14CS413

B.E

PES Institute of Technology

Design Patterns Mini Project
December 2017
7th Semester

Life Pattern

ABHIJITH S D 1PI14CS002
NEHA M M 1PI14CS064

GitHub: Abhijith-S-D/Design-Pattern-Project

- **<u>Intent</u>:**
    - Construct an Object Pool of Resources which maintains seperate pools for every sub-class of Resource that have been registered with the Object Pool.
    - Define dependency between objects in this pool with longetivity such that these objects are destroyed accordingly.

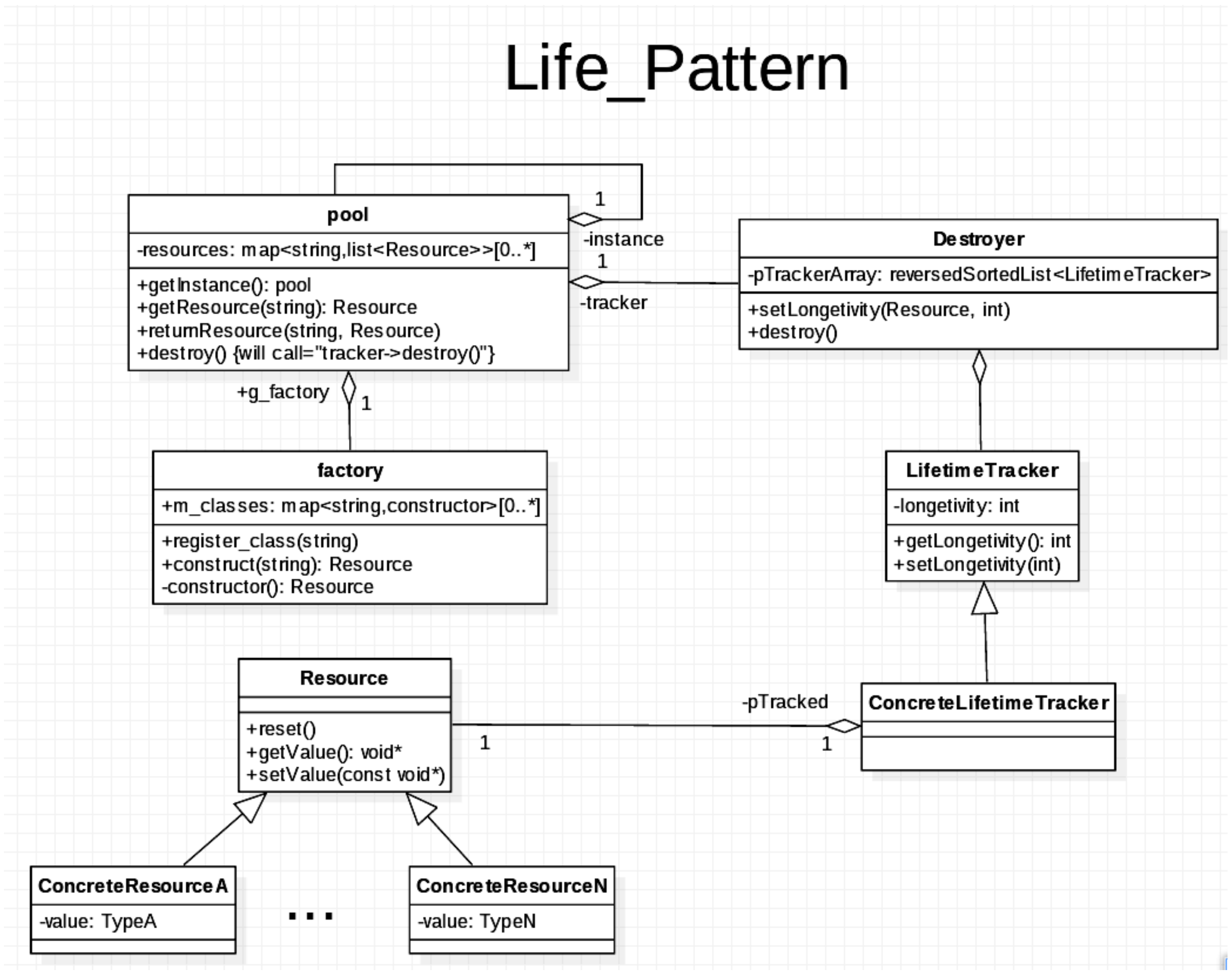
- **<u>Motivation</u>:**

Generally Object Pool exists of only one specific class, but in many cases, we would require an Object Pool which supports all classes that have a particular class as their parent.It would be very much better if users are given the priviledge of registering classes that can be pooled.Object Pools do not generally maintain dependencies between the objects which would generally be essential during destruction.

- **<u>Applicability</u>:**

Resources that are costly to construct and used very frequently can be maintained in this Life Design Pattern, which can maintain dependencies between the Resource objects.This pattern defines a way to construct, store and destroy Resource objects by maintaining longetivity relationship.

