Class.newInstance method. The following implementation sketch illustrates this technique:

```java
// Provider framework sketch
public abstract class Foo {
    // Maps String key to corresponding Class object
    private static Map implementations = null;

    // Initializes implementations map the first time it's called
    private static synchronized void initMapIfNecessary() {
        if (implementations == null) {
            implementations = new HashMap();

            // Load implementation class names and keys from
            // Properties file, translate names into Class
            // objects using Class.forName and store mappings.
            ...
        }
    }

    public static Foo getInstance(String key) {
        initMapIfNecessary();
        Class c = (Class) implementations.get(key);
        if (c == null)
            return new DefaultFoo();

        try {
            return (Foo) c.newInstance();
        } catch (Exception e) {
            return new DefaultFoo();
        }
    }
}
```

The main disadvantage of static factory methods is that classes without public or protected constructors cannot be subclassed. The same is true for nonpublic classes returned by public static factories. For example, it is impossible to subclass any of the convenience implementation classes in the Collections Framework. Arguably this can be a blessing in disguise, as it encourages programmers to use composition instead of inheritance (Item 14).

A second disadvantage of static factory methods is that they are not readily distinguishable from other static methods. They do not stand out in API documentation in the way that constructors do. Furthermore, static factory methods represent a deviation from the norm. Thus it can be difficult to figure out from