The Object Pool Pattern is a creational design pattern that is used to manage a pool of reusable objects. It is particularly useful when the cost of creating and destroying objects is high, and the number of objects in use at any given time is relatively small compared to the total number of objects that could be created

* Initialization: When the object pool is created, it typically pre-allocates a certain number of objects and adds them to the pool.
* Thread Safety: Depending on the application's requirements, the object pool may need to be thread-safe if it's accessed concurrently by multiple threads.
* Object Reuse: Once an object is no longer needed by the client, it's returned to the pool for reuse. This helps in reducing the overhead of object creation and destruction.
* Object Lifespan Management: Some object pools may implement strategies for managing the lifespan of objects in the pool, such as timeout mechanisms to remove objects that have been idle for too long.
* Dynamic Pool Sizing: Depending on the application's requirements, the object pool may dynamically adjust its size, creating or removing objects based on demand.
* Resource Management: Object pools are commonly used for managing resources with limited availability, such as database connections, network connections, threads, and expensive-to-create objects.

*// Resource class (Example: Database Connection)
class* Resource {
 *// Resource-specific attributes and methods*}

*// Object Pool
class* ObjectPool {
 *private BlockingQueue*<Resource> pool;
 *private int* maxSize;

 *public* ObjectPool(*int* maxSize) {
 *this*.maxSize = maxSize;
 *this*.pool = *new* LinkedBlockingQueue<>(maxSize);
 initializePool();
 }

 *private void* initializePool() {
 *for* (*int* i = 0; i < maxSize; i++) {
 pool.offer(*new* Resource()); *// Use offer() to avoid blocking* }
 }

 *public* Resource acquireResource() *throws* InterruptedException {
 System.*out*.println("Acquiring resource from the pool...");
 Resource resource = pool.take(); *// Blocking call, waits until a resource is available* System.*out*.println("Resource acquired: " + resource);
 *return* resource;
 }

 *public void* releaseResource(Resource resource) {
 System.*out*.println("Releasing resource back to the pool: " + resource);
 pool.offer(resource); *// Use offer() to avoid blocking* }
}

*// Client code
public class* ObjectPoolTest {
 *public static void* main(String[] args) {
 ObjectPool objectPool = *new* ObjectPool(10);

 *try* {
 *// Acquire resources from the pool* Resource resource1 = objectPool.acquireResource();
 Resource resource2 = objectPool.acquireResource();

 *// Use resources

 // Release resources back to the pool* objectPool.releaseResource(resource1);
 objectPool.releaseResource(resource2);
 } *catch* (InterruptedException e) {
 e.printStackTrace();
 System.*err*.println("Failed to acquire resource: " + e.getMessage());
 }
 }
}

Acquiring resource from the pool...

Resource acquired: com.govtech.viswa.designpatterns.objectpool.Resource@32a1bec0

Acquiring resource from the pool...

Resource acquired: com.govtech.viswa.designpatterns.objectpool.Resource@5e8c92f4

Releasing resource back to the pool: com.govtech.viswa.designpatterns.objectpool.Resource@32a1bec0

Releasing resource back to the pool: com.govtech.viswa.designpatterns.objectpool.Resource@5e8c92f4