

Articles and events of interest to the software development community

In This Issue:

- The Object Pool Pattern - p.1
- Employee Spotlight: Rob Myers - p.6
- Seminars We Can Give - p.7
- Personal Book Recommendation - p.8
- Technical Book Recommendation - p.8
- What's Happening At Net Objectives This Month - p.9
- What We Do: Net Objectives' Courses - p.12



Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

The Object Pool Pattern

Alan Shalloway, *Net Objectives*, alshall@netobjectives.com

James R. Trott, *Millennium Relief & Development Services*, jtrott@mrds.org

(Chapter 22 from the Second Edition of their book: "Design Patterns Explained: A New Perspective on Object-Oriented Design" released October 2004)

Design patterns do not exist in isolation, but work in concert with other design patterns to help you create more robust applications. In this book, you will gain a solid understanding of twelve core design patterns and a pattern used in analysis. You will gain enough of a foundation that you will be able to read the design pattern literature, if you want to, and possibly discover patterns on your own. Most importantly, you will be better equipped to create flexible and complete software that is easier to maintain.

Overview	1
A Problem Requiring the Management of Objects	2
The Object Pool Pattern	9
Observation: Factories Can Do Much More Than Instantiation	9
Summary	11
Review Questions	11

Overview

In this chapter

This chapter discusses the Object Pool pattern. Rather than using the international e-commerce case study, I will illustrate this pattern with an actual project I worked on several years ago. This project gives evidence of the proposal I made in chapter 10 on the Bridge pattern. That was that if you understand the principles of patterns and work on a project where a pattern that is unknown to applies, you are likely to derive it yourself. In particular, on this project, I ended up deriving the Object Pool pattern from the basic principles of design patterns. Later, I discovered that what I had written had been written by others as well, and named the "Object Pool pattern." The point I want to reinforce for you is that it is more important for you to understand the basic principles of design patterns than it is to memorize a bunch of diagrams and patterns or to have a giant reference book. Knowing how to think with patterns will make you more likely to find the solutions or derive the patterns that are right for your situation.

In this chapter,

- I describe the Object Pool Pattern
- I show how patterns can be used to focus on the most important aspects of the project—deferring things that cause no risk.
- I show how factories can not only manage objects but can also be the best place to describe the logic about their proper functioning.
- I show how patterns and knowing how to encapsulate issues facilitates agile coding practices.

A Problem Requiring the Management of Objects

A pool of connections

Several years ago, I was contracting for a company that was making a web-based, personal investment system. Although such systems are common today, back then, this was new territory. The general intent for this application was to give users the ability to look at their personal investments over the web and enter orders to buy and sell the stocks, bonds, etc., that they had in their accounts. The information about the user's accounts was kept on a mainframe computer. The physical architecture is shown in Figure 22-1.

The front-end that we were using was written with Java servlets; this fed the middle layer that I was writing in C++. My tier communicated with the mainframe to retrieve information about the user's investments or to submit orders. The only way to communicate with the mainframe was through TCP/IP connections using a special messaging protocol that was specified by the company that supported the mainframe. The middleware code that I was writing had the responsibility of taking the input from the user and verifying its validity. It did this by checking with the mainframe computer which had all the information. The servlet front-end essentially just did formatting and basic checking.

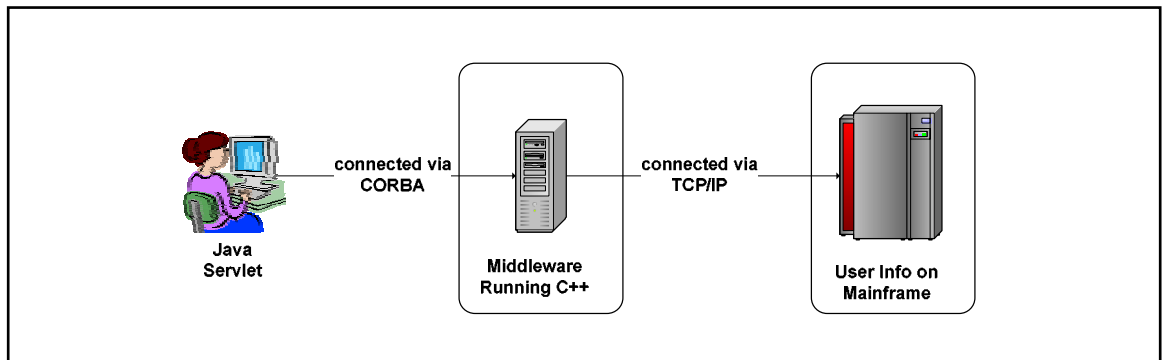


Figure 22-1 n-Tier architecture of the personal investment system

A typical user session would look like:

