Inheritance

- One of the most important concepts in object-oriented programming is that of inheritance.
- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
- This also provides an opportunity to reuse the code functionality and fast implementation time.
- When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class.
- This existing class is called the ***base*** or ***parent*** class, and the new class is referred to as the ***derived*** or ***extending*** class.
- Here is an example `Shape` base class that keeps track of x and y location.

```
//
// Base class
//
class Shape
{
      public:
            Shape(int x, int y)   // constructor
            {
                  _x = x;
                  _y = y;
            }

            int getX() {
                  return _x;
            }

            int getY() {
                  return _y;
            }

            float getArea()
            {
                  cout << "getArea():";
                  return 0;
            }

      private:
            int _x;
            int _y;
};
```

- This example `Rectangle` derived class extends `Shape`, taking advantage of the x and y location already in `Shape`. Note the constructor for `Rectangle` must call the `Shape` constructor using `: Shape(x, y)`. `Rectangle` adds functionality having its own width and height variables that are used in a `getArea()` method.

```
    // Derived class
    class Rectangle: public Shape
    {
          public:
                Rectangle(int x, int y,     // constructor
                      int width, int height) : Shape(x, y)
                {
                      _width = width;
                      _height = height;
                }

                float getArea()
                {
                      cout << "Rectangle getArea():";
                      return _width*_height;
                }
          private:
                int _width;
                int _height;
    };
```

- Similarly, this example `Circle` derived class extends `Shape`, taking advantage of the x and y location. `Circle`, like `Rectangle`, must call the `Shape` constructor using `: Shape(x, y)`. `Circle` adds functionality having its own radius variable that is used in a `getArea()` method.

```
    const float PI = 3.14159265;
    // Derived class
    class Circle: public Shape
    {
          public:
                Circle(int x, int y,        // constructor
                      int radius) : Shape(x, y)
                {
                      _radius = radius;
                }

                float getArea()
                {
                      cout << "Circle getArea():";
                      return PI*_radius*_radius;
                }
          private:
                int _radius;
    };
```

- This `main()` example creates instances called `spot`, `rect`, and `circ` of each of the above classes and prints the various location and area of those instances.

```
int main(void)
{
      Shape spot(6, 6);
      Rectangle rect(5, 7, 8, 8);
      Circle circ(2, 3, 4);

      cout  << "spot- ("
            << spot.getX() << ","
            << spot.getY() << ") "
            << spot.getArea() << endl;

      cout  << "rect- ("
            << rect.getX() << ","
            << rect.getY() << ") "
            << rect.getArea() << endl;

      cout  << "circ- ("
            << circ.getX() << ","
            << circ.getY() << ") "
            << circ.getArea() << endl;

      return 0;
}
```