As far as I understand the task, the main problem in common resources.  
When we start two different threads with the same key, they try to change something common.  
I didn't use classes from concurrency package in the ExternalSystem, because, as I understand the task,  
resolving problem should be in KeyHandler class.

My decision:

**public class** KeyHandler {  
  
 **private** ExternalSystem **externalSystem**;  
 **private final** ConcurrentMap <Key, Key> **keysForLock** = **new** ConcurrentHashMap();  
  
 {  
 **externalSystem** = **new** ExternalSystem();  
 }

**public void** handle(Key key) {  
 **keysForLock**.putIfAbsent(key, key);  
 **synchronized** (**keysForLock**.get(key)) {  
 **externalSystem**.process(key);  
 }  
 }  
  
}

Below I described how I thought.

One of decision: mark method synchronized. It's mean that only one thread can execute this method at one time. Example:

**public synchronized void** handle(Key key) {  
 **externalSystem**.process(key);  
}

It works, but it is bad decision, because our application will not be multithreading in fact.  
All actions will execute as in the standard application in one thread (if we will see on time)

Second decision: synchronization by key.

**public void** handle(Key key) {  
 **synchronized**(key) {  
 **externalSystem**.process(key);  
 }  
}

We can do it only if we sure, that if key1 equals key2 it's mean, that key1 and key2 it the same object in Java memory (the same references).  
For example: new Key("key1"), new Key("key1"). For our application they are equal keys, but for Java it is two different objects!  
  
But I thought that idea is good. We make synchronization only for the same object!  
  
Result. Base on second idea I create list of object for locking in ConcurrentHashMap.  
 We receive key, find locking object and make synchronize.

You can find my test project by the next link on GitHUB:

or in dbGrip\_testtask.zip file.

I'll wait feedback from you. As I told on my first interview, my knowledge in multithreading programming is more theoretical and your opinion by my resolving is really interesting for me.