作用：保留系统当前的安全上下文细节，其中就包括当前使用系统的用户的信息。

#### 上下文细节怎么表示？

用SecurityContext对象来表示。

#### 每个用户都会有它的上下文，那这个SecurityContext保存在哪里呢？

存储在一个SecurityContextHolder中，整个应用就一个SecurityContextHolder。

#### SecurityContextHolder存储SecurityContext的方式？

这要考虑到应用场景：

**1）单机系统**

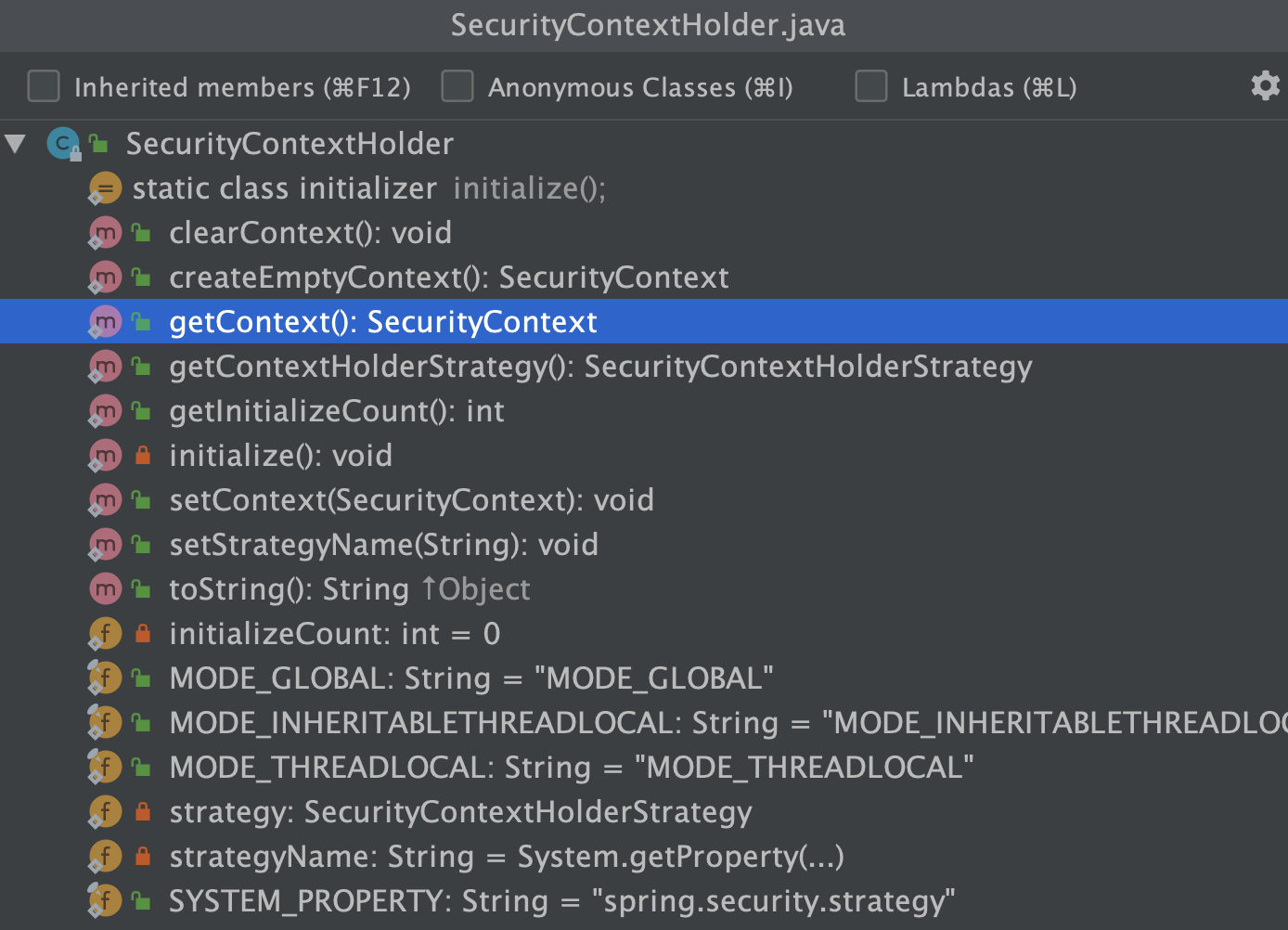
应用从开启到关闭的整个生命周期只有一个用户在使用。整个应用只需要保存一个SecurityContext（安全上下文即可）。

