**Chain of Responsibility**

The chain of responsibility is very useful in object-oriented programming when it comes to determining how to handle different situations. Another way to look at it would be as a chain of authority. For example, let’s focus on a department within a business. Say that one employee wants a new computer monitor. If the employee’s budget is large enough to buy the monitor, then the employee can take care of the purchase. However, if the employee’s budget isn’t large enough, they have to go to their department head to make the purchase. But what if the whole department needs new monitors? The department head must ask the district manager to authorize the purchase. This model can be applied to a hierarchy: it begins with the employee, goes to the head of the department, and finally to the district manager.

Let’s look at this from a computer science standpoint. This same sort of purchase model can be used, but we will look at it in a sense of objects and classes. There will be five main classes to make this model work: *Handler*, *EventHandler1*, *EventHandler2*, *EventHandler3*, and the *PurchaseDemo*.

To begin, the *Handler* is an abstract class that provides certain data and formats, such as a successor variable and abstract methods, for each EventHandler object. *EventHandler1* will process orders if the total is below $75, *EventHandler2* will process orders that are bigger than $75 and less than $500, and EventHandler3 will handle orders above $500.

Looking back at our purchase example, EventHandler1 represents the employee with a $75 budget, *EventHandler2* represents the department head, with a budget up to $500, and *EventHandler3* represents the district manager, who has authority to spend above $500.

Now let’s apply these classes to *PurchaseDemo*. PurchaseDemo will have a method, createChain(), that creates the chain of authority by creating Handler *employee*, which is an EventHandler1, Handler *deptHead*, which is an EventHandler2, and it will create Handler *distManager*, which is an EventHandler3. When *deptHead* is created, its reference address is assigned to the successor variable in *employee*. When *distManager* is created, its reference address gets assigned to the successor variable in *deptHead*.

The actual execution of the program is as follows: A purchase request is made to buy a keyboard that costs $125. This request begins in EventHandler1. EventHandler1 checks the incoming price, $125, to see if it is falls within the given budget ($75 for EventHandler1). Because $125 is greater than $75, EventHandler1 passes the purchase order on to its successor, EventHandler2. EventHandler2 is allowed to make purchases that are between up to $500, so EventHandler2, the deptHead, can submit and finalize the purchase order.

Reviewing the way in which a chain of responsibility works, it is obvious that it resembles the execution of a linked list. In a linked list, we create nodes that store data as well as a link (reference) to other nodes. In a chain of responsibility, we create event handlers (similar to nodes) that hold instructions on how to process input, given that the input passes the pre-condition. If the input passes the handler’s pre-condition, then the handler can process the input information, if not, the input gets passed to the handler’s successor (which holds a reference to the next level handler).

References

"Chain-of-responsibility Pattern." Wikipedia. Wikimedia Foundation, 9 Oct. 2014. Web. 10 Oct. 2014.

"Chain of Responsibility." *Chain of Responsibility*. Web. 10 Oct. 2014.

"Java Design Pattern 1.11 - Chain Of Responsibility." YouTube, 31 Mar. 2013. Web. 10 Oct. 2014.

**PurchaseDemo**