Layer: Browser		Layer: Middleware		Layer: Mainframe
User logs in	→	Middleware gets user ID and password. Formats message and sends it to Mainframe via TCP/IP to see if they are valid	→	Mainframe accepts request and checks for its validity. Responds accordingly.
User requests to see their portfolio	→	Middleware verifies this is a validly logged on user. If so, it passes the request down to the Mainframe.	→	Mainframe loads up the user's portfolio and sends it back to the middleware application.
Display the information	←	Middleware gets Mainframe's response and sends it back to the front-end	←	

Volume 1, Issue 9
September 2004
©2004, Net Objectives,
All Rights Reserved



Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

Throughput was the primary concern

There was some concern about the performance of this system. I was told to be more interested in throughput and not to worry about response time. Since my middleware application was taking requests from many, many people, this requirement meant that I wanted to handle as many transactions as possible but that I did not need to worry about any particular thread's time through the system.

I had never written an application like this before. I had no good ideas for how to balance this load. I knew it would require more than one TCP/IP connection to the mainframe, but how many? I was pretty sure that the right number was somewhere between 2 and 100 – one was too few and more than 100 would not do me any good because the available bandwidth could not handle more than 20 anyway.

So how many TCP/IP connections should I use? Although I knew I did not know the answer right at the start, I would have to figure it out before the end of the project. This was a dilemma for me. I didn't know how to start and did not have the time to do a lot of testing or emulation. And at that time, I did not really know that much about TCP/IP. This meant that the TCP/IP connection was a high risk for the project, even more so

because no one on the project knew all that was entailed in it.

I prefer to tackle high-risk issues quickly. If they cannot be addressed, then I try to isolate them, to try to turn them into lower-risk items. To solve the TCP/IP connection issue, I knew what I needed was an end-to-end solution as soon as possible, something that could take a request from the browser, perform some logic, pass it through to the mainframe, get a response, and send that information back to the browser. Once that had been accomplished, this risk would be reduced because I would have demonstrated the ability to communicate through a TCP/IP connection.

To implement this end-to-end solution with the TCP/IP connection, I had to do the following:

1. Establish a robust TCP/IP connection
2. Determine the number of TCP/IP connections to be used
3. Establish methods to load balance the connections
4. Handle errors on the TCP/IP connections (these were known to sometimes fail)

About the authors --

Alan Shalloway

Alan Shalloway is the CEO and senior consultant of Net Objectives. Since 1981, he has been both an OO consultant and developer of software in several industries. Alan is a frequent speaker at prestigious conferences around the world, including: SD Expo, Java One, OOP, OOPSLA. He is the primary author of Design Patterns Explained: A New Perspective on Object-Oriented Design and is currently co-authoring three other books in the software development area. He is a certified Scrum Master and has a Masters in Computer Science from M.I.T.

James Trott

James Trott is a knowledge management consultant, collaboration specialist, and knowledge engineer. He has Master of Science in Applied Mathematics, an MBA, and a Master of Arts in Intercultural Studies. He has twenty years experience as a pattern-based analyst, working in the petroleum, aerospace, software, and financial industries. He divides his consulting efforts between corporate work and humanitarian work with Millennium Relief and Development Services.





Volume 1, Issue 9
September 2004
©2004, Net Objectives,
All Rights Reserved



Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

All four of these items were important. The risk issues I considered were:

1. How much time would I need to become competent with TCP/IP?
2. If I changed the number of connections, would that affect the using code?
3. Would load balancing affect the using code?
4. Would error handling affect the using code?