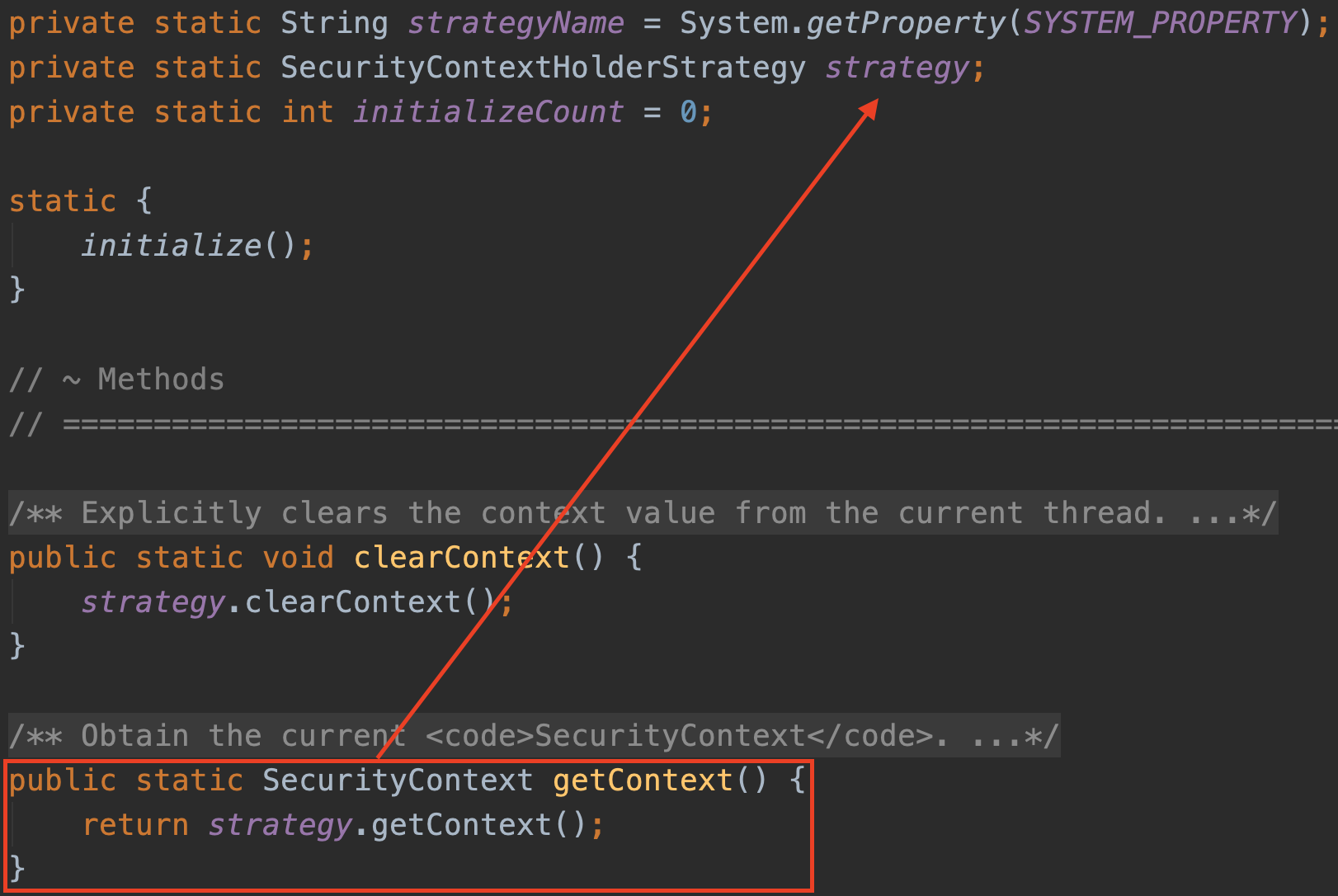
**2）多用户系统**

Web系统，整个生命周期可能同时有多个用户在使用。这时候应用需要保存多个SecurityContext（安全上下文），需要利用ThreadLocal进行保存，每个线程都可以利用ThreadLocal获取其自己的SecurityContext。