- **<u>Structure</u>:**



- **<u>Participants</u>:**
  - **<u>factory</u>**: This is the class which provides the feature of registering a new class to the pattern.This stores the constructors of all the registered classes and given the string of class name as parameter, it builds the Resource object.
  - **<u>Pool</u>:** This is the main singleton class that maintains pools for all the registered classes. One can get Resource objects of registered classes and return them back into the pool.Whenever a Resource object is returned it’s value is reset. It contains a factory object and a Destroyer object which destroys all objects according to their longetivity.

- **Destroyer:** This is a class that maintains all the Resource Objects encapsulated in a LifeTimeTracker object in a reverse-sorted vector according to their longetivity and can destroy a resource with least longetivity and among the object of same longetivity, deletes in FIFO order.

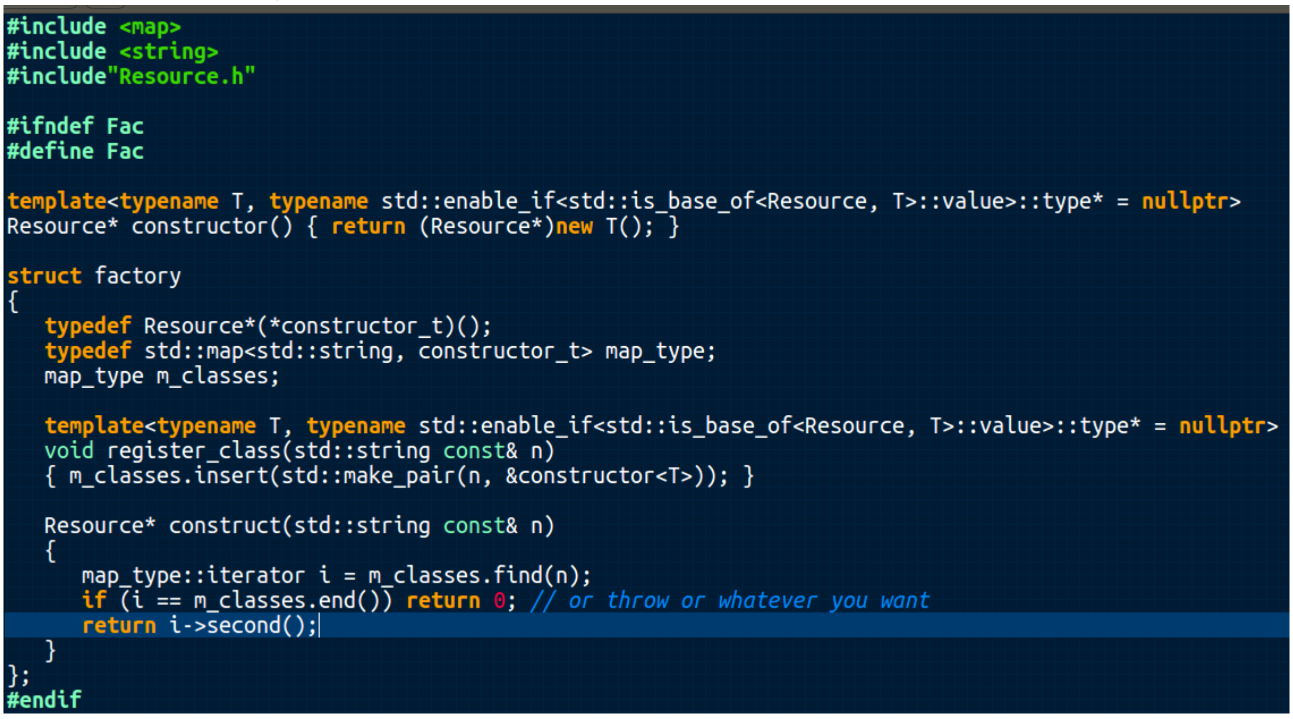
- **LifetimeTracker:** An abstract class mainly used to maintain longetivity.

- **ConcreteLifetimeTracker:** Class used to maintain longetivity of any Resource Object.

- **Resource:** This is the base class which specifies the main operations that all the objects in the pool should follow.

