Category: Creational Design Pattern

- Manages the pool of reusable objects like DBConnection object.
- Borrow from the pool -> use it -> then return it back to the pool.

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 creation.

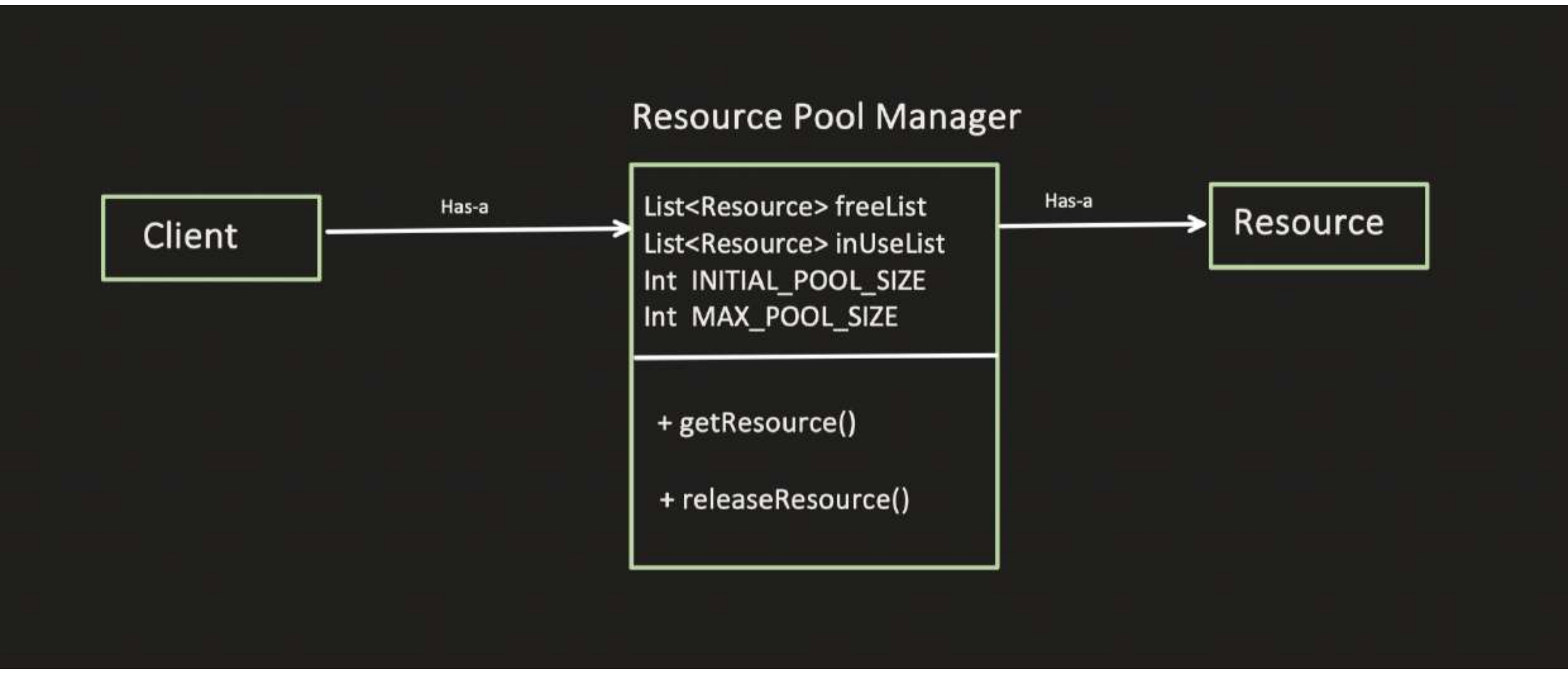
Disadvantages:

- Resource Leakage can happen, if object is not handled properly and not being returned to the pool.

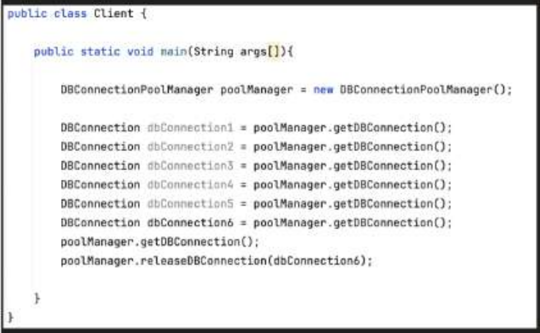
- Required more memory because of managing the pool.