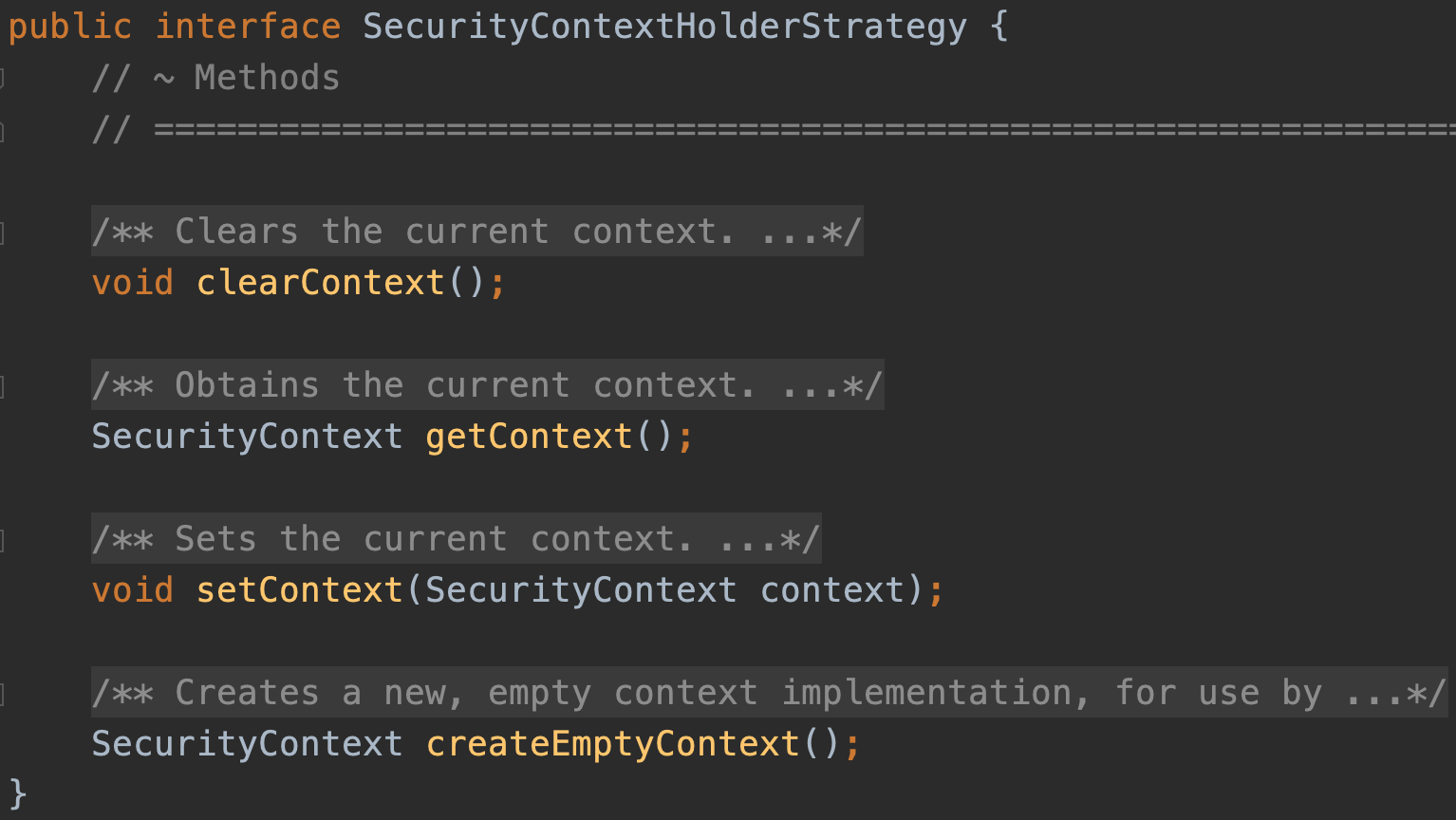
#### 源码分析

SecurityContextHolder结构：

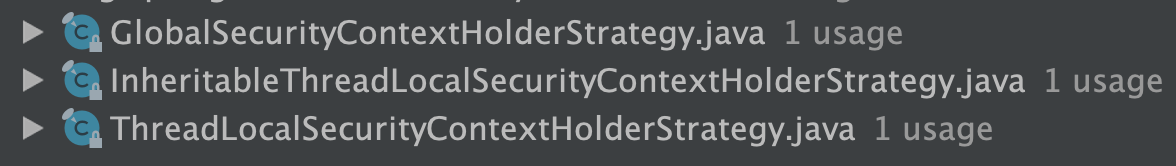




