Chapter 12

Object-Oriented Programming: Polymorphism

OBJECTIVES

- What polymorphism(多态) is, how it makes programming more convenient, and how it makes systems more extensible and maintainable.
- To declare and use virtual functions(虚函数) to effect polymorphism.
- The distinction between abstract and concrete classes(抽象类和具体类).
- To declare pure virtual functions(纯虚函数) to create abstract classes.

Topics

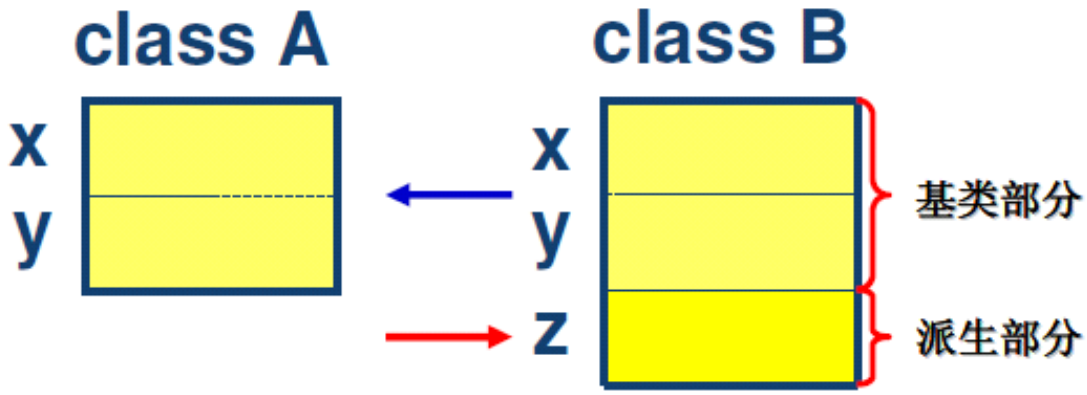
- **12.1 Introduction**
- 12.2 Relationships Among Objects in an Inheritance Hierarchy
- 12.3 Abstract Classes and Pure virtual Functions
- 12.4 Case Study: Payroll System Using Polymorphism

12.1 Introduction—基础

- class B 继承 class A, 即B *is-a* A

```
1. B big;
2. A small = big;
3. A &refSmall = big;
4. A *pSmall = &big;
```

```
1. big = small;
2. B *pBig = &small;
```
错误



12.1 Introduction

- 用户通过键盘输入多个员工信息, 统计收入数据:
- (1) CommissionEmployee

  name, ssn, grossSales, commisionRate
- (2) BasePlusCommissionEmployee

  name, ssn, grossSales, commisionRate, baseSalary
- 用vector或者array来保存指向员工对象的指针
  - CommissionEmployee Pointer
- 希望通过这些指针来调用各自的earnings()函数以进行统计

12.1 Introduction

- **Polymorphism(多态):**
- 通过指向派生类的基类指针, 调用派生类的函数; 将不同的派生类对象都当作基类来处理, 可以屏蔽不同派生类对象之间的差异, 写出通用的代码, 进行通用化编程, 以适应需求的不断变化
- **Virtual Function(虚函数)**
- **Pure Virtual Function(纯虚函数):** 没有给出实现的虚函数
- **Abstract Class(抽象类) vs Concrete Class(具体类)**

Topics

- 12.1 Introduction
- **12.2 Relationships Among Objects in an Inheritance Hierarchy**
- 12.3 Abstract Classes and Pure virtual Functions
- 12.4 Case Study: Payroll System Using Polymorphism

