# Practical 9

COS212

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Deadline: 09/06/2023 at 23:59

Marks: 200

## 1 General instructions:

- This assignment should be completed individually; no group effort is allowed.
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of Java's built-in data structures. Doing so will result in a mark of zero. You may only make use of native arrays where applicable. If you require additional data structures, you must implement them yourself.
- If your code does not compile, you will be awarded a zero mark. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- All submissions will be checked for plagiarism.
- Read the entire assignment before you start coding.
- You will be afforded five upload opportunities.
- Make sure your IDE did not create any packages or import any libraries which are not allowed.

# 2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with permission) and copying material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm. **If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.** Also, note that the OOP principle of code reuse does not mean you should copy and adapt code to suit your solution.

# 3 Outcomes

Upon completing this assignment, you will have implemented the following:

- hashing functions to be used in a hashmap.
- A hashmap class.

# 4 Background

Hashmaps are datastructures which are optimised for searching for data. An ideal hashmap has a lookup complexity of O(1). This is most often not achieved in practice because of imperfect hashing functions, but there are solutions to try and work around this. To keep this practical simple, we will be implementing a Hashlist, which simply means we will only be storing the keys inside our structure instead of key-value pairs.

# 5 Tasks

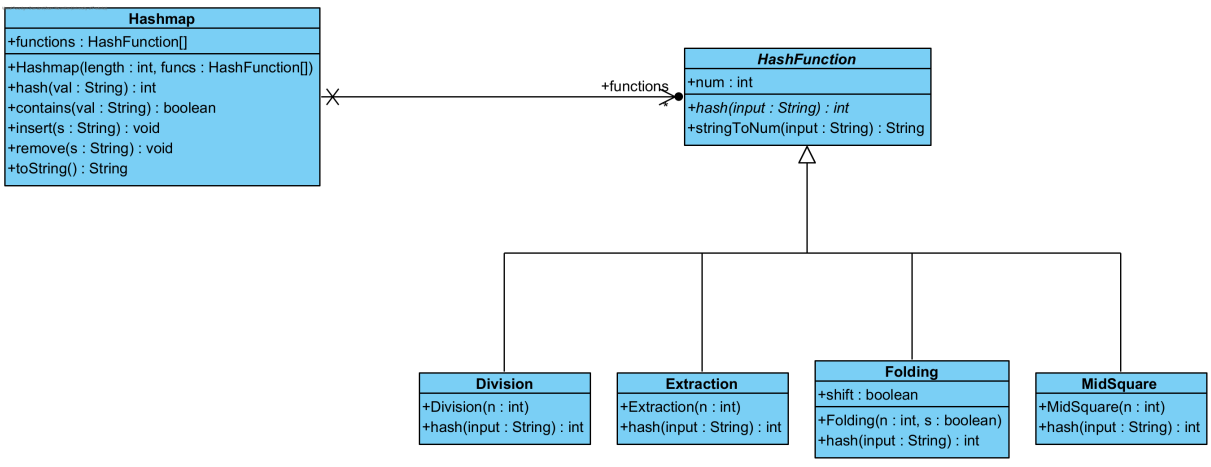


Figure 1: UML

## 5.1 HashFunction

- This class has been given to you.
- It is the parent class for the hash functions.
- The individual classes will use the num variable in their own way.
- The stringToNum() function takes in a String and then returns another String. The returned String only consists of numbers, and you may assume that it can always be converted to a valid Long.
- Note that casting from the Long to the int class isn't allowed. Take a look at `https://docs.oracle.com/javase/8/docs/api/java/lang/Long.html` for the list of functions that can be used on a Long.

### 5.1.1 Division

- Functions
  - Division(n:int)
    - This is the constructor for the class. It sets the num variable to the passed-in parameter.
  - hash(input:String):int
    - Take the input parameter and call stringToNum on it.
    - Convert this String to a long.
    - Return the remainder when the passed-in value is divided by num.
    - You may assume that num will not be 0.

### 5.1.2 Extraction

- Functions
  - Extraction(n:int)
    - This is the constructor for the class. It sets the num variable to the passed-in parameter.
  - hash(input:String):int
    - Take the input parameter and call stringToNum on it.
    - This should return the last *num* numbers in the String.
    - Thus, if num = 3, you should return the last three numbers.
    - You may assume that num will be larger than 0 but small enough that the substring can be converted to an int.
    - If the String returned from stringToNum, is shorter than num, then return that whole String.

