Protokoll – Hashtabelle

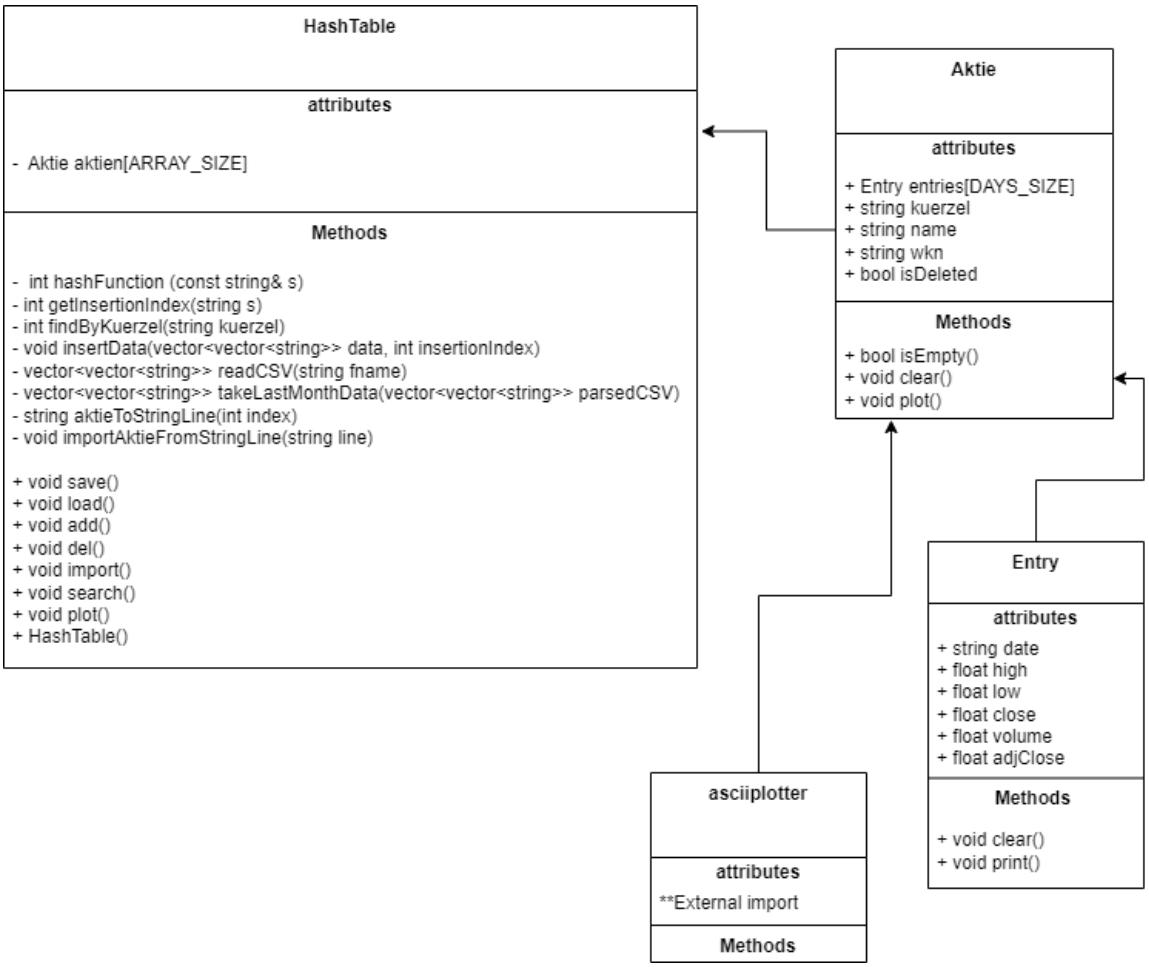
Hashfunction

```
int HashTable::hashFunction(string kuerzel) {
    int hash_val = 0;
    int n = s.length();
    for (int i = 0; i < n; i++) {
        // static_cast<int> to get the ASCII number of the char s[i]
        hash_val += static_cast<int>(kuerzel[i]) * pow(31, n-1-i);
        }
    // Take the modulus to restrict the values from 0 to 1092
    return hash_val%ARRAY_SIZE;
}
```

The hash function takes a Kuerzel and returns the corresponding hash value. To do so, it calculates the weighted sum of the ASCII symbols within the word. The resulting value will be taken modulo 1093 to always get a valid array-index, which needs to be in the range (0-1092).