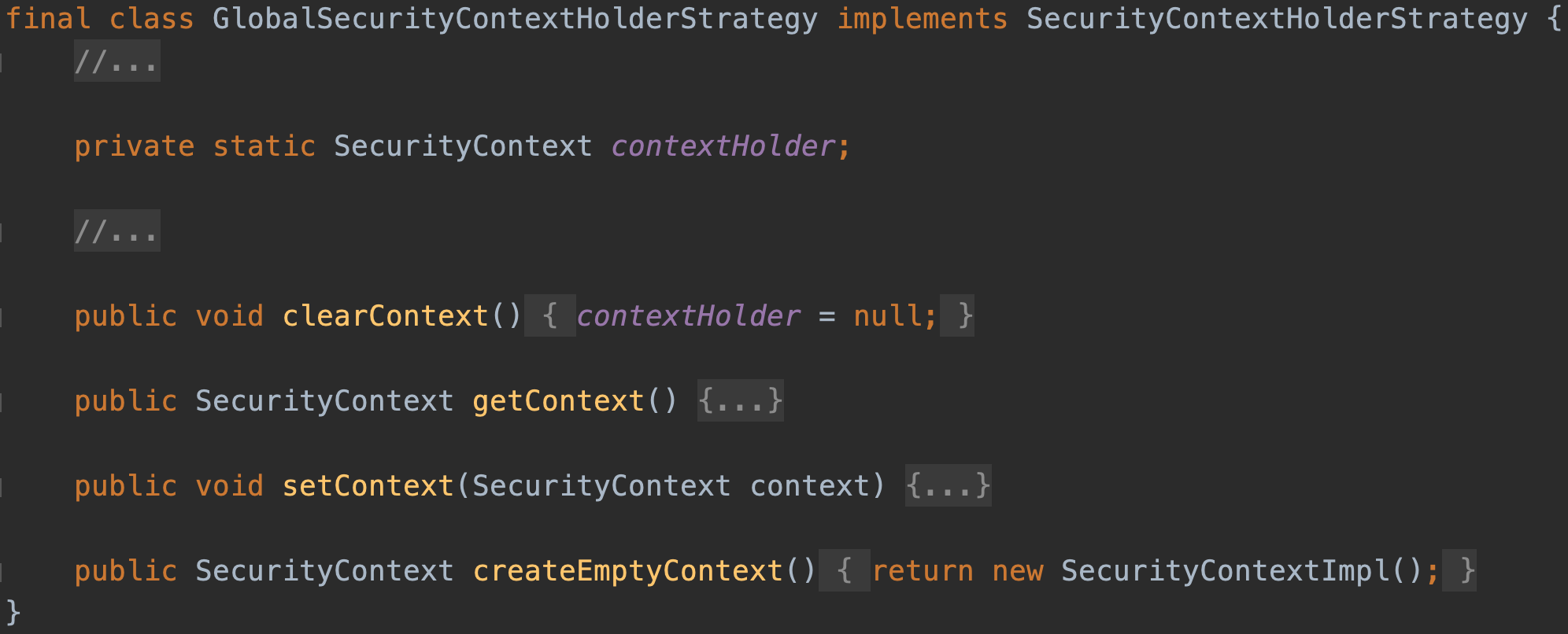
SecurityContextHolder利用SecurityContextHolderStrategy（存储策略）进行上下文的存储。



