



# Week 17: Map

---

4009 DATA STRUCTURES & ALGORITHM (DSA)

# Learning Outcomes

---

- ❖ Create, manipulate and understand the data structure of Map
- ❖ Create a simple student directory app using Map data structure
- ❖ Define and understand the Map data structure
- ❖ Implement a Map ADT in Java
- ❖ Manipulate Map coding in Java and use of Java's inbuilt methods

# Content Overview

---

- ❖ Map ADT
- ❖ Application of Map ADT in the real world
- ❖ The list abstract data type (ADT)
- ❖ Example of map operations
- ❖ `java.util.Map` class
- ❖ A simple map implementation
- ❖ Student Directory App using Map data structure

# Data Structure Trade-offs

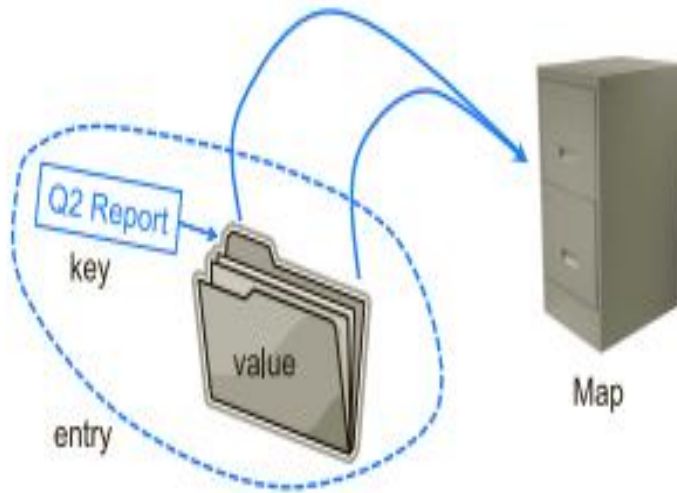
Data structures	Advantages	Disadvantages
<b>Array</b>	Quick insertion, very fast access to index known	Slow search, slow deletion, fixed size
<b>Ordered Array</b>	Quicker Search than unsorted array	Slow insertion and deletion, fixed size
<b>Stack</b>	Provide Last-in First-out access	Slow access to other items
<b>Queue</b>	Provide First-in First out access	Slow access to other items
<b>Linked List</b>	Quick Insertion, Quick deletion	Slow search
<b>Binary Tree</b>	Quick search, insertion, deletion (if tree balanced)	Deletion algorithm is complex
<b>Hash Table</b>	Very fast access if key known, fast insertion	Slow deletion, access slow if key not known, inefficient memory usage
<b>Heap</b>	Fast insertion, deletion, access to large item	Slow access to other items
<b>Graphs</b>	Model real world situations	Some algorithms are slow and complex

# The Map ADT (1/2)

---

- A map is an abstract data type designed to efficiently store and retrieve values based upon a uniquely identifying search key for each.
- A map stores key-value pairs  $(k, v)$ , which we call entries, where  $k$  is the key and  $v$  is its corresponding value.
- Keys are required to be unique, so that the association of keys to values defines a mapping.
- Figure 10.1 provides a conceptual illustration of a map using the file-cabinet metaphor.

# The Map ADT (2/2)



**Figure 10.1:** A conceptual illustration of the map ADT. Keys (labels) are assigned to values (folders) by a user. The resulting entries (labeled folders) are inserted into the map (file cabinet). The keys can be used later to retrieve or remove values.

- Maps are also known as associative arrays, because the entry's key serves somewhat like an index into the map, in that it assists the map in efficiently locating the associated entry. However, unlike a standard array, a key of a map need not be numeric, and it does not directly designate a position within the structure.

# Application of Maps in the real world

---

- A university's information system relies on some form of a student ID as a key that is mapped to that student's associated record (such as the student's name, address, and course grades) serving as the value.
- The domain-name system (DNS) maps a host name, such as `www.wiley.com`, to an Internet-Protocol (IP) address, such as `208.215.179.146`.
- A social media site typically relies on a (nonnumeric) username as a key that can be efficiently mapped to a particular user's associated information.
- A company's customer base may be stored as a map, with a customer's account number or unique user ID as a key, and a record with the customer's information as a value. The map would allow a service representative to quickly access a customer's record, given the key.

