```
    public Class createClass(String name, byte[] b)
    {
        if (++classesCount == CLASSES_PER_LOADER)
            newClassLoader();

        return defineClass(name, b, 0, b.length);
    }

    protected Class findClass(String name) throws ClassNotFoundException
    {
        throw new ClassNotFoundException(name);
    }
}
```

HOTSPOT CODE CACHE

In JPC, when running complex jobs that load such large numbers of classes, there is another memory limit that will get hit. In Sun HotSpot JVMs, the just-in-time compiled code is stored in a nonheap area called the code cache. Not only does JPC produce a large number of classes, it is also unusual in that a high fraction of these are candidates for HotSpot. This means that the HotSpot caches get filled rapidly. This just means that any boost to the permanent generation size normally is also accompanied by an increase in the size of the code cache.

Codeblock replacement

We now have a compiled, loaded, and instantiated custom code block instance. Somehow we have to get this block in place where it's needed. How this is done is also closely related to how the blocks are initially scheduled for execution; see Example 9-5.

EXAMPLE 9-5. Decorator pattern for compilation scheduling

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

    public int execute()
    {
        makeSchedulingDecision();
        target.execute();
    }

    public void replaceTarget(CodeBlock replacement)
    {
        target = replacement;
    }
}
```