7COM1025
Programming for Software Engineers
Lecture 27

Dr. Renato C. de Amorim

SINGLETON (USING REFERENCE)

```
#include <iostream>
using namespace std;
class myclass{
    private:
        myclass(){cout<<"constructor"<<endl;}
        myclass (myclass const&){}
        myclass& operator=(myclass const &){}
    public:
        ~myclass(){
            cout<<"destructor"<<endl;}
        void method(){cout<<"test"<<endl;}
     static myclass& GetUniqueInstance(){
            static myclass Instance;
            return Instance;}};
int main(){
    myclass::GetUniqueInstance().method();
    return 0;}
```

University of Hertfordshire UH

SINGLETON (USING POINTER)

```
#include <iostream>
using namespace std;
class myclass{
    private:
        static myclass *_instance;
        myclass(){}
        myclass (myclass const&){}
        myclass& operator=(myclass const &){}
    public:
        ~myclass(){}
        void method(){cout<<"test"<<endl;}
        static myclass *GetUniqueInstance(){
            if (_instance==NULL)
                _instance = new myclass;
            return _instance;}};
    myclass *myclass::_instance = NULL;
int main(){
    myclass::GetUniqueInstance()->method();
    return 0;}
```

PROBLEM 27.1

Change your classification class to a singleton

FACTORY

```
#include <iostream>
using namespace std;
class myclass{
    public:
        void method(){cout<<"test"<<endl;}};
class myclassFactory{
        myclassFactory(){}
        myclassFactory (myclassFactory const&){}
        myclassFactory& operator=(myclassFactory const &){}
        public:
        static myclass *GetNewInstance(){
            return new myclass;}};
int main(){
    myclass *ptr = myclassFactory::GetNewInstance();
    ptr->method();
    delete ptr;
    return 0;}
```

FACTORY AND INHERITANCE

```
#include<iostream>
enum ComputerType {eLaptop, eDesktop};
class Computer{
 public:
    virtual ~Computer() {};
    virtual void method(){std::cout<<"This is a Computer"<<std::endl;}};
 class Laptop: public Computer {
 public:
    virtual ~Laptop() {};
    virtual void method(){std::cout<<"This is a Laptop"<<std::endl;}};
 class Desktop: public Computer{
 public:
    virtual ~Desktop() {}
    virtual void method(){std::cout<<"This is a Desktop"<<std::endl;}
    void method2(){std::cout<<"Method 2"<<std::endl;}};
 class ComputerFactory{
 public:
    static Computer *GetNewInstance(const ComputerType _type){
       switch(_type){
           case eLaptop:
               return new Laptop;
           case eDesktop:
               return new Desktop;}}};
int main() {
   Computer *ptr_Desktop = ComputerFactory::GetNewInstance(eDesktop);
   ptr_Desktop->method();
   delete ptr_Desktop;
   return 0;}
```

PROBLEM 27.2

Create a factory for you Table class.