

Object Pool

Definition

The Problem (Ex: DBConnection Pool)

Code

Solution (Ex: DBConnection Pool)

Class Diagram

Implementation

Advantages

Disadvantages

▼ Resources

- Video → [43. LLD: Object Pool Design Pattern | Creational Design Pattern | Low Level Design](#)

Definition

The **Object Pool Design Pattern** is a **creational pattern** that manages a set (pool) of reusable objects (like DBConnection Objects). Instead of creating and destroying objects frequently, we **reuse pre-created instances** from a pool.

❗ Borrow an instance from the pool → Use it and → Return it to the pool

Object Pool Design Pattern is used when:

- Object **creation is expensive** (CPU/memory).
- The **same type of object** is needed repeatedly.
- You want to **limit the number of instances** (e.g., database connections).

The Problem (Ex: DBConnection Pool)

Code

```
1 // Resource - Reusable Object
2 public class DBConnection {
3     Connection mySqlConnection;
4     public DBConnection() {
5         try {
6             mySqlConnection = DriverManager
7                 .getConnection("jdbc:mysql://localhost:3306/DB",
8 "root", "root");
9         } catch (Exception e) {
10             e.printStackTrace();
11         }
12 }
```

```
1 // Object Pool
2 public class DBConnectionPoolManager {
3     List<DBConnection> freeConnections = new ArrayList<>();
4     List<DBConnection> inUseConnections = new ArrayList<>();
5     int INITIAL_POOL_SIZE = 3;
6     int MAX_POOL_SIZE = 6;
7
8     public DBConnectionPoolManager() {
```

```

9         for (int i = 0; i < INITIAL_POOL_SIZE; i++) {
10             freeConnections.add(new DBConnection());
11         }
12     }
13
14     public DBConnection getDBConnection() {
15         DBConnection dbConnection = null;
16         if (freeConnections.isEmpty() && inUseConnections.size() <
17 MAX_POOL_SIZE) {
18             freeConnections.add(new DBConnection());
19             System.out.println("New DBConnection created and added to
20 freeConnections list.");
21             System.out.println("freeConnections size: " +
22 freeConnections.size());
23             System.out.println("inUseConnections size: " +
24 inUseConnections.size());
25         } else if (freeConnections.isEmpty() &&
26 inUseConnections.size() >= MAX_POOL_SIZE) {
27             System.out.println("Pool is full. Cannot create new
28 DBConnection.");
29             return null;
30         }
31         dbConnection = freeConnections.remove(freeConnections.size() -
32 1);
33         inUseConnections.add(dbConnection);
34         System.out.println("DBConnection retrieved from
35 freeConnections list and added to inUseConnections list.");
36         System.out.println("freeConnections size: " +
37 freeConnections.size());
38         System.out.println("inUseConnections size: " +
39 inUseConnections.size());
40         return dbConnection;
41     }
42
43     public void releaseDBConnection(DBConnection dbConnection) {
44         if (dbConnection != null) {
45             inUseConnections.remove(dbConnection);
46             freeConnections.add(dbConnection);
47             System.out.println("DBConnection released from
48 inUseConnections list and added to freeConnections list.");
49             System.out.println("freeConnections size: " +
50 freeConnections.size());
51             System.out.println("inUseConnections size: " +
52 inUseConnections.size());
53         } else {
54             System.out.println("DBConnection is null. Cannot
55 release.");
56         }
57     }
58 }

```

```

1 // Client - Object Pool Problem Demo
2 public class Client {
3     public static void main(String[] args) {
4         // Creating a DBConnectionPoolManager
5         DBConnectionPoolManager poolManager = new
6 DBConnectionPoolManager();
7
8         // Creating 6 DBConnections (MAX_POOL_SIZE is 6)
9         DBConnection dbConnection1 = poolManager.getDBConnection();
10        DBConnection dbConnection2 = poolManager.getDBConnection();
11        DBConnection dbConnection3 = poolManager.getDBConnection();
12        DBConnection dbConnection4 = poolManager.getDBConnection();
13        DBConnection dbConnection5 = poolManager.getDBConnection();
14        DBConnection dbConnection6 = poolManager.getDBConnection();
15
16        // 7th DBConnection will not be created as the pool is full
17        (returns null)
18        DBConnection nullDBConnection = poolManager.getDBConnection();
19        System.out.println(nullDBConnection == null ? "DBConnection is
20 null as POOL is full." : "DBConnection is not null");

