Object Pool in C++

Back to **Object Pool** description

```
#include <string>
#include <iostream>
#include <list>
class Resource
{
    int value;
    public:
        Resource()
        {
            value = 0;
        }
        void reset()
        {
            value = 0;
        }
        int getValue()
        {
            return value;
        }
        void setValue(int number)
        {
            value = number;
        }
};
/* Note, that this class is a singleton. */
class ObjectPool
{
    private:
        std::list<Resource*> resources;

        static ObjectPool* instance;
        ObjectPool() {}
    public:
        /**
         * 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 Resource.
         *
         * New resource will be created if all the resources
         * were used at the time of the request.
         *
         * @return Resource instance.
         */
        Resource* getResource()
        {
            if (resources.empty())
            {
                std::cout << "Creating new." << std::endl;
                return new Resource;
            }
            else
            {
                std::cout << "Reusing existing." << std::endl;
                Resource* resource = resources.front();
                resources.pop_front();
                return resource;
            }
        }
        /**
         * Return resource back to the pool.
         *
         * The resource must be initialized back to
         * the default settings before someone else
         * attempts to use it.
         *
         * @param object Resource instance.
         * @return void
         */
        void returnResource(Resource* object)
        {
            object->reset();
            resources.push_back(object);
        }
};
ObjectPool* ObjectPool::instance = 0;
int main()
{
    ObjectPool* pool = ObjectPool::getInstance();
    Resource* one;
    Resource* two;
    /* Resources will be created. */
    one = pool->getResource();
    one->setValue(10);
    std::cout << "one = " << one->getValue() << " [" << one << "]" << std::endl;
    two = pool->getResource();
```

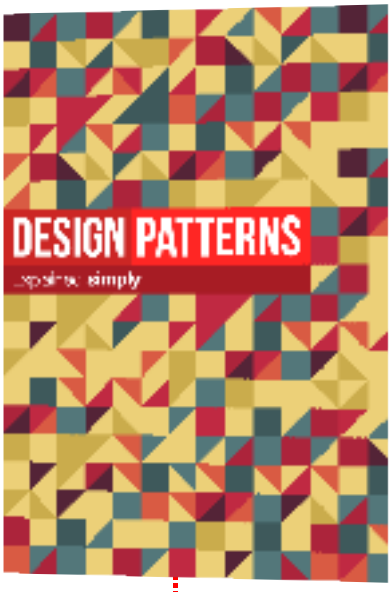
```
    two->setValue(20);
    std::cout << "two = " << two->getValue() << " [" << two << "]" << std::endl;
    pool->returnResource(one);
    pool->returnResource(two);
    /* Resources will be reused.
     * Notice that the value of both resources were reset back to zero.
     */
    one = pool->getResource();
    std::cout << "one = " << one->getValue() << " [" << one << "]" << std::endl;
    two = pool->getResource();
    std::cout << "two = " << two->getValue() << " [" << two << "]" << std::endl;

    return 0;
}
```

Read next

This article is taken from our book **Design Patterns Explained Simply**.

All of the design patterns are compiled there. The book is written in clear, simple language that makes it easy to read and understand (just like this article).

We distribute it in PDF & EPUB formats so you can get it onto your iPad, Kindle, or other portable device immediately after your purchase.



Learn more

Code examples

Java	Object Pool in Java
C++	Object Pool in C++

Design Patterns
AntiPatterns
Refactoring
UML

My account
Forum
Contact us
About us