There was not much I could do about issue 1. I had to enhance my TCP/IP skills quickly. Issues 2 through 4 however, could be mitigated by encapsulating them from the using code. In other words, my main business logic really just needed to deal with *one* TCP/IP connection. How it functioned, how many connections there were, and how it handled errors, were issues for the connection to connection code to handle. If I could isolate these issues from my business logic – if my business logic could ignore them – then I could change them later, *after* I had solved issue 1 without affecting the rest of the code. This told me I had to hide (encapsulate) them all.

This is a general approach for me: I try to look for ways to insulate myself from the impacts of changes to my system. I know I am going to have to change things as I figure out what to do because I rarely get it right the first time. And in this case, I had to be fast, which meant it was even more important to reduce debugging and rework.

With this approach in mind, and armed with a general understanding of patterns, I was ready to begin.

Don't rely on hope

I figured since I didn't have any hope of guessing what the right solution was, I had to make my system so that I could *change* the number of TCP/IP connections without affecting my code (except maybe in the smallest way). This meant that my client code (the code that used the TCP/IP connections) should not be required to change when the number of connections changed. To me, this implied that these client objects *could not even know* how many TCP/IP objects there were. Who would know? *Someone else!*

Create a TCP/IP Manager

That "someone else" would be the manager of my TCP/IP objects. The result was that I had two main functions going on here:

1. The TCP/IP objects that would communicate with the mainframe
2. The management of these objects

While I could have put all of this logic in the TCP/IP object itself, the principle of cohesion prescribed using two separate objects. I called the object that controlled one TCP/IP object a "Port". I did this because TCP/IP is simply one implementation of how a program communicates with the mainframe. While I was using TCP/IP at the moment (and probably always would), in my mind, the object was a port to the mainframe. I really thought of this connection as a port to the user's information. Naturally enough, I decided to name my port manager, "**PortManager**".

While there would be many ports, there would be only one **PortManager**. This one object would

*"It is better to know some of the questions than all of the answers".
– James Thurber*

handle all of my **Ports**. What is the pattern to use if you need to ensure that there is at most one object? The Singleton pattern. That is what I decided to use.

Now, Singleton ensured that I had only one **PortManager**; I also wanted this **PortManager** to be the only one who could create **Ports**¹. Encapsulating the management of **Ports** allowed me to be arbitrary in picking the number of **Ports** to use. I reached up and pulled the number "5" out of the air. That seemed to be as good a number as any to start with.

I implemented the **PortManager** as follows²:

Data Members

I gave it a private array of 5 **Port** references, using a constant to define this number. I instantiated these in the **PortManager**'s constructor. If any of them did not instantiate, I threw an exception because this was a bad start: not being able to get 5 connections did not bode well for keeping communications up!

Methods

I had a method that could be called to ask for a **Port**, called *getInstanceOfPort()*

I had another method that was called (with the **Port** reference) that said the port was no longer active: *returnInstanceOfPort(Port portToRelease)*

Implementing each of these was very simple:

getInstanceOfPort(): The first one simply looped through the **Port** array and found the first available port (each port had a status as to whether it was in use or not). If one was found, its status was updated to active and it was returned. If one wasn't found, I put the thread to sleep for one second and tried again.

returnInstanceOfPort(Port portToRelease): This one wasn't much harder. I simply looped through my ports until I found the reference given and updated its status to inactive. I threw an exception if I could not find it, because that should not happen.

The client code was very straightforward. When a message needed to be handled, the client would:

1. ask the **PortManager** for a port
2. use the port as needed
3. release the port

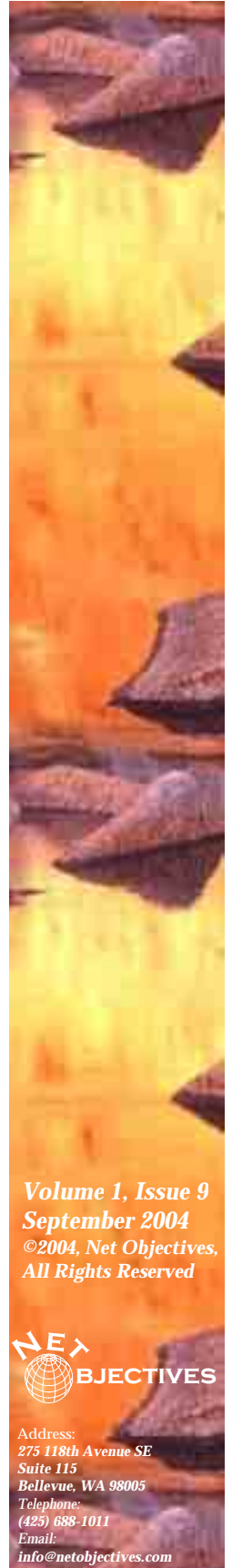
The client code did not have to concern itself with creating ports or how many of them would ever be used.

