Shepherding workshop at EuroPLoP 2008

Some notes that might be useful for you both during the workshop and afterwards when you are actually shepherding.

Day 1 - Welcome of participants

Who are you: Hopefully many people, from beginners to experienced shepherds

Who am I: I've taken part in EuroPLoP conferences since 2003 and started to become a shepherd three years ago. Last year, I was honored to receive the Neil Harrison Shepherding Award. On my homepage, you can find my collected work: http://www.tim-wellhausen.de (there is a link to a page for English speaking visitors).

What can you expect from this workshop?

Expect to leave this workshop with a good gut feeling of what shepherding is about and with the wish to become a shepherd if you are not one already.

I'll try not to give a talk. Instead I'd prefer a more interactive workshop style. Therefore I prepared some exercises and am willing to be responsive for whatever you suggest.

I took and re-used material from Neil's workshop in 2002 and Allan's workshop in 2007 (see resources). I also incorporated many insights from Neil's Pattern Language of Shepherding (also see resources).

Do we have a common understanding of what shepherding is like?

What was your experience in having been a shepherd or having had a shepherd: What was it like? What was important to you? What worked well? What was difficult?

What information do not-yet-shepherds need to actually become a shepherd?

Exercise: Please make a few notes and let us then collect our thoughts on a blackboard. Let's see which points are most important to us.

Some things that are important to me as a shepherd:

Most obviously: Shepherding helps a lot to improve the quality of submitted papers.

Being a shepherd helps to better understand the nature of patterns, to more deeply think about them, and, as consequence, to write better patterns.

It's an important aspect of our community that's rather unique: helping each other to improve our work. There is no paper that could not be improved by shepherding!

While shepherding, you also learn from other people's work.

It's a fun way to give back to the community.

How do you become a shepherd?

Get on the shepherding mailing list to be notified when a PLoP conference chair is looking for shepherds. Ask a shepherd you know to be put on that list.

When you receive a call for shepherds, go to the web page, browse the submitted papers, choose a couple of potential papers you would like to shepherd, and send an email to the chairs.

The program chair assigns both a paper and a program committee (PC) member to you. The PC member needs to be involved in all communication between shepherd and sheep. He or she is there to help both you and the sheep if there is any need.

How does the shepherding process start?

It's your turn to contact the author(s).

Exercise: Suppose you've become shepherd of the paper “Beer after Midnight”. What would you write as your first email to the authors? Could you please write down a short text and read it afterwards?

Here are some points that are important to me:

Introduce myself and what I'm doing.

Explain why I chose the paper and why I have a personal interest in it.

Tell them what to expect from the shepherding process (how much time, how many iterations, what kind of feedback, ...)

Anything else: How much spare time I have, when I'm going to send the first set of comments, who is the PC member, ...

See the attached example email.

Some more things you should be aware of:

Time is always short: try to get going quickly and set explicit deadlines – both for you and for the sheep.

Some authors are more responsive than others: In the end, it is also your decision whether the paper is ready for the conference. The responsiveness of an author to shepherding is an important aspect in accepting a paper.

Shepherding patterns: THREE ITERATIONS, THE SHEPHERD KNOWS THE SHEEP (see “The Language of Shepherding”).

How do you write your first set of comments?

First of all: read the paper in depth and try to get a good understanding of what the paper is all about.

Concentrate on the paper as a whole. If you have trouble understanding the patterns, try to find out what causes the trouble and point it out to the authors.

Concentrate on the content of the paper. Leave remarks on the form for the end.

Rather give small pieces of advice that the authors can incorporate quickly than a full load of detailed feedback.

Adjust your style of feedback: Who are the authors?

> If they are experienced authors, you may quickly and directly get to the main points.
>
> If they are beginners, you should explain your suggestions in more depth.
>
> If they are academics, you might have to clarify an important issue: Academic values *new* knowledge, pattern writers value *existing* knowledge!

Shepherding patterns: BIG PICTURE, HALF A LOAF

What should you concentrate on in your first set of comments?

