# Java Collections API: Symbol Tables

## 1 HashMap

In Chapter 3 of Sedgewick, we introduced *symbol tables*, a generic way to associate keys with values, and delved into how they can be implemented in various ways. In practice, most symbol tables are implemented with *hash tables* (Section 3.4), so much so that the term hashtable is synonymous with symbol table for most people.

In the Java API, the terminology is slightly different. Instead of using the term symbol table, Javas uses *map*, after the mathematical idea of a function that associates pairs between a domain and a range. Hence, the standard Java implementation of a symbol table is called a **HashMap**. An overview of the hash table APIs for Sedgewick versus Java is shown in Table 1. From a usage standpoint, the difference is very slight: contains is renamed containsKey, and keys is renamed keySet. There are also differences in parameter types but this will be transparent to the developer. Note that to use this collection, you will need to import the **java.util.HashMap** package.

| Textbook method | Description | HashMap method |
|---|---|---|
| void put(Key key, Value val) | put key-value pair into the table | Value put(Key key, Value value) |
| Value get(Key key) | get value paired with key | Value get(**Object** key) |
| void delete(Key key) | remove key (and its value) from table | Value remove(**Object** key) |
| boolean contains(Key key) | is there a value paired with key? | boolean **containsKey**(**Object** key) |
| boolean isEmpty() | is the table empty? | boolean isEmpty() |
| int size() | number of key-value pairs | int size() |
| Iterable<Key> keys() | all keys in the table, in sorted order | **Set**<Key> **keySet**() |

Table 1: API comparison for SymbolTable from Algorithms by Sedgewick and Wayne, versus Java's HashMap. Differences with using HashMap are highlighted with bold.

### 1.1 Java API Example

The following code examples compare the usage of hash tables between Sedgewick and the Java API:

**Algorithm 1** Example of a hashtable using Sedgewick's API.

```
public class HashtableTest {
    public static void main(String[] args) {
        SymbolTable<String, Integer> ST = new HashtableST<>();
        ST.put("SER216", 48);
        ST.put("SER222", 80);
        ST.put("SER232", 78);
        ST.put("SER322", 60);
        ST.put("SER332", 24);
        System.out.println(ST.get("SER232"));
        ST.remove("SER216");
        if(ST.contains("SER216"))
            System.out.println(ST.get("Warning: SER216 still exists!"));

        System.out.println(ST.get("Courses:"));
        for(String k : ST.keys())
          System.out.println(k + " : " + ST.get(k));
    }
}
```

**Algorithm 2** Example of a hashtable using Java's API.

```
import java.util.HashMap;

public class HashtableTest {
    public static void main(String[] args) {
        HashMap<String, Integer> ST = new HashMap<>();
        ST.put("SER216", 48);
        ST.put("SER222", 80);
        ST.put("SER232", 78);
        ST.put("SER322", 60);
        ST.put("SER332", 24);
        System.out.println(ST.get("SER232"));
        ST.remove("SER216");
        if(ST.containsKey("SER216"))
            System.out.println(ST.get("Warning: SER216 still exists!"));

        System.out.println(ST.get("Courses:"));
        for(String k : ST.keySet())
          System.out.println(k + " : " + ST.get(k));
    }
}
```