Chapter 4 - Collections

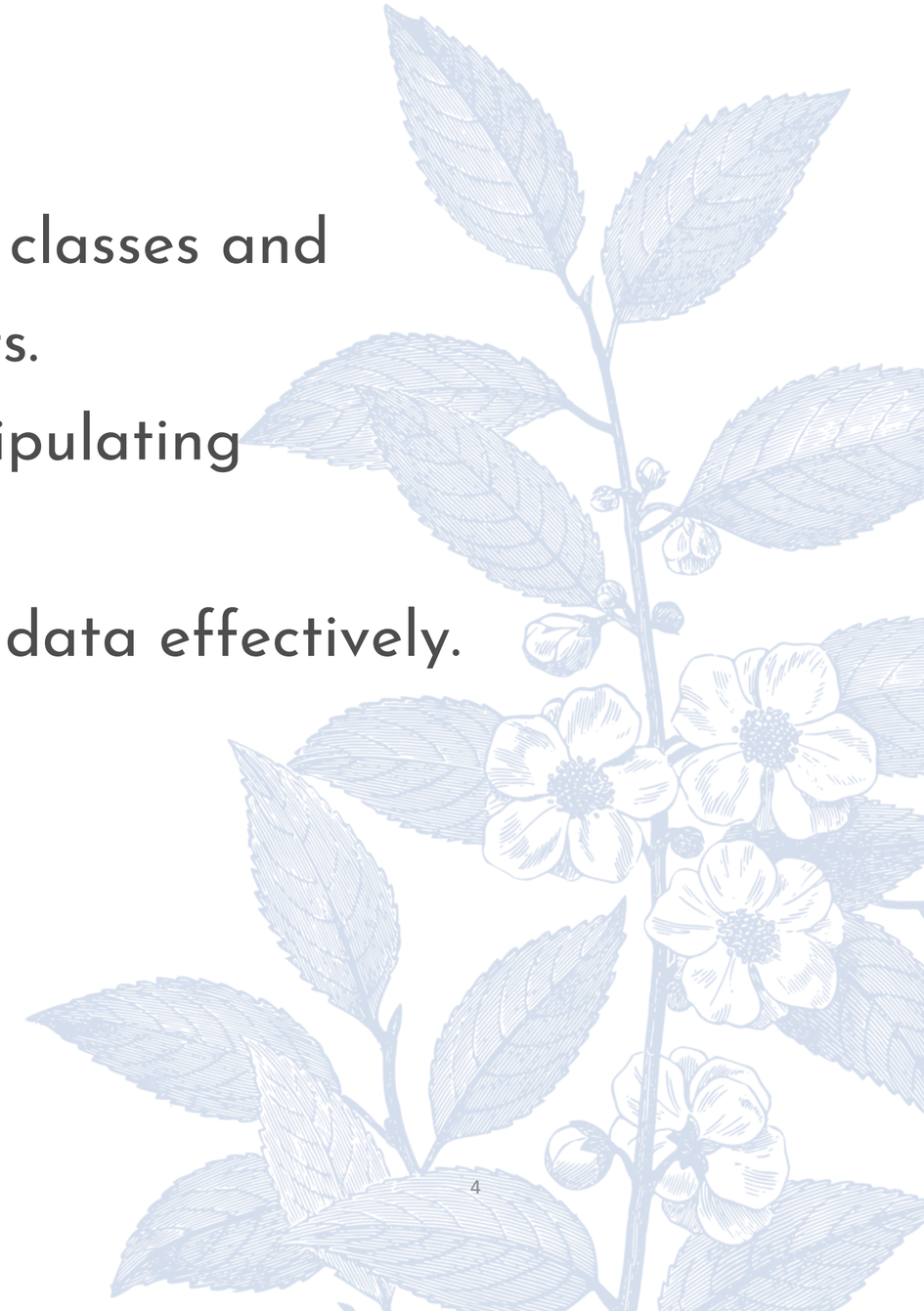
Chapter 4 - Collections

Collections

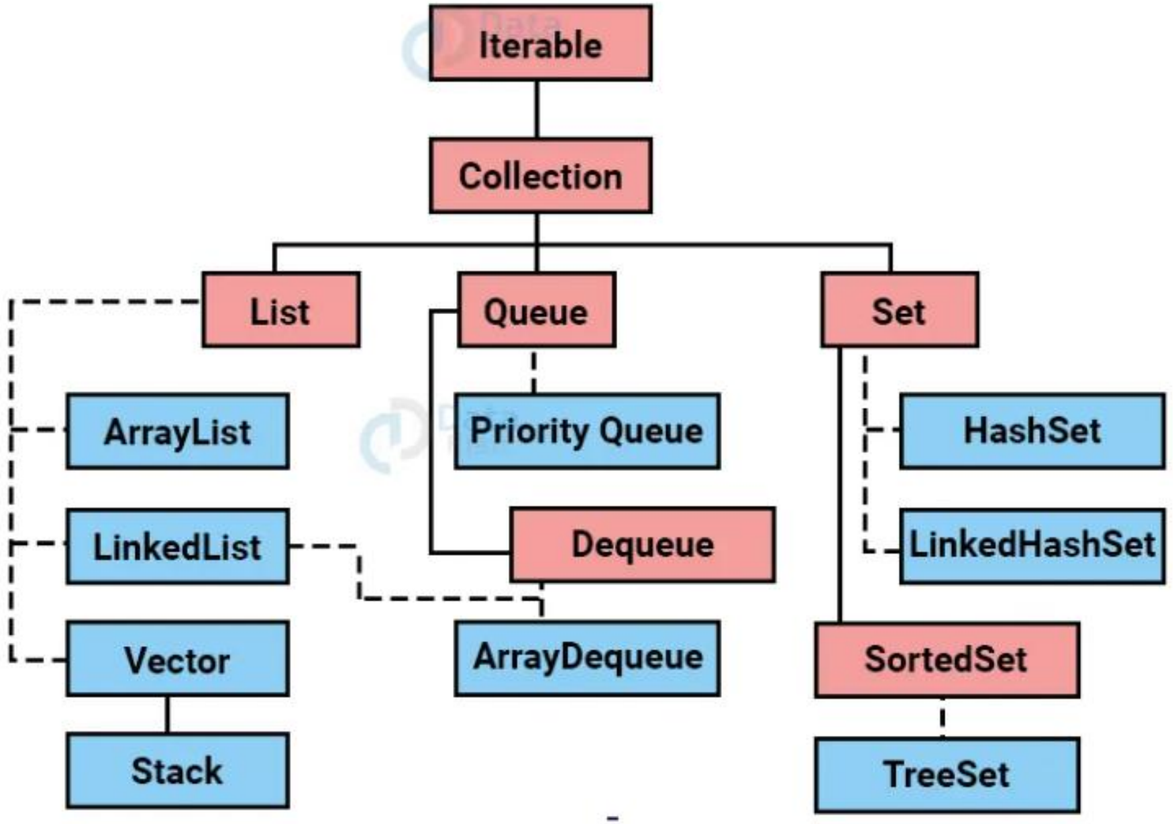
- ✓A collection is an object that groups multiple elements into a single unit
- ✓Very useful
  - store, retrieve and manipulate data
  - transmit data from one method to another
- ✓Collection interface(java.util.collection)
- ✓Map interface(java.util.Map)

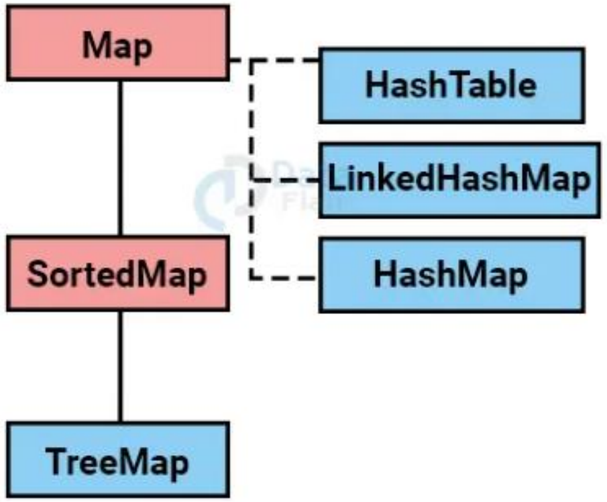
Collections Framework

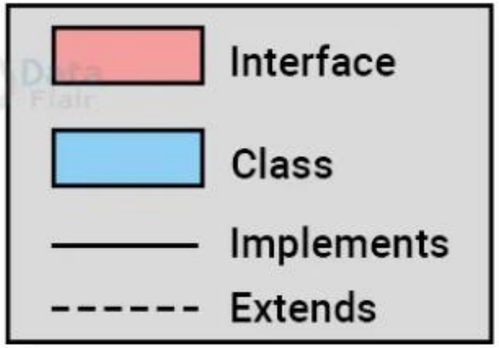
- ✓The Java Collections Framework is a library of classes and interfaces for working with collections of objects.
- ✓Unified architecture for representing and manipulating collections.
- ✓It is used in storing, maintaining and handling data effectively.
- ✓A collections framework contains three things
  - Interfaces
  - Implementations
  - Algorithms



Hierarchy of Collection Framework in Java







Collection Framework in Java

- ✓Interfaces: java.util.Collections. It consists of several important methods that the programmer uses in his day to day life. Some of these methods include add(), size(), remove(), etc.
- ✓Each and every other interface implements the java.util.Collection interface, for example, Set, Queue, etc.
- ✓The only interface that does not implement the collection interface but is part of the framework is the Map interface.

Interfaces in the Collection framework.

