HW4 - Answers

Answer 1:

As per the definition of Proxy pattern, it is meant to provide a surrogate or a placeholder for another object to control access to it.

A Remote Proxy is used to control access to a remote object. So basically it acts as a local representative for an object that resides on a different JVM. When a method call is made on the proxy, this is then transmitted through the proxy to the remote object and the results returned are displayed to the local client. The client does not even need to know the implementation details of the actual remote method and needs to be concerned only with the proxy object.

A Virtual Proxy, on the other hand, acts a representative for an object that may be expensive to create. It defers the creation of the object until it is needed thereby serving as a surrogate for the object before and while the object is created. In this case also, the client does not need to be aware of how the proxy creates the objects but only with how to invoke the proxy.

This goes to show how the proxy pattern manifests itself in many forms while still following the general proxy design. So even though the purposes served by the Remote Proxy and the Virtual Proxy are so different, they still adhere to the proxy design and thereby are really one pattern.

Answer 2:

When we think of patterns that “wrap” classes, one of the first patterns that comes to mind is the Decorator Pattern. The Decorator pattern serves to decorate an object and add behavior to the class. The other pattern that “wraps” classes is the Proxy pattern, though it serves a completely different purpose. It serves to control access to the objects by decoupling the client.

So an example of two patterns which are structurally similar but having different intents would be a Decorator pattern and a Proxy pattern

Answer 3:

This is not necessarily true. A design pattern is meant to be used when there is sufficient evidence to prove that it makes sense. Patterns are general solutions to recurring problems. But most of the problems can be easily solved just by good OO design and not necessarily by the use of design patterns.

We can’t expect to take a pattern and plug it in and expect a good result. Design patterns come into picture only after we are sure that it addresses a problem with our design. Even if a pattern is good match, we need to make sure it has a set of consequences we are willing to bear and understand its effects on our design.

In short, a Design patterns is not meant to be used as a consummate solution to every problem but rather only when using it makes sense to achieve the design goals.

Answer 4:

java.util.Arrays#asList()

Pattern Used: Adapter Pattern

As per the Javadoc definition, the asList() method of the java.util.Array class returns a fixed-size list backed by the specified array. It is able to provide a convenient way to create a fixed-size list initialized to contain several elements.

The asList() method basically takes an input in one format and converts it into another format that is desired. This behavior is reminiscent of how an Adapter Pattern works.

java.awt.Container#add(Component)

Pattern Used: Composite Pattern

The add(Component) method of the java.awt.Container class serves the purpose of adding the specified component to the end of a container, as per the Java docs. A Composite pattern allows us to compose objects into tree structures to represent the part whole hierarchies. Since adding the specified component is similar to adding objects to a tree, this is an example of the Composite pattern

java.lang.Integer#valueOf(int)

Pattern Used: Flyweight

The valueOf(int) method returns an Integer instance of the specified int value. As per the javadoc, this method is preferred compared to using the constructor Integer(int) because it can yield significantly better space and time performance by caching frequently.

The whole idea behind a Flyweight pattern is to reduce the number of object instances at runtime, saving memory and caching. Since java.lang.Integer#valueOf(int) serves to perform a similar operation, this is an example of the Flyweight pattern.

java.lang.Runnable

Pattern Used: Command

As per the Javadoc, the Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. It also provides a common protocol for objects that wish to execute code while they are active. A Command pattern is used to encapsulate a request as an object allowing to parameterize other objects with different requests etc., Since the Runnable interface serves to provide a common protocol for objects, it is an example of the Command pattern.

java.util.Pattern

Pattern Used: Interpreter

The javax.util.pattern class provides a compiled representation of a regular expression. The only pattern which performs such a similar functionality is the Interpreter Pattern. It provides a class-based representation for its grammar along with the interpreter to interpret its sentences. The javax.util.pattern class also provides a similar functionality and thereby is an example of the Interpreter pattern.

java.util.TreeSet#descendingIterator()

Pattern Used: Iterator Pattern

The javadoc for descendingIterator() method indicates that it is used to return an iterator over the elements in a descending order. The iterator pattern basically provides us a way to access the elements of an aggregate object sequentially. Since we use the descendingIterator to access elements in a descending order, it is an example of the Iterator pattern.

javax.servlet.http.HttpServlet

Pattern Used: Template Method Pattern

The HttpServlet class seems like a good example of a Template Pattern. As per its java docs, it provides an abstract class to be subclassed to create a HTTP servlet suitable for a website. The subclass is expected to override at least one method.

The Template pattern is meant to provide a similar functionality. It has a base class which provide basic steps using abstract methods. Later on, the subclasses change the abstract methods to implement the desired functionality without actually changing the algorithms structure, that was initially provided by the Template method.