

# p4 Hash Table

[Overview](#) | [Specifications](#) | [Files](#) | [Steps](#) | [Submission](#)


UPDATE 7/26: We have put together a Canvas page about the runtime analysis with Java Mission Control that has additional pointers and information on what we expect for the analysis. You can find it [here \(https://canvas.wisc.edu/courses/202692/pages/java-mission-control-analysis-additional-information\)](https://canvas.wisc.edu/courses/202692/pages/java-mission-control-analysis-additional-information).

## Overview:

For this project, you will be implementing a hash table, testing its functionality with black-box JUnit tests, and writing a small program to analyze the performance of your hash table against Java's built-in TreeMap. [TreeMap \(https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/TreeMap.html\)](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/TreeMap.html) is a known and in-built data structure of Java.

To analyze the performance, you need to write a small program that performs the same operations on both your custom **HashTable** class and Java's **TreeMap** class. You will be using a program profile class, and Java Mission Control to analyze the profile information.

For this assignment, you are required to:

1. implement a generic **HashTable** class
2. implement **HashTableTest** class to test the generic **HashTable** class
3. test the performance of **HashTable** by
  1. writing **MyProfiler.java**, a program that will be profiled to compare your **HashTable** and Java's **TreeMap**
  2. running your program **MyProfiler** to generate the **jfr** file
  3. using [Oracle Java Mission Control \(https://canvas.wisc.edu/courses/202692/pages/oracle-java-mission-control\)](https://canvas.wisc.edu/courses/202692/pages/oracle-java-mission-control) (**jmc**) to analyze the generated **my\_profile.jfr** data
  4. answering the questions in [conclusions.txt \(https://canvas.wisc.edu/courses/202692/files/13057031/download?wrap=1\)](https://canvas.wisc.edu/courses/202692/files/13057031/download?wrap=1) 
5. taking screenshots of the relevant parts of your profile data as viewed from **Java Mission Control**

All files must be named correctly (case-sensitive). You may define and submit other package-level (not public) classes as needed and you may add and edit any private members to your classes.

The goal for your **HashTable** is to build a searchable data structure that achieves constant time  $O(1)$  for lookup, insert, and delete operations.










Duplicate keys are allowed. If key is already in the HashTable, replace associated value with the new value.

Use what you have learned about writing tests and the JUnit testing framework to ensure that your hash table implementation works correctly prior to analyzing its performance.

## Files

Note: updates to these files are possible, and will be posted in announcements on this page.

In the [p4.zip](https://canvas.wisc.edu/courses/202692/files/13057041/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057041/download?wrap=1>) file:

- [conclusions.txt](https://canvas.wisc.edu/courses/202692/files/13057031/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057031/download?wrap=1>)  (<https://canvas.wisc.edu/courses/202692/files/13057031/download?wrap=1>) (do answer the questions after profiling and analyzing, do SUBMIT)
- [DataStructureADT.java](https://canvas.wisc.edu/courses/202692/files/13057032/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057032/download?wrap=1>)  (<https://canvas.wisc.edu/courses/202692/files/13057032/download?wrap=1>) (provided interface - do NOT EDIT or SUBMIT)
- [HashTableADT.java](https://canvas.wisc.edu/courses/202692/files/13057034/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057034/download?wrap=1>)  (<https://canvas.wisc.edu/courses/202692/files/13057034/download?wrap=1>) (provided interface - do NOT EDIT or SUBMIT)
- [HashTableTest.java](https://canvas.wisc.edu/courses/202692/files/13057035/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057035/download?wrap=1>)  (<https://canvas.wisc.edu/courses/202692/files/13057035/download?wrap=1>) (starter for JUnit test class - write your tests in here and SUBMIT this file)
- [HashTable.java](https://canvas.wisc.edu/courses/202692/files/13057033/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057033/download?wrap=1>)  (<https://canvas.wisc.edu/courses/202692/files/13057033/download?wrap=1>) (starter class - do EDIT and SUBMIT - define your hash table here)
- [IllegalNullKeyException.java](https://canvas.wisc.edu/courses/202692/files/13057036/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057036/download?wrap=1>)  (<https://canvas.wisc.edu/courses/202692/files/13057036/download?wrap=1>) (provided class - do NOT EDIT or SUBMIT)
- [junit-platform-console-standalone-1.5.2.jar](https://canvas.wisc.edu/courses/202692/files/13057037/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057037/download?wrap=1>) (for running JUnit5 tests -- do not submit)
- [KeyNotFoundException.java](https://canvas.wisc.edu/courses/202692/files/13057038/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057038/download?wrap=1>)  (<https://canvas.wisc.edu/courses/202692/files/13057038/download?wrap=1>) (provided class - do NOT EDIT or SUBMIT)
- [MyProfiler.java](https://canvas.wisc.edu/courses/202692/files/13057039/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057039/download?wrap=1>)  (<https://canvas.wisc.edu/courses/202692/files/13057039/download?wrap=1>) (starter class - do write your own profiling code and do SUBMIT)
- [SampleProfilerApplication.java](https://canvas.wisc.edu/courses/202692/files/13057040/download?wrap=1) (<https://canvas.wisc.edu/courses/202692/files/13057040/download?wrap=1>) 