```

```

18     poolManager.releaseDBConnection(dbConnection6); // Releasing a
    DBConnection
19     DBConnection dbConnection = poolManager.getDBConnection(); //
    Reusing the released DBConnection
20
21     // ***** Issues with this code *****
22     // What happens if another client tries to create a new
    DBConnectionPoolManager?
23     DBConnectionPoolManager poolManager2 = new
    DBConnectionPoolManager();
24     // more connections added to the pool that exceeds the
    MAX_POOL_SIZE
25     System.out.println("===== Same Instance? =====");
26     System.out.println(poolManager == poolManager2 ? "Same
    instance of DBConnectionPoolManager" : "Different instances of " +
27         "DBConnectionPoolManager");
28
29 }
30 }

```

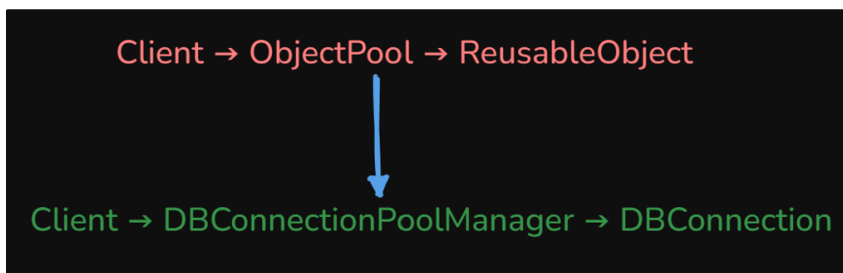
❌ What's **wrong** with the above code? What happens if another client tries to create a new `DBConnectionPoolManager` ?

- More connections were added to the pool that exceed the `MAX_POOL_SIZE` .
- Multiple connection list objects are being created to track connection usage.
- The system will eventually lead to a memory leak.
- Unreliable design.

Solution (Ex: DBConnection Pool)

✅ **This Object Pool Design Pattern is used in conjunction with the Singleton Design Pattern and requires thread safety when acquiring and releasing resources.**

Class Diagram



- **ObjectPool** (`ObjectPool`): Maintains available and in-use objects.
- **ReusableObject** (`DBConnection`): The object being pooled.

- **Client (Client)**: Uses and returns objects from the pool.

