

TABLE 5.3 Budd’s Inheritance Design Approaches

	Characteristics	Parent Interface	Relationship
Specialization	Redefines behavior	Retained	Is-a
Specification	Completes abstract base	Implemented	Is-a
Extension	Type expansion	Extended	Is-a

Ejercicio de polimorfismo y herencia

- Investigue y explique en una página: que son las referencias en C++ y para que sirve el operador const en c++ en variables, en párametros y en retornos
- Implemente el código que aquí se presenta
- Extienda el código con un nueva clase: Daily Employee

	Earnings	print
DailyEmployee	dailyWage workedDays If workedDays > 0 dailyWage* workedDays	Daily employee: fistName, lastName Social security number: SSN Salary: workedDays: workedDays dailyWage = dailyWage

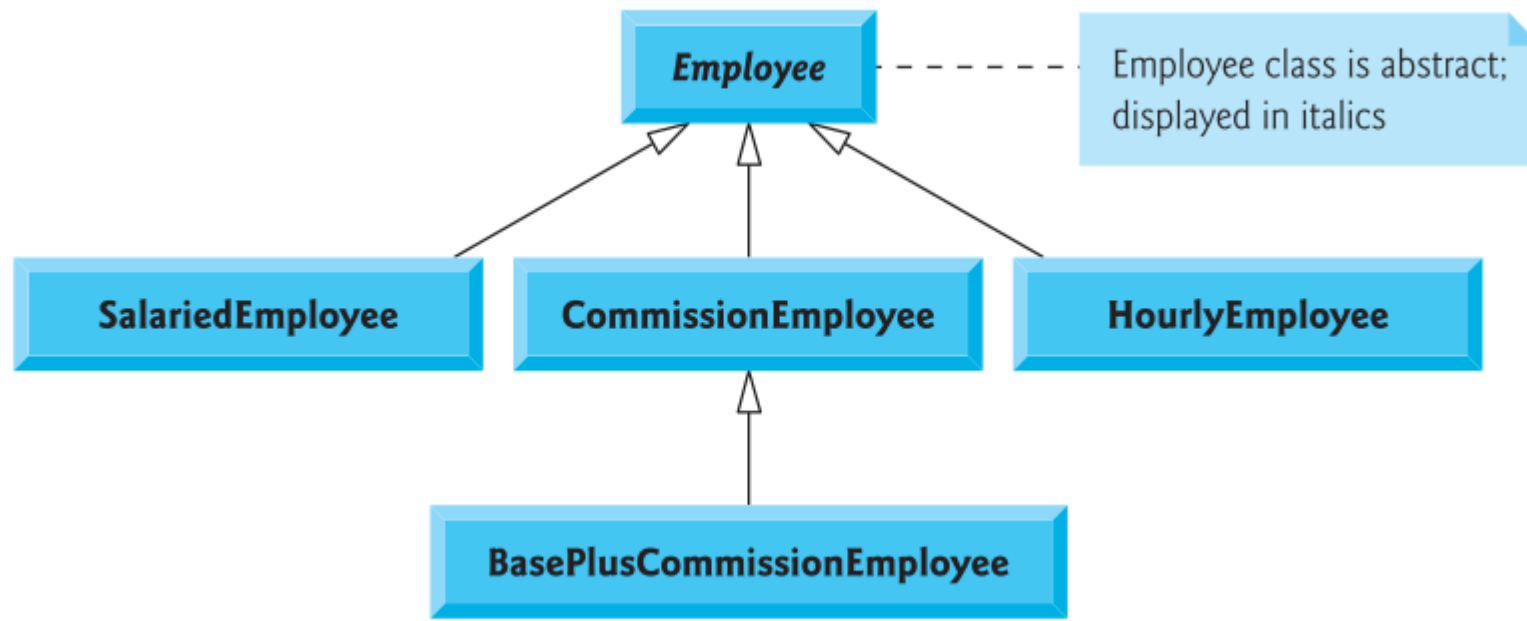


Fig. 9.11.1 Employee Hierarchy

Ejercicio tomado de Deitel, P., & Deitel, H. (2010). *C how to program* (Sixth Edit; Deitel, ed.).