(<https://canvas.wisc.edu/courses/202692/files/13057040/download?wrap=1>) (provided to help you complete MyProfiler.java - do NOT SUBMIT)

### Other files you will need to create and submit with your code:

- **screenshot\_001.png** (screenshot relevant parts of java mission control - DO SUBMIT)
- **screenshot\_002.png** (screenshot relevant parts of java mission control - DO SUBMIT)

### Hints:

- You can create the headers for all unimplemented methods specified in the interface, by clicking 'add unimplemented methods' in the suggestion given by eclipse.
- Also, remember to add appropriate private variables that are necessary for implementing the required methods

## MyProfiler

This is the program that you will profile to determine relative performance between your HashTable and Java's TreeMap structures. The program must perform a "bunch" of inserts, lookups, and removes to the data structures.

Keep in mind, you are trying to figure out which data structure performs best. Because modern computers are so fast, you will likely need to add a lot of items to see enough difference.

Given the complexity analysis of different lookups (single item vs range of values) for the two data structures is different, you will also want to experiment with different lookup operations. Consider too how to lookup many individual values, and many different ranges of values from each structure.

## Be scientific and iterative

<https://www.sciencebuddies.org/science-fair-projects/science-fair/steps-of-the-scientific-method> (<https://www.sciencebuddies.org/science-fair-projects/science-fair/steps-of-the-scientific-method>)

1. Make a hypothesis.
2. Design an experiment (a.k.a. write code, a single test, or set of tests).
3. Run your experiment (code) and record the results.
4. Learn how to interpret your results.
5. Compare your results against your predictions.
6. How well do you understand your data? your results? Can you predict results?
7. Repeat above to collect more data and refine your hypotheses and come to some conclusions.
8. Record your conclusions.

# Java Flight Recorder

Read [Oracle Java Flight Recorder \(https://canvas.wisc.edu/courses/202692/pages/oracle-java-flight-recorder\)](https://canvas.wisc.edu/courses/202692/pages/oracle-java-flight-recorder)

To run the sample profiler (SampleProfileApplication.java), make sure to edit the run configurations of this class in Eclipse as described in [Oracle Java Flight Recorder \(https://canvas.wisc.edu/courses/202692/pages/oracle-java-flight-recorder\)](https://canvas.wisc.edu/courses/202692/pages/oracle-java-flight-recorder).

# Java Mission Control

Now, you are ready to view and start analyzing the results of your program. Read about and use [Oracle Java Mission Control \(https://canvas.wisc.edu/courses/202692/pages/oracle-java-mission-control\)](https://canvas.wisc.edu/courses/202692/pages/oracle-java-mission-control) to do this.