Implementation

```
1 // Resource - Reusable Object
2 public class DBConnection {
3     Connection mySqlConnection;
4
5     public DBConnection() {
6         try {
7             mySqlConnection = DriverManager
8                 .getConnection("jdbc:mysql://localhost:3306/DB",
9 "root", "root");
10        } catch (Exception e) {
11            e.printStackTrace();
12        }
13    }
14 }
15
16 // Object Pool - Singleton Implementation
17 public class DBConnectionPoolManager {
18
19     // Singleton
20     private static DBConnectionPoolManager
21     dbConnectionPoolManagerInstance = null;
22
23     List<DBConnection> freeConnections = new ArrayList<>();
24     List<DBConnection> inUseConnections = new ArrayList<>();
25     int INITIAL_POOL_SIZE = 3;
26     int MAX_POOL_SIZE = 6;
27
28     // private constructor
29     private DBConnectionPoolManager() {
30         for (int i = 0; i < INITIAL_POOL_SIZE; i++) {
31             freeConnections.add(new DBConnection());
32         }
33     }
34
35     // Singleton - thread-safe double-checked locking
36     public static DBConnectionPoolManager getInstance() {
37         if (dbConnectionPoolManagerInstance == null) {
38             synchronized (DBConnectionPoolManager.class) {
39                 if (dbConnectionPoolManagerInstance == null) {
40                     dbConnectionPoolManagerInstance = new
41 DBConnectionPoolManager();
42                 }
43             }
44         }
45         return dbConnectionPoolManagerInstance;
46     }
47
48     // Thread-safe: Only one thread can access this method at a time
49     // and modify the freeConnections and inUseConnections lists
50     public synchronized DBConnection getDBConnection() {
51         DBConnection dbConnection = null;
52         if (freeConnections.isEmpty() && inUseConnections.size() <
53 MAX_POOL_SIZE) {
54             freeConnections.add(new DBConnection());
55             System.out.println("New DBConnection created and added to
56 freeConnections list.");
57             System.out.println("freeConnections size: " +
58 freeConnections.size());
59             System.out.println("inUseConnections size: " +
60 inUseConnections.size());
61         } else if (freeConnections.isEmpty() &&
62 inUseConnections.size() >= MAX_POOL_SIZE) {
63             System.out.println("Pool is full. Cannot create new
64 DBConnection.");
65             return null;
66         }
67     }
68 }
```

```

43     }
44     dbConnection = freeConnections.remove(freeConnections.size() -
1);
45     inUseConnections.add(dbConnection);
46     System.out.println("DBConnection retrieved from
freeConnections list and added to inUseConnections list.");
47     System.out.println("freeConnections size: " +
freeConnections.size());
48     System.out.println("inUseConnections size: " +
inUseConnections.size());
49     return dbConnection;
50 }
51
52 // Thread-safe: Only one thread can access this method at a time
53 // and modify the freeConnections and inUseConnections lists
54 public synchronized void releaseDBConnection(DBConnection
dbConnection) {
55     if (dbConnection != null) {
56         inUseConnections.remove(dbConnection);
57         freeConnections.add(dbConnection);
58         System.out.println("DBConnection released from
inUseConnections list and added to freeConnections list.");
59         System.out.println("freeConnections size: " +
freeConnections.size());
60         System.out.println("inUseConnections size: " +
inUseConnections.size());
61     } else {
62         System.out.println("DBConnection is null. Cannot
release.");
63     }
64 }
65 }

```

```

1 // Client - Object Pool Solution Demo
2 public class Client {
3     public static void main(String[] args) {
4         System.out.println("===== Object Pool Design Pattern
=====");
5         // Creating a DBConnectionPoolManager
6         DBConnectionPoolManager poolManager =
DBConnectionPoolManager.getInstance();
7
8         // Creating 6 DBConnections (MAX_POOL_SIZE is 6)
9         DBConnection dbConnection1 = poolManager.getDBConnection();
10        DBConnection dbConnection2 = poolManager.getDBConnection();
11        DBConnection dbConnection3 = poolManager.getDBConnection();
12        DBConnection dbConnection4 = poolManager.getDBConnection();
13        DBConnection dbConnection5 = poolManager.getDBConnection();
14        DBConnection dbConnection6 = poolManager.getDBConnection();
15
16        // 7th DBConnection will not be created as the pool is full
17        (returns null)
18        DBConnection nullDBConnection = poolManager.getDBConnection();
19        System.out.println(nullDBConnection == null ? "DBConnection is
null as POOL is full." : "DBConnection is not null");
20        poolManager.releaseDBConnection(dbConnection6); // Releasing a
DBConnection
21        DBConnection dbConnection7 = poolManager.getDBConnection(); //
Reusing the released DBConnection
22
23        // ***** Solution Demo *****
24        // What happens if another client tries to create a new
DBConnectionPoolManager?
25        DBConnectionPoolManager poolManager2 =
DBConnectionPoolManager.getInstance();
26        System.out.println("===== Same Instance? =====");
27        System.out.println(poolManager == poolManager2 ? "Same
instance of DBConnectionPoolManager" : "Different instances of " +
"DBConnectionPoolManager");
28    }

```

Advantages

- Reduce the overhead of creating and destroying the frequently required object (generally resource-intensive objects).
- Reduce the latency, as it uses the pre-initialized object.
- Prevent resource exhaustion by managing the number of resource-intensive object creations.

Disadvantages

- Resource leakage can happen if the object is not handled properly and is not returned to the pool.
- Required more memory to manage the pool.
- Pool management required thread safety, which is additional overhead.
- Adds application complexity because of managing the pool.