Patterns help with agile

The important thing to note here is that not only did this make the client code de-coupled from the **Port** code, it also meant I could defer much of my work until later in the project. I have already shown how I could ignore the port handling.

¹ Since I was using C++, I made the **Port** constructor protected and made the **PortManager** a friend of it. Had I been using C#, I could have used delegates to handle this restriction. See this book's companion web site at <http://www.netobjectives.com/dpexplained> to see how to do this. It is not obvious but is easily doable. In Java, I would have put the **PortManager** and the **Port** classes in a package.

² Note that while I used C++ on this project, the code I show here is in Java, to be consistent with the rest of the book.





Volume 1, Issue 9
September 2004
©2004, Net Objectives,
All Rights Reserved



Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

Even better, in the client code, I could also ignore error handling, which is one of the tougher problems with TCP/IP in C++. Why could I do this? Because I was confident that the **Port** and **PortManager** either could handle errors completely or handle them to such an extent that the client code would be virtually unaffected when I did implement it. I was free to focus on the more important aspects of the problem – the client code.

Focusing on what is most important is essential to agile development. In XP, they even have a slogan for it: YAGNI (Ya Ain't Gonna Need It). It reflects the idea that you should build what you need now while ignoring the rest. You should work on the most important things early, when solving them can make the greatest impact. It also means you avoid working on things are at a minimum distracting, and typically never used (and therefore building is a waste of resources).

The most important thing I needed to do was to get the communication between my middleware application and the mainframe going. Things like load-balancing and error handling, while extremely important, would have delayed me from building the system. By focusing on the communication first in a way that decoupled it from the rest of the system, I could get others working on responding to my needs. I could handle these other needs later.

What I had was the Object Pool pattern

Without knowing it at the time, I ended up with the Object Pool pattern. I had achieved this quality design simply by following the principles of design patterns. This is what the Object Pool pattern enables you to do. Although you could say that my derivation implies that the pattern isn't needed, I feel that knowing it would ensure its use when that would be applicable. Also, a less experienced designer may not have taken my approach. It takes less effort to learn from others than to break new ground yourself. Also, now that I know this pattern, I can easily communicate my chosen approach to others who know it, with a high degree of confidence that we will be talking about the same thing.

Handling the errors

After I got the communications working, I decided I had better turn my focus to handling errors. This meant different things to different parts of my code:

1. To the **Client**, it meant being able to get another **Port** to use and to resend the message.
2. To the **Port** that experienced the error, it meant disabling itself and not trying to use it anymore.
3. To the **PortManager**, it meant to remember the Port is bad and to get a new one to replace it.

Employee Spotlight: Rob Myers

Net Objectives is pleased to announce the addition of Rob Myers as a Consultant. Rob will present courses and seminars in the San Francisco Bay area.

Rob Myers has over fifteen years of professional experience in software development, including projects for industry leaders in medical, aerospace, and financial services. An eXtreme Programming coach since the late 90's, Rob has a passion for helping teams adopt the best practices of agile processes, object-oriented design, and software design patterns.



Check www.netobjectives.com/events/pr_main.htm#FreeSeminars
for Public Seminars in
Western Washington State, Midwest, and Northern California

Seminars We Can Give

Transitioning to Agile – More and more companies are beginning to see the need for Agile Development. In this seminar, we discuss what problems agility will present and how to deal with these.

Test-First Techniques Using xUnit and Mock Objects – This seminar explains the basics of unit testing, how to use unit tests to drive coding forward (test-first), and how to resolve some of the dependencies that make unit testing difficult.

