

Singleton Design Pattern

黃世強 (Sai-Keung Wong)

National Yang Ming Chiao Tung University

Taiwan

Singleton

Example:

```
SceneManager::getInstance( )->updateMonster( );
```

Singleton

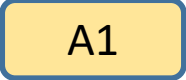
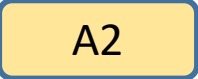
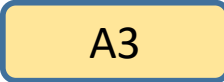
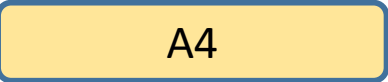
Example:

```
SceneManager::getInstance( )->updateMonster( );
```

```
OOPCourse::getInstance( )->reportStudentInformation( );
```

Singleton

A singleton is a class:

- allow only a  instance of itself to be created.
- give  to that single instance (e.g., give access to class variables)
-   of a class to the single instance

Example:

`SceneManager::getInstance()->updateMonster();`

`OOPCourse::getInstance()->reportStudentInformation();`

Singleton

A singleton is a class:

- allow only a single instance of itself to be created.
- give access to that single instance (e.g., give access to class variables)
- restrict instantiation of a class to the single instance

Example:

```
SceneManager::getInstance( )->updateMonster( );
```

```
OOPCourse::getInstance( )->reportStudentInformation( );
```

Singleton

A::getInstance()->method_name()

```
class A {  
    private:  
    static A *__instance;  
    public:  
    static A *getInstance() const {  
        return __instance;  
    }  
};  
  
//initialization  
A *A::__instance = new A;
```

Singleton

```
class A {  
    private:  
    static A *__instance;  
    public:  
    static A *getInstance() const {  
        return __instance;  
    }  
};  
  
//initialization  
A *A::__instance = new A;
```

A::getInstance()->method_name()

```
class A {  
    private:  
    static A *__instance;  
    public:  
    static A *getInstance() const {  
        if (__instance==0)  
            __instance = A1  
        return __instance;  
    }  
};  
  
// instantiate an object when it is used  
A *A::__instance = 0;
```

Singleton

```
class A {  
    private:  
    static A *__instance;  
    public:  
    static A *getInstance() const {  
        return (!__instance) ?  
            __instance = new A:  
            __instance;  
    }  
    static void method() .....  
};  
A *A::__instance = nullptr;
```

Ternary operator

A::getInstance()->method_name()

```
class A {  
    private:  
    static A *__instance;  
    public:  
    static A *getInstance() const {  
        if (__instance==0)  
            __instance = new A;  
        return __instance;  
    }  
};  
// instantiate an object when it is used  
A *A::__instance = 0;
```