```
1 /* Employee.h
2  * CS 106B, Marty Stepp
3  * This file declares the Employee class, the parent (base) class in
4  * our inheritance hierarchy.
5  */
6
7 #pragma once
8
9 #include <string>
10 using namespace std;
11
12 // A class to represent employees in general.
13 class Employee {
14 public:
15     Employee(string name, int years);
16     virtual int hours() const;
17     virtual string name() const;
18     virtual double salary() const;
19     virtual int vacationDays() const;
20     virtual string vacationForm() const;
21     virtual int years() const;
22     // virtual string getFavoritePokemon() = 0;
23
24 private:
25     string myName;
26     int myYears;
27 };
28
```

```
29 /* Employee.cpp
30  * CS 106B, Marty Stepp
31  * This file implements the members of the Employee class, the parent (base)
32  * class in our inheritance hierarchy.
33  */
34
35 #include "Employee.h"
36
37 Employee::Employee(string name, int years) {
38     myName = name;
39     myYears = years;
40 }
41
42 int Employee::hours() const {
43     return 40;
44 }
45
46 string Employee::name() const {
47     return myName;
48 }
49
50 double Employee::salary() const {
51     return 50000.0 + (500 * myYears);
52 }
53
54 int Employee::vacationDays() const {
55     return 10;
56 }
57
58 string Employee::vacationForm() const {
59     return "yellow";
60 }
61
62 int Employee::years() const {
63     return myYears;
64 }
65
```

```
/* Lawyer.h
 * CS 106B, Marty Stepp
 * This file declares the Lawyer class, a subclass
 * (child class, derived class) in our inheritance hierarchy.
 */

#pragma once

#include "Employee.h"
#include <string>

// Employee edna("Edna Smith", 5);
// Lawyer lisa("Lisa Fiedler", "Stanford", 7);
// lisa.salary()

class Lawyer : public Employee {
    // I now have an hours, name, salary, etc. method. yay!
public:
    Lawyer(string name, string lawSchool, int years);
    virtual double salary() const;
    void sue(string person);

private:
    string myLawSchool;
};
```

```
/* Lawyer.cpp
 * CS 106B, Marty Stepp
 * This file implements the members of the Lawyer class, a subclass
 * (child class, derived class) in our inheritance hierarchy.
 */

#include <iostream>
#include "Lawyer.h"

// call the constructor of Employee superclass?
Lawyer::Lawyer(string name, string lawSchool, int years)
        : Employee(name, years) {
    myLawSchool = lawSchool;
}

// overriding: replace version from Employee class
double Lawyer::salary() const {
    return Employee::salary() * 2;
}

void Lawyer::sue(string person) {
    cout << "See you in court, " << person << endl;
}
```

```
/* Programmer.h
 * CS 106B, Marty Stepp
 * This file declares the Programmer class, a subclass
 * (child class, derived class) in our inheritance hierarchy.
 */

#pragma once

#include "Employee.h"

class Programmer : public Employee {
public:
    Programmer(string name, int years);
    virtual string vacationForm() const;
    virtual void code();
};


```

```
/* Programmer.cpp
 * CS 106B, Marty Stepp
 * This file implements the members of the Programmer class, a subclass
 * (child class, derived class) in our inheritance hierarchy.
 */

#include <iostream>
#include "Programmer.h"

Programmer::Programmer(string name, int years) : Employee(name, years) {}

string Programmer::vacationForm() const {
    return "pink";
}

void Programmer::code() {
    cout << "I'm coding up a storm!" << endl;
}

```

```
/* polymorphism.h
 * CS 106B, Marty Stepp
 * This file declares several classes in an inheritance hierarchy.
 * We write the method implementations in the .h file for brevity
 * rather than separating it into a .h and .cpp.
 */

#pragma once

#include <iostream>
using namespace std;

class Snow {
public:
    virtual void method2() {
        cout << "Snow 2" << endl;
    }

    void method3() {
        cout << "Snow 3" << endl;
    }
};

class Rain : public Snow {
public:
    virtual void method1() {
        cout << "Rain 1" << endl;
    }

    virtual void method2() {
        cout << "Rain 2" << endl;
    }
};

class Sleet : public Snow {
public:
    virtual void method2() {
        cout << "Sleet 2" << endl;
        Snow::method2();
        method3();
    }

    void method3() {
        cout << "Sleet 3" << endl;
    }
};

class Fog : public Sleet {
public:
    virtual void method1() {
        cout << "Fog 1" << endl;
    }

    void method3() {
        cout << "Fog 3" << endl;
    }
};
```

```
/* polymorphism.cpp
 * CS 106B, Marty Stepp
 * This client program creates several variables of the classes in our
 * inheritance hierarchy and then calls methods on them.
 */

#include <iostream>
#include "console.h"
#include "polymorphism.h"
#include "simpio.h"
using namespace std;

int main() {
    int choice = -1;
    while (choice != 0) {
        choice = getInteger("Example to run (1-7, 0 to quit): ");

        if (choice == 1) {
            // Example 1
            cout << "Example 1:" << endl;
            Snow* var1 = new Sleet();
            var1->method2();

            cout << endl;
        }

        if (choice == 2) {
            // Example 2
            cout << "Example 2:" << endl;
            Snow* var2 = new Rain();
            //var2->method1();

            cout << endl;
        }

        if (choice == 3) {
            // Example 3
            cout << "Example 3:" << endl;
            Snow* var3 = new Rain();
            var3->method2();

            cout << endl;
        }

        if (choice == 4) {
            // Example 4
            cout << "Example 4:" << endl;
            Snow* var4 = new Rain();
            ((Rain*) var4)->method1();

            cout << endl;
        }

        if (choice == 5) {
            // Example 5
            cout << "Example 5:" << endl;
            Snow* var5 = new Fog();
            // ((Sleet*) var5)->method1();

            cout << endl;
        }

        if (choice == 6) {
            // Example 6
            cout << "Example 6:" << endl;
            Snow* var6 = new Sleet();
            //((Rain*) var6)->method4();

            cout << endl;
        }

        if (choice == 7) {
            // Example 7
            cout << "Example 7:" << endl;
            Snow* var7 = new Sleet();
            ((Rain*) var7)->method1();

            cout << endl;
        }

        cout << "Finished Example " << choice << endl << endl;
    }
    cout << "Done!  Exiting." << endl;
    return 0;
}
```