generic class.



Abstract classes … contd.

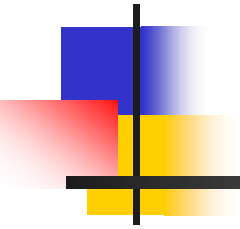
- Properties of abstract classes.
  - Defines a generic base class
  - Class definition has attributes and methods
  - Other classes are derived from it.
  - Derived classes implement the methods defined in abstract class.
  - Can NOT instantiate objects of base class.
  - Can instantiate only objects of derived classes.

How do we create abstract classes?

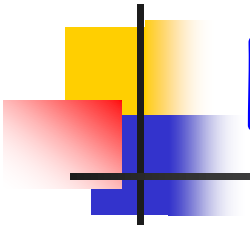
- Set ANY virtual function to 0.
  - Pure virtual function – value of function = 0
  - NO BODY for function

```
class Account
{
    virtual void showAccountType ( ) = 0;
}
class CheckingAccount : public Account
{
    void showAccountType ( ) { … }
}

Account x;                  // Will NOT work.
CheckingAccount y;          // Will work.
```



Miscellaneous Topics



Function overloading

- Functions with the same name but with
  - Different number of arguments or
  - Different types of arguments

E.g. int add (int a, int b, int c) { return (a+b+c); }
int add (int a, int b) { return (a + b); }
double add (double a, double b) { return (a + b); }

- Here "add" is an overloaded function

Function overriding

- Functions defined in parent class and re-implemented by the child class.

E.g. class Bird

```
{
    int canFly( ) { return (1); }
}
class Penguin : public Bird
{
   int canFly ( ) { return (0); }
}
```

Here, "canFly" is an overridden by the child class, Penguin

this keyword

- this refers to the address of the current object

- E.g.

```
class Account
{
   private:
      int balance;
   public:
    setBalance (int amount)
    {
       this->balance = amount;
    }
};
```

C++ static members

- static members in C++
  - Shared by all the objects of a class
  - Specific to a class, NOT object of a class
  - Access them using className::static_member
  - E.g., myClass::staticVar, or myClass::f1( )

```
class myClass
{
  public:
        static int staticVar;
        static void f1( );
};
```

Inline functions

- Normal functions
  - Carry operational overhead
  - Function call, parameter passing, etc.
- Inline functions
  - No overhead related to function calls
  - Might be as simple as a memory access

```
class myClass
{
        public:
                int x;
                inline int getX ( ) { return x; }
};
```

Inline functions … contd.

- Compiler decides if a function defined as "inline" can be "inline" or not.
- Too much of code for any function defined as inline
  - Compiler may treat as a regular (non-inline) function
- Note: Code for an inline function
  - Can be in the class itself, or
  - Can be in the same file as the class definition.
  - CANNOT be defined in any file outside the class definition

Passing args. to a function … by value

- Compiler creates a copy when function is called.
- Any changes made inside the function are not reflected after the function.

```
class myClass
{
        void f1(int  i ) // i is passed by value
         { i = 3; }
};
int x = 5;
myClass obj;
obj.f1( x );
cout << "value of x: " << x << endl; // x is still 5
```

Passing args. to a function … by reference

- Compiler takes the original object.
- Any changes made inside the function are reflected after the function.

```
class myClass
{
        void f1(int& i ) // i is passed by reference.
         { i = 3; }
};
int x = 5;
myClass obj;
obj.f1( x );
cout << "value of x: " << x << endl; // x is 3
```