CS 225 Spring 2019 :: TA Lecture Notes
1/28 Lifecycle

By Wenjie

- **Copy Constructors**
  - Automatic copy constructor
    - Generated if we do not define a copy constructor
    - Copy every instance variable in the object
  - Custom copy constructor
    - pass by reference

```
Cube(const Cube & other){
   ...
};
```

- **Calls to constructors**
  - Copy constructor is called every time when a Cube is copied by value

Constructors	joinCube(Cube c1, Cube c2) {...} By value	joinCube(Cube * c1, Cube * c2) {...} By pointer	joinCube(Cube & c1, Cube & c2) {...} By reference
Cube();	0	0	0
Cube(double length);	1: Cube result(newLength)	1: Cube result(newLength)	1: Cube result(newLength)

Cube(Cube & other)
(copy constructor)
	2: joinCube(Cube c1, Cube c2);
	return result;
		1: return result;	1: return result;

  - In this example below, the copy constructor is called when the parameter is passed in, and when the result is returned. (highlighted)

joinCubes-byValue.cpp

```
15  Cube joinCubes(Cube c1, Cube c2) {
16    double totalVolume = c1.getVolume() + c2.getVolume();
...   ...
20    Cube result(newLength);
21    return result;
22  }
```

CS 225 Spring 2019 :: TA Lecture Notes

1/28 Lifecycle

By Wenjie

- Constructor Initializer list (highlighted below)
    - Required if you have reference variables
    - It tells the compiler to (shallow) copy instance variables to the variables in "other"
    - In this case
        - cube_ = other.cube_
        - ptr_ = other.ptr_
        - ref_ becomes an alias of other.ref_
    - Then nothing is needed in the body, since all variables are copied

Tower.h

```
#pragma once
#include "cs225/Cube.h"
using cs225::Cube;
class Tower {
  public:
    Tower(Cube c, Cube *ptr, const Cube &ref);
                                  // Custom constructor
    Tower(const Tower & other);
                                  // Copy constructor

  private:
    Cube cube_;
    Cube *ptr_;
    const Cube &ref;
};
```

Tower.cpp

```
Tower::Tower(const Tower & other) : cube_(other.cube_),
ptr_(other.ptr_), ref_(other.ref_) {
    //every variable copied
    //nothing needed in the body
}
```

- Deep Copy Constructor

CS 225 Spring 2019 :: TA Lecture Notes
1/28 Lifecycle

By Wenjie

- We do a deep copy of every instance variable (specifically the pointer to the Cube, we want a new Cube)
- Reference variable can only be copied in the Initializer List

Tower.cpp

```
Tower::Tower(const Tower & other) : ref_(other.ref_){
  // Deep copy cube_:
  cube_ = other.cube_;

  // Deep copy ptr_
  ptr_ = new Cube(*other.ptr_);

  // Deep copy ref_ (?)
     // Doesn't make sense to "deep copy" an alias
     // Done in the Initializer List
}
```

- **Destructor**
  - **Purpose** : it cleans up all resources held by the class or objects through cleaning up heap memory and closing all the files
  - If we ever used **new** keyword, we have to free the memory (calling **delete**) so that we don't leak memory.

- **Automatic Destructor**
  - It exists only if no custom destructor is defined
  - **Functionality** - It only calls the destructor of the members without doing anything else ie.cleaning heap memory or closing any files
  - **Invoked** - it is always automatically called when reclaimed
    - Stack memory: reclaimed when function returns
    - Heap memory: reclaimed when calling delete
  - Destructor is the final thing to call in the lifecycle of a class.