Object Oriented Analysis & Design
面向对象分析与设计

Lecture_09 GOF 设计模式（一）

1）单实例　　2）适配器　　3）外观　　4）观察者

主讲: 姜宁康 博士

2、GOF设计模式一: 单实例模式 SingleTon

- 整个美国，只有一个“现任美国总统”
- 比如，在学校，“老师”，有数百个；“校长”，只有一个
- 系统运行时，如何保证某个类只允许实例化一个对象?

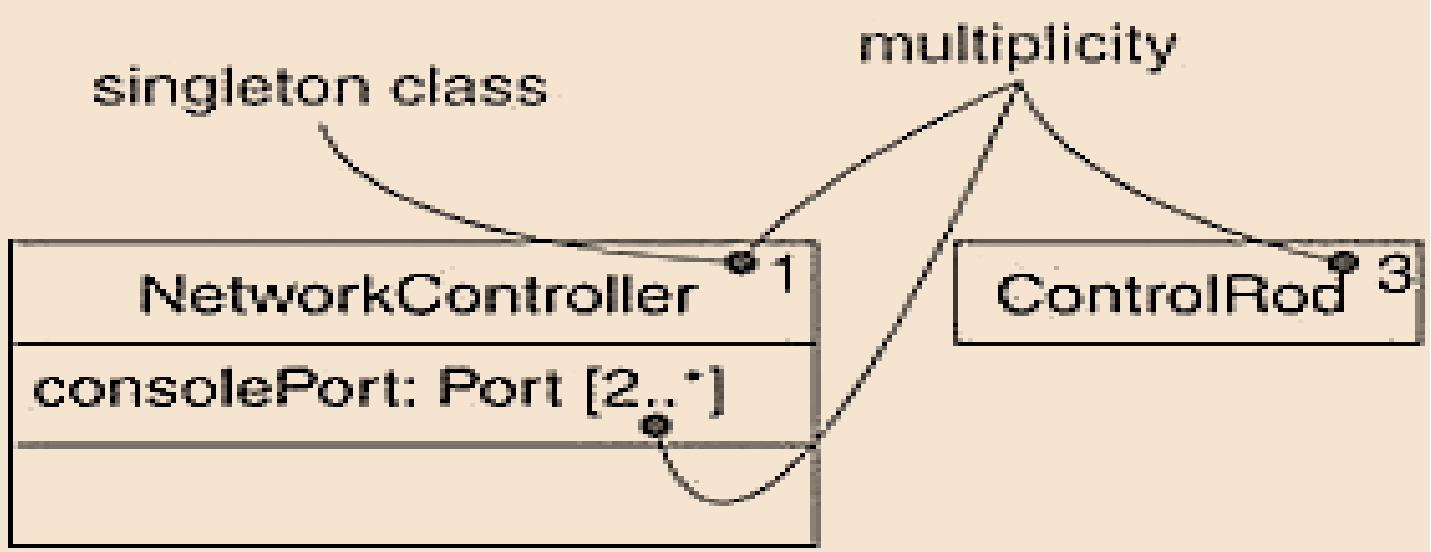
2.1 类的多重性

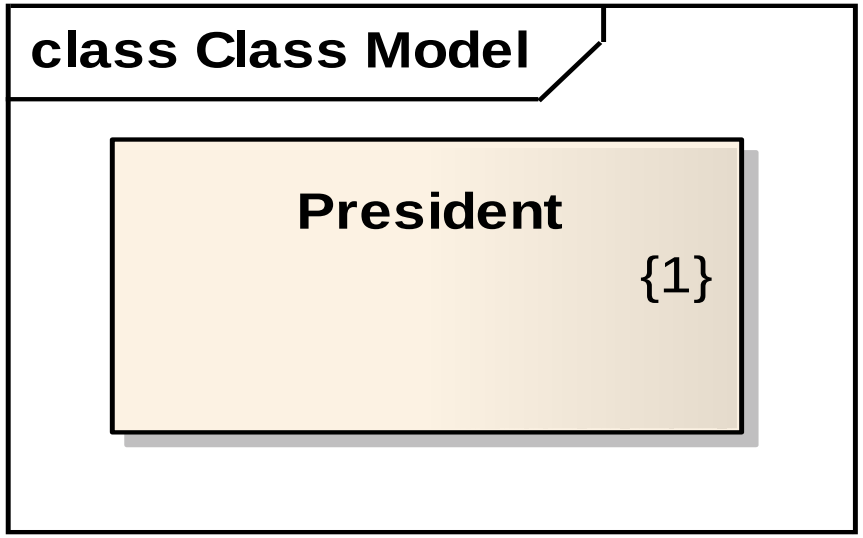
- 类的多重性 multiplicity
  - 在对软件系统进行逻辑设计时，在某些情形之下，可能需要限制类的实例在软件系统中存在的数目

