Object Pool

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 s**ame type of object** is needed repeatedly.
- You want to **limit the number of instances** (e.g., database connections).

The Problem (Ex: DBConnection Pool)

Code

```java
// Resource - Reusable Object
public class DBConnection {
    Connection mySQLConnection;
    public DBConnection() {
        try {
            mySQLConnection = DriverManager
                    .getConnection("jdbc:mysql://localhost:3306/DB", "root", "root");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
// Object Pool
public class DBConnectionPoolManager {
    List<DBConnection> freeConnections = new ArrayList<>();
    List<DBConnection> inUseConnections = new ArrayList<>();
    int INITIAL_POOL_SIZE = 3;
    int MAX_POOL_SIZE = 6;

    public DBConnectionPoolManager() {
```

```
        for (int i = 0; i < INITIAL_POOL_SIZE; i++) {
            freeConnections.add(new DBConnection());
        }
    }

    public DBConnection getDBConnection() {
        DBConnection dbConnection = null;
        if (freeConnections.isEmpty() && inUseConnections.size() < MAX_POOL_SIZE) {
            freeConnections.add(new DBConnection());
            System.out.println("New DBConnection created and added to freeConnections list.");
            System.out.println("freeConnections size: " + freeConnections.size());
            System.out.println("inUseConnections size: " + inUseConnections.size());
        } else if (freeConnections.isEmpty() && inUseConnections.size() >= MAX_POOL_SIZE) {
            System.out.println("Pool is full. Cannot create new DBConnection.");
            return null;
        }
        dbConnection = freeConnections.remove(freeConnections.size() - 1);
        inUseConnections.add(dbConnection);
        System.out.println("DBConnection retrieved from freeConnections list and added to inUseConnections list.");
        System.out.println("freeConnections size: " + freeConnections.size());
        System.out.println("inUseConnections size: " + inUseConnections.size());
        return dbConnection;
    }

    public void releaseDBConnection(DBConnection dbConnection) {
        if (dbConnection != null) {
            inUseConnections.remove(dbConnection);
            freeConnections.add(dbConnection);
            System.out.println("DBConnection released from inUseConnections list and added to freeConnections list.");
            System.out.println("freeConnections size: " + freeConnections.size());
            System.out.println("inUseConnections size: " + inUseConnections.size());
        } else {
            System.out.println("DBConnection is null. Cannot release.");
        }
    }
}
```

```
// Client - Object Pool Problem Demo
public class Client {
    public static void main(String[] args) {
        // Creating a DBConnectionPoolManager
        DBConnectionPoolManager poolManager = new DBConnectionPoolManager();

        // Creating 6 DBConnections (MAX_POOL_SIZE is 6)
        DBConnection dbConnection1 = poolManager.getDBConnection();
        DBConnection dbConnection2 = poolManager.getDBConnection();
        DBConnection dbConnection3 = poolManager.getDBConnection();
        DBConnection dbConnection4 = poolManager.getDBConnection();
        DBConnection dbConnection5 = poolManager.getDBConnection();
        DBConnection dbConnection6 = poolManager.getDBConnection();

        // 7th DBConnection will not be created as the pool is full (returns null)
        DBConnection nullDBConnection = poolManager.getDBConnection();
        System.out.println(nullDBConnection == null ? "DBConnection is 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  }
```

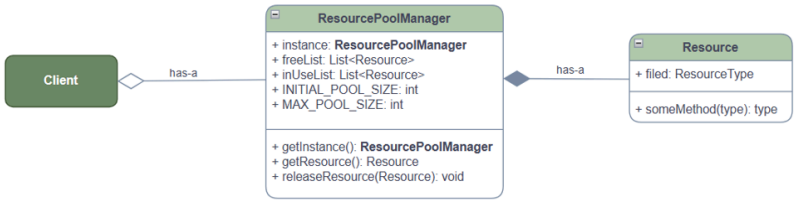
What's **wrong** with the above code? What happens if another client tries to create a new `DBConnectionPoolManager`?

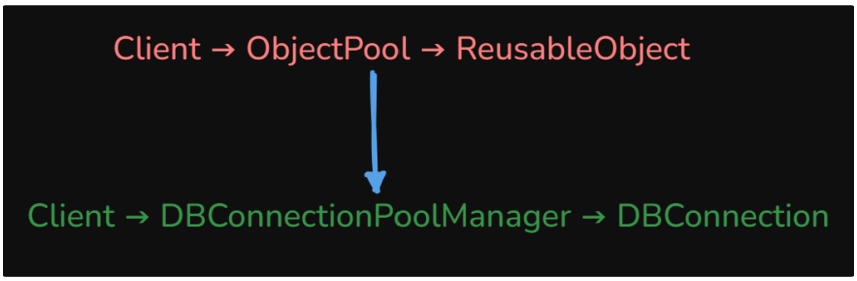
- More connections were added to the pool that exceed the `MAX_POOL_SIZE`.
- Multiple connection list objects are being created to track connection usage.
- The system will eventually lead to a memory leak.
- Unreliable design.