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

Fig. 21.12 | Polymorphic interface for the `Employee` hierarchy classes.

```
1 // Fig. 21.13: Employee.h
2 // Employee abstract base class.
3 #ifndef EMPLOYEE_H
4 #define EMPLOYEE_H
5
6 #include <string> // C++ standard string class
7 using namespace std;
8
9 class Employee
10 {
11 public:
12     Employee( const string &, const string &, const string & );
13
14     void setFirstName( const string & ); // set first name
15     string getFirstName() const; // return first name
16
17     void setLastName( const string & ); // set last name
18     string getLastName() const; // return last name
19
20     void setSocialSecurityNumber( const string & ); // set SSN
21     string getSocialSecurityNumber() const; // return SSN
```

```
22
23     // pure virtual function makes Employee abstract base class
24     virtual double earnings() const = 0; // pure virtual
25     virtual void print() const; // virtual
26 private:
27     string firstName;
28     string lastName;
29     string socialSecurityNumber;
30 }; // end class Employee
31
32 #endif // EMPLOYEE_H
```

Deitel, P., & Deitel, H. (2010). *C how to program* (Sixth Edit; Deitel, ed.).

```

1 // Fig. 21.15: SalariedEmployee.h
2 // SalariedEmployee class derived from Employee.
3 #ifndef SALARIED_H
4 #define SALARIED_H
5
6 #include "Employee.h" // Employee class definition
7
8 class SalariedEmployee : public Employee
9 {

```

g. 21.15 | SalariedEmployee class header file. (Part I of 2.)

21.6 Case Study: Payroll System Using Polymorphism

```

10 public:
11     SalariedEmployee( const string &, const string &,
12                     const string &, double = 0.0 );
13
14     void setWeeklySalary( double ); // set weekly salary
15     double getWeeklySalary() const; // return weekly salary
16
17     // keyword virtual signals intent to override
18     virtual double earnings() const; // calculate earnings
19     virtual void print() const; // print SalariedEmployee object
20 private:
21     double weeklySalary; // salary per week
22 }; // end class SalariedEmployee
23
24 #endif // SALARIED_H

```

```

1 // Fig. 21.16: SalariedEmployee.cpp
2 // SalariedEmployee class member-function definitions.
3 #include <iostream>
4 #include "SalariedEmployee.h" // SalariedEmployee class definition
5 using namespace std;
6
7 // constructor
8 SalariedEmployee::SalariedEmployee( const string &first,
9     const string &last, const string &ssn, double salary )
10     : Employee( first, last, ssn )
11 {
12     setWeeklySalary( salary );
13 } // end SalariedEmployee constructor

```

Fig. 21.16 | SalariedEmployee class implementation file. (Part 1 of 2.)

```

14
15 // set salary
16 void SalariedEmployee::setWeeklySalary( double salary )
17 {
18     weeklySalary = ( salary < 0.0 ) ? 0.0 : salary;
19 } // end function setWeeklySalary
20
21 // return salary
22 double SalariedEmployee::getWeeklySalary() const
23 {
24     return weeklySalary;
25 } // end function getWeeklySalary
26

```

```

27
28 // return salary
29 double SalariedEmployee::getWeeklySalary() const
30 {
31     return weeklySalary;
32 } // end function getWeeklySalary
33
34 // calculate earnings;
35 // override pure virtual function earnings in Employee
36 double SalariedEmployee::earnings() const
37 {
38     return getWeeklySalary();
39 } // end function earnings
40
41 // print SalariedEmployee's information
42 void SalariedEmployee::print() const
43 {
44     cout << "salaried employee: ";
45     Employee::print(); // reuse abstract base-class print function
46     cout << "\nweekly salary: " << getWeeklySalary();
47 } // end function print