- Pool management required thread safety, which is additional overhead.

- Adds application complexity because of managing the pool.


Resource Pool Manager
Client
Has-a
List<Resource> freeList
List<Resource> inUseList
Int INITIAL_POOL_SIZE
Int MAX_POOL_SIZE
+ getResource()
+ releaseResource()
Has-a
Resource

Many engineers makes 1 mistake while coding for this design pattern?



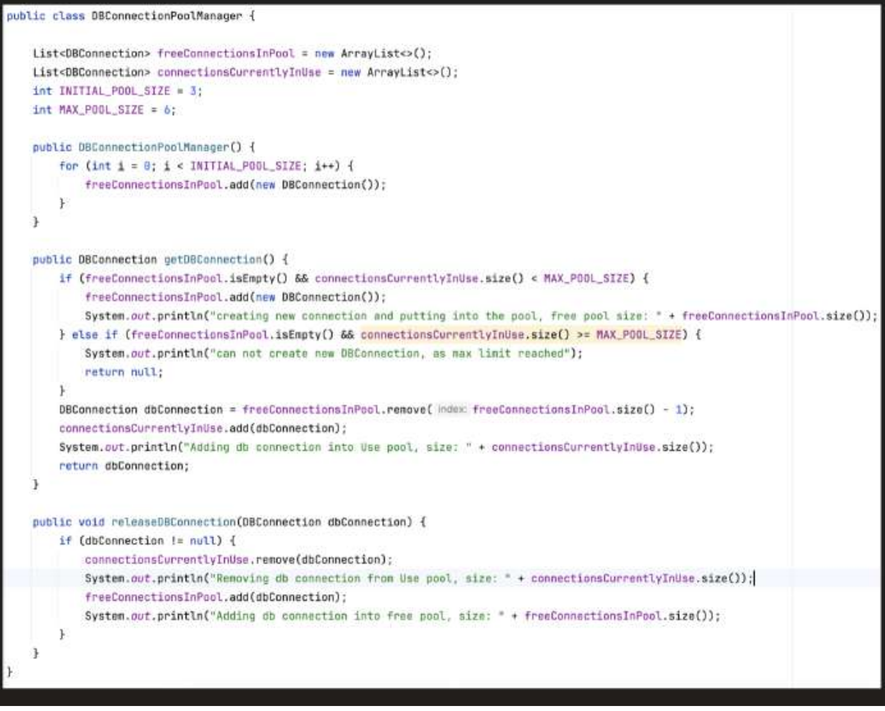
```java
public class Client {

    public static void main(String args[]){

        DBConnectionPoolManager poolManager = new DBConnectionPoolManager();

        DBConnection dbConnection1 = poolManager.getDBConnection();
        DBConnection dbConnection2 = poolManager.getDBConnection();
        DBConnection dbConnection3 = poolManager.getDBConnection();
        DBConnection dbConnection4 = poolManager.getDBConnection();
        DBConnection dbConnection5 = poolManager.getDBConnection();
        DBConnection dbConnection6 = poolManager.getDBConnection();
        poolManager.getDBConnection();
        poolManager.releaseDBConnection(dbConnection6);

    }
}
```



