```
// Fig. 12.9: Employee.h
// Employee abstract base class.
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <string> // C++ standard string class

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 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

```

Fig. 12.9 | Employee abstract base class. (Part 1 of 2.)

```cpp
   // 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

#endif // EMPLOYEE_H
```

Fig. 12.9 | Employee abstract base class. (Part 2 of 2.)

```cpp
// Fig. 12.10: 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

```

Fig. 12.10 | Employee class implementation file. (Part 1 of 3.)

```cpp
// 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

```

Fig. 12.10 | Employee class implementation file. (Part 2 of 3.)

```cpp
// 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
```

Fig. 12.10 | Employee class implementation file. (Part 3 of 3.)

12.6.2 Creating Concrete Derived Class `SalariedEmployee`

- Class `SalariedEmployee` (Figs. 12.11–12.12) derives from class `Employee` (line 9 of Fig. 12.11).

```cpp
// Fig. 12.11: SalariedEmployee.h
// SalariedEmployee class derived from Employee.
#ifndef SALARIED_H
#define SALARIED_H

#include <string> // C++ standard string class
#include "Employee.h" // Employee class definition

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

```

Fig. 12.11 | SalariedEmployee class header. (Part 1 of 2.)

```cpp
   // 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

#endif // SALARIED_H
```

Fig. 12.11 | `SalariedEmployee` class header. (Part 2 of 2.)

12.6.2 Creating Concrete Derived Class `SalariedEmployee` (cont.)

***`SalariedEmployee`* Class Member-Function Definitions**

- Figure 12.12 contains the member-function definitions for `SalariedEmployee`.
- The class's constructor passes the first name, last name and social security number to the `Employee` constructor (line 11) to initialize the `private` data members that are inherited from the base class, but not accessible in the derived class.
- Function `earnings` (line 33–36) overrides pure `virtual` function `earnings` in `Employee` to provide a *concrete* implementation that returns the `SalariedEmployee`'s weekly salary.

12.6.2 Creating Concrete Derived Class `SalariedEmployee` (cont.)

- If we did not define `earnings`, class `SalariedEmployee` would be an *abstract* class.
- In class `SalariedEmployee`'s header, we declared member functions `earnings` and `print` as `virtual`
  - This is *redundant.*
- We defined them as `virtual` in base class `Employee`, so they remain `virtual` functions throughout the class hierarchy.

```cpp
// Fig. 12.12: SalariedEmployee.cpp
// SalariedEmployee class member-function definitions.
#include <iostream>
#include <stdexcept>
#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

// set salary
void SalariedEmployee::setWeeklySalary( double salary )
{
   if ( salary >= 0.0 )
      weeklySalary = salary;
   else
      throw invalid_argument( "Weekly salary must be >= 0.0" );
} // end function setWeeklySalary

```

Fig. 12.12 | SalariedEmployee class implementation file. (Part 1 of 2.)

```cpp
// 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
```

Fig. 12.12 | SalariedEmployee class implementation file. (Part 2 of 2.)

12.6.2 Creating Concrete Derived Class `SalariedEmployee` (cont.)

- Function `print` of class `SalariedEmployee` (lines 39–44 of Fig. 12.12) overrides `Employee` function `print`.
- If class `SalariedEmployee` did not override `print`, `SalariedEmployee` would inherit the `Employee` version of `print`.

12.6.3 Creating Concrete Derived Class `CommissionEmployee`

- Class `CommissionEmployee` (Figs. 12.13–12.14) derives from `Employee` (Fig. 12.13, line 9).
- The constructor passes the first name, last name and social security number to the `Employee` constructor (line 11) to initialize `Employee`'s `private` data members.
- Function `print` calls base-class function `print` (line 57) to display the `Employee`-specific information.

```cpp
// Fig. 12.13: CommissionEmployee.h
// CommissionEmployee class derived from Employee.
#ifndef COMMISSION_H
#define COMMISSION_H

#include <string> // C++ standard string class
#include "Employee.h" // Employee class definition

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

```

Fig. 12.13 | CommissionEmployee class header. (Part 1 of 2.)