11 Creational: Prototype and Object Pool Patterns

**Objective** Convert expensive-to-create “Document” instances to use Prototype, and manage scarce DatabaseConnection objects via an Object Pool.  
 You will deliver three evolutionary levels for each pattern, showing how the design matures from naïve code to production-ready implementation.

#### **Starter code (src/main/java/legacy/)**

package legacy;

public class Document {

private String title, content;

public Document(String t,String c){ this.title=t; this.content=c; }

/\* getters / setters omitted \*/

}

package legacy;

public class DatabaseConnection {

public void executeQuery(String q){ System.out.println("RUN "+q); }

}

#### **Tasks**

1 — analysis/prototype\_objectpool\_problems.md  
 • Explain why repeated new Document(...) and new DatabaseConnection() hurt performance.  
 • List at least two thread-safety risks in naïve pooling.

2 — **Prototype Pattern** • *Level 1* shallow-clone via a copy-constructor.  
 • *Level 2* implement clone() using Cloneable plus manual deep-copy of a mutable list field you will add (List<String> tags).  
 • *Level 3* create a **Prototype Registry** (PrototypeRegistry) that caches “template” documents by key and returns clones on demand.

3 — **Object Pool Pattern** • *Level 1* simple list, no synchronisation.  
 • *Level 2* synchronised acquire/release.  
 • *Level 3* thread-safe pool backed by java.util.concurrent.ArrayBlockingQueue with timeout support.

4 — clean.Main  
 • Clone a “design doc” prototype, change its title, print both objects to show independence.  
 • Spin up 10 threads, each grabbing a connection from Level 3 pool, running SELECT 1, then releasing it.

5 — reflection.md  
 • Compare the three maturity levels for each pattern.  
 • What trade-offs (memory, complexity, contention) appear as you harden the design?

#### **Deliverables**

analysis/prototype\_objectpool\_problems.md

src/main/java/clean/prototype/\*\* ← Document + levels 1-3 + registry

src/main/java/clean/pool/\*\* ← Connection pool levels 1-3

src/main/java/clean/Main.java

reflection.md

README.md

## **Solution reference**

### **Prototype (Levels 1 → 3)**

#### **Level 1 — Shallow copy**

package clean.prototype;

public class Document {

private String title, content;

public Document(String t,String c){ this.title=t; this.content=c; }

public Document(Document source){ // ← shallow copy ctor

this.title = source.title;

this.content = source.content;

}

/\* getters/setters & toString \*/

}

#### **Level 2 — Deep copy with tags**

public class Document implements Cloneable {

private String title, content;

private List<String> tags = new ArrayList<>();

/\* ... ctor & mutators ... \*/

@Override public Document clone(){

try{

Document copy = (Document) super.clone(); // shallow

copy.tags = new ArrayList<>(this.tags); // deep

return copy;

}catch(CloneNotSupportedException e){ throw new AssertionError(e); }

}

}

#### **Level 3 — Prototype Registry**

package clean.prototype;

import java.util.\*;

public final class PrototypeRegistry {

private static final Map<String, Document> cache = new HashMap<>();

public static void addTemplate(String key, Document proto){ cache.put(key, proto); }

public static Document cloneOf(String key){ return cache.get(key).clone(); }

}

### **Object Pool (Levels 1 → 3)**

#### **Level 1 — Unsynchronised**

public class SimplePool {

private final List<DatabaseConnection> list = new ArrayList<>();

public SimplePool(int size){ for(int i=0;i<size;i++) list.add(new DatabaseConnection()); }

public DatabaseConnection acquire(){ return list.remove(0); }

public void release(DatabaseConnection c){ list.add(c); }

}

#### **Level 2 — S synchronised**

public class SynchronizedPool {

private final List<DatabaseConnection> list = new ArrayList<>();

public SynchronizedPool(int n){ for(int i=0;i<n;i++) list.add(new DatabaseConnection()); }

public synchronized DatabaseConnection acquire(){

return list.isEmpty()? null : list.remove(0);

}

public synchronized void release(DatabaseConnection c){ list.add(c); }

}

#### **Level 3 — Blocking queue with timeout**

import java.util.concurrent.\*;

public class BlockingPool {

private final ArrayBlockingQueue<DatabaseConnection> queue;

public BlockingPool(int n){

queue = new ArrayBlockingQueue<>(n);

for(int i=0;i<n;i++) queue.add(new DatabaseConnection());

}

public DatabaseConnection acquire(long timeoutMs) throws InterruptedException {

return queue.poll(timeoutMs, TimeUnit.MILLISECONDS);

}

public void release(DatabaseConnection c){ queue.offer(c); }

}

### **Main demo**

package clean;

import clean.prototype.\*;

import clean.pool.BlockingPool;

import legacy.DatabaseConnection;

import java.util.concurrent.\*;

public class Main {

public static void main(String[] args) throws Exception {

/\* --- Prototype demo -------------------------------------------- \*/

Document template = new Document("Design Patterns","Prototype vs Pool");

PrototypeRegistry.addTemplate("design", template);

Document copy = PrototypeRegistry.cloneOf("design");

copy.setTitle("Cloned Draft");

System.out.println("Template : "+template);

System.out.println("Clone : "+copy);

/\* --- Object-pool demo ----------------------------------------- \*/

BlockingPool pool = new BlockingPool(2);

ExecutorService es = Executors.newFixedThreadPool(10);

for(int i=0;i<10;i++){

es.submit(() -> {

try {

DatabaseConnection c = pool.acquire(500);

if(c!=null){

c.executeQuery("SELECT 1");

Thread.sleep(100); // simulate work

pool.release(c);

}else System.out.println("No connection");

}catch(Exception e){ e.printStackTrace(); }

});

}

es.shutdown();

es.awaitTermination(2,TimeUnit.SECONDS);

}

}

### **Level cheat-sheet**

| **Pattern** | **Level** | **Characteristics** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- |
| **Prototype** | 1 shallow | copy-ctor | quickest | shared refs risk |
|  | 2 deep | clone() + defensive copies | safe copies | extra code |
|  | 3 registry | cache template prototypes | zero creation cost at call-site | central cache to maintain |
| **Object Pool** | 1 list | lightning-fast | unsafe threads | race conditions |
|  | 2 sync | synchronized methods | simple thread-safe | may block long |
|  | 3 blocking queue | ArrayBlockingQueue, timeout | scalable, non-busy wait | a bit more boilerplate |

These incremental levels let you discuss performance, thread-safety, and maintainability — demonstrating how Prototype and Object Pool evolve from “quick-and-dirty” to robust production implementations.