# Creational Patterns

## 1. 1.3. Object Pool Pattern

Çoğunlukla, yazılım geliştirme ve nesne oluşturma sırasında performans önemli bir konudur, bu da yer yer pahalı bir aşama olabilir. Nesne oluşturulmasının çok maliyetli olma durumlarında Object Pool tasarım kalıbı kullanılır.

Temel olarak, Object Pool, belirtilen miktarda nesne içeren bir havuzdur. Havuzdan bir nesne alındığında, geri konulana kadar havuzda erişilebilir değildir, kullanılamaz. Havuz boyutu ile içerisinde barınabilecek maksimum nesne sayısı belirlenir. Bu havuzdaki nesneler istemciler arasında belirli bir düzen ile paylaştırılır.

Veri tabanı için çok fazla bağlantı açmaya ihtiyaç duyulursa, yeni bir bağlantı oluşturmak çok daha uzun sürer ve veri tabanı sunucusu aşırı yüklenir. Bu nedenle her istek için bir veri tabanı bağlantısı nesnesi üretmek çok maliyetli olacaktır. Burada ilgili pattern kullanılabilir.

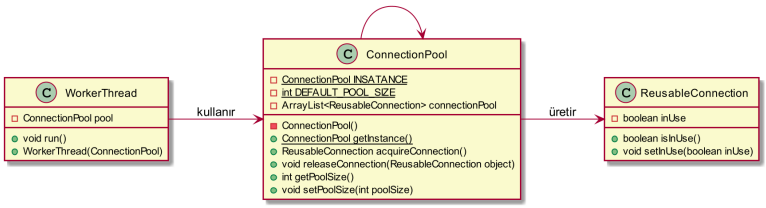


**Figure 1. Object Pool UML Diyagramı**

ConnectionPool sınıfı içerisinde, singleton olacak şekilde pool object oluşturulur.

**ConnectionPool.java.**

```
public class ConnectionPool {
    public static final int DEFAULT_POOL_SIZE = 4;
 private int poolSize = DEFAULT_POOL_SIZE;

 private ArrayList<ReusableConnection> connectionPool;

 private static ConnectionPool instance = null;

 private ConnectionPool() {
        connectionPool = new ArrayList<ReusableConnection>();
```

```
    }

 // SINGLETON
    public static ConnectionPool getInstance() {

     // critical section
     Lock lock = new Lock();

        if (instance == null) {
            instance = new ConnectionPool();
        }

        lock.release();
        // critical section

        return instance;
    }
}
```

Threadlerin pooldan nesne alacakları **acquireConnection()** metodu.

Burada üç senaryo ele alınmıştır:

1. Client, pooldan bir nesne ister. Poolda boşta nesne varsa istek yapan istemciye tahsis edilir.
2. Client, pooldan bir nesne ister. Poolda boşta nesne yok ise, pool size kontrol edilir. Havuzdaki nesne sayısı, havuzun boyutundan küçükse yeni bir nesne oluşturularak ilgili istemciye tahsis edilir.
3. Client, pooldan bir nesne ister. Poolda boşta nesne yok ise, pool size kontrol edilir. Eğer pool size'a da ulaşılmışsa, thread beş saniye olacak şekilde diğer threadlerin kullandığı nesneleri bırakmasını bekleyecektir.

```
public ReusableConnection acquireConnection() {

     Lock lock = new Lock();

        // Bosta bir object ara, kullanimda degilse return et.
        for ( ReusableConnection connection : connectionPool) {
            if ( !connection.isInUse() ) {
             connection.setInUse(true);
                lock.release();
                return connection;
```

```
            }
        }
        // Tum nesneler kullanimda, nesne uretebilmek icin pool size
kontrol et.
        if (connectionPool.size() >= getPoolSize()) {
            //System.out.println("All connections in pool are in
use!!!");
            lock.release();
            return null;
        }

        // pool size dolmamis, yeni nesne uretebiliriz.
        ReusableConnection connection = new ReusableConnection();
        connection.setInUse(true);
        connectionPool.add(connection);

        lock.release();
        return connection;
    }
```

**releaseConnection()** metodu ile threadler, kullandıkları nesnelerin isInUse özelliğini false ederek havuzda ilgili nesnenin artık erişilebilir olduğunu ifade ederler.

```
public void releaseConnection(ReusableConnection object) {
        Lock lock = new Lock();

        int idx  = connectionPool.indexOf(object);
        ReusableConnection connection = connectionPool.get(idx);

        // kullanim durumunu kaldiriyoruz. nesne bosa duser.
        connection.setInUse(false);

        lock.release();
    }
```

ConnectionPool içerisindeki nesneler, ReusableConnection sınıfındaki nesnelerdir. Bu nesnelerden oluşan bir havuz, clientlar arasında paylaştırılır.