Try to get the problem / solution statements right:

> Authors tend to describe a solution they know of but forget to think hard of the problem they are actually solving.
>
> Does the solution really convince you? If not, express your concerns, ask questions that the authors may not have thought about earlier. Make them convince you in the next iteration!

What might be wrong with problem statements?

> They are too broad.
>
> They just rephrase the solution.
>
> They don't match the solution.
>
> They are not well-defined.
>
> There is no problem!

Every pattern tells a story:

> Patterns should build up and resolve tension: The context defines the settings of the story. The problem leaves you puzzled as to how it might be solved. The forces add to the tension until you finally get to know the solution (the Aha moment) but then the consequences make you aware that there is more to come.
>
> If there is not enough tension, make the author aware of it – thinking of tension might reveal more forces or consequences.

Exercise: Read the version of the SINGLETON pattern that you can find attached to the notes. Form groups of two and discuss what kind of feedback you would give. Tell all of us afterwards.

Shepherding patterns: MATCHING PROBLEM & SOLUTION, CONVINCING SOLUTION , WAR STORIES

Day 2 - How do you give feedback?

Exercise: How do you give feedback in general? How do you cope with someone who may not be open to feedback? Please tell us your experience in giving feedback!

Again some points that are important to me:

> The same applies as in every pattern workshop: Tell the authors what you like and make suggestions for improvement.
>
> I prefer asking questions over giving detailed instructions. Sometimes I just have questions and no suggestions...
>
> It's the authors' duty to improve their paper. I only try to point out what could be improved.

The first iteration is over. What's next?

It highly depends on the state of the paper, the responsiveness of the authors, and the amount of time you and the authors are able and willing to spend.

Consider the following when giving more feedback:

> Constrain the pattern. Some patterns have a context that is too broad. It might help to boil down the context.
>
> Let the authors cut down a pattern if it has grown too much.
>
> Help the authors to find the problem through the forces. Forces crystallize the problem.
>
> Do the consequences match the forces? Are there any negative consequences at all? Are important consequences missing?
>
> Check the details of the solution, in particular if there is an implementation section. Will the solution work? Is it speculative?
>
> Are there examples, knows uses?
>
> Are there related patterns? Is there existing work that needs to be referenced?
>
> Is there an explicit target audience?
>
> Has the paper been written for a reader? Is is easy and a pleasure to read?
>
> Is the pattern form appropriate (though advising a different form is difficult)?

Exercise: If time is permitting or if some participants would like to continue exercising after the workshop, read the NULL PATTERN at the end of the notes and try to give feedback on it.

Shepherding patterns: BALANCED CONTEXT, SMALL PATTERNS, FORCES DEFINE PROBLEM

When is the shepherding over?

Shepherding is over as soon as the time is over. Typically, there is a hard deadline when the paper needs to be done for the acceptance decision.

You are asked by the program committee on your opinion how successful the shepherding was: did the authors actually improve their work, did they at least try to?

After a paper has been accepted, you are free to continue working with the author. If you feel like, you can even give more feedback after the conference when the authors have incorporated the feedback from their pattern workshop, though this is rarely done.

What is the relationship between you as a shepherd and the sheep?

The authors own the paper. They are the domain experts, they know what they want to tell. They don't need to follow your advice.

It's your responsibility to help the authors to write their paper, not to take over the writing.

Show the authors that you value what they write. They might be your next shepherds...

But the authors should also acknowledge the work of the shepherd. He or she does this in spare time.

Shepherding patterns: AUTHOR AS OWNER

What else is to say?

"We have a responsibility to give back to the community. Shepherding is one important way." (Neil)

Resources:

Language of Shepherding by Neil: http://hillside.net/language-of-shepherding.pdf

Shepherding workshop by Neil (2002): http://hillside.net/patterns/EuroPLoP2002/ShepheringWorkshop.ppt

Shepherding workshop by Allan (2007): Not available online

An example introductory email to the sheep (PLoP 2006)

“Dear authors,

My name is Tim Wellhausen and I'm your shepherd for the paper that you've submitted to the PLoP conference. I live in Munich, Germany, and work as a freelancer. As a pattern author, I've submitted several papers for EuroPLoP conferences that I'm also going to attend in July.

