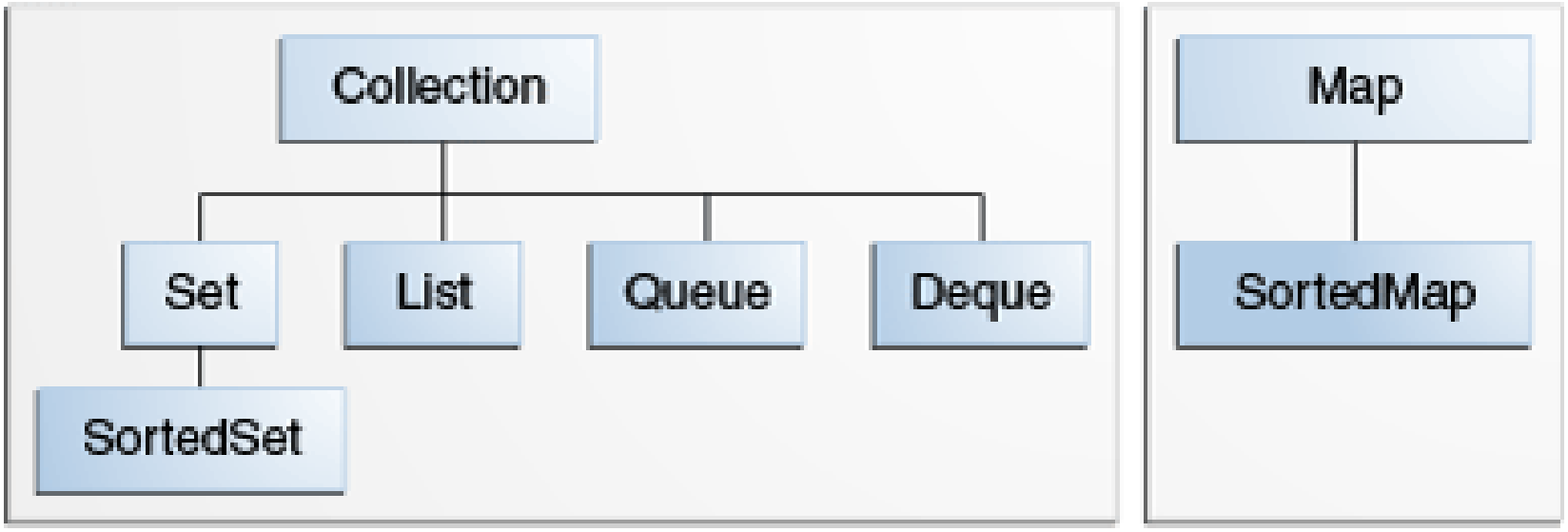
Java Collections

Java Collections Framework

- The Java language API provides many of the data structures from this class for you.
- It defines a “collection” as “an object that represents a group of objects”.
- It defines a collections framework as “a unified architecture for representing and manipulating collections, allowing them to be manipulated independent of the details of their representation.”

Collection interfaces

- The core collection interfaces encapsulate different types of collections.
- They are interfaces so they do not provide an implementation!



Collection<E>

- Collection — the root of the collection hierarchy
- Some types of collections allow duplicate elements, and others do not.
- Some are ordered and others are unordered.
- The Java platform doesn't provide any direct implementations of this interface but provides implementations of more specific subinterfaces, such as Set and List.

Set<E>

- Set — a collection that cannot contain duplicate elements.
- This interface models the mathematical set abstraction and is used to represent sets, such as the cards comprising a poker hand, the courses making up a student's schedule, or the processes running on a machine.

List<E>

- List — an ordered collection (sometimes called a *sequence*).
- Lists can contain duplicate elements.
- The user of a List generally has precise control over where in the list each element is inserted and can access elements by their integer index (position).

Queue<E>

- Queue — a collection used to hold multiple elements prior to processing.
- Besides basic Collection operations, a Queue provides additional insertion, extraction, and inspection operations.
- Queues typically, but do not necessarily, order elements in a FIFO (first-in, first-out) manner.

Deque<E>

- Deque — a collection used to hold multiple elements prior to processing.
- Deques can be used both as FIFO (first-in, first-out) and LIFO (last-in, first-out). In a deque all new elements can be inserted, retrieved and removed at both ends.

Map<K,V>

- Map — an object that maps keys to values.
- A Map cannot contain duplicate keys; each key can map to at most one value.
- If you've used a Hashtable, you're already familiar with the basics of Map.

SortedSet<E>

- SortedSet — a Set that maintains its elements in ascending order.
- Several additional operations are provided to take advantage of the ordering.
- Sorted sets are used for naturally ordered sets, such as word lists and membership rolls.

SortedMap<K,V>

- SortedMap — a Map that maintains its mappings in ascending key order.
- This is the Map analog of SortedSet.
- Sorted maps are used for naturally ordered collections of key/value pairs, such as dictionaries and telephone directories.

General-purpose Implementations					
Interfaces	**Implementations**				
	Hash table	**Resizable array**	**Tree (sorted)**	**Linked list**	**Hash table + Linked list**
Set	HashSet		TreeSet (sorted)		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap (sorted)		LinkedHashMap

Note the naming convention

LinkedList also implements queue and there is a PriorityQueue implementation (implemented with heap)

implementations

- Each of the implementations offers the strengths and weaknesses of the underlying data structure.
- What does that mean for:
  - Hashtable
  - Resizable array
  - Tree
  - LinkedList
  - Hashtable plus LinkedList
- **Think about these tradeoffs when selecting the implementation!**

Choosing the datatype

- When you declare a Set, List or Map, you should use Set, List or Map interface as the datatype instead of the implementing class. That will allow you to change the implementation by changing a single line of code!

```
import java.util.*;

public class Test {
    public static void main(String[] args) {
        Set<String> ss = new LinkedHashSet<String>();

        for (int i = 0; i < args.length; i++)
            ss.add(args[i]);

        Iterator i = ss.iterator();
        while (i.hasNext())
            System.out.println(i.next());
    }
}
```

```
import java.util.*;

public class Test {

	public static void main(String[] args)
	{
		//map to hold student grades
		Map<String, Integer> theMap = new HashMap<String, Integer>();

		theMap.put("Korth, Evan", 100);
		theMap.put("Plant, Robert", 90);
		theMap.put("Coyne, Wayne", 92);
		theMap.put("Franti, Michael", 98);
		theMap.put("Lennon, John", 88);

		System.out.println(theMap);
		System.out.println("-----------------------------------");
		System.out.println(theMap.get("Korth, Evan"));
		System.out.println(theMap.get("Franti, Michael"));

	}

}
```

algorithms

- The collections framework also provides polymorphic versions of algorithms you can run on collections.
  - Sorting
  - Shuffling
  - Routine Data Manipulation
    - Reverse
    - Fill copy
    - etc.
  - Searching
    - Binary Search
  - Composition
    - Frequency
    - Disjoint
  - Finding extreme values
    - Min
    - Max