CSE 12: PA6

2-11-21

Focus: PA6 HashMaps and File Systems

PA6

Overview of PA6

- Part I: An Implementation of `DefaultMap`
  - Given the interface `DefaultMap`, implement `MyHashMap.java`
  - You may use Linear Probing or Separate Chaining for collision handling
  - 9 methods + 1 constructor in all (use the lecture and discussion slides to help you!)
- Part II: `FileSystem` Implementation
  - Implement the `FileData` class (this stores the information for a specific file)
    - Two methods: constructor and toString()
  - Implement the `FileSystem` class
    - Ten methods: some methods are very similar!
- Tester Files
- Part III: Gradescope Questions
- Style - must follow all indicated guidelines

Part 1: DefaultMap

Tips for Part I: MyHashMap.java

- Each entry (key value pair) is represented by the class HashMapEntry. Use this in your implementation
- For the hash function, you may use hashCode()
- Create your own helper methods, one suggestion would be for rehashing
- If you are implementing separate chaining, use the `buckets` instance variable. If you are using linear probing, use the `entries` instance variable. Do not use both!
  - For a refresher, review the lecture and discussion videos to see in depth examples

Amortized Runtime Analysis

Lecture 17 will cover Amortized Runtime Analysis

Think of it as the **average case** when running the HashMap functions. We do not need to consider the expensive methods that are only called occasionally (i.e. expandCapacity) in our analysis.

In a HashMap that has a good hash function and an appropriate load factor, are we going to be closer to the worst or best case?

Only a few methods actually need the amortized analysis, for others there is no method being called that could be affecting the overall runtime.

Part 2: File System

FileData.java

This class represents the file that contains the information for name, directory, and last modified date.

Two Methods

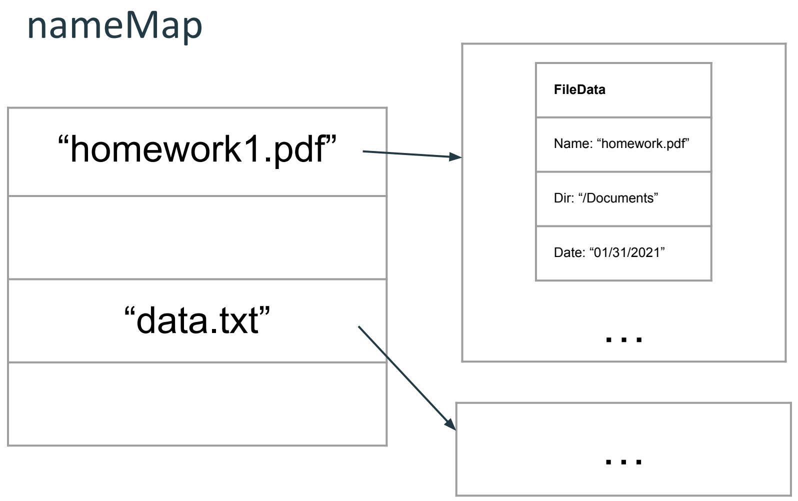
FileData() - constructor, initializes the instance variables

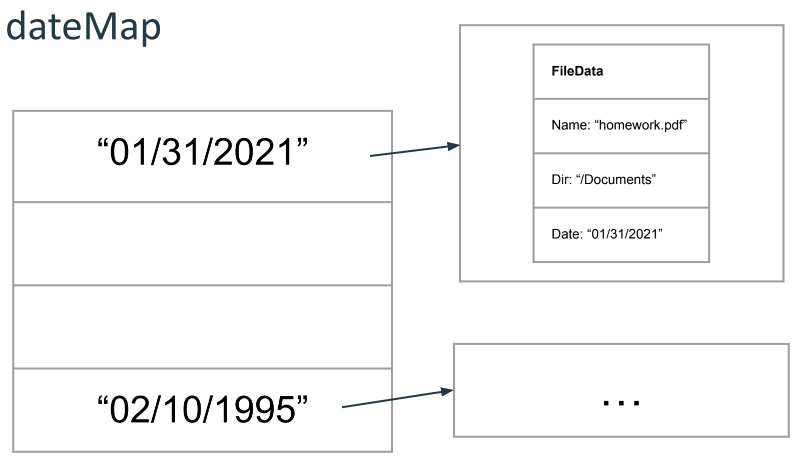
toString() - returns the string representation of the data in the FileData object

