

```
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
```

```

66 /* Lawyer.h
67  * CS 106B, Marty Stepp
68  * This file declares the Lawyer class, a subclass
69  * (child class, derived class) in our inheritance hierarchy.
70  */
71
72 #pragma once
73
74 #include "Employee.h"
75 #include <string>
76
77 // Employee edna("Edna Smith", 5);
78 // Lawyer lisa("Lisa Fiedler", "Stanford", 7);
79 // lisa.salary()
80
81 class Lawyer : public Employee {
82     // I now have an hours, name, salary, etc. method. yay!
83 public:
84     Lawyer(string name, string lawSchool, int years);
85     virtual double salary() const;
86     void sue(string person);
87
88 private:
89     string myLawSchool;
90 };
91
92 _____
93
94 /* Lawyer.cpp
95  * CS 106B, Marty Stepp
96  * This file implements the members of the Lawyer class, a subclass
97  * (child class, derived class) in our inheritance hierarchy.
98  */
99
100 #include <iostream>
101 #include "Lawyer.h"
102
103 // call the constructor of Employee superclass?
104 Lawyer::Lawyer(string name, string lawSchool, int years)
105     : Employee(name, years) {
106     myLawSchool = lawSchool;
107 }
108
109 // overriding: replace version from Employee class
110 double Lawyer::salary() const {
111     return Employee::salary() * 2;
112 }
113
114 void Lawyer::sue(string person) {
115     cout << "See you in court, " << person << endl;
116 }

```

```
117 /* Programmer.h
118 * CS 106B, Marty Stepp
119 * This file declares the Programmer class, a subclass
120 * (child class, derived class) in our inheritance hierarchy.
121 */
122
123 #pragma once
124
125 #include "Employee.h"
126
127 class Programmer : public Employee {
128 public:
129     Programmer(string name, int years);
130     virtual string vacationForm() const;
131     virtual void code();
132 };
133
134
135 _____
136
137 /* Programmer.cpp
138 * CS 106B, Marty Stepp
139 * This file implements the members of the Programmer class, a subclass
140 * (child class, derived class) in our inheritance hierarchy.
141 */
142
143 #include <iostream>
144 #include "Programmer.h"
145
146 Programmer::Programmer(string name, int years) : Employee(name, years) {}
147
148 string Programmer::vacationForm() const {
149     return "pink";
150 }
151
152 void Programmer::code() {
153     cout << "I'm coding up a storm!" << endl;
154 }
155
```

```
156 /* polymorphism.h
157 * CS 106B, Marty Stepp
158 * This file declares several classes in an inheritance hierarchy.
159 * We write the method implementations in the .h file for brevity
160 * rather than separating it into a .h and .cpp.
161 */
162
163 #pragma once
164
165 #include <iostream>
166 using namespace std;
167
168 class Snow {
169 public:
170     virtual void method2() {
171         cout << "Snow 2" << endl;
172     }
173
174     void method3() {
175         cout << "Snow 3" << endl;
176     }
177 };
178
179 class Rain : public Snow {
180 public:
181     virtual void method1() {
182         cout << "Rain 1" << endl;
183     }
184
185     virtual void method2() {
186         cout << "Rain 2" << endl;
187     }
188 };
189
190 class Sleet : public Snow {
191 public:
192     virtual void method2() {
193         cout << "Sleet 2" << endl;
194         Snow::method2();
195         method3();
196     }
197
198     void method3() {
199         cout << "Sleet 3" << endl;
200     }
201 };
202
203 class Fog : public Sleet {
204 public:
205     virtual void method1() {
206         cout << "Fog 1" << endl;
207     }
208
209     void method3() {
210         cout << "Fog 3" << endl;
211     }
212 };
```

```

213 /* polymorphism.cpp
214 * CS 106B, Marty Stepp
215 * This client program creates several variables of the classes in our
216 * inheritance hierarchy and then calls methods on them.
217 */
218
219 #include <iostream>
220 #include "console.h"
221 #include "polymorphism.h"
222 #include "simpio.h"
223 using namespace std;
224
225 int main() {
226     int choice = -1;
227     while (choice != 0) {
228         choice = getInteger("Example to run (1-7, 0 to quit): ");
229
230         if (choice == 1) {
231             // Example 1
232             cout << "Example 1:" << endl;
233             Snow* var1 = new Sleet();
234             var1->method2();
235
236             cout << endl;
237         }
238
239         if (choice == 2) {
240             // Example 2
241             cout << "Example 2:" << endl;
242             Snow* var2 = new Rain();
243             //var2->method1();
244
245             cout << endl;
246         }
247
248         if (choice == 3) {
249             // Example 3
250             cout << "Example 3:" << endl;
251             Snow* var3 = new Rain();
252             var3->method2();
253
254             cout << endl;
255         }
256
257         if (choice == 4) {
258             // Example 4
259             cout << "Example 4:" << endl;
260             Snow* var4 = new Rain();
261             ((Rain*) var4)->method1();
262
263             cout << endl;
264         }
265
266         if (choice == 5) {
267             // Example 5
268             cout << "Example 5:" << endl;
269             Snow* var5 = new Fog();
270             // ((Sleet*) var5)->method1();
271
272             cout << endl;
273         }
274
275         if (choice == 6) {
276             // Example 6
277             cout << "Example 6:" << endl;
278             Snow* var6 = new Sleet();
279             //((Rain*) var6)->method4();
280
281             cout << endl;
282         }
283
284         if (choice == 7) {
285             // Example 7
286             cout << "Example 7:" << endl;
287             Snow* var7 = new Sleet();
288             ((Rain*) var7)->method1();
289
290             cout << endl;
291         }
292
293         cout << "Finished Example " << choice << endl << endl;
294     }
295     cout << "Done! Exiting." << endl;
296     return 0;
297 }

```