Pattern Oriented Development: Design Patterns From Analysis To Implementation – This seminar discusses how design patterns can be used to improve the entire software development process - not just the design aspect of it.

Agile Planning Game – The Planning Game was created by Kent Beck and is well described in his excellent book: Extreme Programming Explained. Unfortunately, the Planning Game as described is not complete enough - even for pure, XP teams. This seminar describes the other factors which must often typically be handled.

Comparing RUP, XP, and Scrum: Mixing a Process Cocktail for Your Team - This seminar discusses how combining the best of some popular processes can provide a successful software development environment for your project.

Design Patterns and Extreme Programming – Patterns are often thought of as an up-front design approach. That is not accurate. This seminar illustrates how the principles and strategies learned from patterns can actually facilitate agile development. This talk walks through a past project of the presenter.

Introduction to Use Cases – In this seminar we present different facets of the lowly Use Case; what they are, why they're important, how to write one, and how to support agile development with them.

Unit Testing For Emergent Design – This seminar illustrates why design patterns and refactoring are actually two sides of the same coin.

Check www.netobjectives.com/events/pr_main.htm#UpcomingPublicCourses
For a complete schedule of upcoming Public Courses
in Western Washington State, Midwest, and Northern California
and information on how to register

Net Objectives Consulting Division

The consultants at Objective Result are industry and technology experts who have been carefully selected and trained by Net Objectives educators. Objective Result provides Project Management, Development, and Quality Assurance consulting services.

The driving force behind Objective Result is a firm belief in the natural synergy that exists between ongoing well trained consultants following the techniques taught through the Net Objectives courses.

Contact Peter Stroeve at Peter.Stroeve@netobjectives.com
if you want to join Net Objectives Consulting

Volume 1, Issue 9
September 2004
©2004, Net Objectives,
All Rights Reserved



Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com



Volume 1, Issue 9
September 2004
©2004, Net Objectives,
All Rights Reserved



Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

Personal Book Recommendation from Scott Bain:
Guns, Germs, and Steel: The Fates of Human Societies by Jared Diamond (ISBN 0393317552). This book examines the largely unexplored concept of causality in history. It changes the way you look at the progress of human events.

The de-coupling I had already achieved made this very straightforward as well. Here is what I had to do:

1. Throw an exception when an error occurred on the **Port**.
2. Catch the exception in the **Client** at the point the transmission of the messages began. It was relatively easy to see if anything needed to be backed out (typically not). After this, just return the **Port** to the **PortManager** indicating it was bad, and ask for another one.
3. Add a new method to the **PortManager** called: `returnBadPort (Port badPort)`. This would identify the **Port** as bad (in much the *same* way that `returnInstanceOfPort` worked). It would then remove that **Port** from the array and insert a new **Port** where it was. If I couldn't get a new **Port** (which is highly unlikely), I would send a message via e-mail to a system administrator – this was the preferred method to use if things looked like they were going to crash and burn. I would then service any other requests with my remaining **Ports** (if I had any).

Going deeper

This should have been enough. But it wasn't. As the system moved forward and it looked like it would actually go into production, I started having difficulty sleeping. Something was nagging me and would not go away. Another design principle I use is to pay attention to your instincts: you often know more than you think you do!

What was bothering me was that hundreds of millions of dollars were going to go through *my* pipeline, unsupervised. While I had written a lot of software systems before, some of which were very mission-critical, the pressure here seemed greater. If something went wrong on this system at midnight, it would not be acceptable to find out about this the next morning and restart the system. Hundreds of millions of dollars in trades might not get executed. This was not a pleasant thought! Some sort of supervision was needed.

I knew I had to solve this problem. In fact, I felt I had to solve it twice. I had to make sure I had a robust system and then I had to be sure it stayed robust. I remembered reading something in Steve Maguire's book, Writing Solid Code³, about having dual checking methods for mission critical parts of the code. It made me think that if I had a separate, independent mechanism of checking the communications, I wouldn't have to worry.

³ Maguire, S. *Writing Solid Code*, Redmond, WA, Microsoft Press, 1993, Redmond, Washington, p.33.