12.2 Relationships Among Objects in an Inheritance Hierarchy

- 12.2.1 Invoking Base-Class Functions from Derived-Class Objects (基类指针指向派生类, 调用基类函数)
- 12.2.2 Aiming Derived-Class Pointers at Base-Class Objects (派生类指针指向基类, 错误)
- 12.2.3 Derived-Class Member-Function Calls via Base-Class Pointers (基类指针指向派生类, 调用派生类函数, 错误)
- 12.2.4 Virtual Functions (应用虚函数, 解决上述问题)
- 12.2.5 Summary of the Allowed Assignments Between Base-Class and Derived-Class Objects and Pointers (基类/派生类对象和指针之间的赋值)

```
12    // create base-class object
13    CommissionEmployee commissionEm
14       "Sue", "Jones", "222-22-222
15
16    // create base-class pointer
17    CommissionEmployee *commissionE
18
19    // create derived-class object
20    BasePlusCommissionEmployee bas
21       "Bob", "Lewis", "333-33-333
22
23    // create derived-class pointe
24    BasePlusCommissionEmployee *ba
35    // aim base-class pointer at b
36    commissionEmployeePtr = &commi
37    cout << "\n\n\nCalling print w
38       << "\nbase-class object inv
39    commissionEmployeePtr->print()
40
41    // aim derived-class pointer a
42    basePlusCommissionEmployeePtr
43    cout << "\n\n\nCalling print w
44       << "\nderived-class object
45       << "print function:\n\n";
46    basePlusCommissionEmployeePtr->print(); // invokes derived-class print
47
48    // aim base-class pointer at derived-class object and print
49    commissionEmployeePtr = &basePlusCommissionEmployee;
50    cout << "\n\n\nCalling print with base-class pointer to "
51       << "derived-class object\ninvokes base-class print "
53    commissionEmployeePtr->print(); // invokes base-class print
```

```
Calling print with base-class pointer to
base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06


Calling print with derived-class pointer to
derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Calling print with base-class pointer to derived-class object
invokes base-class print function on that derived-class object:

commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
```

Notice that the base salary is *not* displayed

12.2 Relationships Among Objects in an Inheritance Hierarchy

- 基类CommissionEmployee

  void print() const;

- 派生类BasePlusCommissionEmployee

  void print() const;

- 通过指向派生类的基类指针, 调用的是基类的函数
- 结论: 对于普通成员函数, 调用基类还是派生类的函数, 取决于句柄的类型, 而不是句柄指向的实际对象类型

12.2 Relationships Among Objects in an Inheritance Hierarchy

```
11    CommissionEmployee *commissionEmployeePtr = nullptr; // base class ptr
12    BasePlusCommissionEmployee basePlusCommissionEmployee(
13       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 ); // derived class
14
15    // aim base-class pointer at derived-class object (allowed)
16    commissionEmployeePtr = &basePlusCommissionEmployee;
17
18    // invoke base-class member functions on derived-class
19    // object through base-class pointer (allowed)
20    string firstName = commissionEmployeePtr->getFirstName();
21    string lastName = commissionEmployeePtr->getLastName();
22    string ssn = commissionEmployeePtr->getSocialSecurityNumber();
23    double grossSales = commissionEmployeePtr->getGrossSales();
24    double commissionRate = commissionEmployeePtr->getCommissionRate();
25
26    // attempt to invoke derived-class-only member functions
27    // on derived-class object through base-class pointer (disallowed)
28    double baseSalary = commissionEmployeePtr->getBaseSalary();
29    commissionEmployeePtr->setBaseSalary( 500 );
```

```
fig12_03.cpp:28:47: error: ‘class CommissionEmployee’ has no member named
   ‘getBaseSalary’
fig12_03.cpp:29:27: error: ‘class CommissionEmployee’ has no member named
   ‘setBaseSalary’
```

12.2 Relationships Among Objects in an Inheritance Hierarchy

- 12.2.1 通过指向派生类的基类指针, 调用的是基类的函数;
- 可否调用派生类自有的函数?

  Compilation Error

- 结论:通过对象句柄, 仅能调用该句柄类型的成员函数

12.2 Relationships Among Objects in an Inheritance Hierarchy

- 派生类指针指向基类对象

Compilation Error

```
1  // Fig. 13.6: fig13_06.cpp
2  // Aiming a derived-class pointer at a base-class object.
3  #include "CommissionEmployee.h"
4  #include "BasePlusCommissionEmployee.h"
5
6  int main()
7  {
8     CommissionEmployee commissionEmployee(
9        "Sue", "Jones", "222-22-2222", 10000, .06 );
10    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
11
12    // aim derived-class pointer at base-class object
13    // Error: a CommissionEmployee is not a BasePlusCommissionEmployee
14    basePlusCommissionEmployeePtr = &commissionEmployee;
15    return 0;
16 } // end main
```