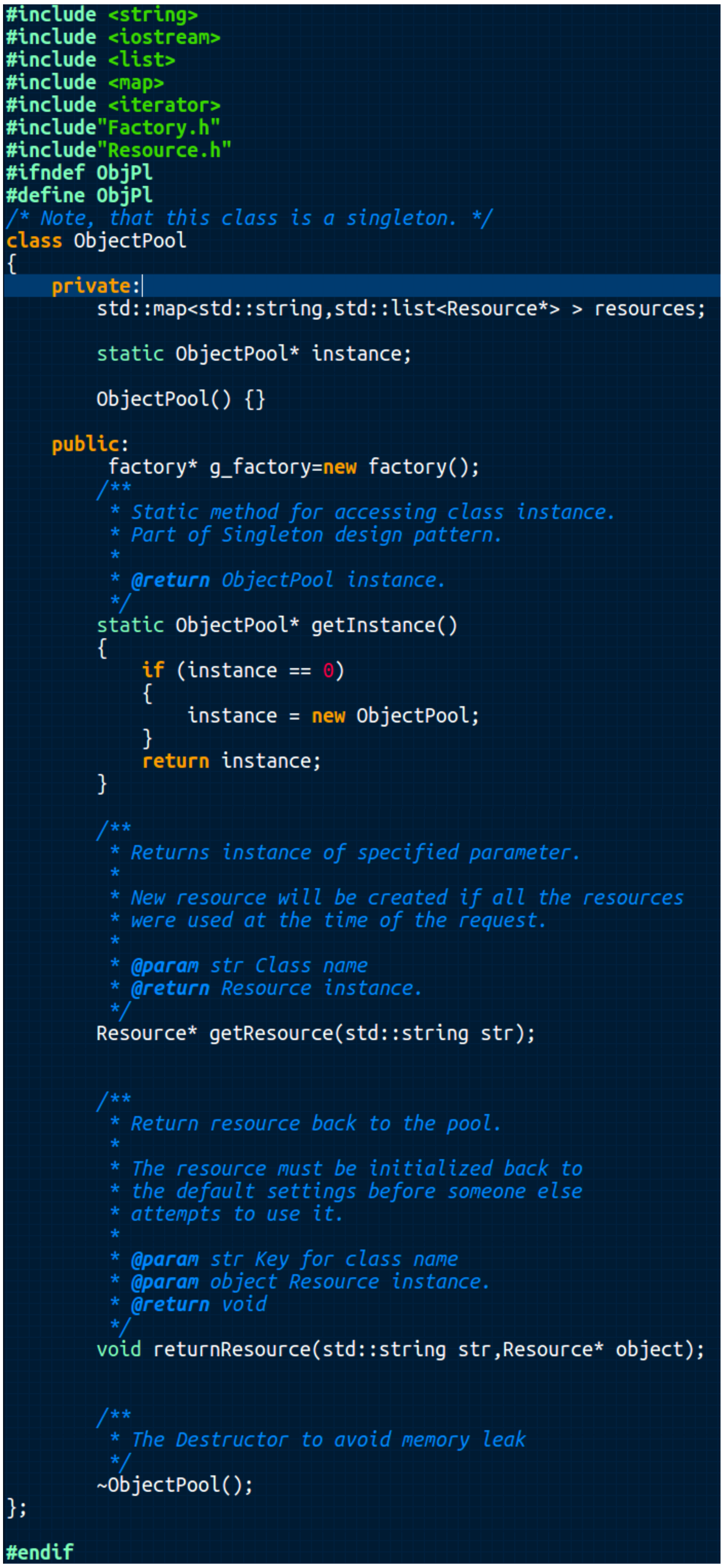
- **ConcreteResource:** This is a concrete resource which has a value of particular type, which can be reset.