I picked your paper because of two reasons. First, I'm interested in user interface related patterns in general. Second, I've been engaged in integrating a docking framework earlier this year.

As a shepherd, I'd like to help you to improve your paper. Regarding the early deadline (end of July) we have time for two or maybe three iterations. During each iteration, I give you feedback on your current version and include suggestions for improvement. You should then try to revise your paper - with or without applying any of my suggestions. That's up to you... :-)

If you have any questions about the shepherding process pleask ask - I don't know whether you are already experienced pattern authors.

During shepherding, I will collaborate with a member of the conference program committee, Mary Lynn Manns. As member of the program committee, she will decide on the acceptance of your paper. Please cc her on every mail.

I'm going to send you another mail with my first set of comments soon.

Best regards,

Tim”

BEER AFTER MIDNIGHT **

Also known as: "Where the f*** is the beer?"

You find yourself in the bar of Kloster Irsee with a couple of thirsty guys around the table. You have already been drinking for a while. Everyone is in a good mood and degree of talking non-sense stories increases steadily. The clock says 00:28, the bar stops serving at 00:30.

◊◊◊

How to have as much beer as possible after the bar stops serving?

Guys don't sit in a bar without having a beer.

A guy that has no more beer tends to grab the nearest, unattended full glass of beer.

Each guy has limited transportation ability. The maximum transportation quantity is five glasses per guy.

The waitress is likely to be suspicious if a guest asks for more than one beer. The level of suspicion increases with every additional beer you are ordering at the same time.

The table you are sitting has an area of around 2 square meters, a lot of white space for creative use.

Beer tastes best when drunk freshly served, right from the cold barrel. The beer's temperature increases gradually with the amount of time that passes once the beer is placed on the table.

The amount of beer to be drunk by each guy is non-deterministic.

Therefore, just before half past twelve every drinking fellow should order as much beer as he is able to carry as long as there is still white space on the table.

This pattern has the following consequences:

+ The table is completely filled with full glasses of beer.
+ As soon as a guy finishes his beer he takes a full one from the table.
+ The guys are sitting in the bar for some more hours.
+ The group cohesion increases significantly.
+ The group adds more patterns to the BEING AT THE BAR pattern language.
- The quality of the beer on the table is lower than from the bar.
- The waitress might be quite uncooperative the next evening.
- The morning after "the session" the guys' ability to participate in the workshop is reduced.

◊◊◊

Known Users: Aaldert, Ewald, Markus, Stefan, Titos, Wolfram, Frank.

Related Patterns: Users of the LAST ORDERS pattern in England and Wales usually make use of a variation of the BEER AFTER MIDNIGHT pattern. Most roles within the pattern also participate in a DISASTER RECOVERY. Sometimes a user of BEER AFTER MIDNIGHT becomes a FLY ON THE FLOOR.

Acknowledgements: Special thanks go to Kevlin for the on-the-fly shepherding of the drunken sheep. The authors would like to thank the waitresses of Kloster Irsee for being patient with us.

Written during EuroPLoP 2003 by Kevlin Henney

The Singleton Pattern

Taken from http://userpages.umbc.edu/~tarr/dp/lectures/Singleton-2pp.pdf

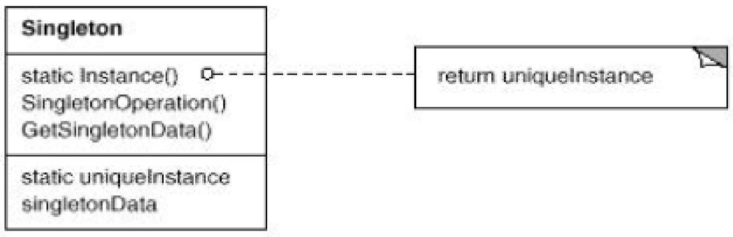
Intent

- Ensure a class only has one instance, and provide a global point of access to it