FileSystem.java

- FileSystem represents the entire structure of the system where you can add, remove, and search for files.
- One of the constructors populates nameMap and dateMap based on a given file, an example is provided in the starter code "input.txt"

MyHashMap<String, ArrayList<FileData>>





Example of add

```
add(“homework1.pdf”, /Documents”, “01/31/2021”);
```

FileData
Name: “homework.pdf”
Dir: “/Documents”
Date: “01/31/2021”

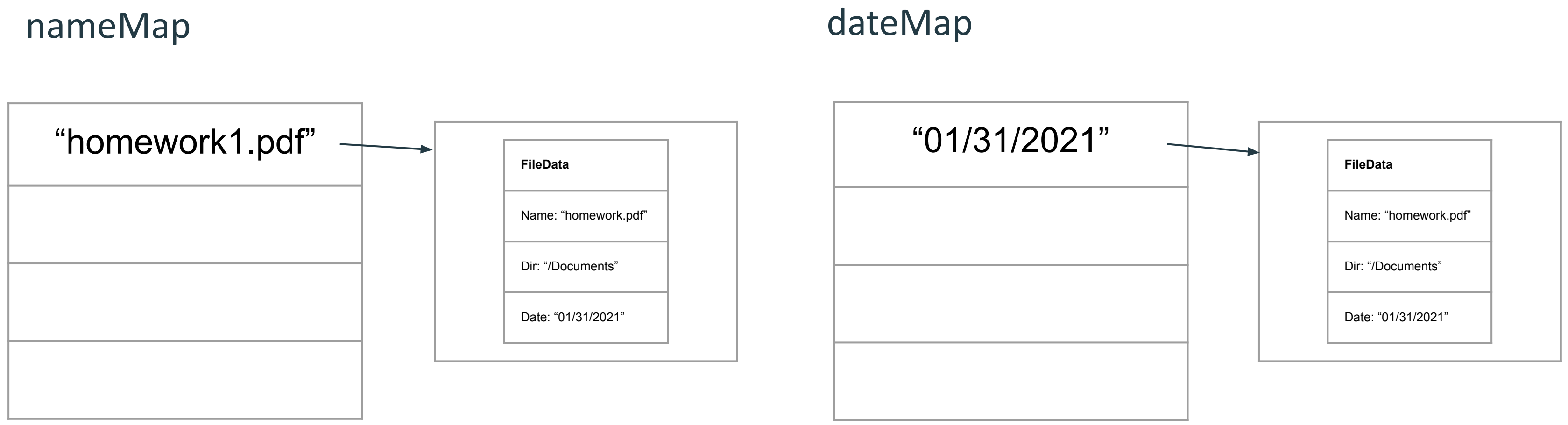
$$\downarrow$$

nameMap

dateMap

After adding

MyHashMap<String, ArrayList<FileData>>



Watch out!

Our File System allows duplicates, ONLY if they are in different directories.

If the file already exists with the same name, then return "false" when adding

Use this chart as a reference to help figure out when to deal with duplicates.

name	date	directory	ok?
same	same	diff	ok
same	diff	diff	ok
diff	same	diff	ok
diff	diff	diff	ok
same	same	same	no!
same	diff	same	no!
diff	same	same	ok
diff	diff	same	ok

Notice Methods are Similar

```
public ArrayList<FileData> findFilesByName(String name)

public ArrayList<FileData> findFilesByDate(String modifiedDate)
```

Same method, just different variable to work with!

Tip: think about what methods from the `DefaultMap` are best suited. Are there times when you can use the keys() method?

Testing

How can we test the FileSystem?

- Create helper methods for comparing FileData objects and ArrayLists of FileData
- Use the constructor that takes an input file to quickly populate your FileSystem
- Utilize the toString() method in the FileData class to get a string representation
- The most crucial method is the add() method, focus on this method and then from there you can use it in your tests for other methods

Overall Tips

- Test your MyHashMap thoroughly before moving on to part 2
    - The autograder will use our implementation of MyHashMap to ensure you do not get penalized for both parts but you may have a hard time working on part 2 if your hash map has bugs
- Look at the method headers, specifically, what does the method return. This can help you keep track of what each method is supposed to do.
- Use Separate Chaining! This way you do not have to deal with tombstones and finding a good key value for place holding.