12.2 Relationships Among Objects in an Inheritance Hierarchy

- 解决办法: **downcasting**
- **If the address of a derived-class object (派生类对象地址) has been assigned to a pointer of one of its direct or indirect base classes (基类指针), it is acceptable to cast that base-class pointer back to a pointer of the derived-class type.**

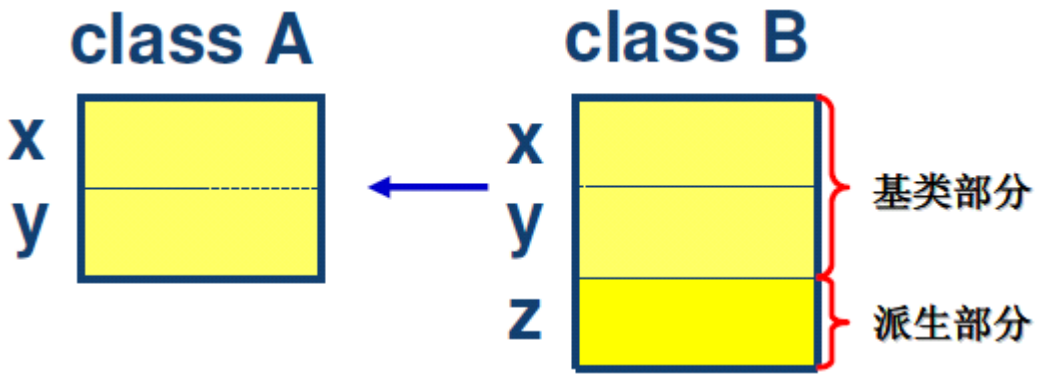
Downcasting

- class B 继承 class A, 即B *is-a* A

1. B big;
2. A small = big;
3. A &refSmall = big; ⟶ B &refBig = (B &) refSmall;
4. A *pSmall = &big; ⟶ B *pBig = (B *) pSmall;

正确

From A to B



Type Fields and switch Statements

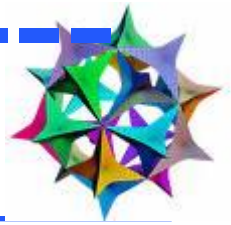
- 如何知道vector中当前的基类指针需要downcast为哪种派生类指针?

12.2.4 Virtual Functions

- With virtual functions, the type of the object being pointed to, not the type of the handle, determines which version of a virtual function to invoke.

- 虚函数: 调用哪个(基类/派生类)虚函数, 由对象类型而不是句柄类型决定.

12.2.4 Virtual Functions ---语法



- 基 类

```
1.  class Shape{
2.  public:
3.      virtual void draw() const;
4.  };
```

- 派生类

```
1.  class Rectangle : public Shape{
2.      virtual void draw() const;
3.  };
```

可省略. 只要基类声明函数为虚函数, 则所有派生类的该函数均为虚函数

12.2.4 Virtual Functions

- 虚函数用于继承结构中的基类和派生类, 以实现多态.
- 派生类中覆盖(Overridden)的虚函数和基类中的虚函数必须函数签名和返回值均相同.
- 函数定义时不需要virtual关键词.

```
1. Rectangle rect;
2. Shape *p = &rect;
3. p->draw();
```

