Object Pool Pattern

Mostly, performance is the key issue during the software development and the object creation, which may be a costly step.

Object Pool Pattern says that **" to reuse the object that are expensive to create".**

Basically, an Object pool is a container which contains a specified amount of objects. When an object is taken from the pool, it is not available in the pool until it is put back. **Objects in the pool have a lifecycle: creation, validation and destroy.**

A pool helps to manage available resources in a better way. There are many using examples: especially in application servers there are data source pools, thread pools etc.

Advantage of Object Pool design pattern

- It boosts the performance of the application significantly.
- It is most effective in a situation where the rate of initializing a class instance is high.
- It manages the connections and provides a way to reuse and share them.
- It can also provide the limit for the maximum number of objects that can be created.

Usage:

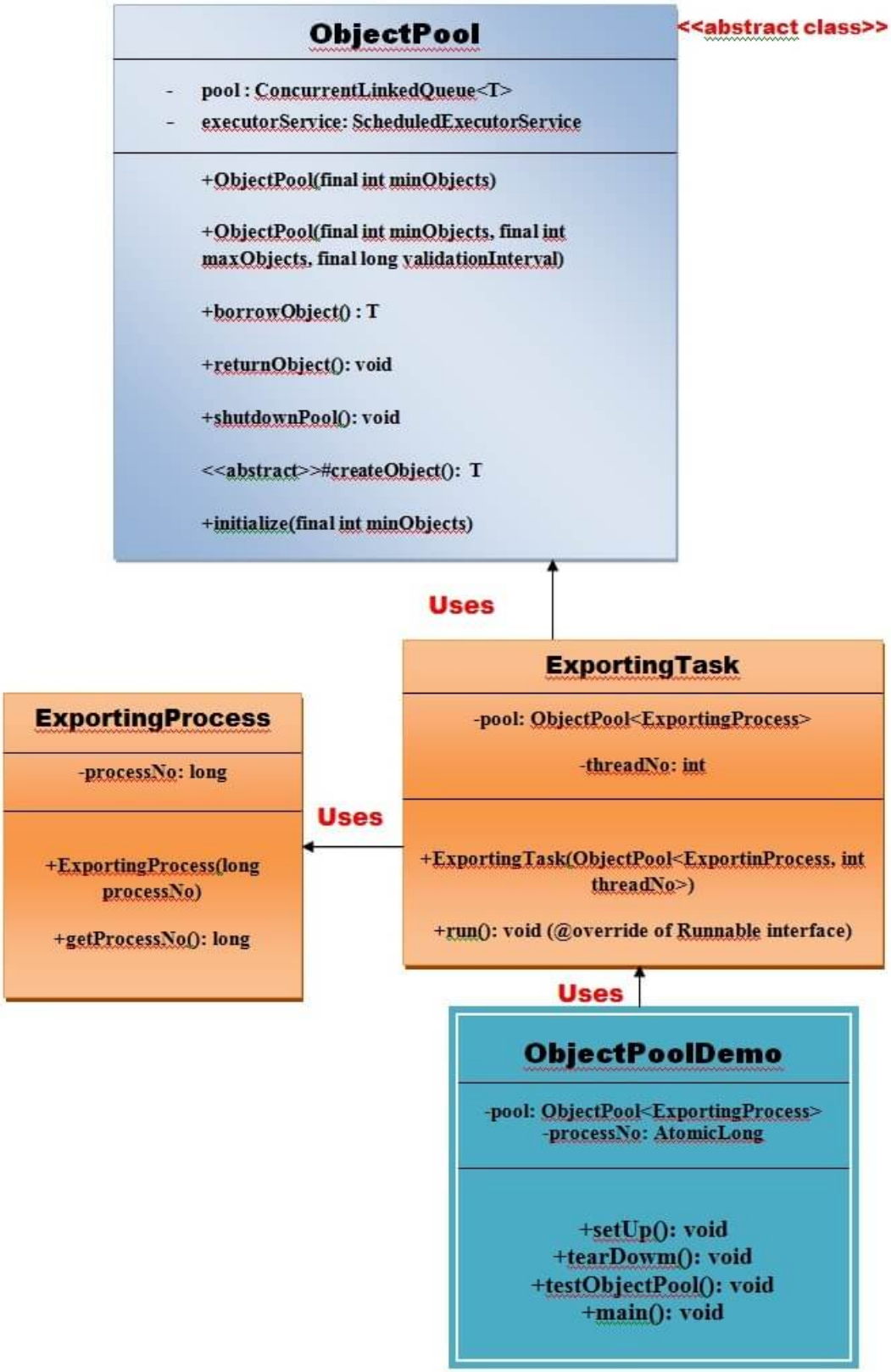
- When an application requires objects which are expensive to create. Eg: there is a need of opening too many connections for the database then it takes too longer to create a new one and the database server will be overloaded.
- When there are several clients who need the same resource at different times.

