Polymorphism (Lecture -3)

(CS 217)

Dr. Muhammad Aleem,

Department of Computer Science,

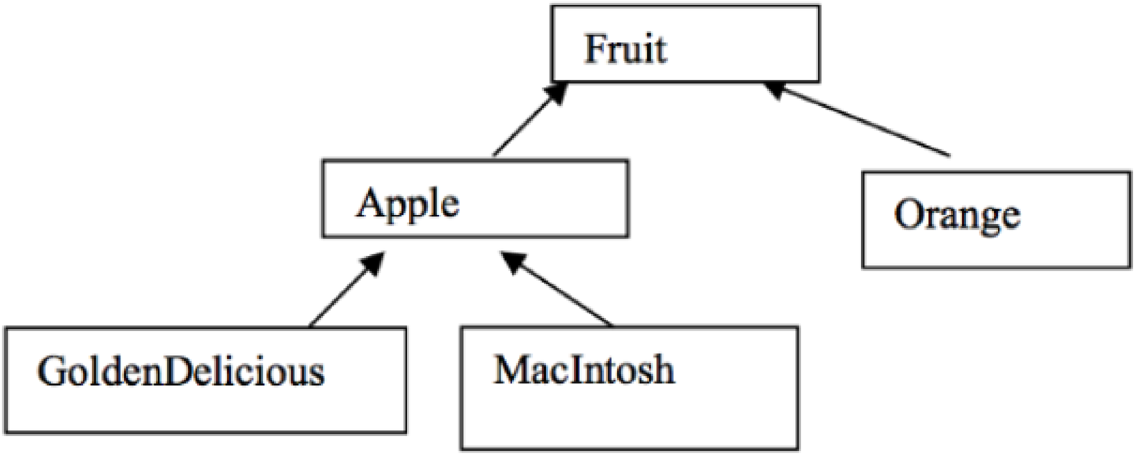
National University of Computer & Emerging Sciences,
Islamabad Campus

Lecture 3 - Contents

- Classes in C++
- Concrete and Abstract Classes
- Pure Virtual Function
- Usage of Pure-Virtual function and Abstract Classes
- Example Code…

Classes in C++

- It is true that all objects are represented by a class, the converse is not true.
- All classes do not necessarily represent objects:



- In C++, we can classify the Classes:
  1. Concrete Classes
  2. Abstract Classes

Concrete Classes

- **Concrete Classes provide full representation of their objects.**
  - **Object can be instantiated**

- **Must provide implementation** for **every member function**

Abstract Classes

- Classes from which it is never intended to instantiate any objects (*Reasons*?):
  - Incomplete: derived classes must define the "missing pieces"
  - Too generic: to define real objects

- Normally used as base classes and called abstract base classes
- It is a common public interface for the entire class hierarchy

Pure `virtual` Functions

- **A class is made *abstract* by declaring one or more of its virtual functions to be "pure"**
  - I.e., by **placing** `"= 0"` in its **declaration**

- Example:

```
virtual void draw() = 0;
```

`"= 0"` is **known** as a ***pure specifier.***

– Tells **compiler** that **there *is no* implementation**.

Pure `virtual` Functions (cont.)

- Every **concrete derived class must override all base-class pure virtual functions**
  - with concrete implementations

- If **even one pure virtual function is not overridden**
  - the **derived-class will also be abstract**
  - **Compiler will refuse to create any objects** of the **class**

- **However, we can create Pointers** and **references** of **abstract classes:**
  - **To point or refer the derived class objects**

Why we need abstract classes?

- To define a **common public interface** in a **class hierarchy**
  - **To create Abstraction Framework (in our software system)**

- It is a **core feature** of **Object-Oriented Design**

- **Simplifies development of big software systems**
  - **Enables code re-use, Readable, maintainable, adaptable code**