SecurityContextHolderStrategy只是一个接口,提供了创建、清空、获取、设置上下文的操作。那么它有哪些实现类呢（即存储策略）？

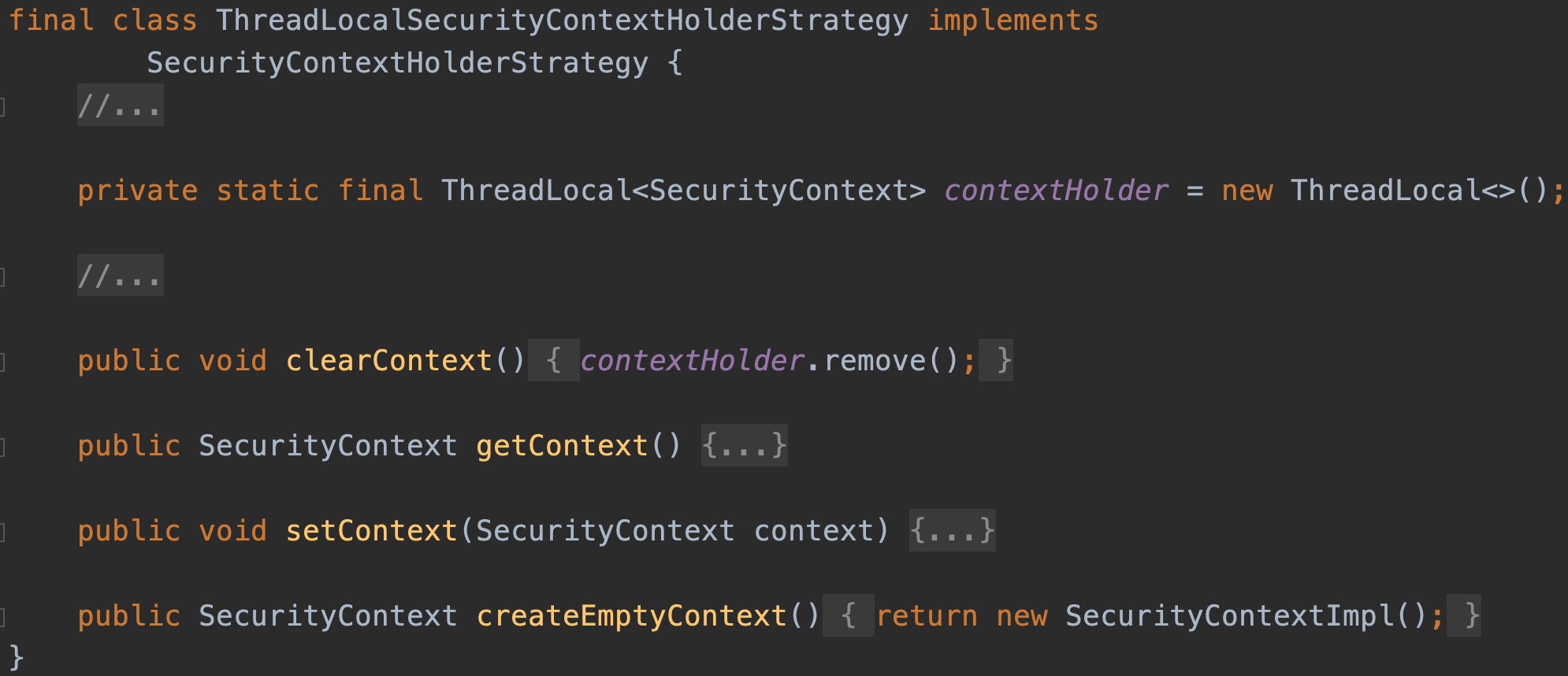


**GlobalSecurityContextHolderStrategy.java**

****

全局的上下文存取策略，只存储一个上下文，对应前面说的单机系统。

**ThreadLocalSecurityContextHolderStrategy.java**

****

基于ThreadLocal的存储策略实现，ThreadLocal内部会用数组来存储多个对象。原理是，ThreadLocal会为每个线程开辟一个存储区域，来存储相应的对象。

#### Authentication——用户信息的表示

SecurityContextHolder中存储了当前与系统交互的用户的信息。Spring Security使用一个Authentication对象来表示这些信息。一般不需要自己创建这个对象，但是查找这个对象的操作对用户来说却非常常见。

Object principal = SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();

String username1 = null;  
if (principal instanceof UserDetails) {  
 username1 = ((UserDetails) principal).getUsername();  
}else {  
 username1 = principal.toString();  
}