**ReusableConnection.java.**

```
public class ReusableConnection {
 private boolean inUse = false;
```

```
    public boolean isInUse() {
        return inUse;
    }

    public void setInUse(boolean inUse) {
        this.inUse = inUse;
    }
}
```

Main metot içerisinde örnek özelinde 8 thread oluşturularak pool nesnesi, bu threadler ile paylaştırılıp örnek çalıştırılır:

**ConnectionPoolMain.java.**

```
ConnectionPool pool = ConnectionPool.getInstance();

 for (int i=0; i<8; i++) {
  WorkerThread thread = new WorkerThread(pool);
  thread.start();
 }
```

Örneğin incelenmesi:

**WorkerThread.java.**

```
class WorkerThread extends Thread {

 ConnectionPool pool;

 public WorkerThread(ConnectionPool pool) {
  this.pool = pool;
 }

 public void run() {
  ReusableConnection connection = pool.acquireConnection();

  if (connection == null) {
   System.out.println("\n" + Thread.currentThread().getName() + " no
 more connection, reached max pool size: " + pool.getPoolSize() +
        "\nWill rety after 5 seconds...");

   try {
    Thread.sleep(5000);
   } catch (InterruptedException e) {
    // TODO Auto-generated catch block
```

```
    e.printStackTrace();
   } finally {
    connection = pool.acquireConnection();
   }
  }
  // use the connection
  System.out.println(Thread.currentThread().getName() + ". Using
 connection: " + connection.hashCode());
  // ...
  // ...
  // release the connObject
  pool.releaseConnection(connection);
  System.out.println(Thread.currentThread().getName() + " has released
 object: " + connection.hashCode());
 }
}
```

Örnek içerisinde her thread,

```
ReusableConnection connection = pool.acquireConnection();
```

komutu ile havuzdan bir nesne almak için istekte bulunur. Eğer null dönmezse, nesne görevini icra eder. Ardından objeyi serbest bırakır.

Eğer acqure metodundan null cevabı gelirse, Thread beş saniye sonra tekrardan nesneye erişim isteğinde bulunacaktır.

- Örnekte pool size 4, üretilen thread sayısı ise 8'dir. Bu durumda çıktıyı yorumlayalım:

```
Thread-0. Using connection: 1064548420

Thread-7 no more connection, reached max pool size: 4
Will rety after 5 seconds...

Thread-6 no more connection, reached max pool size: 4
Will rety after 5 seconds...

Thread-5 no more connection, reached max pool size: 4
Will rety after 5 seconds...

Thread-4 no more connection, reached max pool size: 4
Will rety after 5 seconds...
```

```
Thread-2. Using connection: 68393799
Thread-2 has released object: 68393799
Thread-3. Using connection: 6949854
Thread-1. Using connection: 1712763777
Thread-3 has released object: 6949854
Thread-0 has released object: 1064548420
Thread-1 has released object: 1712763777

-------------SEPERATOR------------------

Thread-7. Using connection: 1064548420
Thread-7 has released object: 1064548420
Thread-6. Using connection: 68393799
Thread-6 has released object: 68393799
Thread-5. Using connection: 1712763777
Thread-5 has released object: 1712763777
Thread-4. Using connection: 1064548420
Thread-4 has released object: 1064548420
```

Görüldüğü üzere pool size 4 olduğundan, Thread 0, 1, 2 ve 3 havudan objeye erişebilmiş ve görevlerini icra edip nesneleri bırakmışlardır. Thread 4, 5, 6 ve 7 ise pool size dolduğundan istekleri karşılanmamıştır.

Seperator sonrası çıktılarda ise beş saniye bekledikten sonra havuzdan nesne isteğinde bulunan threadler görünmektedir. Burada dikkat edilecek husus:

- Thread 0'ın kullanıp ardından release ettiği 1064548420 numaralı nesne, Thread 7 tarafından kullanılmıştır.
- Thread 2'nin kullanıp release ettiği 68393799 numaralı nesneyi ise Thread 6 kulllanmıştır.

Aynı şekilde ilk aşamada dört thread tarafından kullanılıp release edilen dört nesne, yenisi üretilmeksizin diğer dört thread tarafından kullanılmıştır. Böylelikle object pool tasarım kalıbının gereği sağlanmıştır.