The *Mem Ops ObjectPool* class is capable of pooling instances of objects which can be reused. At first the `ObjectPool` is empty. When you request an instance from the `ObjectPool` will create a new instance via an `IObjectFactory` and return to you. When you later free the object again, it is cached internally. When you request an instance again in the future, the `ObjectPool` will first check if it has any instances cached internally. If it has, it will return one of the cached instances. If not, a new instance will be created.

Create an ObjectPool

To use the Mem Ops `ObjectPool` you must first create an instance of the`com.nanosai.memops.objects.ObjectPool` class. Here is how you create a Mem Ops `ObjectPool`' instance:

```
IObjectFactory<MyClass> objectFactory = new IObjectFactory<MyClass>() {
    public MyClass instance() {  return new MyClass(); }
};

int capacity = 8;

ObjectPool<MyClass> objectPool = new ObjectPool<MyClass>(capacity, stringFactory);
```

Notice that the `ObjectPool` class constructor takes two parameters: The capacity and an`IObjectFactory` implementation. These two parameters are explained in the following sections.

Capacity

The capacity is the maximum number of instances the `ObjectPool` will create via its`IObjectFactory`. If you request an object instance from the `ObjectPool` which has no object instances cached, and it has already created `capacity` number of objects, `null` will be returned.

In the example above the capacity is set to 8. That means, that at most 8 `MyClass`instances can be created. These 8 instances can then be reused again and again, but you can never get more than 8 instances out of the pool at the same time. Once 8 objects has been "taken", you will need to free an object before you can take a new. Until then, `null` is returned.

IObjectFactory

The second parameter to the `ObjectPool` constructor is an `IObjectFactory` instance.`IObjectFactory` is a Mem Ops specific **Java interface**. You will have to provide an implementation of `IObjectFactory` yourself. Whatever objects the `ObjectPool` is supposed to return must be provided by an `IObjectFactory`. Here is first how the `IObjectFactory` instance looks:

```
package com.nanosai.memops.objects;

public interface IObjectFactory<T> {
    public T instance();
}
```

As you can see, the `IObjectFactory` only contains a single method named `instance()`. The type returned from the `instance()` method is decided by you when you implement the`IObjectFactory` interface. You can read more about implementing generic types in Java interfaces in my **Generic Java interfaces section of my Java interfaces tutorial**.

Here is an example of implementing the `IObjectFactory` interface:

```
IObjectFactory<String> stringFactory = new IObjectFactory<String>() {
    int instanceNo = 0;

    @Override
    public String instance() {
        String value = "" + instanceNo;
        instanceNo++;
        return value;
    }
};
```

Notice how the generic Java type is set to a **Java String**.

Because the `IObjectFactory` interface only contains a single method, you could also implement it using a **Java Lambda Expression**. Here is an example of implementing the `IObjectFactory` interface using a Java lambda expression:

```
IObjectFactory<String> stringFactory =
    () -> "" + System.currentTimeMillis();
```

Get Object Instance From ObjectPool

Once you have created a Mem Ops `ObjectPool` instance, you can obtain an object instance from it. You do so by calling its `instance()` method. Here is an example of obtaining an object instance from an `ObjectPool`:

```
String instance = objectPool.instance();
```

Free Object

Once you are done using an object instance you should free it back to the `ObjectPool`again. You free an object instance for reuse by calling the `ObjectPool` `free()` method. Here is an example of how freeing an instance looks:

```
objectPool.free(instance);
```