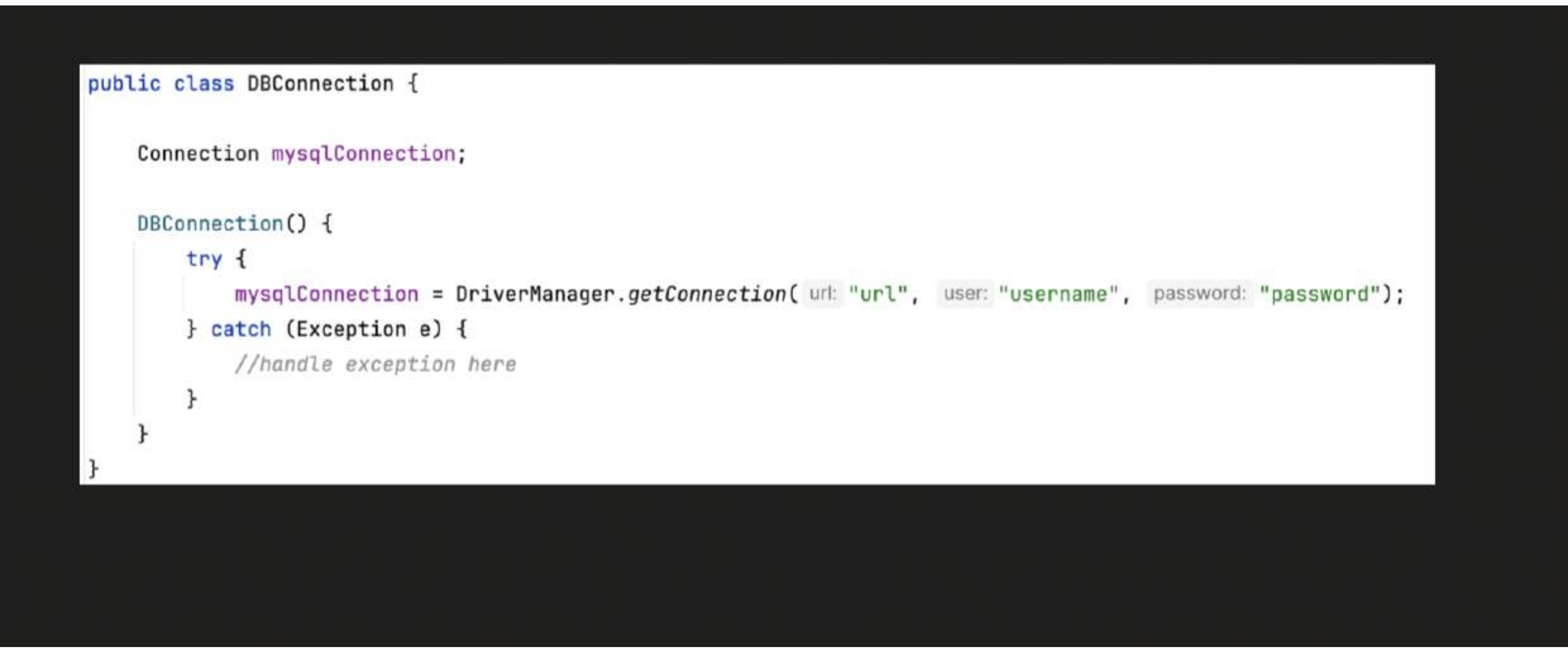
```java
public class DBConnectionPoolManager {

    List<DBConnection> freeConnectionsInPool = new ArrayList<>();
    List<DBConnection> connectionsCurrentlyInUse = new ArrayList<>();
    int INITIAL_POOL_SIZE = 3;
    int MAX_POOL_SIZE = 6;

    public DBConnectionPoolManager() {
        for (int i = 0; i < INITIAL_POOL_SIZE; i++) {
            freeConnectionsInPool.add(new DBConnection());
        }
    }

    public DBConnection getDBConnection() {
        if (freeConnectionsInPool.isEmpty() && connectionsCurrentlyInUse.size() < MAX_POOL_SIZE) {
            freeConnectionsInPool.add(new DBConnection());
            System.out.println("creating new connection and putting into the pool, free pool size: " + freeConnectionsInPool.size());
        } else if (freeConnectionsInPool.isEmpty() && connectionsCurrentlyInUse.size() >= MAX_POOL_SIZE) {
            System.out.println("can not create new DBConnection, as max limit reached");
            return null;
        }
        DBConnection dbConnection = freeConnectionsInPool.remove( index: freeConnectionsInPool.size() - 1);
        connectionsCurrentlyInUse.add(dbConnection);
        System.out.println("Adding db connection into Use pool, size: " + connectionsCurrentlyInUse.size());
        return dbConnection;
    }

    public void releaseDBConnection(DBConnection dbConnection) {
        if (dbConnection != null) {
            connectionsCurrentlyInUse.remove(dbConnection);
            System.out.println("Removing db connection from Use pool, size: " + connectionsCurrentlyInUse.size());
            freeConnectionsInPool.add(dbConnection);
            System.out.println("Adding db connection into free pool, size: " + freeConnectionsInPool.size());
        }
    }
}
```



