// Copying an object from a base class pointer.
// George F. Riley, Georgia Tech, Summer 2006
// ECE3090

// Sometimes, we need to make a copy of an object
// when we have a pointer or a reference to a base class
// of the object. It’s not obvious how to do this correctly
// as the simple solutions are not sufficient.
//
// This example demonstrates the use of an object
// cloning method that solves this problem nicely.

#include <iostream>

using namespace std;

// Define a virtual base class for clonable objects.
class ClonableObject
{
public:
    // Define a pure virtual function called "Clone" method that returns
    // a copy of itself
    virtual ClonableObject* Clone() const = 0;
};

// Define a base class derived from Clon
class Base : public ClonableObject
{
public:
    // All subclasses of Base must define PrintMe (for debugging)
    virtual void PrintMe() const = 0;
};

// Make two distinct subclasses of Base, called A and B
class A : public Base
{
public:
    A(int);
    A(const A&); // Copy constructor
    void PrintMe() const; // Print the object for debugging
    ClonableObject* Clone() const; // Make a clone of the object
private:
    int a;
};

class B : public Base
{
public:
    B(int);
    B(const B&); // Copy constructor
    void PrintMe() const; // Print the object for debugging
    ClonableObject* Clone() const; // Make a clone of the object
private:
    int b;
};
// Implementation of A
A::A(int a1)
: a(a1)
{
}

A::A(const A& old)
: a(old.a)
{ // Copy constructor
}

void A::PrintMe() const
{
    cout << "Hello from A, a is " << a << endl;
}

ClonableObject* A::Clone() const
{ // Return a copy of this A
    return new A(*this);
}

// Implementation of B
B::B(int b1)
: b(b1)
{
}

B::B(const B& old)
: b(old.b)
{ // Copy constructor
}

void B::PrintMe() const
{
    cout << "Hello from B, b is " << b << endl;
}

ClonableObject* B::Clone() const
{ // Return a copy of this B
    return new B(*this);
}

// Subroutine "Sub" has a reference to the Base class as a parameter
void Sub(const Base& base)
{
    // For some reason, Sub needs a copy of the object passed as a parameter
    // However, class "Base" is a virtual base class (due to the PrintMe method)
    // and no objects of class Base can be created. Further, the copy
    // must be either an A or a B, depending on the type of the object
    // passed by the caller. The "Clone" function solves the problem nicely.
    Base* c = (Base*)base.Clone();
    c->PrintMe(); // Print the new object for debugging
}
int main()
{
    A a(1);
    B b(2);
    // Call the Sub method with a and b. Both are valid arguments to
    // Sub since they are subclasses of Base.
    Sub(a);
    Sub(b);
}

// Output from this program is:
// Hello from A, a is 1
// Hello from B, b is 2