Motivation

- Sometimes we want just a single instance of a class to exist in the system
- For example, we want just one window manager. Or just one factory for a family of products.
- We need to have that one instance easily accessible
- And we want to ensure that additional instances of the class can not be created

Structure



Consequences

Benefits

- Controlled access to sole instance
- Permits a variable number of instances

Implementation

[Many implementation details omitted]

Null Object Pattern

Taken from http://www.cs.oberlin.edu/~jwalker/nullObjPattern/

Intent

Provide an object as a surrogate for the lack of an object of a given type. The Null Object provides intelligent do nothing behavior, hiding the details from its collaborators.

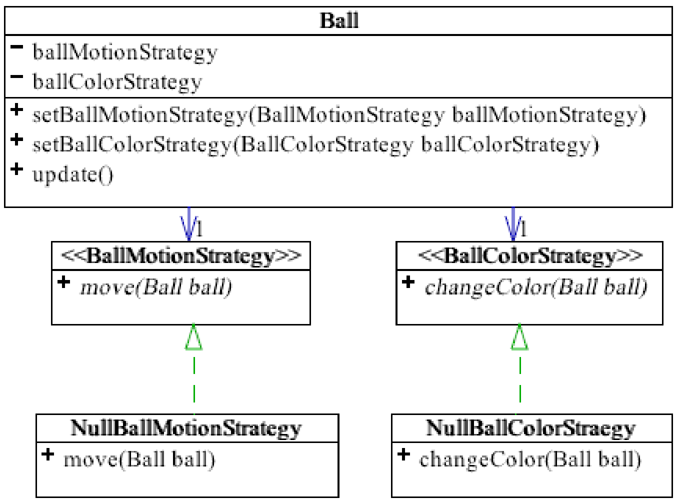
Also Known as

Stub, Active Nothing

Motivation

Sometimes a class that requires a collaborator does not need the collaborator to do anything. However, the class wishes to treat a collaborator that does nothing the same way it treats one that actually provides behavior.

Consider for example a simple screen saver which displays balls that move about the screen and have special color effects. This is easily achieved by creating a Ball class to represent the balls and using a Strategy pattern [GHJV95, page 315] to control the ball's motion and another Strategy pattern to control the ball's color. It would then be trivial to write strategies for many different types of motion and color effects and create balls with any combination of those. However, to start with you want to create the simplest strategies possible to make sure everything is working. And these strategies could also be useful later since you want as strategies as possible strategies.



Now, the simplest strategy would be no strategy. That is do nothing, don't move and don't change color. However, the Strategy pattern requires the ball to have objects which implement the strategy interfaces. This is where the Null Object pattern becomes useful. Simply implement a NullMovementStrategy which doesn't move the ball and a NullColorStrategy which doesn't change the ball's color. Both of these can probably be implemented with essentially no code. All the methods in these classes do "nothing". They are perfect examples of the Null Object Pattern.

The key to the Null Object pattern is an abstract class that defines the interface for all objects of this type. The Null Object is implemented as a subclass of this abstract class. Because it conforms to the abstract class' interface, it can be used any place this type of object is needed. As compared to using a special "null" value which doesn't actually implement the abstract interface and which must constantly be checked for with special code in any object which uses the abstract interface.

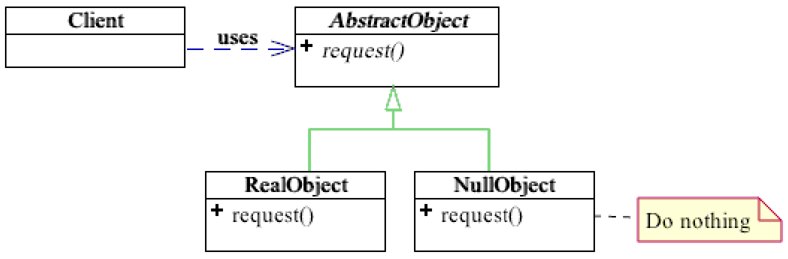
It is sometimes thought that Null Objects are over simple and "stupid" but in truth a Null Object always knows exactly what needs to be done without interacting with any other objects. So in truth it is very "smart."