12.2.4 Virtual Functions

- 调用虚函数的两种情况:
- 通过指向派生类的基类指针(或引用)调用, 程序会在执行时(execution time)根据对象类型动态选择合适的派生类函数– 动态绑定(dynamic binding)或延迟绑定(late binding).
- 通过对象名和点操作符调用, 程序在编译时(compile time)即根据对象类型确定函数– 静态绑定(static binding).

```
class CommissionEmployee
{
public:
   CommissionEmployee( const std::string &, const std::string &,
      const std::string &, double = 0.0, double = 0.0 );

   void setFirstName( const std::string & ); // set first name
   std::string getFirstName() const; // return first name

   void setLastName( const std::string & ); // set last name
   std::string getLastName() const; // return last name

   void setSocialSecurityNumber( const std::string & ); // set SSN
   std::string getSocialSecurityNumber() const; // return SSN

   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   void setCommissionRate( double ); // set commission rate
   double getCommissionRate() const; // return commission rate

   virtual double earnings() const; // calculate earnings
   virtual void print() const; // print object
private:
   std::string firstName;
   std::string lastName;
   std::string socialSecurityNu
   double grossSales; // gross
   double commissionRate; // co
}; // end class CommissionEmplo
```

```
class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
   BasePlusCommissionEmployee( const std::string &, const std::string &,
      const std::string &, double = 0.0, double = 0.0, double = 0.0 );

   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

   virtual double earnings() const override; // calculate earnings
   virtual void print() const override; // print object
private:
   double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee
```

```
commissionEmployeePtr = &basePlusCommissionEmployee;
cout << "\n\nCalling virtual function print with base-class pointer"
   << "\nto derived-class object invokes derived-class "
   << "print function:\n\n";

// polymorphism; invokes BasePlusCommissionEmployee's print;
// base-class pointer to derived-class object
commissionEmployeePtr->print();
```

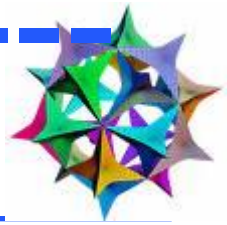
```
Calling virtual function print with base-class pointer
to derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00
```

Notice that the base salary *is* now displayed

12.2.4 Virtual Functions ---

若干限制



- 只有类成员才能声明为虚函数
- 静态成员函数不能是虚函数
- 构造函数不能是虚函数
- 析构函数可以是虚函数

```
class Shape {
public:
    Shape(string s = "") :name(s) { cout << "Shape Constructor" << endl; }
    ~Shape() { cout << "Shape Destructor" << endl; }
    void setN(string s) { name = s; }
    string getN() { return name; }
    double area() { return 0; }
    void print() { cout << "Shape:" << name << endl; }
private:
    string name;
};
class Circle :public Shape {
public:
    Circle(string s, double r) :Shape(s), radius(r) { cout << "Circle Constructor" << e
    ~Circle() { cout << "Circle Destructor" << endl; }
    void setR(double r) { radius = r; }
    double getR() { return radius; }
    double area() { return 3.14 * radius * radius; }
    void print() { Shape::print();  cout << "radius: " << radius << "area: " << area()
private:
    double radius;
};
```

```
class Rectangle :public Shape {
public:
    Rectangle(string s, double w, double l) :Shape(s), width(w), length(l) { cout << "Rect
    ~Rectangle() { cout << "Rectangle Destructor" << endl; }
    double getW() { return width; }
    double area() { return width * length; }
    void print() { Shape:print(); cout << "length: " << length << "width: " << width << "a
private:
    double width, length;
};
class Square :public Rectangle {
public:
    Square(string s, double d) :Rectangle(s, d, d) { cout << "Square Constructor" << endl;
    ~Square() { cout << "Square Destructor" << endl; }
    void print() { Shape::print(); cout << "length: " << getW() << "area: " << area() << e
    Circle inCircle()
    {
        return Circle("circle", getW() / 2);
    }
};
```

12.2.5 Summary of the Allowed Assignments Between Base-Class and Derived-Class Objects and Pointers

	base-class pointer	derived-class pointer
base-class object	OK	ERROR
derived-class object	OK	OK

```
class A{
public:
   void testfuc(){
      func( );
   }

   void func(){
      cout << "A::func called ";
      vfunc();
   }

   virtual void vfunc(){
      cout << "A::vfunc." << endl;
   }
};
```

```
class B : public A{
public:
   void func(){
      cout << "B::func nothing called." << endl;
   }
   virtual void vfunc(){
      cout << "B::vfunc." << endl;
   }
};
```

```
int main()
{
   A a;
   a.func();
   B b;
   b.func();
   b.testfuc();
   A *p = &b;
   p->vfunc();
   p->testfuc();
   p->func();

   return 0;
}
```

```
class A
{
public:
    A() {  aC++; }
    virtual ~A() {  aC--; }
    static int aC;
};
int A::aC = 10;
class B:public A
{
public:
    B() {  aC++; }
    ~B() {  aC--; }
    static int aC;
};
int B::aC = 20;
A a;
B b;
void f(A a) { cout << a.aC << endl; }
```

```
int main()
{
    cout << "A::ac " << a.aC << endl;
    cout << "B::ac " << b.aC << endl;
    A a1;
    A* p = new B();
    cout << "A::ac " << a.aC << endl;
    cout << "B::ac " << b.aC << endl;
    delete p;
    cout << "A::ac " << a.aC << endl;
    cout << "B::ac " << b.aC << endl;
    f(b);
    cout << "A::ac " << a.aC << endl;
    cout << "B::ac " << b.aC << endl;
    return 0;
}
```

