Activity 13-2: Strategy and Design Patterns

Introducing the Patterns

Look at handout:

- 14:37-17:20 : Definition of the problem: Strategy separates high level policy from set of low level details. Example of File Transfer.
- 20:30-23:10 : Example of Template method in the File Transfer example.

Comparison of strategy and template method

- 23:10-27:50 : Discussion of strategy vs template
- Template method is *internal polymorphism*, strategy is *external polymorphism*.
- In Template method there is only one instance/object (but 2 classes), in Strategy there are two objects.
- Template method is *smaller* and a bit *faster* (no reference traversal needed).
- Strategy is more *flexible* (the two parts can be created at different times and at different places).
- Strategies can be *hot-swapped* while the program is running.
- In strategy, high level policy and low level details are independent of each other. In template method, the low level details strongly depend on the high level policy.

Example: Tree traversals

Imagine a tree class in Java, with accompanying node interface and classes (just one way of implementing it, not the best but it will do for now – do you notice the Liskov Substitution principle violation in it?):

```
// E is the type of the elements stored at the nodes.
interface Node<E> {
    boolean isEmpty();
    E getElement();
    Node leftChild();
    Node rightChild();
}

class EmptyNode<E> implements Node<E> { ... }
class NonEmptyNode<E> implements Node<E> { ... }

class Tree<E> {
    private Node<E> head = new EmptyNode();
}
```

We want to define another class which will perform a traversal of the tree. There are two completely independent questions here:

- What kind of traversal to perform (preorder, inorder, postorder).
- What to do on each node.

This is a perfect example of the strategy and template method patterns: We can separate the two operations in the two classes involved in those patterns. For example, here is the strategy pattern solution to it:

```
interface NodeStrategy<E> {
    void accept(E element);
}

class InorderTraversal<E> {
    private NodeStrategy strategy;
    private Tree tree;
    public InorderTraversal<E>(Tree t) {
        tree = t;
    }

    public void traverse() {
        traverseNode(tree.getHead());
    }

    public void traverseNode(Node<E> n) {
        if (n.isEmpty()) { return; }
        traverseNode(n.leftChild());
        strategy.accept(n.getElement());
        traverseNode(n.leftChild());
    }
}
```

The InorderTraversal class knows how to navigate the tree, but it delegates to the NodeStrategy class in order to handle what happens to each element. This way the "looping" logic remains the same while the details on what to do with each element differ.

Here is the same example with the Template Method approach:

```
abstract class InorderTraversal<E> {
    private Tree tree;
    public InorderTraversal<E>(Tree t) {
        tree = t;
    }

    public void traverse() {
        traverseNode(tree.getHead());
    }

    public void traverseNode(Node<E> n) {
        if (n.isEmpty()) { return; }
        traverseNode(n.leftChild());
        accept(n.getElement());
        traverseNode(n.leftChild());
    }
```

```
    abstract void accept(E element);
}

class PrintInorderTraversal<E> extends InorderTraversal<E> {
    void accept(E element) {
        System.out.println(element);
    }
}
```

NOTE: This exhibits the one of the limitations of the template method pattern: The printing mechanism is here intimately related to the traversal strategy; we can't just take that printing code and apply it to the PreorderTraversal strategy, we would need to create a new class for it.

In general, imagine we wanted to work with all 3 traversals, and there were 4 different kinds of operations we wanted to do with the trees (print, add, find max, append to list). We would then need 12 different classes: 3 abstract classes for the 3 different kinds of traversals, and 4 subclasses of each for the different operations.

On the other hand, the strategy pattern would need just 3+4=7 classes: 3 for the different traversal kinds, and 4 for the different operation kinds. The strategy pattern then tells us how to mix them together to get any of the desired 12 combinations.

The ETS drawing example

- 32:40-34:00: Features (show)
- 34:00-36:30: Description of data structure (show)
- 36:30-41:10: Using the template method (show)