Object Pool

Creational Design Pattern

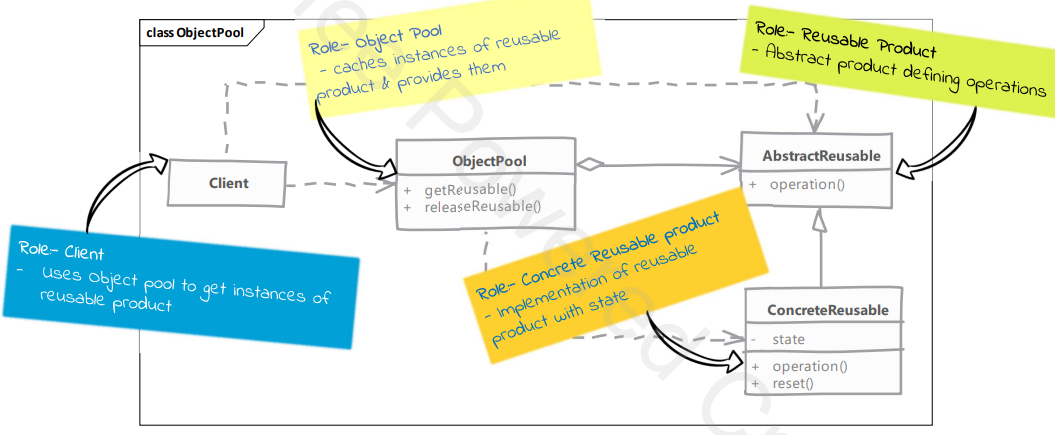
**What is an Object Pool:**

1. In our system if the cost of creating an instance of a class is high and we need a large number of objects of this class for a short duration.
2. Either pre-create objects of the class or collect unused instances in an in memory cache.

**Implementing a Singleton:**

1. Thread-safe caching of objects should be done in pool
2. Methods to acquire and release objects should be provided & pool should reset cached objects before giving them out.
3. Reusable objects must provide methods to reset its state upon release by code.
4. Decide whether to create new pooled objects when the pool is empty or to wait until objects become available.

**UML:**



**Implementation Considerations:**

1. Resetting object state should not be a costly operation otherwise you may end up losing your performance savings.
2. Pre-caching objects: meaning creating objects in advance can be helpful as it won’t slow down the code using these objects. However it may add-up to start up time & memory consumption.
3. Object pool’s synchronization should consider the reset time needed & avoid resetting in synchronized context if possible.

**Comparison with Prototype**

Object pool:

1. Cached objects that frequently live throughout programs' entire run.
2. Code using objects from object pool has to return the objects explicitly to pool. Depending on implementation, failing to return to pool may lead to memory and/or resource leak.

Prototype:

1. Prototype creates objects when needed and no caching is done.
2. Once an object is cloned no special treatment is needed by client code and the object can be used like any regular object.

**Implementation:**

1. Create an interface Poolable with a reset method.
2. Create a class ObjectPool<T extends Poolable> with a BlockingQueue<T> that contains all the objects and implement put and release methods.
3. Create class that extends Poolable.