Assignment2 Formal report

218044198
Xixi Gao
June 25, 2021

Table of contents

Table of contents 2

1 Introduction 3

2 MultipleListingService.java 3

2.1 MultipleListingService.java without Class builder 3

2.2 Builder patter 4

2.3 MultipleListingService.Builder 4

3 Singleton.java 5

4 StudentCode.java 6

5 Conclusion 7

6 Reference 7

1 Introduction

This project is a web application using the Spring boot framework. It aims to find the relative record in our data file mls.txt by uniq UUID. The returned record is a MultipleListingService object with attributes id, address, price.

Before adding code in Singleton.java and StudentCode.java, if we want to find the record of one UUID, it needs to iterate from the beginning of the mls.text to see whether the record is found or not.

After modifying the code, we add a caching mechanism in Singleton.java, which can add the new data into memory and search cache first to see if the record is already in the memory. Compared to the original version, we may not need to search from the beginning of the file every time.

Applying Singleton method in mlsLookup web method in Student.java to make sure when we want to find the responsible report, we search the cache first.

2 MultipleListingService.java

2.1 MultipleListingService.java without Class builder

MultipleListingService is an object with private attributes id, address, and price of the estate. We have default constructor public MultipleListingService() and methods to get or set all these three attributes.

java.lang.String	getAddress()	Get the address of the property
java.util.UUID	getId()	Get the ID of the property
int	getPrice()	Get the price of the property
void	**setAddress** (java.lang.String address)	Set the address of the property
void	**setId** (java.util.UUID id)	Set the ID of the property
void	**setPrice** (int price)	Set the price of the property

Besides that, there is a Class builder inside the MultipleListingService Class. It is a special design pattern named Builder pattern. We will discuss it in Chapter 2.2.

2.2 Builder patter

"Builder pattern builds a complex object using simple objects and using a step by step approach. This comes under creational pattern as this pattern provides one of the best ways to create an object."

We will discuss by example in 2.3.

"A builder Class builds the final object step by step. This builder is independent of other object."

Using Builder pattern we don't need to full all the parameters (some are null) if we need too many parameters. For example, if one constructor have many parameters: new Building("123", "Finch ave", null, null, null, null, null}. By using builder pattern, we can write new Builder("123").address("Finch ave").build(). It's more clean and simple.

2.3 MultipleListingService.Builder

We are using MultipleListingService.Builder to discuss the Builder pattern.

Builder is a static class, it has attributes id, address and price.

It has one constructor and three methods:

Constructor	Description
Builder (java.util.UUID id)	

Modifier and Type	Method	Description
MultipleListingService	build()	Finalize the construction of an MLS record using Builder design pattern.
MultipleListingService.Builder	**locatedAt** (java.lang.String address)	Set the address of the property
MultipleListingService.Builder	**pricedAt** (int price)	Set the price of the property

We said the Builder pattern can build the final object step by step in the last chapter. Let's explain it. To create a new MultipleListingService object, firstly, We can use the constructor to create a Builder object new Builder(id). Then we use methods locatedAt. Then pricedAt method

new Builder(someId).locatedAt(someStreet).priceAt(somePrice) to pass the other attribute street and price to our builder object. Finally, apply the build() method to return a MultipleListingService object that have the same attributes with as builder object. New Builder(someId).locatedAt(someStreet).priceAt(somePrice).build().

3 Singleton.java

In Singleton.java, we have a Class Singleton. If the private constructor new Singleton() is called, the cacheMap will be initialized.

We have methods:

Type	Method	Description
Singlonton	getInstance()	If instance is null, create a new Singleton(), if not null, return the exist instance

Void	addCache(UUID uuid, MultipleListingService record)	add new <key, value> to the cache memory
Void	removeCacheData(UUID uuid)	remove from the cache memory
Int	getSize()	Return the size of cache
MultipleListingService	getMls(UUID uuid)	If no uuid in cacheMap, return null. Else return the value of uuid in cacheMap

In every method, I used the keyword synchronized.

"Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results. “

So it needs to be made sure by some synchronization method that only one thread can access the resource at a given point of time. For example, if more than one person called the getInstance() method, different objects may be returned to different people.

4 StudentCode.java

Firstly, we need to initialize our cacheMap using Singleton.getInstance();

Then we use Singleton.*getMls*(uuid) to find whether this uuid exists in our cacheMap.

if we cannot find the uuid in cache memory, then we need to iterate the data file to find uuid. If uuid is found, add the new <uuid, record> into our cacheMap.and add the new <key, value> into memory, the next time if we want to find the record of this uuid, we only need to see the cacheMap.

If uuid cannot be found in data file, then an Exception is through. if we can find in memory, return the value of the key

I also write two lines of code to let us know whether the caching works in the console:

```
System.out.println(Singleton.getSize() + " new data added into cache”);
System.out.println("Find in cache");
```

5 Conclusion

After running the code, I find that the Singleton cacheMap works in StudentCode.java. And it seems faster than the original version. There are only five data in data file. I need more to test. More learning should to be done.

6 Reference

1.https://www.geeksforgeeks.org/synchronized-in-java/

2.https://www.tutorialspoint.com/design_pattern/builder_pattern.html

3.https://www.programmersought.com/article/25297074308/