

CS 12 • Winter 2003

In-lab Programming Exercise 4

Class Inheritance & Polymorphism

wagner@cs.ucr.edu
Department of Computer Science and Engineering
University of California, Riverside

February 24, 2003

Consider a graphics system that has classes for various shapes such as squares, circles, and so on. For instance, a square might have data members `side_length` and `center_point` (an instance of class `Point` — see below), while a circle might have only a `center_point` and a `radius`. In a well-designed system, these would be derived from a common class, `Shape`. Your task in this week's in-lab programming exercise is to implement such a system, in a simplified way.

The class `Shape` is the base class while both of the other shape classes (`Square` and `Circle`) are derived classes. These classes all have member functions `center`, `erase`, and `draw`, which — since this is a simplified application — will not really do any work but simply should output a message telling which function was called and what the class type of the object calling it is. The function `center`, however, should also call `erase` and `draw`, in addition to displaying the message referred to above. All three functions should take no arguments and return no value.

There are **four** parts to this assignment:

- Write a UML description of each class, indicating which class it's derived from (if any) and which other classes it uses. Since you'll be creating ASCII files, you won't be able to create fancy diagrams, so you should write your UML descriptions as shown in the diagram below. Make sure to include the appropriate access modifier indicators (- for private, + for public, and # for protected), as well as all the necessary constructors, destructors, member functions, and operators. Only include the *necessary* members; do *not* add everything and the kitchen sink to each class. Also, make sure to think carefully where the member functions `draw`, `erase`, and `center` are declared and/or

defined. Make sure to overload the insertion operator `<<` in each class so as to output the appropriate information.

```
*****
*           Point           *
*****
*   - int horiz_coord      *
*   - int vert_coord       *
*****
*   + Point()              *
*   + Point(int h, int v)  *
*   + int getHorizCoord()  *
*   + int getVertCoord()   *
*   friend operator +      *
*   friend operator -      *
*   friend operator <<     *
*****
```

- Create a main file named `not_virtual.cpp` and write your classes using no virtual functions. Compile and test.
- Now create another main file, named `virtual.cpp`, and write your classes using virtual functions in the base class. Compile and test. This step should require only minor changes to the classes from the previous step.
- Explain the difference in the results, if any.

Don't worry about writing the class interfaces and the class implementations in different files. Instead, for each of the two cases (virtual/not virtual), write everything (UML descriptions, all classes, the main function below, and the explanations) in one file, as described. Thus, you should turn in **two** files. Good luck!

```
#include <iostream>
```

```
int main()
{
    int length = 5;
    Point center_pt(10, 18);

    Square sq(length, center_pt);
    sq.draw();
}
```

```

std::cout << std::endl << "Object_" << sq <<
    ",_of_derived_class_Square,_about_to_call_center()."
    << std::endl;
sq.center();

int radius = 4;
center_pt = Point(7, 4);

Circle circ(radius, center_pt);
circ.draw();
std::cout << std::endl << "Object_" << circ <<
    ",_of_derived_class_Circle,_about_to_call_center()."
    << std::endl;
circ.center();

return 0;
}

```

■