> NOTE: Object pool design pattern is essentially used in Web Container of the server for creating thread pools and data source pools to process the requests.

Example of Object Pool Pattern:

Let's understand the example by the given UML diagram.

UML for Object Pool Pattern



Implementation of above UML:

Step 1

Create an ObjectPool class that is used to create the number of objects.

File: ObjectPool.java

```
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

public abstract class ObjectPool<T> {
/*
  pool implementation is based on ConcurrentLinkedQueue from the java.util.concurrent package.
  ConcurrentLinkedQueue is a thread-safe queue based on linked nodes.
   Because the queue follows FIFO technique (first-in-first-out).
 */

   private ConcurrentLinkedQueue<T> pool;

 /*

   ScheduledExecutorService starts a special task in a separate thread and observes
   the minimum and maximum number of objects in the pool periodically in a specified
    time (with parameter validationInterval).
   When the number of objects is less than the minimum, missing instances will be created.
   When the number of objects is greater than the maximum, too many instances will be removed.
   This is sometimes useful for the balance of memory consuming objects in the pool.
*/
    private ScheduledExecutorService executorService;
      /*
     * Creates the pool.
     *
     * @param minObjects : the minimum number of objects residing in the pool
     */

    public ObjectPool(final int minObjects)
    {
       // initialize pool

       initialize(minObjects);

    }

    /*
      Creates the pool.
      @param minObjects:   minimum number of objects residing in the pool.
      @param maxObjects:   maximum number of objects residing in the pool.
      @param validationInterval: time in seconds for periodical checking of
         minObjects / maxObjects conditions in a separate thread.
      When the number of objects is less than minObjects, missing instances will be created.
      When the number of objects is greater than maxObjects, too many instances will be removed.
    */
```

```
    public ObjectPool(final int minObjects, final int maxObjects, final long validationInterval) {
      // initialize pool
       initialize(minObjects);
        // check pool conditions in a separate thread
      executorService = Executors.newSingleThreadScheduledExecutor();
      executorService.scheduleWithFixedDelay(new Runnable()  // annonymous class
      {
          @Override
          public void run() {
              int size = pool.size();

              if (size < minObjects) {
                  int sizeToBeAdded = minObjects + size;
                  for (int i = 0; i < sizeToBeAdded; i++) {
                      pool.add(createObject());
                  }
              } else if (size > maxObjects) {
                  int sizeToBeRemoved = size - maxObjects;
                  for (int i = 0; i < sizeToBeRemoved; i++) {
                      pool.poll();
                  }
              }
          }
      }, validationInterval, validationInterval, TimeUnit.SECONDS);
    }

   /*
      Gets the next free object from the pool. If the pool doesn't contain any objects,
      a new object will be created and given to the caller of this method back.

      @return T borrowed object
   */
    public T borrowObject() {
      T object;
      if ((object = pool.poll()) == null)
      {
          object = createObject();
      }
      return object;
    }
  /*
      Returns object back to the pool.
      @param object object to be returned
   */
    public void returnObject(T object) {
      if (object == null) {
          return;
```

```
        }
        this.pool.offer(object);
    }
   /*
        Shutdown this pool.
    */
      public void shutdown(){
        if (executorService != null){
            executorService.shutdown();
        }
    }
    /*
        Creates a new object.
         @return T new object
     */
    protected abstract T createObject();

    private void initialize(final int minObjects)  {
        pool = new ConcurrentLinkedQueue<T>();
        for (int i = 0; i < minObjects; i++) {
            pool.add(createObject());
        }
    }
}// End of the ObjectPool Class.
```

Step 2

Create an ExportingProcess class that will be used by ExportingTask class.