Technical Book Recommendation:
Effective Java Programming Language Guide by Joshua Bloch (ISBN: 0201310058)
This book is a series of set-pieces, each a short chapter describing a best practice in Object-Oriented programming. Despite the title, it is a great reference for any developer who seeks to improve the practice of OO in any language.

Dual checking

The answer was relatively easy. The **PortManager** was already responsible for creating the **Ports**. I just gave it the additional responsibility for verifying the integrity of those **Ports**. To do this, I chose to have it start a thread that ran a check every 15 minutes; this check would do the following:

- See how many outstanding requests there were for **Ports** (and do something if the number were exceedingly large).
- Query the **Ports** about their status (active or not) and contrast this with the number of requests outstanding to see if something was wrong.
- See how many **Ports** were in an error condition (should be at most 1 or 2).

Basically, I could put anything I wanted to in this code, expanding these ideas if needed. It would provide a higher perspective, one that I could refine as I learned more about the failures possible. I could do this without affecting the rest of the code in any significant way.

Thanks to dual checking, I started sleeping again!

The Object Pool Pattern

Although I used the Object Pool pattern to illustrate how factories can be used and how patterns assist agility, the Object Pool pattern is a useful pattern in and of itself. Typically this pattern is useful whenever there is a shared resource and a single point of contact to that resource would be beneficial. This point of contact (the pool) can also perform other responsibilities (such as error handling). By encapsulating these responsibilities, the client code using these objects is both freed up from concerning itself from them and isolated from any changes to them.

Observation: Factories Can Do Much More than Instantiation

Instantiation, management, error handling

I started out my discourse on factories to illustrate how I can separate the issues of use from construction. I have now illustrated how you can separate the issues of use from construction and from object management. I even extended this to include error handling since bad **Ports** meant I needed new ones.

What's Happening at Net Objectives This Month

We are eager to announce the first installment of our monthly streamzines, Encapsulating Construction:

<http://www.netobjectives.com/streamzines/CurrentStreamzine>

In this presentation, Scott Bain discusses a design technique which will make it easier to evolve a design when requirements change.

"Streamzines" are recorded multimedia presentations, 15 – 60 minutes long, in which one of our consultants will explore a particular software issue in depth. This offers a taste of one of our seminars or courses, but here you can interrupt, rewind, and pause the instructor at will.

If you are not currently receiving links to monthly Ezines and Streamzines, as well as to upcoming public courses and seminars, become a member of our mailing list.

Fill out and submit our online form at: <http://www.netobjectives.com/subscribe.htm>

*Volume 1, Issue 9
September 2004
©2004, Net Objectives,
All Rights Reserved*



Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

The point is, once you start thinking about objects as things with responsibilities, it becomes clearer how to separate these responsibilities. Once the responsibilities have been identified, it is often easy to see the correct structure for the system to have. This results in code that is clearer, more robust, more manageable, and easier to change. And you can sleep at night.

The Object Pool Pattern: Key Features	
Intent	Manage the reuse of objects when it is either expensive to create an object or there is a limit on the number objects of a particular type that can be created.
Problem	The creation and/or management of objects must follow a well-defined set of rules. Typically these rules relate to how to make an object, how many objects can be created and how to reuse existing objects once they have finished their current tasks.
Solution	The Client calls ReusablePool 's <i>acquireReusable</i> method when it needs a Reusable object. If the pool is empty, then the <i>acquireReusable</i> method creates a Reusable object if it can; otherwise, it waits until a Reusable object is returned to the collection.
Participants and Collaborators	The ReusablePool manages the availability of Reusable objects for use by the Client . Client then uses instances of Reusable objects for a limited amount of time. ReusablePool contains all of the Reusable objects so that they can be managed in a unified way.
Consequences	Works best when the demand for objects is fairly consistent over time; large variations in demand can lead to performance problems. To address this issue in the Object Pool pattern, limit the number of objects that can be created. Keeping the logic to manage the creation of instances separate from the class whose instances are being managed results in a more cohesive design.
Implementation	If there is a limit on the number of objects that may be created or if there is a limit on the size of the pool, then use a simple array to implement the pool. Otherwise, use a vector object. The object responsible for managing the object pool must be the only object able to create those objects. The ReusablePool is implemented with a Singleton pattern. Another variant is to put a release method on the Reusable object – and let it return itself to the pool
Reference	This pattern is not in the Gang of Four book. It is described in Mark Grand's book, <i>Patterns in Java, Volume 1</i> ⁴ , pages 135-142. Clifton Nock's book, <i>Data Access Patterns</i> ⁵ , covers this in a good amount of detail in the context of database resources; he refers to it as "Resource Pool".

