



# Lecture-4

---

- Inheritance ... review.
- Polymorphism
  - Virtual functions
- Abstract classes
- const member functions
- const arguments to a function
- Function overloading and overriding



# Inheritance - base class & derived class

- **Base class**

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

- **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
    double computeInterest( );       // 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);
    double computeInterest( );
};
```

## ■ Derived (or child) class-1

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

## ■ Derived (or child) class-2

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



# 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 computeInterest( )" is defined in both base and child classes.
  - Supposed to return different values
    - **virtual** double Account::computeInterest ( )  
    { return 0; }
    - double CheckingAccount::computeInterest ( )  
    { return 10.0; }
    - double IRA\_Account::computeInterest ( )  
    { return 100.0; }



# virtual functions ... contd.

---

```
main( )
```

```
{
```

```
    Account *x = new CheckingAccount();
```

```
    x→computeInterest( );
```

```
    // Will this return 0 or 10.0?
```

```
}
```

- This will return
  - 0, if the function is **NOT virtual**
  - 10.0, if the function 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.
  - `computeInterest( )` should compute 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 double computeInterest ( ) = 0;
}
class CheckingAccount : public Account
{
    double computeInterest ( ) { ... }
}
```

Account x; // Will **NOT** work.

CheckingAccount y; // Will work.



# const arguments and const member functions

---



# const arguments to functions

---

```
void f1(const int a)
{
    a = 3; // Not allowed
}
```

- **const** arguments to a function can't be changed in the function.
- f1 can't change a in the above example



# const member functions

---

```
class myClass
```

```
{
```

```
    int a;
```

```
    ...
```

```
    void f1( ) const
```

```
    { a = 3; } // Not allowed
```

```
};
```

- **const** functions can't change any attributes of myClass.
- f1 can't change a in the above example



# Function overloading and function overriding

---





# 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