File: ExportingProcess.java

```
public class ExportingProcess {
 private long processNo;


    public ExportingProcess(long processNo)  {
         this.processNo = processNo;
        // do some  expensive calls / tasks here in future
        // .........
     System.out.println("Object with process no. " + processNo + " was created");
     }


    public long getProcessNo() {
        return processNo;
    }
}// End of the ExportingProcess class.
```

Step 3

Create an ExportingTask class that will use ExportingProcess and ObjectPool class.

File: ExportingTask.java

```
public class ExportingTask implements Runnable {
        private ObjectPool<ExportingProcess> pool;
        private int threadNo;
        public ExportingTask(ObjectPool<ExportingProcess> pool, int threadNo){
           this.pool = pool;
           this.threadNo = threadNo;
        }

        public void run() {
           // get an object from the pool
           ExportingProcess exportingProcess = pool.borrowObject();
           System.out.println("Thread " + threadNo + ": Object with process no. "
                 + exportingProcess.getProcessNo() + " was borrowed");

           //you can  do something here in future
           // .........

             // return ExportingProcess instance back to the pool
           pool.returnObject(exportingProcess);

           System.out.println("Thread " + threadNo +": Object with process no. "
                 + exportingProcess.getProcessNo() + " was returned");
        }

    }// End of the ExportingTask class.
```

Step 4

Create an ObjectPoolDemo class.