- **Implementation:**
  - factory.h



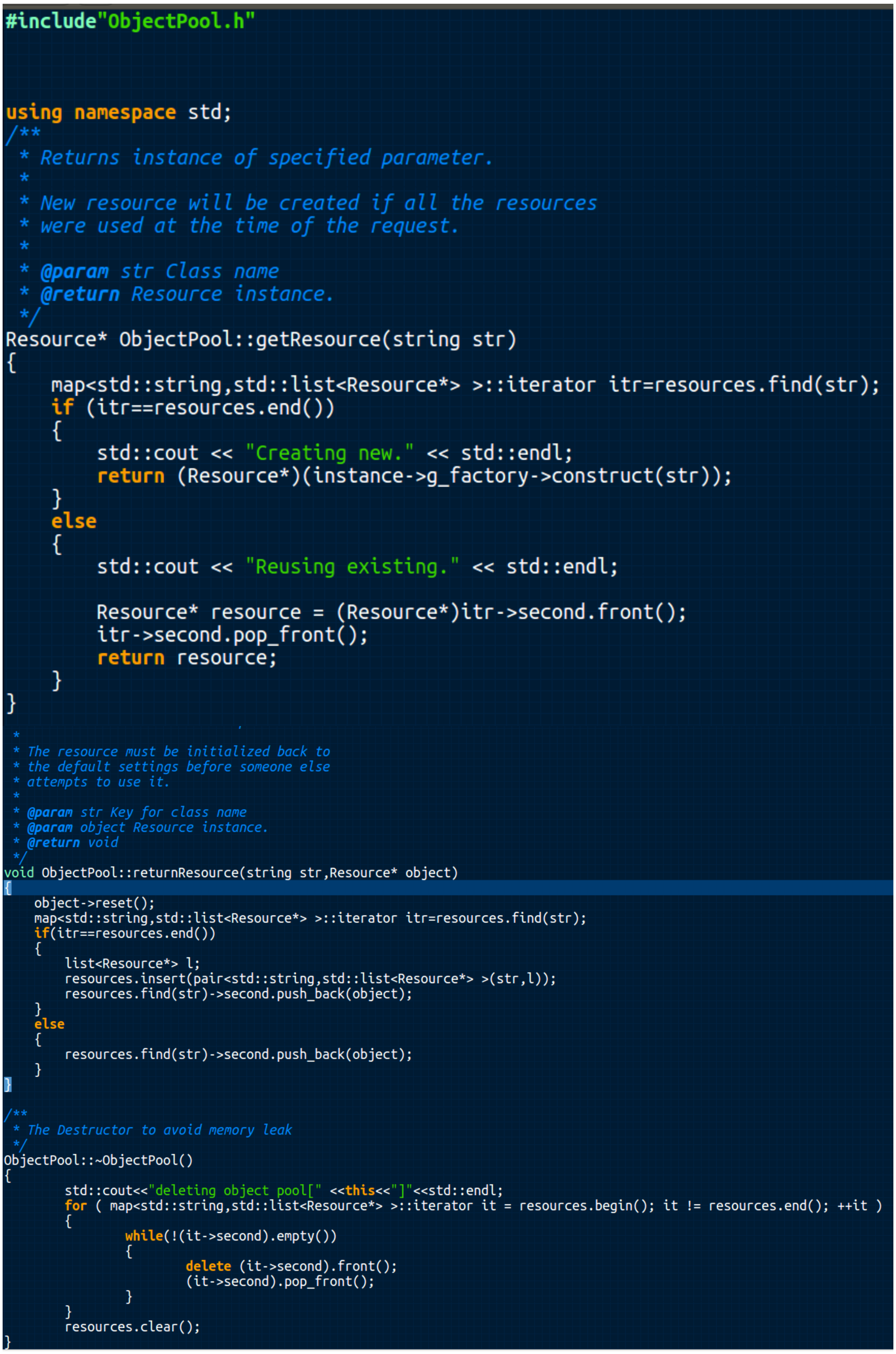
```
#include <map>
#include <string>
#include"Resource.h"

#ifndef Fac
#define Fac

template<typename T, typename std::enable_if<std::is_base_of<Resource, T>::value>::type* = nullptr>
Resource* constructor() { return (Resource*)new T(); }

struct factory
{
   typedef Resource*(*constructor_t)();
   typedef std::map<std::string, constructor_t> map_type;
   map_type m_classes;

   template<typename T, typename std::enable_if<std::is_base_of<Resource, T>::value>::type* = nullptr>
   void register_class(std::string const& n)
   { m_classes.insert(std::make_pair(n, &constructor<T>)); }

   Resource* construct(std::string const& n)
   {
      map_type::iterator i = m_classes.find(n);
      if (i == m_classes.end()) return 0; // or throw or whatever you want
      return i->second();
   }
};
#endif
```