### 5.1.3 Folding

- Members
  - shift:boolean
    - This boolean is used to determine which type of folding will be used.
    - If shift is True, then you should use Shift folding.
    - If shift is False, then you should use Boundary folding.
- Functions
  - Folding(n:int,boolean s)
    - This is the constructor for the class. It sets the num variable to the passed-in parameter.
  - hash(input:String):int
    - Call stringToNum on the passed-in parameter. Pad this with "0" until the length of this string is a multiple of num.(*i.e. add O's to the end of the string until it can be evenly split.*)
    - The num variable is the length of every partition of the string. The partitions are then converted to ints and then added together and returned.
    - If boundary folding is used, then every second partition must first be reversed and then converted to an int.

### 5.1.4 MidSquare

- Functions
  - MidSquare(n:int)
    - This is the constructor for the class. It sets the num variable to the passed-in parameter.
  - hash(input:String):int
    - Use stringToNum and save it inside a Long. Square this value and convert it to a String.
    - If the length of this String is less than or equal to num then return the value of this String.
    - Otherwise use this String to extract the middle. If a valid midpoint does not exist, then add a "0" at the end of the String.
    - A midpoint is defined as follows :
      - It must be a substring of length num.
      - The number of characters to the left and right of the substring should be equal.

## 5.2 Hashmap

- Members
  - functions:HashFunctions[]
    - This is an array of hash functions.
    - This will be passed in the constructor and will be used to hash data.
    - The array will not contain null values.
  - Create your own member here.
    - As explained in the background section, to simplify the prac, we won't be saving key-value pairs. Instead, we will only save the keys, which will be Strings.
    - Note that we will be implementing Chaining; thus, you need a datatype that allows for this. You are free to choose any data structure you want since you must implement your functions based on what you choose.
    - The hashmap will have a fixed length which will be set in the constructor.
    - The indexes will not be sorted. Thus if a collision occurs, the new String should be inserted at the back of that index.
    - *Hint: a 2D String array or an array of linked lists would be the easiest to implement*
- Functions
  - Hashmap(length:int,funcs:HashFunction[])
    - This is the constructor for the hashmap. Set the functions member to the passed-in parameter.
    - Use the length parameter to initialise the data structure that you will use to store the keys. You may assume this number is larger than 0.
  - hash(val:String):int
    - This should calculate the hash for the passed-in parameter.
    - Call the hash function of every function inside the functions member and then add all of these results together.
    - Before returning, take the modulus with the length of the hashmap.
  - contains(val:String):boolean
    - Returns True if the passed-in parameter is inside the Hashmap and returns False otherwise.
  - insert(s:String):void
    - If the passed-in parameter is already inside the Hashmap, then do nothing.
    - Add the passed-in parameter to the hashmap. If a collision occurs, then add the passed-in parameter to the back of the chain at that index.
  - remove(s:String):void
    - This function removes the passed-in parameter from the Hashmap.

- toString():String
  - This returns a String representation of the Hashmap.
  - The String starts with "[" and ends with "]".**WITHOUT A NEWLINE**
  - The indexes are separated with ";".
  - If an index had a collision and multiple values are chained together these should be separated with ",". Note there are no spaces.
  - If an index is empty then nothing is added for it.
  - Thus if we call toString on an empty hashmap with a length of 10 the result is: [;;;;;;;;;]

# 6 Submission instructions

Do not submit any custom function classes.
Your code should be able to be compiled and run with the following commands:
**javac *.java**
**java Main**

Once you are satisfied that everything is working, you must create an archive of all your Java code into one archive called uXXXXXXXX.{tar.gz/tar/zip} where X is your student number. Submit your code for marking under the appropriate link before the deadline. **Make sure your archive consists of only java files and no folders or subfolders.**

Please ensure that you thoroughly test your code before submitting it. Just because your code runs with local testing does not mean that your code works for every situation. **DO NOT USE FITCH FORK AS A DEBUGGER**

# 7 Submission checklist

- Division.java
- Extraction.java
- Folding.java
- HashFunction.java
- Hashmap.java
- MidSquare.java

# 8 Allowed imports

No imports are allowed.