# The list abstract data type (ADT)

- A map stores a collection of objects, it should be viewed as a collection of key-value pairs. As an ADT, a map  $M$  supports the following methods:
  - **size( )**: Returns the number of entries in  $M$ .
  - **isEmpty( )**: Returns a boolean indicating whether  $M$  is empty.
  - **get(k)**: Returns the value  $v$  associated with key  $k$ , if such an entry exists; otherwise returns null.
  - **put(k, v)**: If  $M$  does not have an entry with key equal to  $k$ , then adds entry  $(k,v)$  to  $M$  and returns null; else, replaces with  $v$  the existing value of the entry with key equal to  $k$  and returns the old value.
  - **remove(k)**: Removes from  $M$  the entry with key equal to  $k$ , and returns its value; if  $M$  has no such entry, then returns null.
  - **keySet( )**: Returns an iterable collection containing all the keys stored in  $M$ .
  - **values( )**: Returns an iterable collection containing all the values of entries stored in  $M$  (with repetition if multiple keys map to the same value).
  - **entrySet( )**: Returns an iterable collection containing all the key-value entries in  $M$ .



# Example of Map operations

- The following table shows a series of map operations and their effects

<b>Method</b>	<b>Return Value</b>	<b>Map</b>
isEmpty()	true	{}
put(5,A)	null	{(5,A)}
put(7,B)	null	{(5,A), (7,B)}
put(2,C)	null	{(5,A), (7,B), (2,C)}
put(8,D)	null	{(5,A), (7,B), (2,C), (8,D)}
put(2,E)	C	{(5,A), (7,B), (2,E), (8,D)}
get(7)	B	{(5,A), (7,B), (2,E), (8,D)}
get(4)	null	{(5,A), (7,B), (2,E), (8,D)}
get(2)	E	{(5,A), (7,B), (2,E), (8,D)}
size()	4	{(5,A), (7,B), (2,E), (8,D)}
remove(5)	A	{(7,B), (2,E), (8,D)}
remove(2)	E	{(7,B), (8,D)}
get(2)	null	{(7,B), (8,D)}
remove(2)	null	{(7,B), (8,D)}
isEmpty()	false	{(7,B), (8,D)}
entrySet()	{(7,B), (8,D)}	{(7,B), (8,D)}
keySet()	{7, 8}	{(7,B), (8,D)}
values()	{B, D}	{(7,B), (8,D)}

# A simple version of the Map Interface

---

```
1  public interface Map<K,V> {  
2      int size();  
3      boolean isEmpty();  
4      V get(K key);  
5      V put(K key, V value);  
6      V remove(K key);  
7      Iterable<K> keySet();  
8      Iterable<V> values();  
9      Iterable<Entry<K,V>> entrySet();  
10 }
```

# java.util.Map class

---

- Like the other data structures, Java provides a class named java.util.Map that implements the data structure Map
- NOTE: before creating your program's class, you must import java.util.Map; class in order to use and manipulate Map in Java.

```
1  /* DSA Week 15 Lab 1 */
2
3  import java.util.HashMap;
4  import java.util.Map;
5
6  public class Week15Lab1 {
7
8      public static void main(String[] args) {
9          ..
10         ..
11         ..
12         ..
13     }
```

