Inheritance
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

The idea that a single interface may take multiple types or that a single symbol may be different types.

In Object-Orientated Programming (OOP) a key example is that a single object may take on the type of any of its base types.
Virtual
Method Dispatch

1) Look at the type the method is called on
2) Look for the method in that type if found
   A. If type is virtual use runtime type and goto 2 ignoring virtual from now on
   B. Use method that method
3) No method found change to base type and goto 2
Cube.cpp

1 Cube::print_1() {
2     cout << "Cube" << endl;
3 }
4
5 Cube::print_2() {
6     cout << "Cube" << endl;
7 }
8
9 virtual Cube::print_3() {
10     cout << "Cube" << endl;
11 }
12
13 virtual Cube::print_4() {
14     cout << "Cube" << endl;
15 }
16
17 Cube::print_5() = 0;

Cube.h

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

RubikCube.cpp

1 // No print_1() in RubikCube.cpp
2
3
4
5 RubikCube::print_2() {
6     cout << "Rubik" << endl;
7 }
8
9 // No print_3() in RubikCube.cpp
10
11
12
13 RubikCube::print_4() {
14     cout << "Rubik" << endl;
15 }
16
17 RubikCube::print_5() {
18     cout << "Rubik" << endl;
19 }
## 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>
Abstract Class:

[Requirement]:

[Syntax]:

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

class RubikCube : public Cube {
public:
    ~RubikCube();
};
Templates
template1.cpp

1
2
3 T maximum(T a, T b) {
4 T result;
5 result = (a > b) ? a : b;
6 return result;
7 }
#pragma once

class List {
    public:

    private:

};

#endif