# DSA2

Data Struc & Alg Game & Sim II (IGME.309)

# Singletons

- Scopes
  - { }
- A variable allocated in a scope will live in the Stack.
  - If we work with a pointer.
    - The pointer variable itself will live in the stack
    - The address its pointing to should live in the heap

# Singletons

```
void main(void)
{
    if (true)
    {
        int x = 10;
    }
    x = 20;
}
```

x is out of scope!

x will live within the scope of the if but will be inexistent outside of it.

# Singletons

```
void main(void)
{
    if (true)
    {
        int* x = new int(); // lets say 0x007c4808 (just because)
        *x = 20;
    }
}
```

Memory leak. I lost the address of the location I had in memory.
Even thought the location still holds the value 20

# Singletons

```
void main(void)
{
    int* temp = nullptr;
    if (true)
    {
        int* x = new int(); // lets say 0x007c4808 (just because)
        *x = 20;
        temp = x;
    }
    delete temp;
    temp = nullptr;
}
```

That will clean the content of the location 0x007c4808
Even though we lost track of the x pointer.

# Singletons

- They are a way to keep a single reference among different .cpps
- Work like global variables.
  - There are always a way to go around them.
- Shouldn’t really use them for everything.
  - But will simplify your life.
  - If you don’t use them you need to warranty that no more than one pointer will be created for a class… which is complicated.

# Singletons

- How to make them?
  - They need to have all pointers variables allocated in the different Stacks pointed at them.
    - How to make something shared among different stacks?
      - Put it on the heap! (but kind of defeats the purpose)
      - Make them static!

# Static variables

```
class Foo{
public:
    static int x;
    friend std::ostream& operator<<(std::ostream &os, const Foo other){
        return os << other.x;
    }
};
int Foo::x = 0;
int main(void){
    Foo o1;
    o1.x = 10;
    Foo o2;
    std::cout << o2 << std::endl;
    return 0;
}
```

X will be shared among all objects created using that class as a blueprint.

# Singletons

- How to make them?
  - They need to have all pointers variables allocated in the different Stacks pointed at them.
    - How to make something shared among different stacks?
      - Put it on the heap! (but kind of defeats the purpose)
      - ~~Make them static!~~
    - Other classes should access an object but not create one.

# Singletons

```
class Foo{
private:
    static Foo* instance;
    int var;
    Foo(){};
    Foo(Foo const& other){};
    Foo& operator=(Foo const& other){};
};
Foo* Foo::instance = nullptr;
```

# Singletons

```
int main(void)
{
    Foo::instance = nullptr;
    return 0;
}
int main(void){
    Foo oFoo;
    oFoo.instance = nullptr;
    return 0;
}
```

Not valid syntax

# Singletons

```
class Foo{
private:
	static Foo* instance;
	int var;
	Foo(){ var = 10;};
	Foo(Foo const& other){};
	Foo& operator=(Foo const& other){};
};
Foo* Foo::instance = nullptr;
```

Nothing outside the class can create a class object.

How do I instantiate a class object?

Have the class itself do it!

# Singletons “Constructor”

```
class Foo{
    static Foo* instance;
    int var;
private:
    Foo(){ var = 10; };
    Foo(Foo const& other){};
    Foo& operator=(Foo const& other){};
public:
    static Foo* GetInstance(){
        if (instance == nullptr)
            instance = new Foo();
        return instance;
    }
};
```

```
Foo* Foo::instance = nullptr;
int main(void){
    Foo* oFoo = Foo::GetInstance();
    if (true){
        Foo* oFoo1 = Foo::GetInstance();
    }
    return 0;
}
```

But I allocated memory! Who’s job is it to deallocate it?

# Singletons “Destructor”

```
class Foo{
    static Foo* instance;
    int var;
private:
    Foo(){ var = 10; };
    Foo(Foo const& other){};
    Foo& operator=(Foo const& other){};
public:
    static Foo* GetInstance(){
        if (instance == nullptr)
            instance = new Foo();
        return instance;
    }
    static void ReleaseInstance(){
        if (instance != nullptr){
            delete instance;
            instance = nullptr;
        }
    }
};
```

```
Foo* Foo::instance = nullptr;
int main(void){
    Foo* oFoo = Foo::GetInstance();
    if (true){
        Foo* oFoo1 = Foo::GetInstance();
    }
    oFoo.ReleaseInstance();
    return 0;
}
```

# Singletons

- Now what?
  - I have a fairly complex class that makes sure that:
    - Only itself is able to create a new object.
    - That object is going to be the same no matter how many pointers are pointing to that memory location.
  - Where do I use it?
    - Managers that will be present among all of your application in different cpp files.
      - For example?
        - A model manager, you can have many different models in your scene but a mesh manager will make sure that only one instance of a model is created and it gets rendered over and over
        - A texture manager, that will warranty that only a single texture file is loaded in memory even if more than one model is using it.
        - A light manager, that will warranty that no matter what you do to a light source the changes will be persistent among different cpp files.

# Singletons

- Tips:
  - When you create a class object you will create an instance of that class. And as instances every single one will be independent.
  - When you create a singleton you create a copy of a single object, and as such whatever you do to a copy will affect all of them.

# Singletons

- Class object:

Clone troopers

- Singleton

The Flash – speed mirage