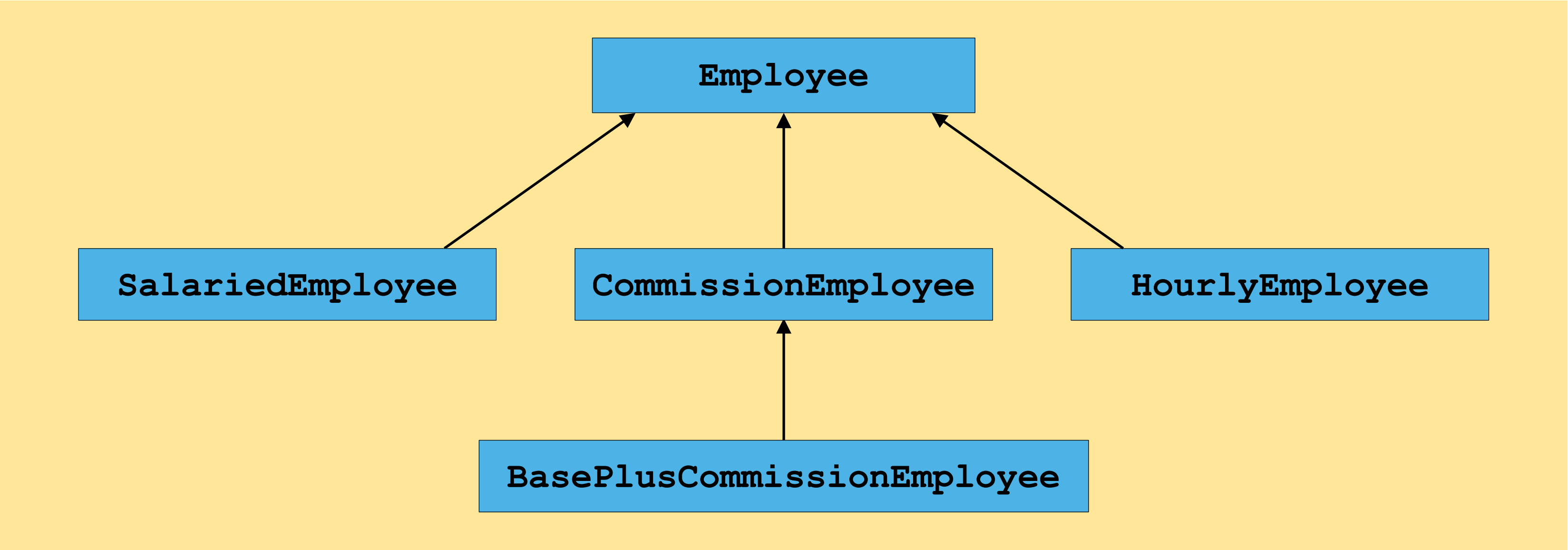
Case Study: Payroll System Using Polymorphism

- Create a payroll program
  - Use **virtual functions** and **polymorphism**

- Problem statement
  - **4 types of employees, paid weekly:**
    1. **Salaried (fixed salary, no matter the hours)**
    2. **Hourly workers**
    3. **Commission (paid percentage of sales)**
    4. **Base-plus-commission (base salary + percentage of sales)**

Case Study: Payroll System Using Polymorphism

- Base class `Employee`:
  - Pure virtual function `earnings` (returns pay)
  - Other classes derive from `Employee`



Payroll System

```
class Employee {
public:
   Employee(const char *, const char *);
   ~Employee();
            char *getFirstName() const;
            char *getLastName() const;

   // Pure virtual functions make Employee abstract base class.
   virtual float earnings() const = 0; // pure virtual
   virtual void print() const = 0;    // pure virtual

protected:
   char *firstName;
   char *lastName;
};
```

