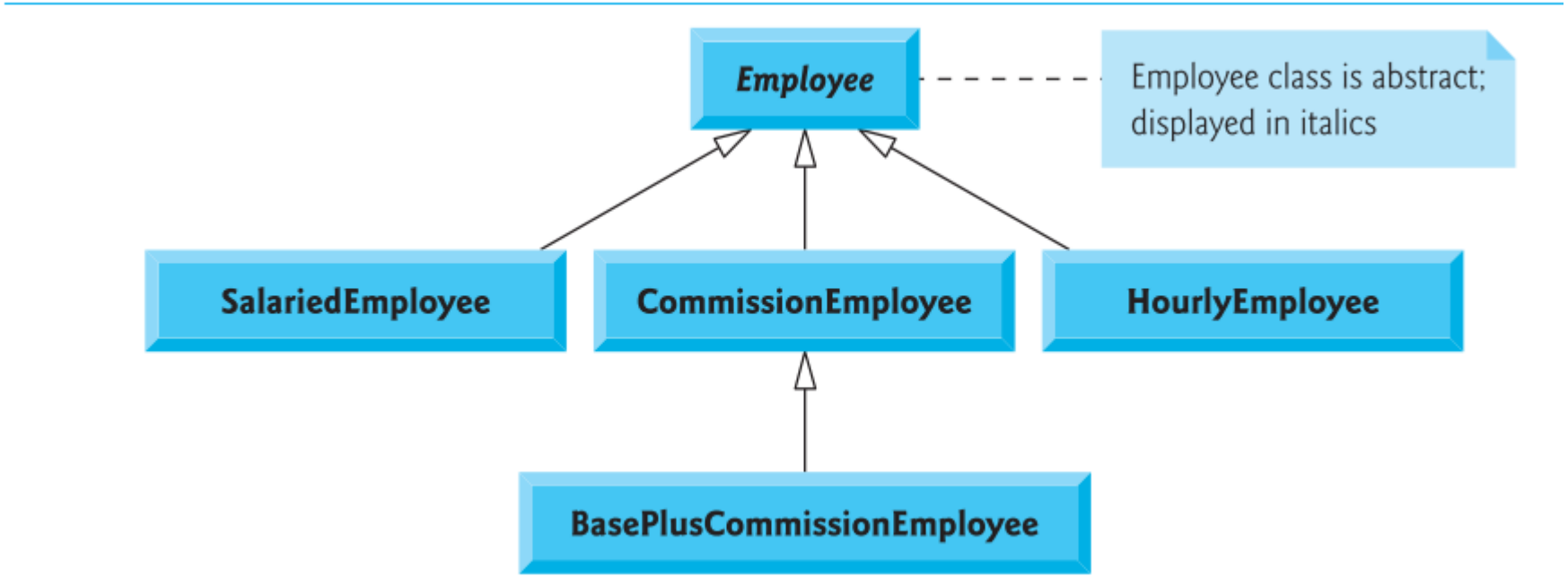
TABLE 5.3 Budd's Inheritance Design Approaches

	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 employe: fistName, lastName Social security number: SSN Salary: workedDays: workedDays dailyWage = dailyWage



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

	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*

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.).

Fig. 21.13 | Employee class header file. (Part 2 of 2.)

```
// Fig. 21.14: Employee.cpp
// Abstract-base-class Employee member-function definitions.
// Note: No definitions are given for pure virtual functions.
#include <iostream>
#include "Employee.h" // Employee class definition
using namespace std;

// constructor
Employee::Employee( const string &first, const string &last,
   const string &ssn )
   : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
{
   // empty body
} // end Employee constructor

// set first name
void Employee::setFirstName( const string &first )
{
   firstName = first;
} // end function setFirstName

// return first name
string Employee::getFirstName() const
{
   return firstName;
} // end function getFirstName

// set last name
void Employee::setLastName( const string &last )
{
   lastName = last;
} // end function setLastName

// return last name
string Employee::getLastName() const
{
   return lastName;
} // end function getLastName

// set social security number
void Employee::setSocialSecurityNumber( const string &ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber

// return social security number
string Employee::getSocialSecurityNumber() const
{
   return socialSecurityNumber;
} // end function getSocialSecurityNumber

// print Employee's information (virtual, but not pure virtual)
void Employee::print() const
{
   cout << getFirstName() << ' ' << getLastName()
      << "\nsocial security number: " << getSocialSecurityNumber();
} // end function print
```

```
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 1 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
```

```
// Fig. 21.16: SalariedEmployee.cpp
// SalariedEmployee class member-function definitions.
#include <iostream>
#include "SalariedEmployee.h" // SalariedEmployee class definition
using namespace std;

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

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