- ObjectPool.h



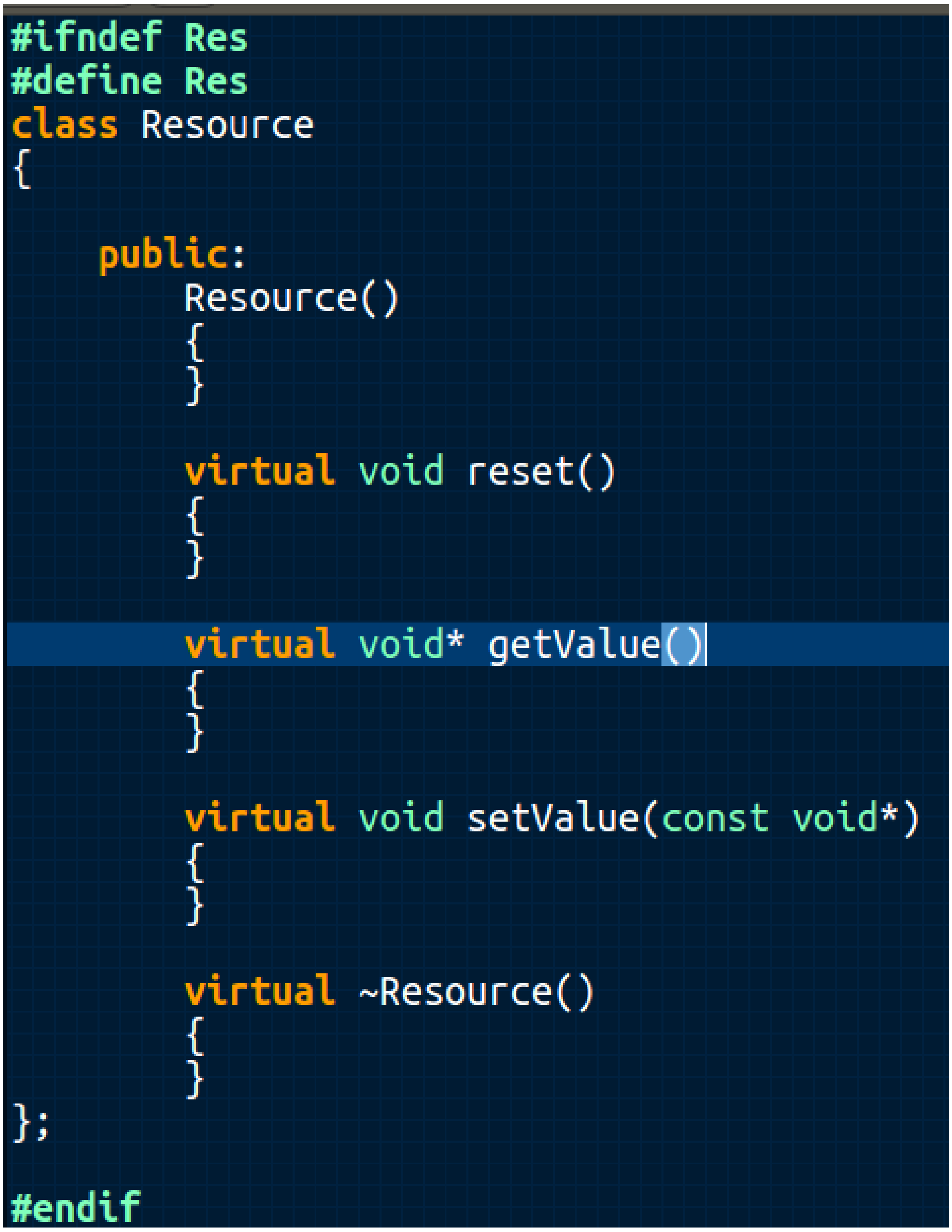
```
#include <string>
#include <iostream>
#include <list>
#include <map>
#include <iterator>
#include"Factory.h"
#include"Resource.h"
#ifndef ObjPl
#define ObjPl
/* Note, that this class is a singleton. */
class ObjectPool
{
    private:
        std::map<std::string,std::list<Resource*> > resources;

        static ObjectPool* instance;

        ObjectPool() {}

    public:
         factory* g_factory=new factory();
        /**
         * Static method for accessing class instance.
         * Part of Singleton design pattern.
         *
         * @return ObjectPool instance.
         */
        static ObjectPool* getInstance()
        {
            if (instance == 0)
            {
                instance = new ObjectPool;
            }
            return instance;
        }

        /**
         * Returns instance of specified parameter.
         *
         * New resource will be created if all the resources
         * were used at the time of the request.
         *
         * @param str Class name
         * @return Resource instance.
         */
        Resource* getResource(std::string str);


        /**
         * Return resource back to the pool.
         *
         * The resource must be initialized back to
         * the default settings before someone else
         * attempts to use it.
         *
         * @param str Key for class name
         * @param object Resource instance.
         * @return void
         */
        void returnResource(std::string str,Resource* object);


        /**
         * The Destructor to avoid memory leak
         */
        ~ObjectPool();
};

#endif
```