Volume 1, Issue 9
 September 2004
 ©2004, Net Objectives,
 All Rights Reserved

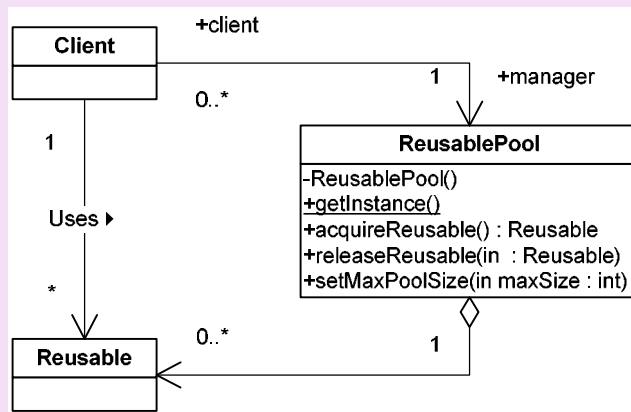


Address:
 275 118th Avenue SE
 Suite 115
 Bellevue, WA 98005
 Telephone:
 (425) 688-1011
 Email:
 info@netobjectives.com

⁴ Grand, Mark. *Patterns in Java—Volume 1, A Catalog of Reusable Design Patterns Illustrated with UML*. New York, NY: John Wiley & Sons, Inc., 1998, p. 135-142.

⁵ Nock, Clifton. *Data Access Patterns*. Boston, Mass: Addison-Wesley, 2003.

Figure 22-2 Generic structure of the Object Pool.



Summary

Creation, management, error handling

I illustrated in this chapter how factories can be used for much more than just creating and managing the reuse of objects. The Object Pool pattern is useful for two major reasons:

1. It shows how a factory can be used to both instantiate and manage objects.
2. It illustrates how encapsulation of responsibility assists developers in focusing on what they most need to.

Review Questions

Observations

1. What are three general strategies to follow when designing?

2. What two patterns does the Object Pool pattern incorporate?
3. What is the intent of the Object Pool pattern?

Interpretations

1. What does the XP community mean by YAGNI?

Opinions and Applications

1. Reading widely is an important discipline. You never know when you will find something you can use, such as the example from Steve Maguire's book, *Writing Solid Code*. Give at least one example from your own experienced where this has been true for you.

To join a discussion about this article, please go to
<http://www.netobjectivesgroups.com/6/ubb.x>
 and look under the E-zines category.

Volume 1, Issue 9
 September 2004
 ©2004, Net Objectives,
 All Rights Reserved



Address:
 275 118th Avenue SE
 Suite 115
 Bellevue, WA 98005
 Telephone:
 (425) 688-1011
 Email:
info@netobjectives.com

Go to <http://www.netobjectives.com/subscribe.htm> to subscribe to our mailing list and receive future ezines and seminar announcements

Volume 1, Issue 9
September 2004
©2004, Net Objectives,
All Rights Reserved



Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

Net Objectives - What We Do

Net Objectives provides enterprises with a full selection of training, coaching and consulting services. Our Vision is "Effective software development without suffering". We facilitate software development organizations migration to more effective and efficient processes. In particular, we are specialists in agility, effective analysis, design patterns, refactoring and test-driven development.

We provide a blend of training, follow up coaching and staff supplementation that enables your team to become more effective in all areas of software development. Our engagements often begin with an assessment of where you are and detail a plan of how to become much more effective. Our trainers and consultants are experts in their fields (many of them published authors).

When you've taken a course from Net Objectives, you will see the world of software development with new clarity and new insight. Our graduates often tell us they are amazed at how we can simplify confusing and complicated subjects, and make them seem so understandable, and applicable for everyday use. Many of our students remark that it is the best training they have ever received.