```

```
1 // Fig. 21.17: HourlyEmployee.h
2 // HourlyEmployee class definition.
3 #ifndef HOURLY_H
4 #define HOURLY_H
5
6 #include "Employee.h" // Employee class definition
7
8 class HourlyEmployee : public Employee
9 {
10 public:
11     static const int hoursPerWeek = 168; // hours in one week
12
13     HourlyEmployee( const string &, const string &,
14                   const string &, double = 0.0, double = 0.0 );
15
16     void setWage( double ); // set hourly wage
17     double getWage() const; // return hourly wage
18
19     void setHours( double ); // set hours worked
20     double getHours() const; // return hours worked
21
22     // keyword virtual signals intent to override
23     virtual double earnings() const; // calculate earnings
24     virtual void print() const; // print HourlyEmployee object
25 private:
26     double wage; // wage per hour
27     double hours; // hours worked for week
28 }; // end class HourlyEmployee
29
30 #endif // HOURLY_H
```

Fig. 21.17 | HourlyEmployee class header file.


```

1 // Fig. 21.18: HourlyEmployee.cpp
2 // HourlyEmployee class member-function definitions.
3 #include <iostream>
4 #include "HourlyEmployee.h" // HourlyEmployee class definition
5 using namespace std;
6
7 // constructor
8 HourlyEmployee::HourlyEmployee( const string &first, const string &last,
9     const string &ssn, double hourlyWage, double hoursWorked )
10     : Employee( first, last, ssn )
11 {
12     setWage( hourlyWage ); // validate hourly wage
13     setHours( hoursWorked ); // validate hours worked
14 } // end HourlyEmployee constructor
15
16 // set wage
17 void HourlyEmployee::setWage( double hourlyWage )
18 {
19     wage = ( hourlyWage < 0.0 ? 0.0 : hourlyWage );
20 } // end function setWage
21
22 // return wage
23 double HourlyEmployee::getWage() const
24 {
25     return wage;
26 } // end function getWage
27
28 // set hours worked
29 void HourlyEmployee::setHours( double hoursWorked )
30 {
31     hours = ( ( ( hoursWorked >= 0.0 ) &&
32         ( hoursWorked <= hoursPerWeek ) ) ? hoursWorked : 0.0 );
33 } // end function setHours
34

```

```

34 // return hours worked
35 // return hours worked
36 double HourlyEmployee::getHours() const
37 {
38     return hours;
39 } // end function getHours
40
41 // calculate earnings;
42 // override pure virtual function earnings in Employee
43 double HourlyEmployee::earnings() const
44 {
45     if ( getHours() <= 40 ) // no overtime
46         return getWage() * getHours();
47     else
48         return 40 * getWage() + ( ( getHours() - 40 ) * getWage() * 1.5 );
49 } // end function earnings
50
51 // print HourlyEmployee's information
52 void HourlyEmployee::print() const
53 {

```

Fig. 21.18 | HourlyEmployee class implementation file. (Part 1 of 2.)

```

54     cout << "hourly employee: ";
55     Employee::print(); // code reuse
56     cout << "\nhourly wage: " << getWage() <<
57         "; hours worked: " << getHours();
58 } // end function print