- ObjectPool.cpp



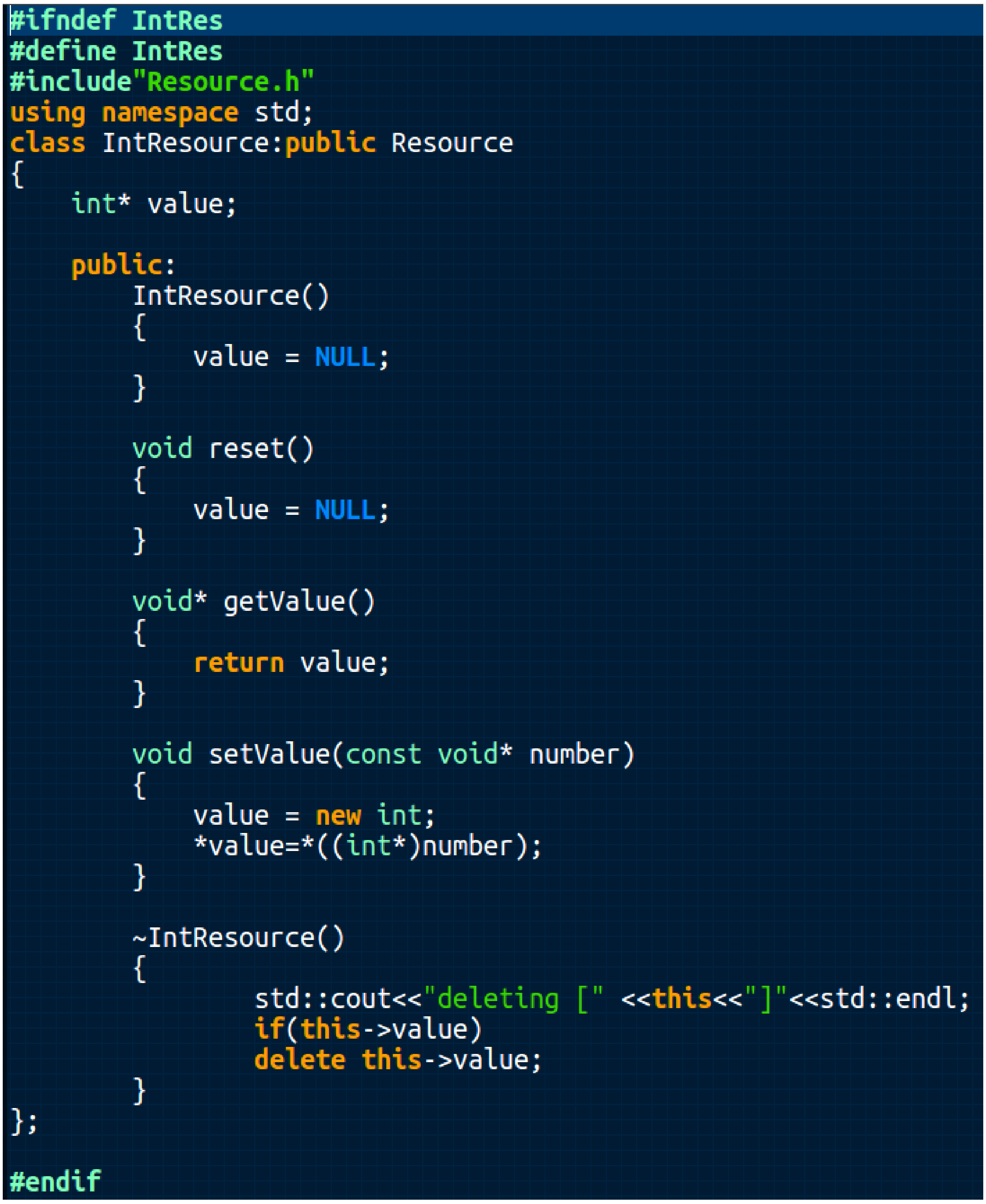
```cpp
#include"ObjectPool.h"


using namespace std;
/**
 * Returns instance of specified parameter.
 *
 * New resource will be created if all the resources
 * were used at the time of the request.
 *
 * @param str Class name
 * @return Resource instance.
 */
Resource* ObjectPool::getResource(string str)
{
    map<std::string,std::list<Resource*> >::iterator itr=resources.find(str);
    if (itr==resources.end())
    {
        std::cout << "Creating new." << std::endl;
        return (Resource*)(instance->g_factory->construct(str));
    }
    else
    {
        std::cout << "Reusing existing." << std::endl;

        Resource* resource = (Resource*)itr->second.front();
        itr->second.pop_front();
        return resource;
    }
}

 *
 * The resource must be initialized back to
 * the default settings before someone else
 * attempts to use it.
 *
 * @param str Key for class name
 * @param object Resource instance.
 * @return void
 */
void ObjectPool::returnResource(string str,Resource* object)
{
    object->reset();
    map<std::string,std::list<Resource*> >::iterator itr=resources.find(str);
    if(itr==resources.end())
    {
        list<Resource*> l;
        resources.insert(pair<std::string,std::list<Resource*> >(str,l));
        resources.find(str)->second.push_back(object);
    }
    else
    {
        resources.find(str)->second.push_back(object);
    }
}

/**
 * The Destructor to avoid memory leak
 */
ObjectPool::~ObjectPool()
{
        std::cout<<"deleting object pool[" <<this<<"]"<<std::endl;
        for ( map<std::string,std::list<Resource*> >::iterator it = resources.begin(); it != resources.end(); ++it )
        {
                while(!(it->second).empty())
                {
                        delete (it->second).front();
                        (it->second).pop_front();
                }
        }
        resources.clear();
}
```