Applicability

Use the Null Object pattern when:

- an object requires a collaborator. The Null Object pattern does not introduce this collaboration--it makes use of a collaboration that already exists.
- some collaborator instances should do nothing.
- you want to abstract the handling of null away from the client.

Structure



Participants

- Client
    - requires a collaborator.

- AbstractObject
    - declares the interface for Client's collaborator.
    - implements default behavior for the interface common to all classes, as appropriate.

- RealObject
    - defines a concrete subclass of AbstractObject whose instances provide useful behavior that Client expects.

- NullObject
    - provides an interface identical to AbstractObject's so that a null object can be substituted for a real object.
    - implements its interface to do nothing. What exactly it means to do nothing depends on what sort of behavior Client is expecting.
    - when there is more than one way to do nothing, more than one NullObject class may be required.

Collaborations

- Clients use the AbstractObject class interface to interact with their collaborators. If the receiver is a RealObject, then the request is handled to provide real behavior. If the receiver is a NullObject, the request is handled by doing nothing or at least providing a null result.

Consequences

The Null Object pattern:

- defines class hierarchies consisting of real objects and null objects. Null objects can be used in place of real objects when the object is expected to do nothing. Whenever client code expects a real object, it can also take a null object.
- makes client code simple. Clients can treat real collaborators and null collaborators uniformly. Clients normally don't know (and shouldn't care) whether they're dealing with a real or a null collaborator. This simplifies client code, because it avoids having to write testing code which handles the null collaborator specially.
- encapsulates the do nothing code into the null object. The do nothing code is easy to find. Its variation with the AbstractObject and RealObject classes is readily apparent. It can be efficiently coded to do nothing. It does not require variables that contain null values because those values can be hard-coded as constants or the do nothing code can avoid using those values altogether.
- makes the do nothing behavior difficult to distribute or mix into the real behavior of several collaborating objects. The same do nothing behavior cannot easily be added to several classes unless those classes all delegate the behavior to a class which can be a null object class.
- can necessitate creating a new NullObject class for every new AbstractObject class.

- can be difficult to implement if various clients do not agree on how the null object should do nothing as when your AbstractObject interface is not well defined.
- always acts as a do nothing object. The Null Object does not transform into a Real Object.

Implementation

[Omitted]

Sample Code (Java)

[Omitted]

Known Uses

See [Woolf96]

Related Patterns (shortened)

Singleton [GHJV95, page 127] is often used to implement a Null Object since multiple instances would act exactly the same and have no internal state that could change.

Strategy [GHJV95, page 315] patterns often have one Null Object representing the strategy of doing nothing.

State [GHJV95, page 305] often uses Null Object to represent the state in which the client should do nothing.

Iterators [GHJV95, page 257] may have Null Object as a special case which doesn't iterate over anything.

Adapters [GHJV95, page 142] may have Null Object as a special case which pretend to adapt another object without actually adapting anything.

Bruce Anderson has also written about the Null Object pattern, which he also refers to as "Active Nothing." [Anderson95]

NullObject is a special case of the Exceptional Value pattern in The CHECKS Pattern Language [Cunningham95]. An Exceptional Value is a special Whole Value (another pattern) used to represent exceptional circumstances. It will either absorb all messages or produce Meaningless Behavior (another pattern). A NullObject is one such Exceptional Value.

References (shortened)

[Anderson95] Anderson, Bruce. "Null Object." UIUC patterns discussion mailing list (patterns@cs.uiuc.edu), January 1995.

[Woolf96] Woolf, Bobby. "The Null Object Pattern"