Topics

- 12.1 Introduction
- 12.2 Relationships Among Objects in an Inheritance Hierarchy
- **12.3 Abstract Classes and Pure virtual Functions**
- 12.4 Case Study: Payroll System Using Polymorphism

12.3 Abstract Classes and Pure virtual Functions

```
1. class Shape{
2. public:
3.     virtual void draw() const;
4. };
```

- 一些成员函数对于基类来说是没有意义的, 将其声明为虚成员函数的目的是要求派生类给出其实现.

12.3 Abstract Classes and Pure virtual Functions

- **Pure Virtual Function(纯虚函数)**

  A pure virtual function is specified by placing "= 0" in its declaration, as in

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

- **对于纯虚函数, 不需要在类源码中给出其实现.**

12.3 Abstract Classes and Pure virtual Functions

- **Abstract Class(抽象类):** 包含一个或多个纯虚函数的类. 无法实例化, 但可以声明指针和引用, 只能用于继承.
- **1. Shape obj;** // Error, 不能实例化
- **2. Rectangle objRectangle;**
- **3. Shape *ptr = &objRectangle;** // OK, 可指针
- **4. Shape &ref = objRectangle;** // OK, 可引用

- **Concrete Class(具体类):** 不包含纯虚函数, 可以实例化

12.3 Abstract Classes and Pure virtual Functions

- 成员函数是否声明为虚函数, 取决于是否需要多态性支持
- 虚函数是否声明为纯虚函数, 取决于该函数对于当前类是否有意义, 以及当前类是否需要实例化

基类	派生类
虚函数 has an implementation	gives the derived class the option of overriding the function
纯虚函数 does not provide an implementation	requires the derived class to override the function (for that derived class to be concrete; otherwise the derived class remains abstract)

Topics

- 12.1 Introduction
- 12.2 Relationships Among Objects in an Inheritance Hierarchy
- 12.3 Abstract Classes and Pure virtual Functions
- **12.4 Case Study: Payroll System Using Polymorphism**

