Example 9-5 shows how the code block decorator intercepts calls for execution and is then able to make a decision as to whether to queue the block for compilation. In addition to this, we also have a method that can replace the decorator's target with a different block instance. Once the block has been compiled, the decorator is useless, and so ideally we would like to replace it. This replacement is actually achieved quite easily. By replacing the original interpreted block with a block like Example 9-6, we can propagate the notice about the new block back up the call stack. Once the exception reaches the right level, we can replace the reference to the decorator with a direct reference to the compiled block.

EXAMPLE 9-6. Block replacing CodeBlock

```
public class CodeBlockReplacer implements CodeBlock
{
    private CodeBlock target;

    public int execute()
    {
        throw new CodeBlockReplacementException(target);
    }

    static class CodeBlockReplacementException extends RuntimeException
    {
        private CodeBlock replacement;

        public CodeBlockReplacementException(CodeBlock compiled)
        {
            replacement = compiled;
        }

        public CodeBlock getReplacementBlock()
        {
            return replacement;
        }
    }
}
```

TIP #6: USE DECORATOR PATTERNS WITH CARE

The decorator pattern is nice from a design point of view, but the extra indirection can be costly. Remember that it is permitted to remove decorators as well as add them. This removal may be considered an "exceptional occurrence" and can be implemented with a specialized exception throw.