Under the Java 1.1 and later platforms, the call to `defineClass` is replaced with

```
    c = defineClass(name, bytes, 0, bytes.length);
```

The first example is acceptable under Java 1.1 but deprecated (that is, it's considered bad style, and one day it may no longer be acceptable). If the `name` doesn't match the name of the class found in the bytes, then a `ClassFormatError` is thrown.

If the class is found neither in the cache nor wherever `findClass` looks for it, the class loader calls `findSystemClass` to see whether the system can locate a definition for the class. If `findSystemClass` doesn't find it, it throws a `ClassNotFoundException`, since that was the last chance to find the class.

8.3.1 Caching Classes

It's important that a class loader return the same `Class` object each time it's given a particular name. If the same class were loaded more than once, it would be confusing to users who might find that two classes with identical names aren't identical. Class static constructors might be invoked multiple times, causing problems for classes that were designed to expect them to be called only once.

Under Java 1.0, it was the responsibility of the class loader to cache classes itself. This is usually done with a `Hashtable`, as shown in the template. However, this still leaves the possibility of confusion, since two different class loaders might each load a class into the system with the same name. Java 1.1 resolves this problem by handling the caching itself. It makes this cache available to the class loader developer through a method called `findLoadedClass`:

```
    Class findLoadedClass(String name);
```

A call to `findLoadedClass` replaces the cache lookup. When `defineClass` is called, it maps the name of the class to the `Class` that is returned. After that, `findLoadedClass` always returns that `Class` whenever it's given the same name, no matter which class loader invokes it.

When implementing your class loader, you will have to decide whether to use the Java 1.0 interface or the 1.1 interface. The 1.0 interface is supported on virtual machines supporting Java 1.1 but not vice versa. However, using the 1.0 interface will have different results on a JVM 1.1 if the class loader tries to define a class more than once. On a JVM 1.0, it would actually load the classes multiple times, and the system would have two different classes with the same name. These classes wouldn't share `static` fields or use `private` fields or methods on the other. On 1.1 and later JVMs, however, `defineClass` throws an exception when it's asked to define the class a second time anywhere in the virtual machine, even if the bytes are identical.