```

```
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include "Employee.h" // Employee class definition
7
8  class CommissionEmployee : public Employee
9  {
10 public:
11     CommissionEmployee( const string &, const string &,
12                        const string &, double = 0.0, double = 0.0 );
13
14     void setCommissionRate( double ); // set commission rate
15     double getCommissionRate() const; // return commission rate
16
17     void setGrossSales( double ); // set gross sales amount
18     double getGrossSales() const; // return gross sales amount
19
20     // keyword virtual signals intent to override
21     virtual double earnings() const; // calculate earnings
22     virtual void print() const; // print CommissionEmployee object
23 private:
24     double grossSales; // gross weekly sales
25     double commissionRate; // commission percentage
26 }; // end class CommissionEmployee
27
28 #endif // COMMISSION_H
```

```

1 // Fig. 21.20: CommissionEmployee.cpp
2 // CommissionEmployee class member-function definitions.
3 #include <iostream>
4 #include "CommissionEmployee.h" // CommissionEmployee class definition
5 using namespace std;
6
7 // constructor
8 CommissionEmployee::CommissionEmployee( const string &first,
9     const string &last, const string &ssn, double sales, double rate )
10 : Employee( first, last, ssn )
11 {
12     setGrossSales( sales );
13     setCommissionRate( rate );
14 } // end CommissionEmployee constructor
15
16 // set commission rate
17 void CommissionEmployee::setCommissionRate( double rate )
18 {
19     commissionRate = ( ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0 );
20 } // end function setCommissionRate
21
22 // return commission rate
23 double CommissionEmployee::getCommissionRate() const
24 {
25     return commissionRate;
26 } // end function getCommissionRate
27
28 // set gross sales amount
29 void CommissionEmployee::setGrossSales( double sales )
30 {
31     grossSales = ( ( sales < 0.0 ) ? 0.0 : sales );
32 } // end function setGrossSales
33
34 // return gross sales amount
35 double CommissionEmployee::getGrossSales() const
36 {
37     return grossSales;
38 } // end function getGrossSales
39
40 // calculate earnings; override pure virtual function earnings in Employee
41 double CommissionEmployee::earnings() const
42 {
43     return getCommissionRate() * getGrossSales();
44 } // end function earnings
45
46 // print CommissionEmployee's information
47 void CommissionEmployee::print() const
48 {
49     cout << "commission employee: ";
50     Employee::print(); // code reuse
51     cout << "\ngross sales: " << getGrossSales()
52         << "; commission rate: " << getCommissionRate();
53 } // end function print

```

```

1 // Fig. 21.21: BasePlusCommissionEmployee.h
2 // BasePlusCommissionEmployee class derived from CommissionEmployee.
3 #ifndef BASEPLUS_H
4 #define BASEPLUS_H
5
6 #include "CommissionEmployee.h" // CommissionEmployee class definition
7
8 class BasePlusCommissionEmployee : public CommissionEmployee
9 {
10 public:
11     BasePlusCommissionEmployee( const string &, const string &,
12         const string &, double = 0.0, double = 0.0, double = 0.0 );
13
14     void setBaseSalary( double ); // set base salary
15     double getBaseSalary() const; // return base salary
16
17     // keyword virtual signals intent to override
18     virtual double earnings() const; // calculate earnings
19     virtual void print() const; // print BasePlusCommissionEmployee object
20 private:
21     double baseSalary; // base salary per week
22 }; // end class BasePlusCommissionEmployee
23
24 #endif // BASEPLUS_H

```

```

1 // Fig. 21.22: BasePlusCommissionEmployee.cpp
2 // BasePlusCommissionEmployee member-function definitions.
3 #include <iostream>
4 #include "BasePlusCommissionEmployee.h"
5 using namespace std;
6
7 // constructor
8 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10     double sales, double rate, double salary )
11     : CommissionEmployee( first, last, ssn, sales, rate )
12 {
13     setBaseSalary( salary ); // validate and store base salary
14 } // end BasePlusCommissionEmployee constructor
15
16 // set base salary
17 void BasePlusCommissionEmployee::setBaseSalary( double salary )
18 {
19     baseSalary = ( ( salary < 0.0 ) ? 0.0 : salary );
20 } // end function setBaseSalary
21
22 // return base salary
23 double BasePlusCommissionEmployee::getBaseSalary() const
24 {
25     return baseSalary;
26 } // end function getBaseSalary
27
28 // calculate earnings;
29 // override virtual function earnings in CommissionEmployee
30 double BasePlusCommissionEmployee::earnings() const
31 {
32     return getBaseSalary() + CommissionEmployee::earnings();
33 } // end function earnings
34
35 // print BasePlusCommissionEmployee's information
36 void BasePlusCommissionEmployee::print() const
37 {
38     cout << "base-salaried ";
39     CommissionEmployee::print(); // code reuse
40     cout << "; base salary: " << getBaseSalary();
41 } // end function print

