CS 225

#6: C++ Overloading and Inheritance

February 2, 2022 · *G Carl Evans*

Inheritance

In nearly all object-oriented languages (including C++), classes can be extended to build other classes. We call the class being extended the **base class** and the class inheriting the functionality the **derived class**.

Shape.h

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

  private:
    double length_;
};
```

Square.h

```
#include "Shape.h"

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

  private:
    // Nothing!
};
```

In the code, **`Square`** is derived from the base class **`Shape`**:

- All **public** functionality of **`Shape`** is part of **`Square`**:

main.cpp

```
5  int main() {
6    Square sq;
7    sq.getLength(); // Returns 1, the len init'd
8                    // by Shape's default ctor
…    ...
```

- [Private Members of **`Shape`**]:

Virtual

- The **virtual** keyword allows us to override the behavior of a class by its derived type.

Example:

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 print_5() = 0;
```

RbikCube.cpp

```
// No print_1()


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

// No print_3()


RubikCube::print_4() {
  cout << "Rubik" << endl;
}

RubikCube::print_5() {
  cout << "Rubik" << endl;
}
```

	`Cube c;`	`RubikCube c;`	`RubikCube rc;` `Cube &c = rc;`
`c.print_1();`			
`c.print_2();`			
`c.print_3();`			
`c.print_4();`			
`c.print_5();`			

Polymorphism

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

- A **RubikCube** may polymorph itself to a Cube
- A Cube cannot polymorph to be a **RubikCube** *(base types only)*

Pure Virtual Methods

In `Cube`, `print_5()` is a **pure virtual** method:

Cube.h

```
1  virtual Cube::print_5() = 0;
```

A pure virtual method does not have a definition and makes the class and **abstract class**.

C++ Templates:

1.

2.

3.

Templated Functions:

functionTemplate1.cpp

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

Where to put templated code?

Templated Classes:

List.h

```
1  #pragma once
2
3
4  class List {
5    public:
6
7
8
9
10
11
12   private:
13
14
15 };
```

List.hpp

```
1
2
3
4
5
```

CS 225 – Things To Be Doing:

1. mp_stickers due next Monday
2. lab_intro extended deadline Sunday
3. new lab released today also due Sunday
4. Daily POTDs