File: ObjectPoolDemo.java

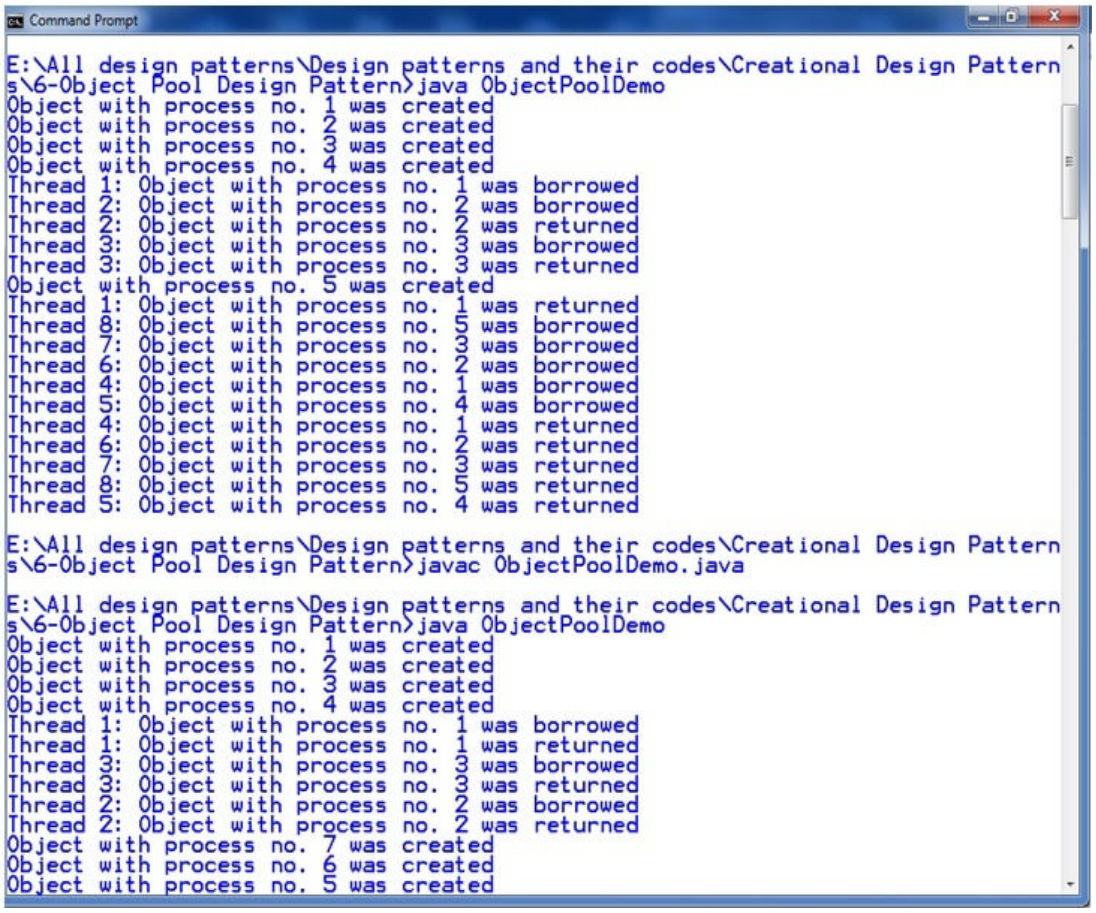
```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicLong;
public class ObjectPoolDemo{
      private ObjectPool<ExportingProcess> pool;
      private AtomicLong processNo=new AtomicLong(0);
      public void setUp() {
           // Create a pool of objects of type ExportingProcess.
          /*Parameters:
            1) Minimum number of special ExportingProcess instances residing in the pool = 4
            2) Maximum number of special ExportingProcess instances residing in the pool = 10
            3) Time in seconds for periodical checking of minObjects / maxObjects conditions
              in a separate thread = 5.
            -->When the number of ExportingProcess instances is less than minObjects,
```

```
            missing instances will be created.
        -->When the number of ExportingProcess instances is greater than maxObjects,
            too many instances will be removed.
        -->If the validation interval is negative, no periodical checking of
            minObjects / maxObjects conditions in a separate thread take place.
          These boundaries are ignored then.
       */
  pool = new ObjectPool<ExportingProcess>(4, 10, 5)
    {
       protected ExportingProcess createObject()
       {
          // create a test object which takes some time for creation
          return new ExportingProcess( processNo.incrementAndGet());
       }
    };
}
public void tearDown() {
    pool.shutdown();
}
public void testObjectPool() {
    ExecutorService executor = Executors.newFixedThreadPool(8);

    // execute 8 tasks in separate threads

    executor.execute(new ExportingTask(pool, 1));
    executor.execute(new ExportingTask(pool, 2));
    executor.execute(new ExportingTask(pool, 3));
    executor.execute(new ExportingTask(pool, 4));
    executor.execute(new ExportingTask(pool, 5));
    executor.execute(new ExportingTask(pool, 6));
    executor.execute(new ExportingTask(pool, 7));
    executor.execute(new ExportingTask(pool, 8));

    executor.shutdown();
    try {
       executor.awaitTermination(30, TimeUnit.SECONDS);
       } catch (InterruptedException e)

         {
          e.printStackTrace();
         }
}
public static void main(String args[])  {
    ObjectPoolDemo op=new ObjectPoolDemo();
    op.setUp();
    op.tearDown();
    op.testObjectPool();
```

```
    }
}//End of the ObjectPoolDemo class.
```

download this Object Pool Pattern Example

Output



← prev next →

Help Others, Please Share

Join Javatpoint Test Series

Placement Papers	AMCAT	Bank PO/Clerk	GATE
TCS	eLitmas	UPSSSC	NEET
HCL	Java	Government	CAT
Infosys	Python	Exams	Railway
IBM	C Programming	SSC	CTET
Accenture	Networking	Civil Services	IIT JEE
		SBI	

Learn Latest Tutorials

Ansible | Mockito | Talend | Azure

SharePoint | Powershell | Kali Linux | OpenCV

Kafka | Pandas | Joomla | Reinforcement

Preparation


Aptitude


Reasoning

Verbal A.


Interview


Company

Trending Technologies


AI

AWS

Selenium


Cloud

Hadoop

ReactJS

D. Science


Angular 7

Blockchain

Git

ML

DevOps

B.Tech / MCA

DBMS

DS

DAA


OS


C. Network


Compiler D.


Computer Organization and Architecture

COA


Discrete Mathematics Tutorial

D. Math.


Ethical Hacking Tutorial

E. Hacking


Computer Graphics Tutorial

C. Graphics


Software Engineering Tutorial

Software E.


html tutorial

Web Tech.