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

```java
// Resource - Reusable Object
public class DBConnection {
  Connection mySQLConnection;

  public DBConnection() {
      try {
          mySQLConnection = DriverManager
                  .getConnection("jdbc:mysql://localhost:3306/DB",
"root", "root");
      } catch (Exception e) {
          e.printStackTrace();
      }
  }
}
```

```java
// Object Pool - Singleton Implementation
public class DBConnectionPoolManager {

    // Singleton
    private static DBConnectionPoolManager
dbConnectionPoolManagerInstance = null;

    List<DBConnection> freeConnections = new ArrayList<>();
    List<DBConnection> inUseConnections = new ArrayList<>();
    int INITIAL_POOL_SIZE = 3;
    int MAX_POOL_SIZE = 6;

    // private constructor
    private DBConnectionPoolManager() {
        for (int i = 0; i < INITIAL_POOL_SIZE; i++) {
            freeConnections.add(new DBConnection());
        }
    }

    // Singleton - thread-safe double-checked locking
    public static DBConnectionPoolManager getInstance() {
        if (dbConnectionPoolManagerInstance == null) {
            synchronized (DBConnectionPoolManager.class) {
                if (dbConnectionPoolManagerInstance == null) {
                    dbConnectionPoolManagerInstance = new
DBConnectionPoolManager();
                }
            }
        }
        return dbConnectionPoolManagerInstance;
    }

    // Thread-safe: Only one thread can access this method at a time
    // and modify the freeConnections and inUseConnections lists
    public synchronized DBConnection getDBConnection() {
        DBConnection dbConnection = null;
        if (freeConnections.isEmpty() && inUseConnections.size() <
MAX_POOL_SIZE) {
            freeConnections.add(new DBConnection());
            System.out.println("New DBConnection created and added to
freeConnections list.");
            System.out.println("freeConnections size: " +
freeConnections.size());
            System.out.println("inUseConnections size: " +
inUseConnections.size());
        } else if (freeConnections.isEmpty() &&
inUseConnections.size() >= MAX_POOL_SIZE) {
            System.out.println("Pool is full. Cannot create new
DBConnection.");
            return null;
```

```
        }
        dbConnection = freeConnections.remove(freeConnections.size() - 1);
        inUseConnections.add(dbConnection);
        System.out.println("DBConnection retrieved from freeConnections list and added to inUseConnections list.");
        System.out.println("freeConnections size: " + freeConnections.size());
        System.out.println("inUseConnections size: " + inUseConnections.size());
        return dbConnection;
    }

    // Thread-safe: Only one thread can access this method at a time
    // and modify the freeConnections and inUseConnections lists
    public synchronized void releaseDBConnection(DBConnection dbConnection) {
        if (dbConnection != null) {
            inUseConnections.remove(dbConnection);
            freeConnections.add(dbConnection);
            System.out.println("DBConnection released from inUseConnections list and added to freeConnections list.");
            System.out.println("freeConnections size: " + freeConnections.size());
            System.out.println("inUseConnections size: " + inUseConnections.size());
        } else {
            System.out.println("DBConnection is null. Cannot release.");
        }
    }
}
```

```
// Client - Object Pool Solution Demo
public class Client {
    public static void main(String[] args) {
        System.out.println("======= Object Pool Design Pattern ======");
        // Creating a DBConnectionPoolManager
        DBConnectionPoolManager poolManager = DBConnectionPoolManager.getInstance();

        // Creating 6 DBConnections (MAX_POOL_SIZE is 6)
        DBConnection dbConnection1 = poolManager.getDBConnection();
        DBConnection dbConnection2 = poolManager.getDBConnection();
        DBConnection dbConnection3 = poolManager.getDBConnection();
        DBConnection dbConnection4 = poolManager.getDBConnection();
        DBConnection dbConnection5 = poolManager.getDBConnection();
        DBConnection dbConnection6 = poolManager.getDBConnection();

        // 7th DBConnection will not be created as the pool is full (returns null)
        DBConnection nullDBConnection = poolManager.getDBConnection();
        System.out.println(nullDBConnection == null ? "DBConnection is null as POOL is full." : "DBConnection is not null");
        poolManager.releaseDBConnection(dbConnection6); // Releasing a DBConnection
        DBConnection dbConnection7 = poolManager.getDBConnection(); // Reusing the released DBConnection

        // ******** Solution Demo **********
        // What happens if another client tries to create a new DBConnectionPoolManager?
        DBConnectionPoolManager poolManager2 = DBConnectionPoolManager.getInstance();
        System.out.println("====== Same Instance? ======");
        System.out.println(poolManager == poolManager2 ? "Same instance of DBConnectionPoolManager" : "Different instances of " +
                "DBConnectionPoolManager");
    }
```

```
29 }
```

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.