Object Oriented Programming

Bilal Khalid Dar

Inheritance IV

Virtual Functions

- If the member function definition is outside the class, the keyword virtual must not be specified again.

```
class Shape{
public:
  virtual void sayHi ();
};
virtual void Shape::sayHi (){  // error
  cout << ‘‘Just hi! \n’’;
}
```

- **Virtual functions can not be stand-alone or static functions.**

- **A virtual function can be inherited from a base class by a derived class, like other class member functions.**

Virtual Functions

- The ***virtualness*** of an **operation** is **always inherited**
- if a **function** is **virtual** in the **base class**, it **must be virtual** in the **derived class**,
- **Even if** the **keyword** "**virtual**" **not specified** (But always use the keyword in children classes for clarity.)
- If **no overridden function** is **provided**, the **virtual function** of **base class** is **used**

Introduction to Virtual Functions

- **Terminology in C++:**
  - **redefine** a **method** that uses **static binding**
  - **override** a **method** that uses **dynamic binding** (i.e., ***virtual functions***)

Virtual Functions

- To **override** a **base class virtual function**, the **virtual function instance** in **derived class must match** the **base class virtual function exactly**.

- **The overriding functions are virtual automatically. The use of keyword virtual is optional in derived classes.**

Virtual Functions

How to declare a member function virtual:

```
class Animal{
  public:
       virtual void id(){cout << "animal";}
};


class Cat : public Animal{
  public:
       virtual void id(){cout << "cat";}
};


class Dog : public Animal{
  public:
       virtual void id(){cout << "dog";}
};
```

Polymorphism Example (using Base Class's Pointers and References)

```
class Shape{
public:
   virtual void sayHi() { cout <<‘‘Just hi! \n’’;}
};
class Triangle : public Shape{
public:
   // overrides Shape::sayHi(), automatically virtual
   void sayHi() { cout <<‘‘Hi from a triangle! \n’’;}
};

void print(Shape obj, Shape *ptr, Shape &ref){
   ptr -> sayHi();   // bound at run time
   ref.sayHi();      // bound at run time
   obj.sayHi();      // bound at compile time
}

int main(){
  Triangle mytri;
  print( mytri, &mytri, mytri );
}
```

DEMO: PolyExample2.cpp

Virtual Destructors

- **Constructors cannot be virtual**, but **destructors can be virtual** when a **constructor** of a class is executed there is no **virtual** table in the memory, means no **virtual** pointer defined yet.

- **Ensures:** the **derived class destructor** is **called** when a **base class pointer** is **used,** while ***deleting a dynamically created derived class object***.

```
virtual ~Shape();
```

Reason: to **invoke the correct destructor**, **no matter how object is accessed**

Virtual Destructors (contd.)

```
class base {
public:
   ~base() {
      cout <<  “destructing base\n”;
   }
};

class derived : public base {
public:
   ~derived() {
      cout << “destructing derived\n”;
   }
};
```

```
int main()
{
   base *p = new derived;
   delete p;
  return 0;
}
```

Output:

```
destructing base
```

Using non-virtual destructor

Virtual Destructors (contd.)

```
class base {

public:
   virtual ~base() {
      cout <<  “destructing base\n”;
   }
};

class derived : public base {
public:
   ~derived() {
      cout << “destructing
  derived\n”;
   }
};
```

```
int main()
{
   base *p = new derived;
   delete p;

  return 0;
}
```

Output:

```
destructing derived
destructing base
```

Using virtual destructor

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

Concrete Classes

- **Concrete Classes: used to instantiate objects**
- **Must provide implementation** for **every member function they define**

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() const = 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**
  - **Cannot call a constructor**

Purpose

- **When** it **does not make sense** for **base class** to **have an implementation of a function**

- Software **design requires *all* concrete derived classes** to **implement the function**
    - **Themselves (to meet customized needs)**

Why Do we Want to do This?

- To define a ***common public interface*** for the **various classes** in a **class hierarchy**
  - **Create framework** for **abstractions defined** in **our software system**

- The **heart** of ***object-oriented programming***

- **Simplifies** a **lot of big software systems**
    - **Enables code** re-use in a **major way**
    - **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:**
    - **Salaried (fixed salary, no matter the hours)**
    - **Hourly workers**
    - **Commission (paid percentage of sales)**
    - **Base-plus-commission (base salary + percentage of sales)**