# A simple Map implementation

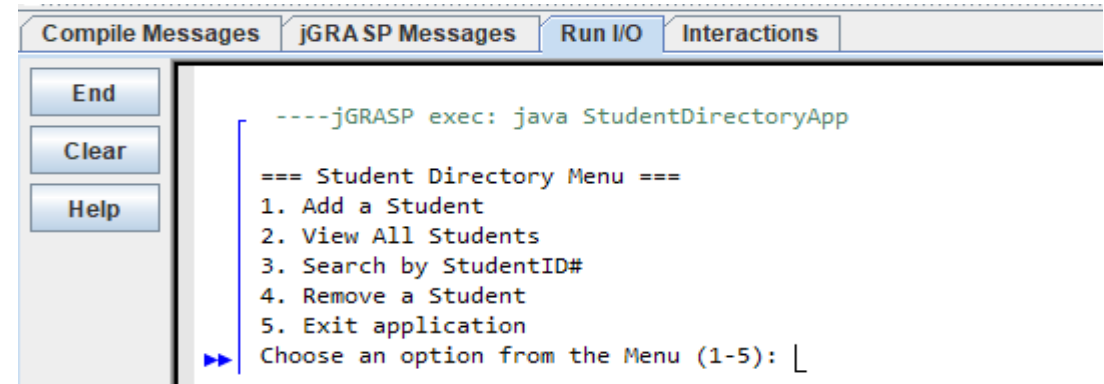
---

```
1  /* DSA Week 15 Lab 1 */
2
3  import java.util.HashMap;
4  import java.util.Map;
5
6  public class Week15Lab1 {
7
8      public static void main(String[] args) {
9          // Create a Map to store course codes with index keys
10         Map<Integer, String> itiCourses = new HashMap<>();
11
12         // Simulate "pushing" elements with incremental keys
13         itiCourses.put(0, "DIT");
14         itiCourses.put(1, "DHRM");
15         itiCourses.put(2, "DACC");
16         itiCourses.put(3, "DICT");
17
18         // Get the top element (the last inserted one)
19         int topKey = itiCourses.size() - 1;
20         String topElement = itiCourses.get(topKey);
21         System.out.println("Top element: " + topElement);
22
23         // Check if the map is empty
24         if (itiCourses.isEmpty()) {
25             System.out.println("Map is empty");
26         } else {
27             // Print the contents of the map
28             System.out.print("Map contents: " + itiCourses);
29         }
30     }
31 }
```

# Student Directory App using Map data structure

```
1  /* DSA Week 15 Final Lab */
2
3  import java.util.HashMap;
4  import java.util.Map;
5  import java.util.Scanner;
6
7  public class StudentDirectoryApp {
8
9      public static void main(String[] args) {
10         Scanner scanner = new Scanner(System.in);
11
12         //use map <hashmap> data structure to store data
13         Map<String, String> studentDirectory = new HashMap<>();
14
15         while (true) {
16             System.out.println("\n=== Student Directory Menu ===");
17             System.out.println("1. Add a Student");
18             System.out.println("2. View All Students");
19             System.out.println("3. Search by StudentID#");
20             System.out.println("4. Remove a Student");
21             System.out.println("5. Exit application");
22             System.out.print("Choose an option from the Menu (1-5): ");
23             int choice = scanner.nextInt();
24             scanner.nextLine();
25
26             switch (choice) {
27                 case 1:
28                     System.out.print("Enter a studentID#: ");
29                     String id = scanner.nextLine();
30                     System.out.print("Enter a student name: ");
31                     String name = scanner.nextLine();
32                     studentDirectory.put(id, name);
33                     System.out.println("Student added.");
34                     break;
35                 case 2:
36                     System.out.println("\nView All students:");
37                     for (Map.Entry<String, String> entry : studentDirectory.entrySet()) {
38                         System.out.println("ID: " + entry.getKey() + " | Name: " + entry.getValue());
39                     }
40                     break;
41                 case 3:
42                     System.out.print("Enter a studentID# to search: ");
43                     String searchId = scanner.nextLine();
44                     if (studentDirectory.containsKey(searchId)) {
45                         System.out.println("Student Name: " + studentDirectory.get(searchId));
```

```
46         } else {
47             System.out.println("Student not found.");
48         }
49         break;
50     case 4:
51         System.out.print("Enter a studentID# to remove: ");
52         String removeId = scanner.nextLine();
53         if (studentDirectory.remove(removeId) != null) {
54             System.out.println("Student removed.");
55         } else {
56             System.out.println("Student ID not found.");
57         }
58         break;
59     case 5:
60         System.out.println("Exiting program. Goodbye!");
61         scanner.close();
62         return;
63     default:
64         System.out.println("Invalid option. Try again.");
65     }
66 }
67 }
68 }
```



```
Compile Messages  jGRASP Messages  Run I/O  Interactions

End
Clear
Help

----jGRASP exec: java StudentDirectoryApp

=== Student Directory Menu ===
1. Add a Student
2. View All Students
3. Search by StudentID#
4. Remove a Student
5. Exit application
Choose an option from the Menu (1-5): |
```

# DSA Week 17 activities

---

## **Refer to print out materials.**

- This week, you are required to complete the questions and two labs.
  - Refer to the print out, answer all week 17 questions.
  - Refer to the print out, complete the two labs activities using the lab computers.

***Note:*** You can complete the activities in any order, however, make afford to complete and understand everything which prepares you for well for the Final Exam.