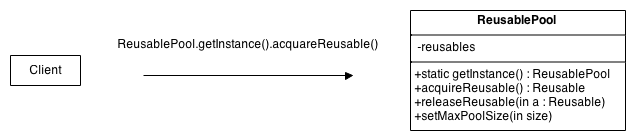
Object pooling can offer a significant performance boost; it is most effective in situations where the cost of initializing a class instance is high, the rate of instantiation of a class is high, and the number of instantiations in use at any one time is low.

Object pools (otherwise known as resource pools) are used to manage the object caching. A client with access to a Object pool can avoid creating a new Objects by simply asking the pool for one that has already been instantiated instead. Generally the pool will be a growing pool, i.e. the pool itself will create new objects if the pool is empty, or we can have a pool, which restricts the number of objects created.

It is desirable to keep all Reusable objects that are not currently in use in the same object pool so that they can be managed by one coherent policy. To achieve this, the Reusable Pool class is designed to be a singleton class.

The general idea for the Connection Pool pattern is that if instances of a class can be reused, you avoid creating instances of the class by reusing them.



* **Reusable** - Instances of classes in this role collaborate with other objects for a limited amount of time, then they are no longer needed for that collaboration.
* **Client** - Instances of classes in this role use Reusable objects.
* **ReusablePool** - Instances of classes in this role manage Reusable objects for use by Client objects.

**Program Example**

**Program example is in python**

**class** Resource:

*""" Some resource, that clients need to use.*

*The resources usualy have a very complex and expensive*

*construction process, which is definitely not a case*

*of this Resource class in my example.*

*"""*

\_\_value = 0

**def** reset(self):

*""" Put resource back into default setting. """*

print(**"reset happens"**)

self.\_\_value = 0

**def** setValue(self, number):

self.\_\_value = number

**def** getValue(self):

**return** self.\_\_value

**class** ObjectPool:

*""" Resource manager.*

*Handles checking out and returning resources from clients.*

*It's a singleton class.*

*"""*

\_\_instance = **None**

\_\_resources = list()

**def** \_\_init\_\_(self):

**if** ObjectPool.\_\_instance != **None**:

**raise** NotImplemented(**"This is a singleton class."**)

@staticmethod

**def** getInstance():

**if** ObjectPool.\_\_instance == **None**:

ObjectPool.\_\_instance = ObjectPool()

**return** ObjectPool.\_\_instance

**def** getResource(self):

**if** len(self.\_\_resources) > 0:

print(**"Using existing resource."**)

**return** self.\_\_resources.pop(0)

**else**:

print(**"Creating new resource."**)

**return** Resource()

**def** returnResource(self, resource):

resource.reset()

self.\_\_resources.append(resource)

**def** main():

pool = ObjectPool.getInstance()

*# These will be created*

one = pool.getResource()

two = pool.getResource()

one.setValue(10)

two.setValue(20)

print(**"%s = %d"** % (one, one.getValue()))

print(**"%s = %d"** % (two, two.getValue()))

pool.returnResource(one)

pool.returnResource(two)

one = **None**

two = **None**

*# These resources will be reused*

one = pool.getResource()

two = pool.getResource()

one.setValue(15)

two.setValue(25)

print(**"%s = %d"** % (one, one.getValue()))

print(**"%s = %d"** % (two, two.getValue()))

**if** \_\_name\_\_ == **"\_\_main\_\_"**:

main()