Well, little that we can do without violating one of the absolute maxims and declaring a class in the `java.lang` package. So we now know that our superclass is going to helpfully hold references for us. What does this mean for the class unloading?

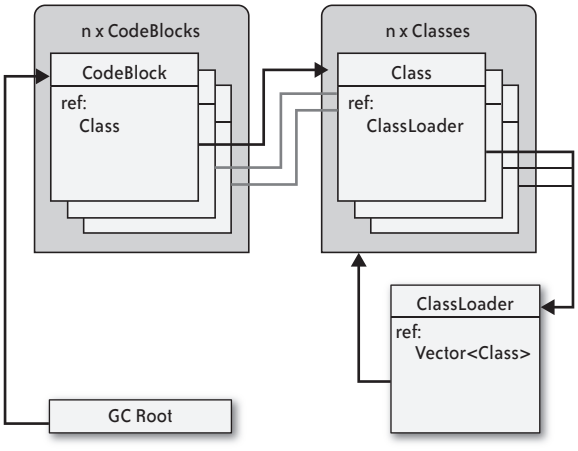

FIGURE 9-12. The GC root path for classes

In the GC root path in Figure 9-12, we can see that until all the instances of all the classes loaded by a classloader are GC candidates, all the classes will remain loaded. So one active codeblock can stop `n` classes from being unloaded. That's no good at all.

There is one simple way to mitigate this problem. No one said that we had to let `n` rise uncontrollably. If we limit the number of classes loaded by any one loader, then we decrease the chances of classes being held hostage. In JPC we have a custom classloader that will only load up to 10 classes by default. Loading the 10th class triggers the construction of a new loader, which will be used for the next 10, and so on. This method means that any one class can only hold up to 10 others hostage; see Example 9-4.

EXAMPLE 9-4. ClassLoader implementation in JPC

```
private static void newClassLoader()
{
    currentClassLoader = new CustomClassLoader();
}

private static class CustomClassLoader extends ClassLoader
{
    private int classesCount;

    public CustomClassLoader()
    {
        super(CustomClassLoader.class.getClassLoader());
    }
```