The following courses are among our most often requested. This is not a complete list, though it is representative of the types of courses we offer

Agile Project Management - This 2-day course analyzes what it means to be an agile project, and provides a number of best practices that provide and/or enhance agility. Different agile practices from different processes (including RUP, XP and Scrum) are discussed.

Agile Use Case Analysis - This 3-day course provides theory and practice in deriving software requirements from use cases. This course is our recommendation for almost all organizations, and is intended for audiences that mix both business and software analysts.

Design Patterns Explained: A New Perspective on Object-Oriented Design - This 3-day course teaches several useful design patterns as a way to explain the principles and strategies that make design patterns effective. It also takes the lessons from design patterns and expands them into both a new way of doing analysis and a general way of building application architectures.

Test-Driven Development: Iterative Development, Refactoring and Unit Testing - This 3-day course teaches how to use either Junit, NUnit or CppUnit to create unit tests. Lab work is done in both Refactoring and Unit Testing together to develop very solid code by writing tests first. The course explains how the combination of Unit Testing Refactoring can result in emerging designs of high quality.

Effective Object-Oriented Analysis, Design and Programming In C++, C#, Java, or VB.net - This 5-day course goes beyond the basics of object-oriented design and presents 14 practices which will enable your developers to do object-oriented programming effectively. These practices are the culmination of the best coding practices of eXtreme Programming and of Design Patterns.

Software Dev Using an Agile (RUP, XP, SCRUM) Approach and Design Patterns - This 5-day course teaches several design patterns and the principles underneath them, the course goes further by showing how patterns can work together with agile development strategies to create robust, flexible, maintainable designs

If you are interested in any of these offerings, if your user group or company is interested in Net Objectives making a free technical presentation to them, or if you would like to be notified of upcoming Net Objectives events, please visit our website, or contact us by the email address or phone number below:

www.netobjectives.com • mike.shalloway@netobjectives.com • 404-593-8375

Business-Driven Software Development (BDS) is Net Objectives' proprietary integration of Lean-Thinking with Agile methods across the business, management and development teams to maximize the value delivered from a software development organization. BDS has built a reputation and track record of delivering higher quality products faster and with lower cost than other methods

BDS goes beyond the first generation of Agile methods such as Scrum and XP by viewing the entire value stream of development. Lean-Thinking enables product portfolio management, release planning and critical metrics to create a top-down vision while still promoting a bottom-up implementation.

BDS integrates business, management and teams. Popular Agile methods, such as Scrum, tend to isolate teams from the business side and seem to have forgotten management's role altogether. These are critical aspects of all successful organizations. In BDS:

- **Business** provides the vision and direction; properly selecting, sizing and prioritizing those products and enhancements that will maximize your investment
- **Teams** self-organize and do the work; consistently delivering value quickly while reducing the risk of developing what is not needed
- **Management** bridges the two; providing the right environment for successful development by creating an organizational structure that removes impediments to the production of value. This increases productivity, lowers cost and improves quality

Become a Lean-Agile Enterprise

All levels of your organization will experience impacts and require change management. We help prepare executive, mid-management and the front-line with the competencies required to successfully change the culture to a Lean-Agile enterprise.

Prioritization is only half the problem. Learn how to both prioritize and size your initiatives to enable your teams to implement them quickly.

Learn to come from business need not just system capability. There is a disconnect between the business side and development side in many organizations. Learn how BDS can bridge this gap by providing the practices for managing the flow of work.

Why Net Objectives

While many organizations are having success with Agile methods, many more are not. Much of this is due to organizations either starting in the wrong place (e.g., the team when that is not the main problem) or using the wrong method (e.g., Scrum, just because it is popular). Net Objectives is experienced in all of the Agile team methods (Scrum, XP, Kanban, Scrumban) and integrates business, management and teams. This lets us help you select the right method for you.

<p>Assessments</p> <p>See where you are, where you want to go, and how to get there.</p> <p>Business and Management Training</p> <p>Lean Software Development Product Portfolio Management Enterprise Release Planning</p>	<p>Productive Lean-Agile Team Training</p> <p>Team training in Kanban, Scrum Technical Training in ATDD, TDD, Design Patterns</p> <p>Roles Training</p> <p>Lean-Agile Project Manager Product Owner</p>
--	---