```

```

1 // Fig. 20.15: BasePlusCommissionEmployee.cpp
2 // Class BasePlusCommissionEmployee member-function definitions.
3 #include <iostream>
4 #include "BasePlusCommissionEmployee.h"
5 using namespace std;
6
7 // constructor
8 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
9     const string &first, const string &last, const string &ssn,
10    double sales, double rate, double salary )
11    // explicitly call base-class constructor
12    : CommissionEmployee( first, last, ssn, sales, rate )
13 {
14     setBaseSalary( salary ); // validate and store base salary
15 } // end BasePlusCommissionEmployee constructor
16
17 // set base salary
18 void BasePlusCommissionEmployee::setBaseSalary( double salary )
19 {
20     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
21 } // end function setBaseSalary
22
23 // return base salary
24 double BasePlusCommissionEmployee::getBaseSalary() const
25 {
26     return baseSalary;
27 } // end function getBaseSalary
28
29 // calculate earnings
30 double BasePlusCommissionEmployee::earnings() const
31 {
32     // can access protected data of base class
33     return baseSalary + ( commissionRate * grossSales );
34 } // end function earnings
35

```



```

#include <iostream>
#include <iomanip>
#include <vector>
#include <typeinfo>
#include "Employee.h"
#include "SalariedEmployee.h"
#include "HourlyEmployee.h"
#include "CommissionEmployee.h"
#include "BasePlusCommissionEmployee.h"
using namespace std;

int main()
{
    // set floating-point output formatting
    cout << fixed << setprecision( 2 );

    // create vector of four base-class pointers
    vector < Employee * > employees( 4 );

    // initialize vector with various kinds of Employees
    employees[ 0 ] = new SalariedEmployee(
        "John", "Smith", "111-11-1111", 800 );
    employees[ 1 ] = new HourlyEmployee(
        "Karen", "Price", "222-22-2222", 16.75, 40 );
    employees[ 2 ] = new CommissionEmployee(
        "Sue", "Jones", "333-33-3333", 10000, .06 );
    employees[ 3 ] = new BasePlusCommissionEmployee(
        "Bob", "Lewis", "444-44-4444", 5000, .04, 300 );

    // polymorphically process each element in vector employ
    for ( size_t i = 0; i < employees.size(); i++ )
    {
        employees[ i ]->print(); // output employee informati
        cout << endl;
    }
}

```

```

34 // polymorphically process each element in vector employees
35 for ( size_t i = 0; i < employees.size(); i++ )
36 {
37     employees[ i ]->print(); // output employee information
38     cout << endl;
39
40     // downcast pointer
41     BasePlusCommissionEmployee *derivedPtr =
42         dynamic_cast < BasePlusCommissionEmployee * >
43         ( employees[ i ] );
44

```

Fig. 21.25 | Demonstrating downcasting and runtime type information. (Part 1 of 2.)

824 Chapter 21 Object-Oriented Programming: Polymorphism

```

45 // determine whether element points to base-salaried
46 // commission employee
47 if ( derivedPtr != 0 ) // 0 if not a BasePlusCommissionEmployee
48 {
49     double oldBaseSalary = derivedPtr->getBaseSalary();
50     cout << "old base salary: $" << oldBaseSalary << endl;
51     derivedPtr->setBaseSalary( 1.10 * oldBaseSalary );
52     cout << "new base salary with 10% increase is: $"
53         << derivedPtr->getBaseSalary() << endl;
54 } // end if

55
56 cout << "earned $" << employees[ i ]->earnings() << "\n\n";
57 } // end for

58
59 // release objects pointed to by vector's elements
60 for ( size_t j = 0; j < employees.size(); j++ )
61 {
62     // output class name
63     cout << "deleting object of "
64         << typeid( *employees[ j ] ).name() << endl;
65
66     delete employees[ j ];
67 } // end for

```