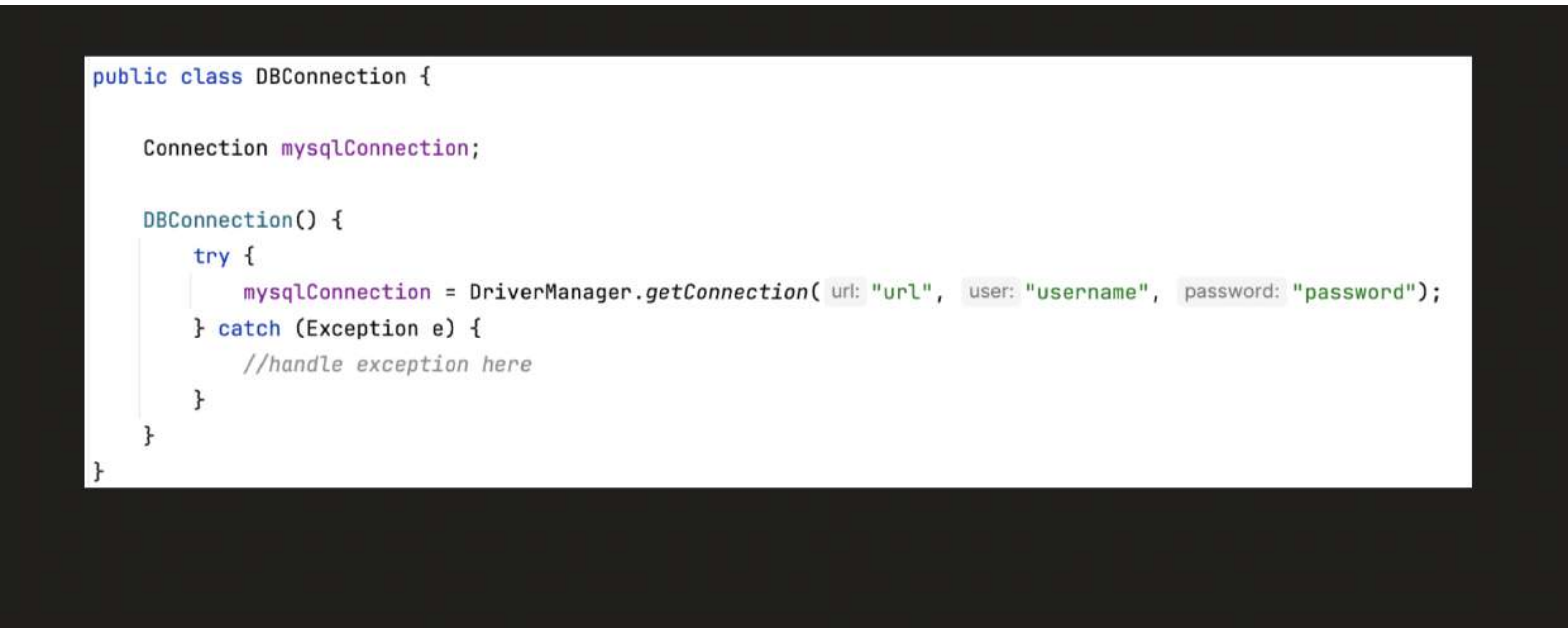
```java
public class DBConnection {

    Connection mysqlConnection;

    DBConnection() {
        try {
            mysqlConnection = DriverManager.getConnection( url: "url",  user: "username",  password: "password");
        } catch (Exception e) {
            //handle exception here
        }
    }
}
```

What's wrong with the above code?

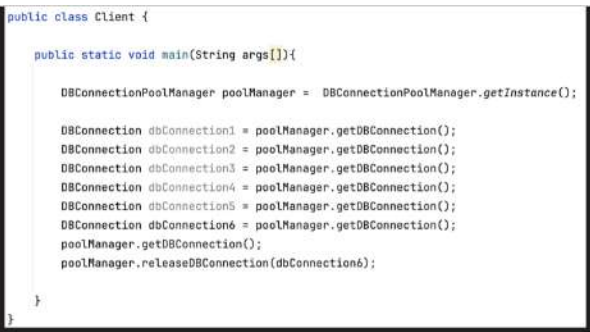
This Object Pool Design pattern is used with Singleton design pattern and required thread safety while acquiring and releasing the resource.



```
public class DBConnection {

    Connection mysqlConnection;

    DBConnection() {
        try {
            mysqlConnection = DriverManager.getConnection( url: "url",  user: "username",  password: "password");
        } catch (Exception e) {
            //handle exception here
        }
    }
}
```



