#pragma once

#include "Shape.h"

class Square : public Shape {
public:
    double getArea() const;

private:
    // Nothing!
};

class Shape {
public:
    Shape();
    Shape(double length);
    double getLength() const;

private:
    double length_;
Derived Classes

[Public Members of the Base Class]:

```cpp
int main() {
    Square sq;
    sq.getLength(); // Returns 1, the length init'd
    // by Shape's default ctor
    ...
    ...
}
```

[Private Members of the Base Class]:
Polymorphism

Object-Orientated Programming (OOP) concept that a single object may take on the type of any of its base types.
Virtual
Cube.cpp

```cpp
// No print_1() in Cube.cpp

Cube::print_1() {
    cout << "Cube" << endl;
}

cube::print_2() {
    cout << "Cube" << endl;
}

virtual Cube::print_3() {
    cout << "Cube" << endl;
}

virtual Cube::print_4() {
    cout << "Cube" << endl;
}

// In .h file:
virtual Cube::print_5() = 0;
```

RubikCube.cpp

```cpp
// No print_1() in RubikCube.cpp

RubikCube::print_2() {
    cout << "Rubik" << endl;
}

cube::print_4() {
    cout << "Cube" << endl;
}

// In .h file:
RubikCube::print_5() {
    cout << "Rubik" << endl;
```
## Runtime of Virtual Functions

<table>
<thead>
<tr>
<th>virtual-main.cpp</th>
<th>Cube c;</th>
<th>RubikCube c;</th>
<th>RubikCube rc; Cube &amp;c = rc;</th>
</tr>
</thead>
<tbody>
<tr>
<td>c.print_1();</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_2();</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_3();</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_4();</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.print_5();</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why Polymorphism?
class Animal {
  public:
    void speak() {
    }
};

class Dog : public Animal {
  public:
    void speak() {
    }
};

class Cat : public Animal {
  public:
    void speak() {
    }
};
Abstract Class:

[Requirement]:

[Syntax]:

[As a result]:
class Cube {
  public:
  ~Cube();
};

class RubikCube : public Cube {
  public:
  ~RubikCube();
};
class PNG {
  public:
    PNG();
    PNG(unsigned int width, unsigned int height);
    PNG(PNG const & other);
    ~PNG();

    PNG & operator= (PNG const & other);
    bool operator== (PNG const & other) const;

    bool readFromFile(string const & fileName);
    bool writeToFile(string const & fileName);
    HSLAPixel & getPixel(unsigned int x, unsigned int y) const;
    unsigned int width() const;
  // ...

  private:
    unsigned int width_; 
    unsigned int height_; 
    HSLAPixel * imageData_; 
    void _copy(PNG const & other);
};
List ADT
What types of “stuff” do we want in our list?
Templates
T maximum(T a, T b) {
    T result;
    result = (a > b) ? a : b;
    return result;
}
#pragma once

class List {
  public:
  private:
};

#endif