12.4 Case Study: Payroll System Using Polymorphism

- 目的: 输出各类员工的基本信息和薪金信息
- **Salaried employees (普通薪金制员工)**

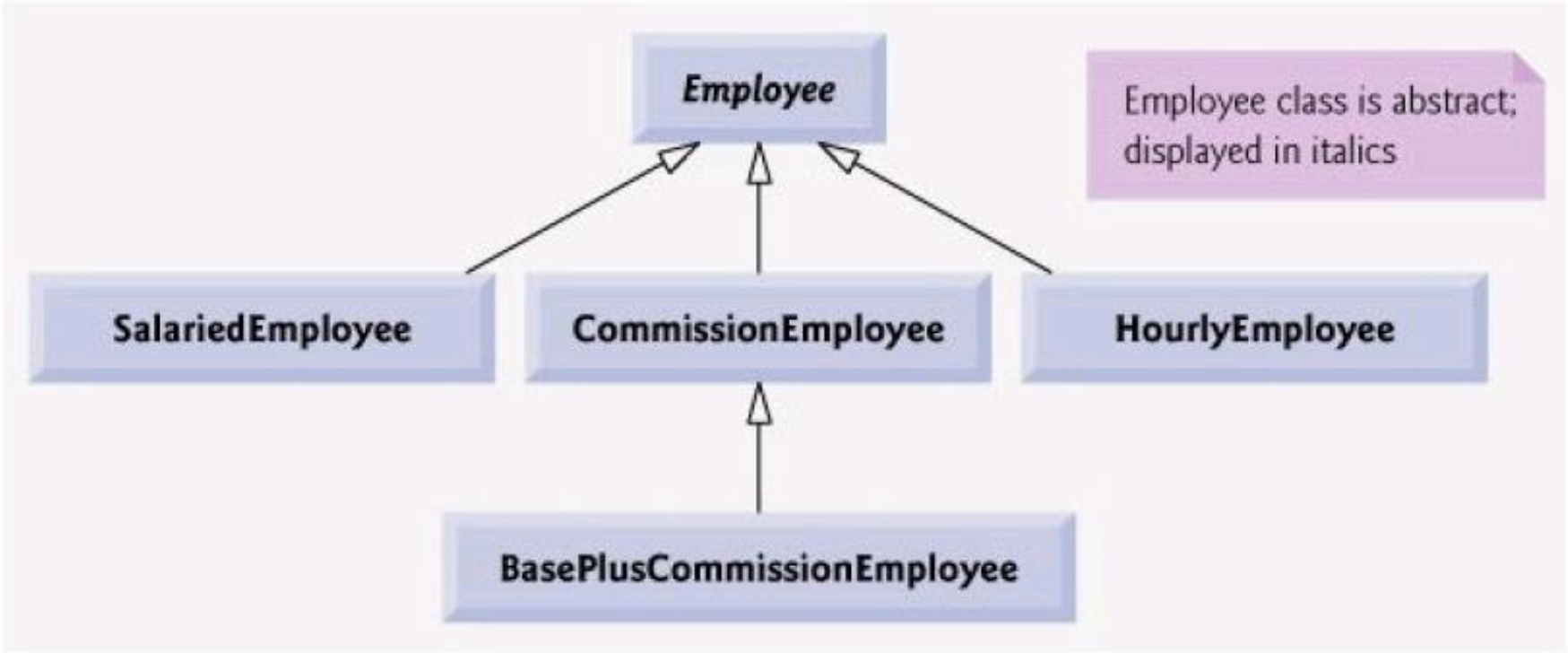
  Name, SSN, Weekly Salary
- **Hourly employees (计时工)**

  Name, SSN, Wage per hour, Hours
- **Commission employees (佣金制员工)**

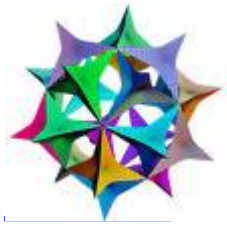
  Name, SSN, Gross sales amount, Commission rate
- **Base-salary-plus-commission employees (带底薪的佣金制员工)**

  Name, SSN, Gross sales amount, Commission rate, Base Salary

12.4 Case Study: Payroll System Using Polymorphism



12.4.1 Creating Abstract Base Class Employee



	earnings	print
Employee	= 0	*firstName lastName* social security number: *SSN*
Salaried-Employee	weeklySalary	salaried employee: *firstName lastName* social security number: *SSN* weekly salary: *weeklysalary*
Hourly-Employee	*If hours <= 40* wage * hours *If hours > 40* (40 * wage) + ((hours - 40) * wage * 1.5)	hourly employee: *firstName lastName* social security number: *SSN* hourly wage: *wage*; hours worked: *hours*
Commission-Employee	commissionRate * grossSales	commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*
BasePlus-Commission-Employee	baseSalary + (commissionRate * grossSales)	base salaried commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*; base salary: *baseSalary*

12.4.1 Creating Abstract Base Class Employee

- Employee Class
- • Name, SSN: 各类员工的共有属性
- • print(): 输出
- • earnings():
  函数

```
class Employee
{
public:
   Employee( const std::string &, const std::string &,
      const std::string & );
   virtual ~Employee() { } // virtual destructor

   void setFirstName( const std::string & ); // set first name
   std::string getFirstName() const; // return first name
```

```
void Employee::print() const
{
   cout << getFirstName() << ' ' << getLastName()
      << "\nsocial security number: " << getSocialSecurityNumber();
} // end function print
```

```
:string & ); // set last name
nst; // return last name

( const std::string & ); // set SSN
yNumber() const; // return SSN
```

```
   // pure virtual function makes Employee an abstract base class
   virtual double earnings() const = 0; // pure virtual
   virtual void print() const; // virtual
private:
   std::string firstName;
   std::string lastName;
   std::string socialSecurityNumber;
}; // end class Employee
```