- ○ Resource.h



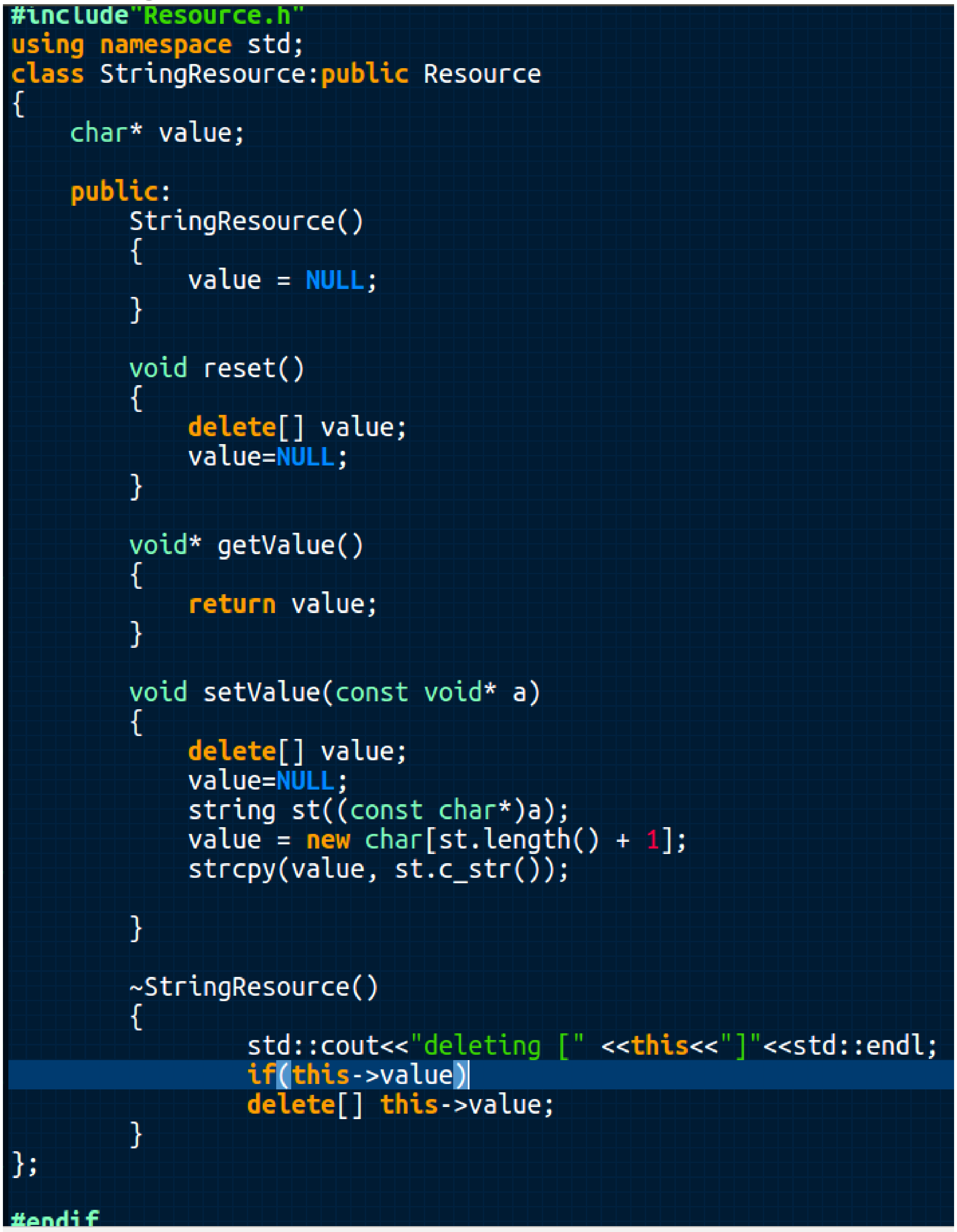
```cpp
#ifndef Res
#define Res
class Resource
{

    public:
        Resource()
        {
        }

        virtual void reset()
        {
        }

        virtual void* getValue()
        {
        }

        virtual void setValue(const void*)
        {
        }

        virtual ~Resource()
        {
        }
};

#endif
```

