This page last changed on Dec 07, 2006 by dswood.

Graph - Cost

## Session Objectives	## Session Overview
By the end of this session students should ... • Understand the Strategy pattern • Be able to refactor to the Strategy pattern	**Activity** • Lab - Cost • Discussion - Strategy • Discussion - Mutual Recursion

Session Notes

Lab - Cost

Introduce costs for each hop (see diagram below). Ask students to find the minimum cost path. Things to look for: • Injecting a Link class to model the cost of a link to a neighbor without breaking existing code or introducing new functionality. • Link's job: Understands a connection to a node • It's better to copy the hopsTo() code and then refactor the duplication later, than to modify hopsTo() and break existing tests. • Link can be a package level class (as opposed to inner class) as it does not need knowledge of anything in Node. • Students may first pass a flag down the methods. This will later be refactored to a Strategy. Work the students towards a solution using the strategy pattern. This will use an anonymous inner class which may not be familiar to the students. If the students do not get to the strategy then the instructors will have to demo it on the screen.	!graph with costs.png!

Discussion - Strategy

Discuss the *strategy* pattern. Put a good example of the Cost code on the screen and talk through these points:

- Ask the students when it is commonly used? To decouple an algorithm from its host, and encapsulate the algorithm into a separate class
    - What is the algorithm in the Cost example? There are 2: '1' and 'cost'. Note that the tree transversal algorithm is something completely different.
    - Ask for some more examples of when strategy might be used. Comparator
    - They tend to kill 'if' statements as we may have seen with passing a flag down.
- Describe to the students the common usage patterns of a strategy:
    - Typically implemented as anonymous inner classes which gives them access to the internals of their parent class.
    - Typically don't have state.
    - Typically declared as constants so that they can be passed in to change behaviour.
    - Often pass 'this' into the strategy since they are static and need access to a particular instance.

One thing to note is that we haven't had to change our tests significantly. This is because we haven't changed the public interface of Node significantly. Knowing that the implementation of canReach(), hopsTo() and costTo() is shared, we might want to refactor our tests down to some shared tests. However the tests are of the public interfaces, not of the specific implementation and so we should resist the urge to refactor the tests. With this we are testing our intentions rather than just shooting for 100% code coverage. When we refactored the code we did not change the intentions and so do not need to change the tests.

Discussion - Mutual Recursion

Discuss the *mutual recursion* pattern.
Where recursion spans more than one class/method. The first class/method will call the second class/method which will call back to the first. This can be within a single class or between two classes.
Ask the students to identify the mutual recursion in graph? Node.costTo() & Link.costTo(). So you don't have to recurse on yourself and this allows you to get quite a bit of work done.
This is an excellent example of collaboration. Node and Link are working closely together to solve a problem.