Clayton Turner
CSIS 603 - Design Patterns

Creational Patterns: Singleton

Implemented pattern:
https://android.googlesource.com/platform/tools/tradefederation/+/5f019e3ff0853a01330d6a3fd931160fd89dbb87/src/com/android/tradefed/targetsetup/FileDownloadCacheFactory.java (Code on second page)

The singleton in question for this response is from the source code of the Android operating system. The singleton class is named FileDownloadCacheFactory. The factory is used to interact with FileDownloadCache instances. FileDownloadCaches are directory-dependent so it makes sense that it is possible to have multiple instances of FileDownloadCache yet just one factory. This can be useful when going between external and internal storages or just within each storage type itself - possibilities are endless with Android. Additionally, this is lazy initialization due to the if == null, then create the singleton section.

A FileDownloadCache is “a helper class that maintains a local filesystem LRU cache of downloaded files” as per the Android sourcecode site. FileDownloadCache instances are used in order to retrieve specific files from the given remotePath. If the File (referring to the class type, so a specific instance) exists in cache, then it is pulled from there and, if not, then the file is pulled into the cache and then pulled down. A FileDownloadCache also has a public method to alter its max cache size.

```java
package com.android.tradefed.targetsetup;

import java.io.File;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

/**
* A factory for creating {@link FileDownloadCache}
*/
public class FileDownloadCacheFactory {

	// use the "singleton inner class" pattern
	// http://en.wikipedia.org/wiki/Singleton_pattern#The_solution_of_Bill_Pugh
	private static class SingletonHolder {
		public static final FileDownloadCacheFactory INSTANCE = new
	FileDownloadCacheFactory();
	}

	private Map<String, FileDownloadCache> mCacheObjectMap =
	Collections.synchronizedMap(
		new HashMap<String, FileDownloadCache>());

	/**
	* Get the singleton instance of FileDownloadCacheFactory
	* @return
	*/
	public static FileDownloadCacheFactory getInstance() {
		return SingletonHolder.INSTANCE;
	}

	/**
	* Retrieve the {@link FileDownloadCache} with the given cache directory, creating if
	necessary.
	* <p/>
	* Note that the cache assumes that this process has exclusive access to the
	<var>cacheDir</var>
	* directory. If multiple TF processes will be run on the same machine, they MUST
	each use
	* unique cache directories.
	*
	* @param cacheDir the local filesystem directory to use as a cache
	* @return the {@link FileDownloadCache} for given cacheDir
	*/
	public synchronized FileDownloadCache getCache(File cacheDir) {
		FileDownloadCache cache =
		mCacheObjectMap.get(cacheDir.getAbsolutePath());
		if (cache == null) {
			cache = new FileDownloadCache(cacheDir);
			mCacheObjectMap.put(cacheDir.getAbsolutePath(), cache);
		}
		return cache;
	}
}
```