```
//
//  main.cpp
//  AbsoluteCpp_ch15_1
//
//

//Demonstrates the performance of the virtual function bill.
#include <iostream>
#include "sale.h" //Not really needed, but safe due to ifndef.
#include "discountsale.h"
using std::cout;
using std::endl;
using std::ios;
using namespace SavitchSale;

int main( )
{
    Sale simple(10.00);//One item at $10.00.
    DiscountSale discount(11.00, 10);//One item at $11.00 with a 10%
     discount.

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);

    if (discount < simple)
    {
        cout << "Discounted item is cheaper.\n";
        cout << "Savings is $" << simple.savings(discount) << endl;
    }
    else
        cout << "Discounted item is not cheaper.\n";

    return 0;
}
```

```
//This is the header file sale.h.
//This is the interface for the class Sale.
//Sale is a class for simple sales.

#ifndef SALE_H
#define SALE_H


namespace SavitchSale
{

    class Sale
    {
    public:
        Sale( );
        Sale(double thePrice);
        double getPrice( ) const;
        void setPrice(double newPrice);
        virtual double bill( ) const;
        double savings(const Sale& other) const;
        //Returns the savings if you buy other instead of the calling
         object.
    private:
        double price;
    };

    bool operator < (const Sale& first, const Sale& second);
    //Compares two sales to see which is larger.

}//SavitchSale

#endif // SALE_H
```

```
//This is the file sale.cpp.
//This is the implementation for the class Sale.
//The interface for the class Sale is in the file sale.h.

#include <iostream>
#include "sale.h"
#include <cstdlib>
using std::cout;

namespace SavitchSale
{

    Sale::Sale( ) : price(0)
    {
        //Intentionally empty
    }

    Sale::Sale(double thePrice)
    {
        if (thePrice >= 0)
            price = thePrice;
        else
        {
            cout << "Error: Cannot have a negative price!\n";
            exit(1);
        }
    }

    double Sale::bill( ) const
    {
        return price;
    }

    double Sale::getPrice( ) const
    {
        return price;
    }

    void Sale::setPrice(double newPrice)
    {
        if (newPrice >= 0)
            price = newPrice;
        else
        {
            cout << "Error: Cannot have a negative price!\n";
            exit(1);

        }
    }

    double Sale::savings(const Sale& other) const
    {
        return (bill( ) - other.bill( ));
    }
```

```
    bool operator < (const Sale& first, const Sale& second)
    {
        return (first.bill( ) < second.bill( ));
    }

}//SavitchSale
```

```
//This is the file discountsale.h.
//This is the interface for the class DiscountSale.

#ifndef DISCOUNTSALE_H
#define DISCOUNTSALE_H
#include "sale.h"

namespace SavitchSale
{

    class DiscountSale : public Sale
    {

    public:
        DiscountSale( );
        DiscountSale(double thePrice, double theDiscount);
        //Discount is expressed as a percent of the price.
        //A negative discount is a price increase.
        double getDiscount( ) const;
        void setDiscount(double newDiscount);
        double bill( ) const;
    private:
        double discount;

    };

}//SavitchSale

#endif //DISCOUNTSALE_H
```

```
//This is the implementation for the class DiscountSale.
//This is the file discountsale.cpp.
//The interface for the class DiscountSale is in the header file
 discountsale.h.
#include "discountsale.h"

namespace SavitchSale
{

    DiscountSale::DiscountSale( ) : Sale( ), discount(0)
    {
        //Intentionally empty
    }

    DiscountSale::DiscountSale(double thePrice, double theDiscount)
              : Sale(thePrice), discount(theDiscount)
    {
        //Intentionally empty
    }

    double DiscountSale::getDiscount( ) const
    {
        return discount;
    }

    void DiscountSale::setDiscount(double newDiscount)
    {
        discount = newDiscount;
    }

    double DiscountSale::bill( ) const
    {
        double fraction = discount/100;
        return (1 - fraction)*getPrice( );
    }

}//SavitchSale
```

```
//This is the header file employee.h.
//This is the interface for the abstract class Employee.

#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <string>
using std::string;

namespace SavitchEmployees
{

    class Employee
    {

    public:
        Employee( );
        Employee(string theName, string theSsn);
        string getName( ) const;
        string getSsn( ) const;
        double getNetPay( ) const;
        void setName(string newName);
        void setSsn(string newSsn);
        void setNetPay(double newNetPay);
        virtual void printCheck( ) const = 0;
    private:
        string name;
        string ssn;
        double netPay;

    };

}//SavitchEmployees

#endif //EMPLOYEE_H
```

```
//This is the IMPLEMENTATION FILE: employee.cpp
//This is the IMPLEMENTATION for the class Employee.
//The interface for the class Employee is in the header file employee.h.
#include <string>
#include <cstdlib>
#include <iostream>
#include "employee.h"
using std::string;
using std::cout;

namespace SavitchEmployees
{
    Employee::Employee( ) : name("No name yet"), ssn("No number yet"),
     netPay(0)
    {
        //deliberately empty
    }

    Employee::Employee(string theName, string theNumber) :
        name(theName), ssn(theNumber), netPay(0)
    {
        //deliberately empty
    }

    string Employee::getName( ) const
    {
        return name;
    }

    string Employee::getSsn( ) const
    {
        return ssn;
    }

    double Employee::getNetPay( ) const
    {
        return netPay;
    }

   void Employee::setName(string newName)
    {
        name = newName;
    }

    void Employee::setSsn(string newSsn)
    {
        ssn = newSsn;
    }

    void Employee::setNetPay (double newNetPay)
    {
        netPay = newNetPay;
    }

}//SavitchEmployees
```

```
//
//  main.cpp
//  AbsoluteCpp_ch15_7
//


//Program to illustrate use of a virtual function to defeat the slicing
 problem.
#include <string>
#include <iostream>
using std::string;
using std::cout;
using std::endl;

class Pet
{
public:
    string name;
    virtual void print( ) const;
};

class Dog : public Pet
{
public:
    string breed;
    virtual void print( ) const;
};

int main( )
{
    Dog vdog;
    Pet vpet;

    vdog.name = "Tiny";
    vdog.breed = "Great Dane";
    vpet = vdog;
    cout << "The slicing problem:\n";
    // vpet.breed; //is illegal since class Pet has no member named breed.
    vpet.print( );
    // Dog vdog1 = (Dog)vpet;
    // vdog1.print();
    cout << "Note that it was print from Pet that was invoked.\n";

    cout << "The slicing problem defeated:\n";
    Pet *ppet;
    Dog *pdog;
    pdog = new Dog;

    pdog->name = "Tiny";
    pdog->breed = "Great Dane";
    ppet = pdog;
    ppet->print( );
    pdog->print( );

    //The following, which accesses member variables directly
```

```
    //rather than via virtual functions would produce an error:
    //cout << "name: " << ppet->name << "  breed: "
    //     << ppet->breed << endl;
    //It generates an error message saying
    //class Pet has no member named breed.

    return 0;
}

void Dog::print( ) const
{
    cout << "name: " << name << endl;
    cout << "breed: " << breed << endl;
}

void Pet::print( ) const
{
    cout << "name: " << name << endl;
}
```