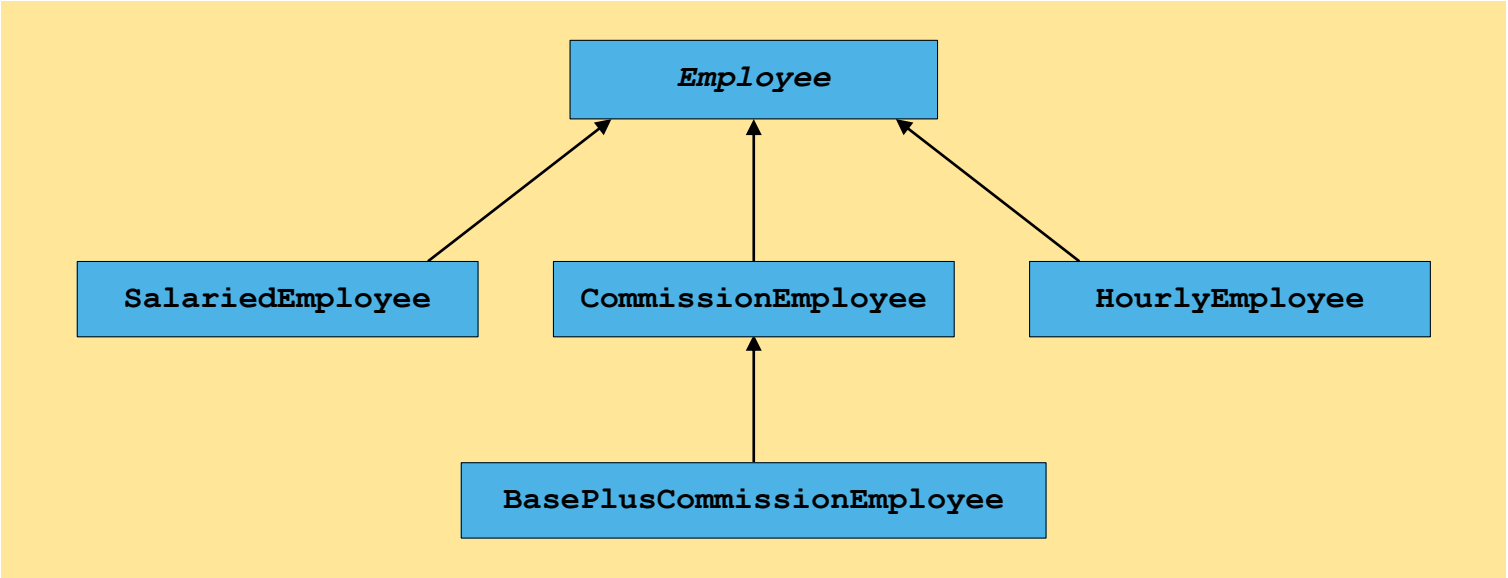
Case Study: Payroll System Using Polymorphism

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

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

Case Study: Payroll System Using Polymorphism

- Base class `Employee`:
  - Pure virtual function `earnings` (returns pay)
    - Pure virtual because need to know employee type
    - Cannot calculate for generic employee
  - Other classes derive from `Employee`



Employee Example

```
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;
};
```

Virtual function	Pure virtual function
A virtual function is a member function in a base class that can be redefined in a derived class.	A pure virtual function is a member function in a base class whose declaration is provided in a base class and implemented in a derived class.
The classes which are containing virtual functions are not abstract classes.	The classes which are containing pure virtual function are the abstract classes.
In case of a virtual function, definition of a function is provided in the base class.	In case of a pure virtual function, definition of a function is not provided in the base class.
The base class that contains a virtual function can be instantiated.	The base class that contains a pure virtual function becomes an abstract class, and that cannot be instantiated.
If the derived class will not redefine the virtual function of the base class, then there will be no effect on the compilation.	If the derived class does not define the pure virtual function; it will not throw any error but the derived class becomes an abstract class.
All the derived classes may or may not redefine the virtual function.	All the derived classes must define the pure virtual function.


Exercise

Shapes

- Shape:
  - color, fields
  - draw() draw itself on the screen
  - calcArea() calculates its own area.

Kinds of Shapes

- Rectangle
- Triangle
- Circle

Each could be a kind of shape (could be specializations of the shape class).

Each knows how to draw itself, etc.


Exercise

Coding Task

- A grocery shop want to make a software that enables them to save information of their **Customers**. The Customer class keeps tracks of the names and addresses of the customers. (all data members are private). The customer information printInformation() method that prints out their names along with their addresses.

- We have special **OnlineCustomers (**that inherits from the Customer **),** it adds a new instance variable for the email address of a online customer. Also, the online customers can add their contact number. The class has a function that send notification message to the customers.

- Override the printInformation() method in the OnlineCustomer class to print all his/her information.

- The main of the programs as follows

```
main()
    {
        Customer c(“Ahmed Khan", “Murree Road Rawalpindi“);
        c.printInformation();
        OnlineCustomer c2(“Memmona Khan", “i9 Markaz Islamabad", “mk6@gmail.com”, “03245010000”);
 c2.printInformation();
    }
 }
```