```

// set salary
void SalariedEmployee::setWeeklySalary( double salary )
{
   weeklySalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setWeeklySalary

// return salary
double SalariedEmployee::getWeeklySalary() const
{
   return weeklySalary;
} // end function getWeeklySalary

// return salary
double SalariedEmployee::getWeeklySalary() const
{
   return weeklySalary;
} // end function getWeeklySalary

// calculate earnings;
// override pure virtual function earnings in Employee
double SalariedEmployee::earnings() const
{
   return getWeeklySalary();
} // end function earnings

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

```cpp
// Fig. 21.20: CommissionEmployee.cpp
// CommissionEmployee class member-function definitions.
#include <iostream>
#include "CommissionEmployee.h" // CommissionEmployee class definition
using namespace std;

// constructor
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

// set commission rate
void CommissionEmployee::setCommissionRate( double rate )
{
   commissionRate = ( ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0 );
} // end function setCommissionRate

// return commission rate
double CommissionEmployee::getCommissionRate() const
{
    return commissionRate;
} // end function getCommissionRate

// set gross sales amount
void CommissionEmployee::setGrossSales( double sales )
{
   grossSales = ( ( sales < 0.0 ) ? 0.0 : sales );
} // end function setGrossSales

// return gross sales amount
double CommissionEmployee::getGrossSales() const
{
    return grossSales;
} // end function getGrossSales

// calculate earnings; override pure virtual function earnings in Employee
double CommissionEmployee::earnings() const
{
   return getCommissionRate() * getGrossSales();
} // end function earnings

// print CommissionEmployee's information
void CommissionEmployee::print() const
{
   cout << "commission employee: ";
   Employee::print(); // code reuse
   cout << "\ngross sales: " << getGrossSales()
      << "; commission rate: " << getCommissionRate();
} // end function print
```

```cpp
// Fig. 21.21: BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class derived from CommissionEmployee.
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include "CommissionEmployee.h" // CommissionEmployee class definition

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

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

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

#endif // BASEPLUS_H
```

```cpp
// Fig. 21.22: BasePlusCommissionEmployee.cpp
// BasePlusCommissionEmployee member-function definitions.
#include <iostream>
#include "BasePlusCommissionEmployee.h"
using namespace std;

// constructor
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

// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( ( salary < 0.0 ) ? 0.0 : salary );
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
    return baseSalary;
} // end function getBaseSalary

// calculate earnings;
// override virtual function earnings in CommissionEmployee
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
```

```cpp
// Fig. 20.15: BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
#include "BasePlusCommissionEmployee.h"
using namespace std;

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

// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary

// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   // can access protected data of base class
   return baseSalary + ( commissionRate * grossSales );
} // end function earnings

```

```
// before you can execute this application.
#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;
```

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

      // downcast pointer
      BasePlusCommissionEmployee *derivedPtr =
         dynamic_cast < BasePlusCommissionEmployee * >
            ( employees[ i ] );
```

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


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

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

   // release objects pointed to by vector's elements
   for ( size_t j = 0; j < employees.size(); j++ )
   {
      // output class name
      cout << "deleting object of "
         << typeid( *employees[ j ] ).name() << endl;

      delete employees[ j ];
   } // end for
```