12.4.1 Creating Abstract Base Class SalariedEmployee

- SalariedEmployee, 继承Employee
- • weeklySalary: 普通薪金制员工的独有属性
- • print(): Override基类函数, 输出基本信息和薪酬信息
- • earnings(): 必须Override基类的纯虚函数. 计算薪酬

```
class SalariedEmployee : public Employee
{
public:
   SalariedEmployee( const std::string &, const std::string &,
      const std::string &, double = 0.0 );
   virtual ~SalariedEmployee() { } // virtual destructor

   void setWeeklySalary( double ); // set weekly salary
   double getWeeklySalary() const; // return weekly salary

   // keyword virtual signals intent to override
   virtual double earnings() const override; // calculate earnings
   virtual void print() const override; // print object
private:
   double weeklySalary; // salary per week
}; // end class SalariedEmployee
```

12.4.1 Creating Abstract Base Class SalariedEmployee

```
class SalariedEmployee : public Employee
{
public:
   SalariedEmployee( const std::string &, const std::string &,
      const std::string &, double = 0.0 );
   virtual ~SalariedEmployee() { } // virtual destructor

   void setWeeklySalary( double ); // set weekly salary
   double getWeeklySalary() const; // return weekly salary

   // keyword virtual signals intent to override
   virtual double earnings() const override; // calculate earnings
   virtual void print() const override; // print object
private:
   double weeklySalary; // salary per week
}; // end class SalariedEmployee
```

```
// constructor
SalariedEmployee::SalariedEmployee( const string &first,
   const string &last, const string &ssn, double salary )
   : Employee( first, last, ssn )
{
   setWeeklySalary( salary );
} // end SalariedEmployee constructor

void SalariedEmployee::print() const
{
   cout << "salaried employee: ";
   Employee::print(); // reuse abstract base-class print function
   cout << "\nweekly salary: " << getWeeklySalary();
} // end function print
```

12.4.1 Creating Abstract Base Class HourlyEmployee

- HourlyEmployee, 继承Employee
- • Wage, hours: 计时工的独有属性
- • print(): Override基类函数, 输出基本信息和薪酬信息
- • earnings(): 必须Override基类的纯虚函数, 计算薪酬

12.4.1 Creating Abstract Base Class CommissionEmployee

- CommissionEmployee, 继承Employee
- • grossSales, commisionRate: 佣金制员工的独有属性
- • print(): Override基类函数, 输出基本信息和酬信息
- • earnings(): 必须算薪酬

```
class CommissionEmployee : public Employee
{
public:
   CommissionEmployee( const std::string &, const std::string &,
      const std::string &, double = 0.0, double = 0.0 );

   virtual ~CommissionEmployee() { } // virtual destructor

   void setCommissionRate( double ); // set commission rate
   double getCommissionRate() const; // return commission rate

   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   // keyword virtual signals intent to override
   virtual double earnings() const override; // calculate earnings
   virtual void print() const override; // print object
private:
   double grossSales; // gross weekly sales
   double commissionRate; // commission percentage
}; // end class CommissionEmployee
```

12.4.1 Creating Abstract Base Class CommissionEmployee

```
class CommissionEmployee : public Employee
{
public:
   CommissionEmployee( const std::string &, const std::string &,
      const std::string &, double = 0.0, double = 0.0 );

   virtual ~CommissionEmployee() { } // virtual destructor

   void setCommissionRate( double ); // set commission rate
   double getCommissionRate() const; // return commission rate

   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   // keyword virtual signals intent to override
   virtual double earnings() const override; // calculate earnings
   virtual void print() const override; // print object
private:
   double grossSales; // gross weekly sales
   double commissionRate; // commission perce
}; // end class CommissionEmployee
```

```
CommissionEmployee::CommissionEmployee( const string &first,
   const string &last, const string &ssn, double sales, double rate )
   : Employee( first, last, ssn )
{
   setGrossSales( sales );
   setCommissionRate( rate );
} // end CommissionEmployee constructor

double CommissionEmployee::earnings() const
{
   return getCommissionRate() * getGrossSales();
} // end function earnings
```