## Specifications

- **Define (code) the HashTableTest class**
  - add your own tests in addition to ours to get you started
  - start small
    - get the easy tests and implementation working first
- **Define (code) the HashTable class**
  - should be able to instantiate with generic Key, Value pairs
  - must implement operations described in the provided **DataStructureADT** interface in an efficient way ( $O(1)$ ).
  - may use any of the following for internal data structure(s) as you like: arrays, ArrayList, LinkedList, a new node type, etc
  - must NOT USE Java's HashTable, HashMap, or TreeMap types to implement your HashTable (ask if you are not sure)
  - must handle edge or corner cases (first, last, empty, full - must expand)
  - must handle collisions (you can choose how, but must be one of the ways presented in lecture)
  - must handle resizing (we will set the *initial capacity* "table size" and add enough to cause re-sizing)
  - must document design choices that you make for hashing and collision resolution in your **HashTable** implementation (see comments and document there)
- **[Recommended] Try Test-Driven Development (TDD)**
  1. get all classes to compile (hint: add stubs for unimplemented methods to **HashTable** class)

2. choose a feature or method of **DataStructureADT** or **HashTableADT** to implement in your **HashTable** class.
3. write a test for it in **HashTableTest**
4. run the test (from a Linux command line: **make junit5** )
5. see that your test fails as expected
6. add code to your **HashTable** class to implement the feature so that it passes the test
7. run the test(s) again
8. see that your tests pass as expected
9. repeat steps 1-8 for all required functionality of your class

- (<https://canvas.wisc.edu/courses/202692/pages/professional-source-code>) **Write the**

### **MyProfiler** class

- The **MyProfiler** class contains methods for inserting and retrieving data from the hashtable and tree map
- You are required to instantiate **MyProfiler** class in the main method with the appropriate data type
- In the insert method, you will need to insert into both the hash table and tree map
- Similarly in the retrieve method, you need to get from both the hash table and the tree map
- The profile class takes one argument **<num\_elements>**, you need to insert and retrieve these many numbers of elements from both hash table and tree map
- Instructions to perform the profiling are provided here: [Oracle Java Flight Recorder](https://canvas.wisc.edu/courses/202692/pages/oracle-java-flight-recorder) (<https://canvas.wisc.edu/courses/202692/pages/oracle-java-flight-recorder>)
- Instructions to view and analyze results are provided here: [Oracle Java Mission Control](https://canvas.wisc.edu/courses/202692/pages/oracle-java-mission-control) (<https://canvas.wisc.edu/courses/202692/pages/oracle-java-mission-control>)
- Once, you complete the profiling, answer the questions asked in the file: **conclusions.txt**
  - Focus on the **Memory** usage and **Method profiling** options in the drop down for "**Java Application**" in the left menu of Java mission control. You should be able to determine which classes and then which methods are using the most resources whether that is space (bytes) or CPU cycles (counts).
  - Then, try to answer the questions. You may need to edit your My\_Profiler and HashTable classes re-profile to see understand your results better.
    - Consider how changes that you make in your code affects the results.
    - ProTip: Save each jfr file with new names and document your experiments so that you can compare the usage for each configuration or profiler code you try.
  - Based on your figures, you can tell if the time being spent in HashTable.insert is less than, similar to, or greater than the time spent in TreeMap.put. Does that make sense from what you know about HashTables and TreeMap (Red-Black Tree). If so, you should be able to explain why. If it doesn't make sense, you may wish to investigate that and see if you can improve the performance of your HashTable. (You can not improve the TreeMap).

- Generate images to support your conclusions. You can get points for conclusions based on your results even if those results are not consistent with what we should expect when profiling HashTables against TreeMap. To get full credit, you must also pass GradeScope performance tests and have an efficient HashTable.

## Steps

1. Read the entire assignment.
2. Review the grading rubric. Note: the rubric is subject to some changes.
3. Use [Development Environment of your choice](https://canvas.wisc.edu/courses/202692/pages/development-environment-of-your-choice)  
(<https://canvas.wisc.edu/courses/202692/pages/development-environment-of-your-choice>)
4. Create a **p4** project folder for your work.
5. Copy the provided source files into your project folder and refresh it.
6. Get the file **HashTable.java** to compile (add unimplemented methods).
7. Run the SampleProfilerApplication from Eclipse as instructed above
8. Complete **MyProfiler.java** the code and complete the testing assignment (review Specifications and assignment for more details)

Submit your work to: [p4 Hash Table](https://canvas.wisc.edu/courses/202692/assignments/833517)

(<https://canvas.wisc.edu/courses/202692/assignments/833517>)