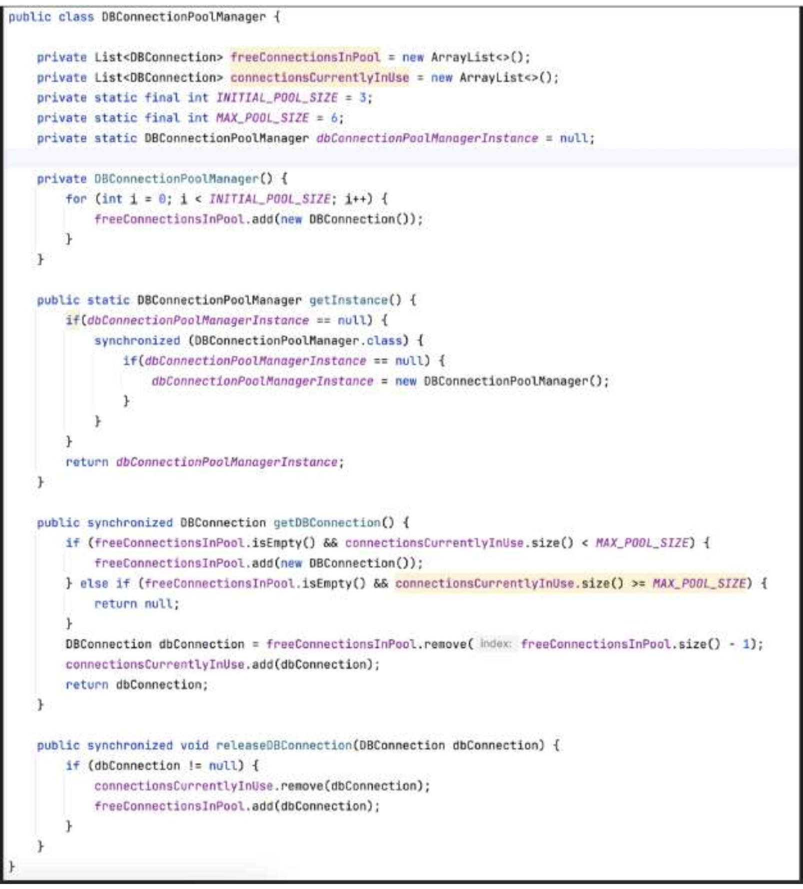
```java
public class Client {

    public static void main(String args[]){

        DBConnectionPoolManager poolManager =  DBConnectionPoolManager.getInstance();

        DBConnection dbConnection1 = poolManager.getDBConnection();
        DBConnection dbConnection2 = poolManager.getDBConnection();
        DBConnection dbConnection3 = poolManager.getDBConnection();
        DBConnection dbConnection4 = poolManager.getDBConnection();
        DBConnection dbConnection5 = poolManager.getDBConnection();
        DBConnection dbConnection6 = poolManager.getDBConnection();
        poolManager.getDBConnection();
        poolManager.releaseDBConnection(dbConnection6);

    }
}
```



```java
public class DBConnectionPoolManager {

    private List<DBConnection> freeConnectionsInPool = new ArrayList<>();
    private List<DBConnection> connectionsCurrentlyInUse = new ArrayList<>();
    private static final int INITIAL_POOL_SIZE = 3;
    private static final int MAX_POOL_SIZE = 6;
    private static DBConnectionPoolManager dbConnectionPoolManagerInstance = null;

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

    public static DBConnectionPoolManager getInstance() {
        if(dbConnectionPoolManagerInstance == null) {
            synchronized (DBConnectionPoolManager.class) {
                if(dbConnectionPoolManagerInstance == null) {
                    dbConnectionPoolManagerInstance = new DBConnectionPoolManager();
                }
            }
        }
        return dbConnectionPoolManagerInstance;
    }

    public synchronized DBConnection getDBConnection() {
        if (freeConnectionsInPool.isEmpty() && connectionsCurrentlyInUse.size() < MAX_POOL_SIZE) {
            freeConnectionsInPool.add(new DBConnection());
        } else if (freeConnectionsInPool.isEmpty() && connectionsCurrentlyInUse.size() >= MAX_POOL_SIZE) {
            return null;
        }
        DBConnection dbConnection = freeConnectionsInPool.remove( index: freeConnectionsInPool.size() - 1);
        connectionsCurrentlyInUse.add(dbConnection);
        return dbConnection;
    }

    public synchronized void releaseDBConnection(DBConnection dbConnection) {
        if (dbConnection != null) {
            connectionsCurrentlyInUse.remove(dbConnection);
            freeConnectionsInPool.add(dbConnection);
        }
    }
}
```