DEMO: Payroll.cpp

```
Employee::Employee(const char *first, const char *last)
{
            firstName = new char[ strlen(first) + 1 ];
            strcpy(firstName, first);
            lastName = new char[ strlen(last) + 1 ];
            strcpy(lastName, last);
}

// Destructor deallocates dynamically allocated memory
Employee::~Employee() {
  delete [] firstName;   delete [] lastName;
}

//Return a pointer to the first name
char *Employee::getFirstName() const {
return firstName;   // caller must delete memory
}

char *Employee::getLastName() const {
  return lastName;   // caller must delete memory
}
```

```
class SalariedEmployee: public Employee {
public:
        SalariedEmployee(const char *, const char *, float = 0.0);
        void setWeeklySalary(float);
         virtual float earnings() const;
         virtual void print() const;
private:
        float weeklySalary;
};
```

```
// Constructor function for class
SalariedEmployee:: SalariedEmployee(const char *first,
                                     const char *last, float s)
   : Employee(first, last)  // call base-class constructor
{ weeklySalary = s > 0 ? s : 0; }

// Set the SalariedEmployee’s salary
void SalariedEmployee::setWeeklySalary(float s)
   { weeklySalary = s > 0 ? s : 0; }

// Get the SalariedEmployee’s pay
float SalariedEmployee::earnings() const { return weeklySalary; }

// Print the SalariedEmployee’s name
void SalariedEmployee::print() const
{
   cout << endl << " Salaried Employee: " << getFirstName()
        << ' ' << getLastName();
}
```

```
class CommissionWorker : public Employee {
public:
     CommissionWorker(const char *, const char *, float = 0.0, unsigned = 0);
    void setCommission(float);
    void setQuantity(unsigned);
    virtual float earnings() const;
    virtual void print() const;

private:
    float commission;   // amount per item sold
    unsigned quantity;  // total items sold for week
};
```

```
CommissionWorker::CommissionWorker(const char *first,  const char *last, float c, unsigned q)
   : Employee(first, last)  // call base-class constructor
{
                  commission = c > 0 ? c : 0;
                  quantity = q > 0 ? q : 0;
}

void CommissionWorker::setCommission(float c)
   { commission = c > 0 ? c : 0; }

void CommissionWorker::setQuantity(unsigned q)
   { quantity = q > 0 ? q : 0; }

float CommissionWorker::earnings() const
   { return commission * quantity; }

void CommissionWorker::print() const
{
   cout << endl << "Commission worker: " << getFirstName()
        << ' ' << getLastName();
}
```

```
class HourlyWorker : public Employee {
public:
  HourlyWorker(const char *, const char *,
          float = 0.0, float = 0.0);
  void setWage(float);
  void setHours(float);
  virtual float earnings() const;
  virtual void print() const;

private:
  float wage;   // wage per hour
  float hours;  // hours worked for week

};
```

```
HourlyWorker::HourlyWorker(const char *first, const char *last   float w, float h)
   : Employee(first, last)   // call base-class constructor
{
   wage = w > 0 ? w : 0;
   hours = h >= 0 && h < 168 ? h : 0;
}

void HourlyWorker::setWage(float w) { wage = w > 0 ? w : 0; }

// Set the hours worked
void HourlyWorker::setHours(float h)
   { hours = h >= 0 && h < 168 ? h : 0; }

// Get the HourlyWorker's pay
float HourlyWorker::earnings() const { return wage * hours; }

// Print the HourlyWorker's name
void HourlyWorker::print() const
{
   cout << endl << "    Hourly worker: " << getFirstName()
        << ' ' << getLastName();
}
```

```
class BasePlusCommissionEmployee:public CommissionWorker
{
private:
          float baseSalary;
public:
          BasePlusCommissionEmployee(const char* , const
char* , float =0.0, unsigned =0,float =0.0);

          void setBaseSalary(float sal)          {
                    baseSalary = sal;
          }
          float getBaseSalary(void) const     {
                    return baseSalary;
          }
          void print() const;
          float earnings() const;
};
```

```
BasePlusCommissionEmployee::BasePlusCommissionEmployee(const
char* first, const char* last, float c,
                              unsigned q,float sal)
                              :CommissionWorker(first,last,c,q)
{
                baseSalary=(sal);
}

void BasePlusCommissionEmployee::print() const
{
               cout << "\nbase-salaried commission employee: ";
               CommissionWorker::print();  // code reuse
} // end function print

float BasePlusCommissionEmployee::earnings() const
{
               return getBaseSalary() + CommissionWorker::earnings();

} // end function earnings
```

```
void main(void)
{
   Employee *ptr;  // base-class pointer

   SalariedEmployee b(“Nauman", "Sarwar", 800.00);
   ptr = &b;  // base-class pointer to derived-class object
   ptr->print();                         // dynamic binding
   cout << " earned $" << ptr->earnings(); // dynamic binding
   b.print();                          // static binding
   cout << " earned $" << b.earnings();    // static binding

   CommissionWorker c(“Qasim", “Ali", 3.0, 150);
   ptr = &c;  // base-class pointer to derived-class object
   ptr->print();                         // dynamic binding
   cout << " earned $" << ptr->earnings(); // dynamic binding
   c.print();                          // static binding
   cout << " earned $" << c.earnings();    // static binding
   BasePlusCommissionEmployee p("Mehshan", "Mustafa", 2.5, 200, 1000.0);
   ptr = &p;  // base-class pointer to derived-class object
   ptr->print();                          // dynamic binding
   cout << " earned $" << ptr->earnings(); // dynamic binding
   p.print();                            // static binding
   cout << " earned $" << p.earnings();    // static binding

   HourlyWorker h(“Samer", “Tufail", 13.75, 40);
   ptr = &h;  // base-class pointer to derived-class object
   ptr->print();                          // dynamic binding
   cout << " earned $" << ptr->earnings(); // dynamic binding
   h.print();                            // static binding
   cout << " earned $" << h.earnings();    // static binding

   cout << endl;

   return 0;
}
```

End of Lecture 3