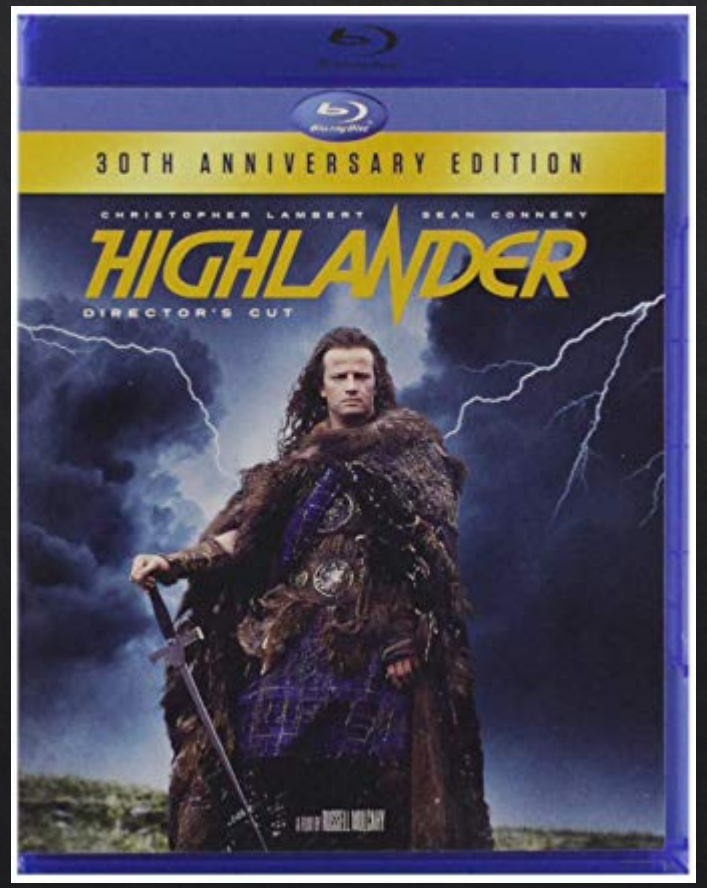
ETGG3802

Lecture3: Singletons + the GOM



Singletons

- A common (controversial?) design pattern
  - Language agnostic
- Goals:
  - Ensure only one instance of a class
  - Allow controlled, global-ish access to the one instance

C++ static member review

- Declaring something static

```
class Foo {
    static int msData;
    static int getData();   // Note: no static keywork in cpp
};
```

- Initializing a static attribute
  - Not in constructor…see why?
  - Proper way (in .cpp normally)

```
int Foo::msData = 1000;
```

- Rules:
  - Non-static (i.e. normal) can access static and non-static members
  - Static methods can only access static members
  - Can call a static method through an instance or the class (using :: operator)

C++ template review

- Purpose?
- Template functions

```
template <class T>
T do_something(T param1, int param2, T param3) {}

float f = do_something(float1, 42, float2); // infer type
string s = do_something<string>(“abc”, 42, “def”);  // explicit

// Template specialization - allows us to provide a special version
// for select types.
template<>
Foo do_something(Foo param1, int param2, Foo param3) {}
```

C++ template review

- Template classes

```
template <class T>
class Foo {
    T mData;
    T func() {}
};

// Specialization (usually of an attribute or method).  Can be in .h
// or .cpp where needed.
template<>
string Foo<string>::func() { /* do something different*/ }
```

- Reflection: why we only put templated stuff in headers?

Singleton Implementation

- Design goal: Make it a “drop-in” (easy to transform a class in a Singleton)
- Two schools of thought (we’ll do B):
  - A: Make the singleton the first time it’s asked for
    - Pro: it exists in the Singleton!
    - Con: constructor can’t take arguments (unless they *all* take arguments)
  - B: Make the singleton externally and store it in the singleton
- Basic idea:
  - Templated class
  - Derive class A from Singleton<A> to turn into a singleton
  - Store a static T* (the singleton instance) in the class.
    - Set it in the constructor
    - Raise std::runtime_error if there already is one.
  - Have a static getSingletonPtr method in Singleton
  - Create the A instance elsewhere (e.g. LogManager is created in Application)

GameObject

- very similar to Unity's definition
- A thing in the game
- Basically just a transform (sceneNodes contain this in Ogre)
- We'll add components to it in the next lab.
- [Look at the starting class]

The GOM (Game Object Manager)

- GOM = A centralized hub to store all GameObjects.
  - By named groups (e.g. “Level1” or “Player”)
  - Has ways to access a game object (by name or group+name)
  - Use std::map for O(log n) [probably] access times.
- [look at the header file]
- Review of std::map?