

Lecture 19: Sets and Maps

CS 2110

October 29, 2026

Today's Learning Outcomes

73. Describe the operations of the Set and Map ADTs, and utilize them in programs.

74. Compare the efficiency of Set implementations backed by arrays, sorted arrays, and (balanced) binary search trees.

57. Identify and define functional interfaces and implement them using lambda expressions.

The `Set<T>` ADT

A **set** is an *unordered* collection of *distinct* elements.

The `Set<T>` ADT is used to model a set of elements with type `T`.

Approach 1: Compose with `List`

This approach allows any type T to be stored. Since the elements may not be `Comparable`, we can't impose invariants on the element order.

`add()`:

`remove()`:

`contains()`:

`size()`:

`iterator()`:

Approach 2: Compose with sorted `ArrayList`

For "sorted" to make sense, we must impose the generic type bound `T extends Comparable<T>`.

`contains()`:

`add()`:

`remove()`:

`size()`:

`iterator()`:

Approach 3: Compose with a BST

Again, we need the generic type bound `T extends Comparable<T>`.

Additional Set Operations: Union

The **union** of two sets S and T is the set that contains all elements that belong to either S or T (or both).

Let's add a `union()` instance method to our `Set` implementations:

```
public Set<T> union(Set<T> other) { ... }
```



Coding Demo: Three Union Definitions



N = `this.size()`
M = `other.size()`

1: Naive iterative `add()`

2: Smarter iterative `add()`

3: Incorporating sorting

Additional Set Operations: Restriction

A **predicate** maps each element of a type T to a `boolean` value.

Recall: Java's `Predicate<T>` functional interface with method

```
public boolean test(T elem);
```



Coding Demo: Set Restriction



* This is a good review of functional interface syntax.*

Maps

A **Map** is an unordered collection of (**key**, **value**) pairs called its **entries**.

Each key in a map is associated with one value (i.e., belongs to one entry).

(Python's dictionary type is a map)

Choosing Keys and Values

Keys and values play different roles in a Map.

Keys	Values

To figure out which is which:

"Every _____ is paired with exactly one _____. "

"I want to use _____ to look up _____. "

The Map<K, V> ADT

Generic on both key type K and value type V

Modifying Entries:

Querying:

Iterating:

Practice with Map Operations

Given a `Map` associating the names of famous scientists (`String`) with their birthdays (`LocalDate`), write a method to print all the scientists with a particular birthday.

Map
`put(K key, V value): void`
`containsKey(K key): boolean`
`get(K key): V`
`remove(K key): V`
`keySet(): Set<K>`
`size(): int`

```
/** Prints the names of the scientists in `map` who were
 *  born on the same date (month and day) as `bday`. */
void haveBDay(Map<String, LocalDate> map, LocalDate bday) {




}
```

```
d.getMonth() == bday.getMonth && d.getDayOfMonth() == bday.getDayOfMonth()
```



Coding Demo: Implementing Maps



Like `Set`, but with one level of *indirection*

- Auxiliary `Entry` type (stored in underlying collection)
- Use `key` to search the collection
- Use entire `Entry` to meet the method post-condition once we get to the right place

Dynamic Priority Queues

Sometimes, we want to update the priority of an element while it is inside of a priority queue.

> E.g., In Dijkstra's algorithm (Lecture 23), we'll update the best-known distance to a node when we discover a new incoming path to it.

Heaps (alone) don't give us a good way to do this, since searching in a heap is inefficient.