The main class, the HashTable class, contains 1093 Aktien. The number 1093 was chosen because the requirement is to cover 1000 Aktien und 1093 is the following prime number after 1000.

Structs and Classes



- Entry: Represents one row in the csv-file containing the date and different values
- Aktie: Contains a kürzel, a name, a WKN, an array with 30 Entries and a isDeleted flag.
- Hashtabelle: Contains a aktien array with 1093 Aktien. A Hashtabelle implements the following methods:
    - ADD: Prompts the user for a Kürzel, a name and a WKN and populates an Aktie (without any Entries) within the HashTable. To find the index where to insert the new Aktie, we first generate the hash value from the kuerzel and check if the aktien array at this position is free. If yes, we can put that Aktie to this index. If not, we use quadratic probing to find another slot.
    - IMPORT: This function takes a Kürzel and loads a .csv file named <Kürzel>.csv. It takes the last 30 rows of the CSV file and saves them as entries for the corresponding Aktie. If there is no entry for the corresponding Aktie, the command returns with a "not found" message.
    - SEARCH: Takes a Kürzel and prints the newest Entry for the Aktie. If there is no entry for the corresponding Aktie, the command returns with a "not found" message.
    - DELETE: Takes a Kürzel and sets the isDeleted flag for the corresponding Aktie with the value `true`. We do not actually remove the Aktie from the Array. The reason is that deleting would leave an empty Aktie, which can lead to premature stopping of the probing method when looking for entries. The isDeleted flag is also considered when adding elements: if there is collision with a deleted element, it is replaced with the new Aktie. In this way, we avoid littering the hash table with deleted Akties.
    - PLOT: Takes a Kürzel and plots all the Entries using an included C++ plotting library (https://github.com/joehood/asciiplotter). If there is no entry for the corresponding Aktie, the command returns with a "not found" message.
    - SAVE: Saves all the Aktien in a text file, where each row represents one Aktie and all the values are separated by a comma.
    - LOAD: Loads Aktien from a File with the same format as the result of the SAVE file.
    - QUIT: Ends the Program.

Aufwandsabschätzung

Insertion: The linked list can always insert in the first position O(1), an array needs to first find a free place to insert the a new value O(n), the hash table uses the hash function and quadratic probing to find a place, at worst, when table is close to being full (füllgrad), finding an insertion place can take O(n) as the quadratic probing searches for an insertion place.

Aufwand	**Linked List**	**Array**	**Hashtable**
Best Case	O(1)	O(1)	O(1)
Average case	O(1)	O(n)	O(1)
Worst case	O(1)	O(n)	O(n)

Successful Search: The linked list searches through the list in order until finding the target, the array does similarly, the hash table uses the hash function to find them faster. On average, the hash table will do better O(1), but the worst cases of all these is of order O(n), that happens with a higher füllgrad.

Aufwand	**Linked List**	**Array**	**Hashtable**
Best Case	O(1)	O(1)	O(1)
Average Case	O(n)	O(n)	O(1)
Worst case	O(n)	O(n)	O(n)

Unsuccessful search: In unsuccessful search, both the linked list and the array have to transverse all stored values, whereas the hashtable on average will do the task in O(1).

Aufwand	**Linked List**	**Array**	**Hashtable**
Best Case	O(n)	O(n)	O(1)
Average Case	O(n)	O(n)	O(1)
Worst Case	O(n)	O(n)	O(n)

Delete: deletion consists first of a search, with an additional operation. This additional operation for the linked list is to deattach the node and connect the endpoints O(1), where the array would just empty the entry O(1), and our implementation of the HashTable, would set the isDelete flag to `true` O(1). So for all these cases, the cost of deletion for these three methods is the same one as for searching.