Important Details About Destructors

Always define a destructor, and make it `virtual`.

Why `virtual`?

```
class MyClass : public ParentClass …
// dynamic allocation and deallocation
ParentClass* p = new MyClass;
delete p; // which destructor?
```

If `ParentClass`' destructor is not `virtual`, the answer is, "**The wrong one.**"

Example of Destructor Declaration in a Class Definition

Destructors
- are named ~ followed by the class name,
- take no arguments, and
- return nothing.

So we have …

```
class MyClass {
    virtual ~MyClass ();
};
```


Operation of Destructors

Destructors do the following…

1. **Execute the body** of the destructor.
2. **Call destructors for all fields** that are instances, in reverse order of declaration in the class definition.
3. **Call destructors for base classes**, if any.
4. (**For dynamically-allocated instances**, the **deallocation** happens here.)

Destructor Bodies Must Perform Deallocation

Do not call destructors for fields that are instances; such calls are made automatically.

Fields that are pointers to instances are not implicitly destroyed.

If a field
- points to an instance
- that should be destroyed
- when the instance containing the field is destroyed,
- the **destructor body** (the code) **must perform that deallocation explicitly**.

