Object Pool ve Singleton ile Multi-Thread Projesi

Yazılım dünyasında nesneler sınıflardan örnek alındığı sırada 'örnek alınma zamanı' denilebilecek bir süre kaybeder. Object pool tasarım deseni bu problemi ortadan kaldırmaya yönelik çalışan bir tasarım desenidir.

Probleme örnek vermek gerekirse , üniversite sınavına giren ve sonuç bekleyen 2 milyon öğrenci olduğunu varsayalım. Üniversite sonucu açıklanacağı esnada 2 milyon öğrencinin aynı anda sonuçların yayınlandığı internet sitesine giriş yapmaya çalıştığını ele alalım. Bu durumda giriş yapmaya çalışan öğrencilerden çok büyük bir kısmı başarılı giriş yapamayacak ve giriş yapması belki de saatler sürecek.

Bu problemin çözümü olarak sistemin açılmasından saatler öncesinde çalışan bir login servisimiz olduğunu , bu servis çalıştığı her an object pool'a yeni bir login kaydettiğini (thread sayısı sınırlandırılabilir) ve login servisimize istek geldiği anda pool'dan isteği daha hızlı yanıtlamak üzere nesne kullanabilir.

Ben ise bu projede "Stadyum" problemini örneklemeye çalıştım. Bu problemi somutlaştırmak gerekirse , Futbol maçı öncesinde stadyuma giriş yapmaya çalışan 50.000 taraftar olduğunu varsayalım. Stadyum kapılarının (10 adet) ise belli bir zaman aralığında 1000 taraftar kabul edebileceğini ve sadece 1 kapı açıldığını düşünelim.

Belki de maç saatinden 5-6 saat önce stadyumda sıraya girmemiz gerekirdi. Eğer maç saatinden önce stadyuma ait 10 kapı açılmış olsaydı taraftarlar stadyuma daha fazla beklemek gerekmeden girebilirdi.

Başlangıç olarak problemimizde stadyum 1 tane olmak zorunda olduğu için Singleton Pattern kullanılmıştır.

```java
public abstract class Stadium {
    private static ConcurrentLinkedQueue<StadiumGate> gatePool;
    private ScheduledExecutorService scheduledExecutorService;

    public Stadium(final int minObject) {
        initializeGatePool(minObject);
    }

    private Stadium(){

    }

    protected abstract StadiumGate openGate();

    private void initializeGatePool(final int minObject) {
        gatePool = getInstance();
        for (int i = 0; i < minObject; i++) {
            gatePool.add(openGate());
        }
    }

    public static ConcurrentLinkedQueue<StadiumGate> getInstance() {
        synchronized (Stadium.class){
            if(gatePool==null){
                return new ConcurrentLinkedQueue<StadiumGate>();
            }
            return gatePool;
        }
    }
```

Yazmış olduğum "StadiumDemo" classında stadyum kapılarının 10 adet olduğu(pool içerisinde minimum 10 adet kapı) belirtilmiştir. (Stadyuma maç sırasında yeni kapı açılmasının mümkün olduğunu varsayalım)

```java
  public void setUp() {
      pool = new Stadium( minObject: 10) {
          @Override
          protected StadiumGate openGate() {
              return new StadiumGate(processNo.incrementAndGet());
          }
      };
  }
```

Buna rağmen işlem yapacak thread sayımızın (kapı sayısının) ise 15 olduğu fakat 10 tanesinin çalışabileceğini (açıldığını) var sayalım.

```
public void testStadium() {
    ExecutorService executorService = Executors.newFixedThreadPool( nThreads: 10);
    for (int i = 1; i <= 15; i++) {
        executorService.execute(new OpenThread(pool, i));
    }
```

Açılan her kapıdan taraftarlar girmeye devam edebilir. Ta ki kapılardaki yoğunluk azalana kadar. Yoğunluk azaldıktan sonra ise kapılar (threadler) tekrar yoğunluk olana kadar sırayla kapanmaktadır. Tekrar yoğunluk olur ise sırada bekleyen threadler yeni kapıları açmakla görevlidir.

```
public class OpenThread extends Thread {

    private Stadium pool;
    private int threadNumber;

    public OpenThread(Stadium pool, int threadNumber) {
        this.pool = pool;
        this.threadNumber = threadNumber;
    }

    @Override
    public void run() {
        StadiumGate gate = pool.borrowGate();
        while(gate==null){
            try {
                OpenThread.sleep( millis: 1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            gate = pool.borrowGate();
        }
```

Son olarak kodu çalıştırdığımız zaman ise çıktılarımıza bakıp senaryomuzu tamamlayalım.

→ İlk olarak stadyumdaki kapılar açıldı.

```
Gate with gate no : 1 was opened
Gate with gate no : 2 was opened
Gate with gate no : 3 was opened
Gate with gate no : 4 was opened
Gate with gate no : 5 was opened
Gate with gate no : 6 was opened
Gate with gate no : 7 was opened
Gate with gate no : 8 was opened
Gate with gate no : 9 was opened
Gate with gate no : 10 was opened
```

→ Stadyum kapıları kullanılabilir duruma getirildi.

```
Gate(object) opened : 4      Gate process(thread) no : 4 was borrowed
Gate(object) opened : 3      Gate process(thread) no : 3 was borrowed
Gate(object) opened : 5      Gate process(thread) no : 5 was borrowed
Gate(object) opened : 9      Gate process(thread) no : 9 was borrowed
Gate(object) opened : 1      Gate process(thread) no : 1 was borrowed
Gate(object) opened : 7      Gate process(thread) no : 7 was borrowed
Gate(object) opened : 10     Gate process(thread) no : 10 was borrowed
Gate(object) opened : 6      Gate process(thread) no : 6 was borrowed
Gate(object) opened : 8      Gate process(thread) no : 8 was borrowed
Gate(object) opened : 2      Gate process(thread) no : 2 was borrowed
```

→ Taraftarlar kapılardan girişe başladı.

```
Fans enter the stadium through gate : 10
Fans enter the stadium through gate : 8
Fans enter the stadium through gate : 4
Fans enter the stadium through gate : 2
Fans enter the stadium through gate : 5
Fans enter the stadium through gate : 3
Fans enter the stadium through gate : 9
```