UML符号的表示形式	含义
N	对可以同时存在的对象数目没有限制
1	有且只能有一个对象
1..n	必须有至少一个对象存在
0..n	可以有零个或任意多个对象存在
0..1	可以有零个或一个对象存在
<大于等于1的整数>	必须有指定数量的对象存在
2..10	必须至少有2个对象存在，但不能有超过10个对象同时存在

2.1 类的多重性

- 多重性的图形表示
  - 在类图上，类的多重性表达式被放置在类的图标的右上角
  - 如果类的多重性表达式在类图上被省略，那么此类的多重性缺省为n，即对此类的可同时存在的对象的数目没有限制





2.2 类的实例化

- 当类(class)不希望被实例化时
  - 不提供构造函数
  - 缺省的构造函数
  - 定义成抽象类
- 要确保类不被实例化
  - 类包含显式的构造函数
  - 将构造函数声明为私有（private），在该类的外部就不可能访问，也就不能被实例化了

```
//+----------------------+
public class UtilityClass
{
   private UtilityClass()
  {
    ...
  }
 }
//+----------------------+
```

  - 另外要注意的是，这种用法有点副作用，那就是它不能被子类化
  - 因为 superclass 不可访问

2.3 单实例模式 The Singleton Pattern

Intent **目的**	**You want to have only one of an object, but there is no global object that controls the instantiation of this object. You also want to ensure that all entities are using the same instance of this object, without passing a reference to all of them 只希望有一个对象，系统所有地方都可以用到这个对象，又不使用全局变量，也不需要传递对象的引用**
Problem **问题**	**Several different client objects need to refer to the same thing, and you want to ensure that you do not have more than one of them 几个不同的客户对象都希望引用同一个对象，如何保证?**
Solution解决方案	**Guarantees one instance 确保单实例**
Participants and collaborators **参与方**	**Clients create an instance of the Singleton solely through the getInstance method**
Consequences **结果**	**Clients need not concern themselves whether an instance of the Singleton exists. This can be controlled from within the Singleton**
Implementation **实现**	• **Add a private static member of the class that refers to the desired object (Initially, it is null)** • **Add a public static method that instantiates this class if this member is null (and sets this member's value) and then returns the value of this member** • **Set the constructor's status to protected or private so that no one can directly instantiate this class and bypass the static constructor mechanism**

2.3单实例模式 解决方法: Java为例

```
class USTax {
  public:
      static USTAX* getInstance(); // public static getter function
  private:
      USTax();
      static USTax* instance;
}

USTax* USTax::instance = 0;
USTax* USTax::getInstance(){
      if (instance == 0) {
            instance = new USTax;
      }
      return instance;
}
```

