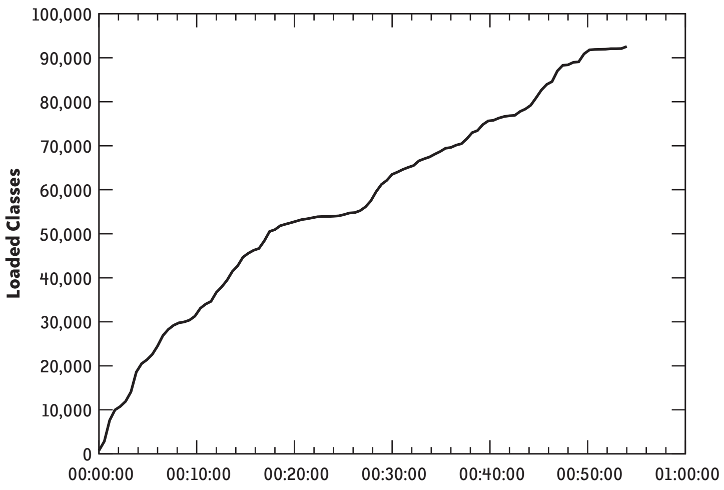

FIGURE 9-11. Class count during a modern GNU/Linux boot

So we need to find the space for all these classes. Class file storage in the JVM occurs in a special memory area outside the normal object heap known as the "Permanent Generation" space. In a typical Sun JVM, the permanent generation starts out at 16 MB and can grow to at most 64 MB. Obviously 100,000 classes are not going to fit into a 64 MB heap. There are two ways we can solve this problem.

The first is with the command *java* `-XX:MaxPermSize=128m`. Clearly, solution one is to increase the size of the permanent generation. Crude though this may be, it does help to solve the problem. Unfortunately, it's no solution on its own, because all we have done is delay the inevitable. Eventually we will load enough classes to fill the new space, and we can't just keep adding more room.

The second half of the solution involves reducing the number of loaded classes. In fact, shouldn't the garbage collector be clearing away all the unused classes? Classes are really no different from heap objects as far as garbage collection goes. A class can be garbage collected (unloaded) only if no live references to the class are held. Of course every instance of a class holds a strong reference to the `Class` object of its type. So for a class to be collected, there must first be no live instances of the class, and then no additional references to the class. Therefore, perhaps we can solve the problem if we define our custom classloader as shown in Example 9-3.

EXAMPLE 9-3. Simple no-holding ClassLoader

```
public class CustomClassLoader extends Classloader
{
    public Class createClass(String name, byte[] classBytes)
    {
        return defineClass(name, classBytes, 0, classBytes.length);
```