12.4.1 Creating Abstract Base Class BasePlusCommissionEmployee

- **BasePlusCommissionEmployee**, 继承 CommissionEmployee Class, 间接继承Employee Class
- • **baseSalary**: 带底薪的佣金制员工的独有属性
- • **print()**: Override基类CommissionEmployee 函数 输出基本信息
- • **earnings()**: 这

```
class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
   BasePlusCommissionEmployee( const std::string &, const std::string &,
      const std::string &, double = 0.0, double = 0.0, double = 0.0 );
   virtual ~CommissionEmployee() { } // virtual destructor

   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

   // keyword virtual signals intent to override
   virtual double earnings() const override; // calculate earnings
   virtual void print() const override; // print object
private:
   double baseSalary; // base salary per week
}; // end class BasePlusCommissionEmployee
```

12.4.1 Creating Abstract Base Class BasePlusCommissionEmployee

```
class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
   BasePlusCommissionEmployee( const std::string &, const std::string &,
      const std::string &, double = 0.0, double = 0.0, double = 0.0 );
   virtual ~CommissionEmployee() { } // virtual destructor

   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

   // keyword virtual signals intent to override
   virtual double earnings() const override; // calculate earnings
   virtual void print() const override; // print object
private:
   double baseSalary; // base salary per week
}; // end class BasePlusCommissionEmployee
```

```
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
   : CommissionEmployee( first, last, ssn, sales, rate )
{
   setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

double BasePlusCommissionEmployee::earnings() const
{
    return getBaseSalary() + CommissionEmployee::earnings();
} // end function earnings

// print BasePlusCommissionEmployee's information
void BasePlusCommissionEmployee::print() const
{
   cout << "base-salaried ";
   CommissionEmployee::print(); // code reuse
   cout << "; base salary: " << getBaseSalary();
} // end function print
```

12.4.6 Demonstrating Polymorphic Processing

- 实例化四种类型员工, 建立四个对象
- • 通过对象名调用print和earnings函数(静态绑定)
- • 通过基类指针调用print和earnings函数(动态绑定)
- • 通过基类引用调用print和earnings函数(动态绑定)

```
SalariedEmployee salariedEmployee(
   "John", "Smith", "111-11-1111", 800 );
CommissionEmployee commissionEmployee(
   "Sue", "Jones", "333-33-3333", 10000, .06 );
BasePlusCommissionEmployee basePlusCommissionEmployee(
   "Bob", "Lewis", "444-44-4444", 5000, .04, 300 );

cout << "Employees processed individually using static binding:\n\n";

// output each Employee's information and earnings using static binding
salariedEmployee.print();
cout << "\nearned $" << salariedEmployee.earnings() << "\n\n";
commissionEmployee.print();
cout << "\nearned $" << commissionEmployee.earnings() << "\n\n";
basePlusCommissionEmployee.print();
cout << "\nearned $" << basePlusCommissionEmployee.earnings()
   << "\n\n";
```

```
salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
earned $500.00
```

```
// create vector of three base-class pointers
vector< Employee * > employees( 3 );

// initialize vector with pointers to Employees
employees[ 0 ] = &salariedEmployee;
employees[ 1 ] = &commissionEmployee;
employees[ 2 ] = &basePlusCommissionEmployee;

cout << "Employees processed polymorphically via dynamic binding:\n\n";

// call virtualViaPointe
// and earnings using dy
cout << "Virtual functic

for ( const Employee *employeePtr : employees )
   virtualViaPointer( employeePtr );
```

```
void virtualViaPointer( const Employee * const baseClassPtr )
{
   baseClassPtr->print();
   cout << "\nearned $" << baseClassPtr->earnings() << "\n\n";
} // end function virtualViaPointer
```

```
Employees processed polymorphically using dynamic binding:

Virtual function calls made off base-class pointers:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
```

Summary

- 虚函数和多态
- 静态绑定和动态绑定
- 纯虚函数
- 抽象类和具体类

Homework

- 实验必做题目：

  12.12

- 实验选做题目：

  12.15