- IntResource.h



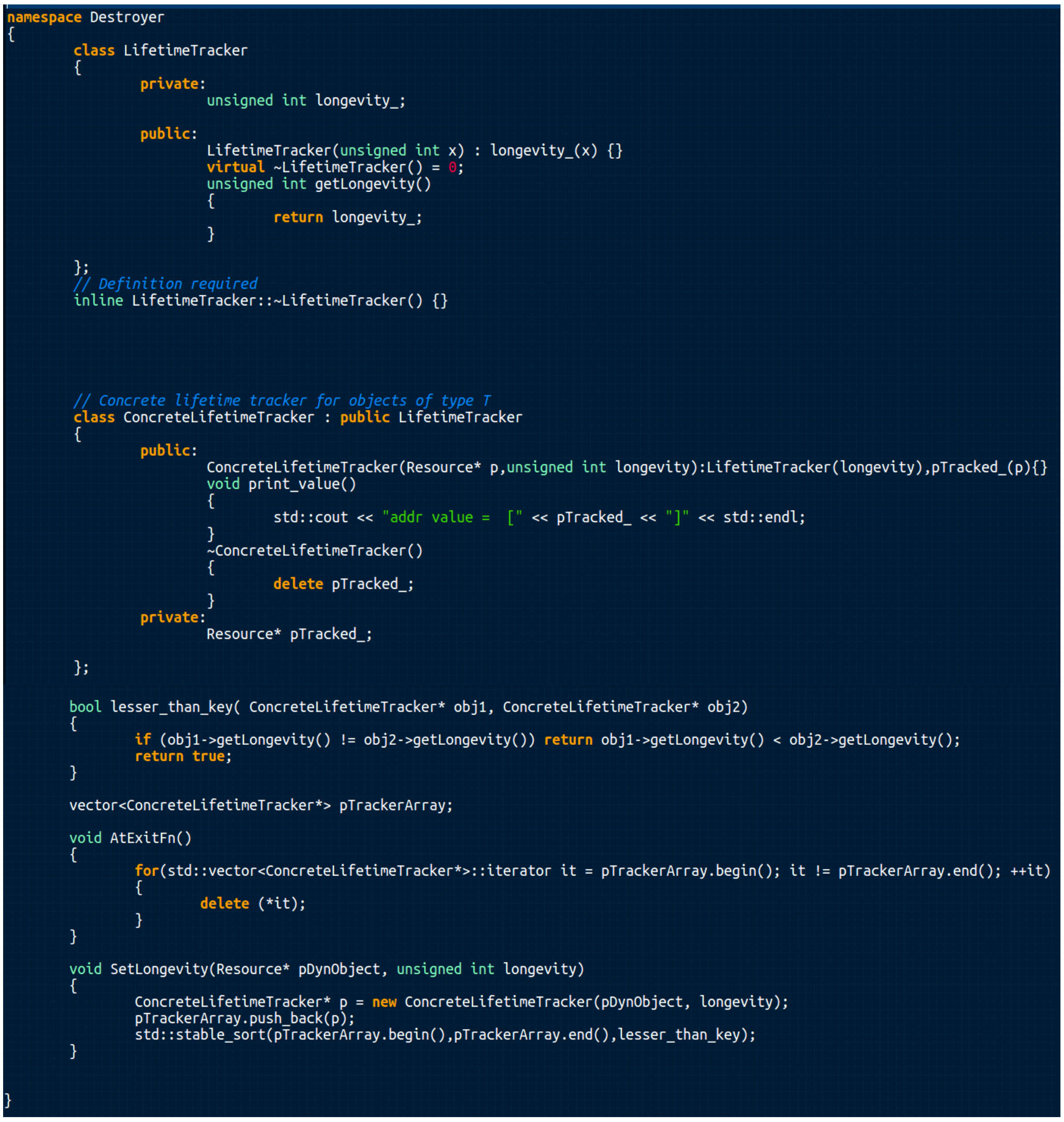
```
#ifndef IntRes
#define IntRes
#include"Resource.h"
using namespace std;
class IntResource:public Resource
{
    int* value;

    public:
        IntResource()
        {
            value = NULL;
        }

        void reset()
        {
            value = NULL;
        }

        void* getValue()
        {
            return value;
        }

        void setValue(const void* number)
        {
            value = new int;
            *value=*((int*)number);
        }

        ~IntResource()
        {
                std::cout<<"deleting [" <<this<<"]"<<std::endl;
                if(this->value)
                delete this->value;
        }
};

#endif
```

- StringResource



```
#include"Resource.h"
using namespace std;
class StringResource:public Resource
{
    char* value;

    public:
        StringResource()
        {
            value = NULL;
        }

        void reset()
        {
            delete[] value;
            value=NULL;
        }

        void* getValue()
        {
            return value;
        }

        void setValue(const void* a)
        {
            delete[] value;
            value=NULL;
            string st((const char*)a);
            value = new char[st.length() + 1];
            strcpy(value, st.c_str());

        }

        ~StringResource()
        {
                std::cout<<"deleting [" <<this<<"]"<<std::endl;
                if(this->value)
                delete[] this->value;
        }
};

#endif
```

- LifeTracker.h



```
namespace Destroyer
{
	class LifetimeTracker
	{
		private:
			unsigned int longevity_;

		public:
			LifetimeTracker(unsigned int x) : longevity_(x) {}
			virtual ~LifetimeTracker() = 0;
			unsigned int getLongevity()
			{
				return longevity_;
			}

	};
	// Definition required
	inline LifetimeTracker::~LifetimeTracker() {}



	// Concrete lifetime tracker for objects of type T
	class ConcreteLifetimeTracker : public LifetimeTracker
	{
		public:
			ConcreteLifetimeTracker(Resource* p,unsigned int longevity):LifetimeTracker(longevity),pTracked_(p){}
			void print_value()
			{
				std::cout << "addr value =  [" << pTracked_ << "]" << std::endl;
			}
			~ConcreteLifetimeTracker()
			{
				delete pTracked_;
			}
		private:
			Resource* pTracked_;

	};

	bool lesser_than_key( ConcreteLifetimeTracker* obj1, ConcreteLifetimeTracker* obj2)
	{
		if (obj1->getLongevity() != obj2->getLongevity()) return obj1->getLongevity() < obj2->getLongevity();
		return true;
	}

	vector<ConcreteLifetimeTracker*> pTrackerArray;

	void AtExitFn()
	{
		for(std::vector<ConcreteLifetimeTracker*>::iterator it = pTrackerArray.begin(); it != pTrackerArray.end(); ++it)
		{
			delete (*it);
		}
	}

	void SetLongevity(Resource* pDynObject, unsigned int longevity)
	{
		ConcreteLifetimeTracker* p = new ConcreteLifetimeTracker(pDynObject, longevity);
		pTrackerArray.push_back(p);
		std::stable_sort(pTrackerArray.begin(),pTrackerArray.end(),lesser_than_key);
	}

}
```

- **Conclusion:**

Hence we were able to create a pattern which gives it's users more flexible control over creation,maintainance and destruction of all Resources maintaining dependencies.