1、公有的成员函数，创建并供客户获取该单实例

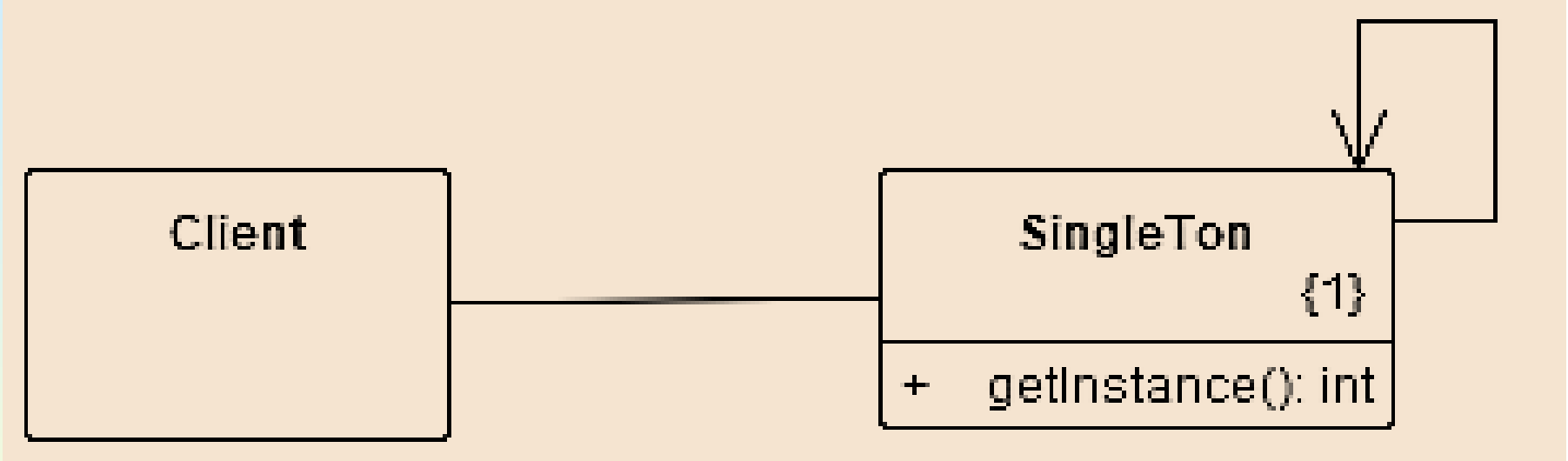
2、私有的构造函数

3、私有的静态成员变量

该单实例唯一的创建之处

2.4 单实例模式 结构

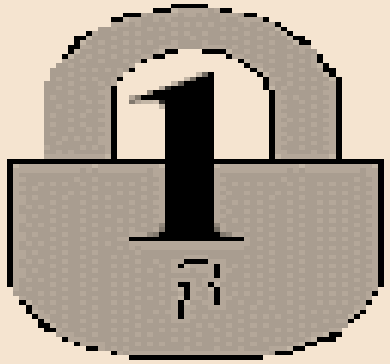
- 单实例模式 模型1



2.4 单实例模式 结构

- 单实例结构
  - 锁住某类的实例化功能。客户类只能使用某类自身实例化的唯一实例

Singleton



1) Lock-up a class so that clients cannot create their own instances, but must use the single instance hosted by the class itself.

2.5 单实例模式例子：(使用前)

定义了一个全局变量，使用之前要测试是否已经实例化。这段代码出现了多次

```
class GlobalClass {
   int  m_value;
   public:
   GlobalClass( int v=0 )  { m_value = v; }
   int  get_value()        { return m_value; }
   void set_value( int v ) { m_value = v; }
};

// Default initialization初始化
GlobalClass*   global_ptr = 0;

void foo( void ) {
   // Initialization on first use
   if ( ! global_ptr )
      global_ptr = new GlobalClass;
   global_ptr->set_value( 1 );
   cout << "foo: global_ptr is "
        << global_ptr->get_value() << '\n';
}
```

```
void bar( void ) {
   if ( ! global_ptr )
      global_ptr = new GlobalClass;
   global_ptr->set_value( 2 );
   cout << "bar: global_ptr is "
        << global_ptr->get_value() << '\n';
}

int main( void ) {
   if ( ! global_ptr )
      global_ptr = new GlobalClass;
     cout << "main: global_ptr is "
          << global_ptr->get_value() << '\n';
     foo();
     bar();
}
// main: global_ptr is 0
// foo: global_ptr is 1
// bar: global_ptr is 2
```

采用单实例模式

```
class GlobalClass {
   int    m_value;
 private: static GlobalClass* s_instance;//1. 私有变量
          GlobalClass( int v=3 ) { m_value = v; }//2.私有构造函数
public:
   int  get_value()       { return m_value; }
   void set_value( int v ) { m_value = v; }
   static GlobalClass* instance() { //3. 公有getter函数
      if ( ! s_instance )
        s_instance = new GlobalClass;
      return s_instance;
   }
};
// Allocating and initializing GlobalClass's
// static data member.  The pointer is being
// allocated - not the object inself.
GlobalClass* GlobalClass::s_instance = 0;

void foo( void ) {
   GlobalClass::instance()->set_value( 1 );
   cout << "foo: global_ptr is "
      << GlobalClass::instance()->get_value() << '\n';
}
```

- 采用设计模式之后 After taking Design Pattern
  - 由某类自己来控制只实例化一个对象
  - 所有的客户对象只使用公有的getter函数：GlobalClass::instance()，以获得该单实例

```
void bar( void ) { //应用之处
   GlobalClass::instance()->set_value( 2 );
   cout << "bar: global_ptr is "
        << GlobalClass::instance()->get_value()
        << '\n';
}

int main( void ) {  //主函数，应用之处
   cout << "main: global_ptr is "
        << GlobalClass::instance()->get_value()
        << '\n';
   foo();
   bar();
}

// main: global_ptr is 3
// foo: global_ptr is 1
// bar: global_ptr is 2
```

2.5 单实例模式 小结

- 单实例模式要点 There are two forces that affect the Singleton
  - 只能有一个实例 There must be exactly one instance of the class
  - 这个实例能够方便地被所有客户访问 The instance must be (easily) accessible to all potential clients

- 解决方法 Solution (Check list)
  - 定义私有的静态成员变量，保存单实例的引用 Define a private static attribute in the "single instance" class
  - 定义公有的Getter函数 Define a public static getter function in the class
  - 该类自己负责“第一次使用时”实例化对象 Do "lazy initialization" (creation on first use) in the getter function
    - Class itself responsible for creating, maintaining, and providing global access to its own single instance
  - 定义私有的构造函数 Define all constructors to be protected or private
  - 客户对象只能通过 getter 函数获得该单实例

- 本讲结束