- ✓Collection - This is the root interface and is present at the top of the Collection hierarchy and allows us to work with a group of objects.
- ✓List - This interface extends the Collection interface and is used to store data in the form of a list. The object of List stores elements in an ordered form.
- ✓Set - This interface extends the Collection interface and handles a set of data with unique elements.
- ✓SortedSet - This interface extends the Set interface and is used to handle the set of elements that are sorted.

Interfaces in the Collection framework.

- ✓Map - This interface does not extend any other interfaces. It is used to map the data in the form of keys and values.
- ✓SortedMap - This interface extends the Map interface and is used to maintain the keys in ascending order.
- ✓Map.Entry - This is an inner class of the Map interface that is used to represent elements(Both keys and values) on a map.

The implemented classes

- ✓AbstractCollection - This class implements most of the Collection interfaces.
- ✓AbstractList - This class extends the AbstractCollecton and implements most of the list interfaces.
- ✓AbstractSequentialList - This class extends the AbstractList class. It is used to perform sequential access to a collection of elements rather than random access
- ✓LinkedList - This class is used to implement a linked list. This class also extends the AbstractList class.

The implemented classes

- ✓ArrayList - This class is used to create a dynamic and flexible array. It extends the AbstractList class.
- ✓Vector - Vector implements a dynamic array which means it can grow or shrink as required. They are very similar to ArrayList, but Vector is synchronized
- ✓AbstractSet - This class extends the AbstractCollection class and implements most of the Set interface.
- ✓HashSet - This class is used to work with Hash Tables. The class extends the AbstractSet.

The implemented classes

- ✓LinkedHashSet - This class allows iteration in insertion order and extends the HashSet class.
- ✓TreeSet - This class is used to implement the set stored in a tree. It extends the AbstractSet Class.
- ✓AbstractMap - This class implements most of the Map interfaces.
- ✓HashMap - This class is used to implement a hash table. It extends the AbstractMap class.
- ✓LinkedHashMap - This class is used to perform iteration in insertion order. This class extends the HashMap class.

The implemented classes

- ✓HashTable - It is similar to HashMap, but is synchronized.
- ✓TreeMap - The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.

Collection Interface

- ✓Defines fundamental methods
  - int size();
  - boolean isEmpty();
  - boolean contains(Object element);
  - boolean add(Object element); // Optional
  - boolean remove(Object element); // Optional
  - Iterator iterator();
- ✓These methods are enough to define the basic behavior of a collection
- ✓Provides an Iterator to step through the elements in the Collection

Iterator Interface

- ✓Defines three fundamental methods
  - Object next()
  - boolean hasNext()
  - void remove()
- ✓These three methods provide access to the contents of the collection
- ✓An Iterator knows position within collection
- ✓Each call to next() “reads” an element from the collection
  - Then you can use it or remove it

Example - SimpleCollection

```
List<Integer> nums = List.of(10, 5, 20, 25, 30, 45);
Iterator<Integer> its = nums.iterator();
```

List Interface

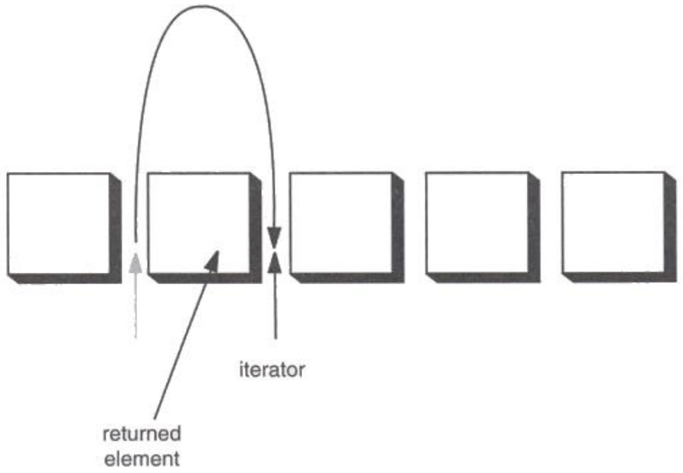
- ✓The List interface adds the notion of order to a collection
- ✓The user of a list has control over where an element is added in the collection
- ✓Lists typically allow duplicate elements
- ✓Provides a ListIterator to step through the elements in the list.

ListIterator Interface

- ✓Extends the Iterator interface
- ✓Defines three fundamental methods
  - void add(Object o) - before current position
  - boolean hasPrevious()
  - Object previous()
- ✓The addition of these three methods defines the basic behavior of an ordered list
- ✓A ListIterator knows position within list

Example - ListIterator

```
List<Integer> nums = List.of(10, 5, 20, 25, 30, 45);
ListIterator<Integer> its = nums.listIterator();
its.next();
its.next();
System.out.println(its.previous()); //?
```



List Implementations

- ✓ArrayList
  - low cost random access
  - high cost insert and delete
  - array that resizes if need be
- ✓LinkedList
  - sequential access
  - low cost insert and delete
  - high cost random access
- ✓Vector
  - Vector is synchronized

ArrayList overview

- ✓Constant time positional access (it's an array)
- ✓One tuning parameter, the initial capacity

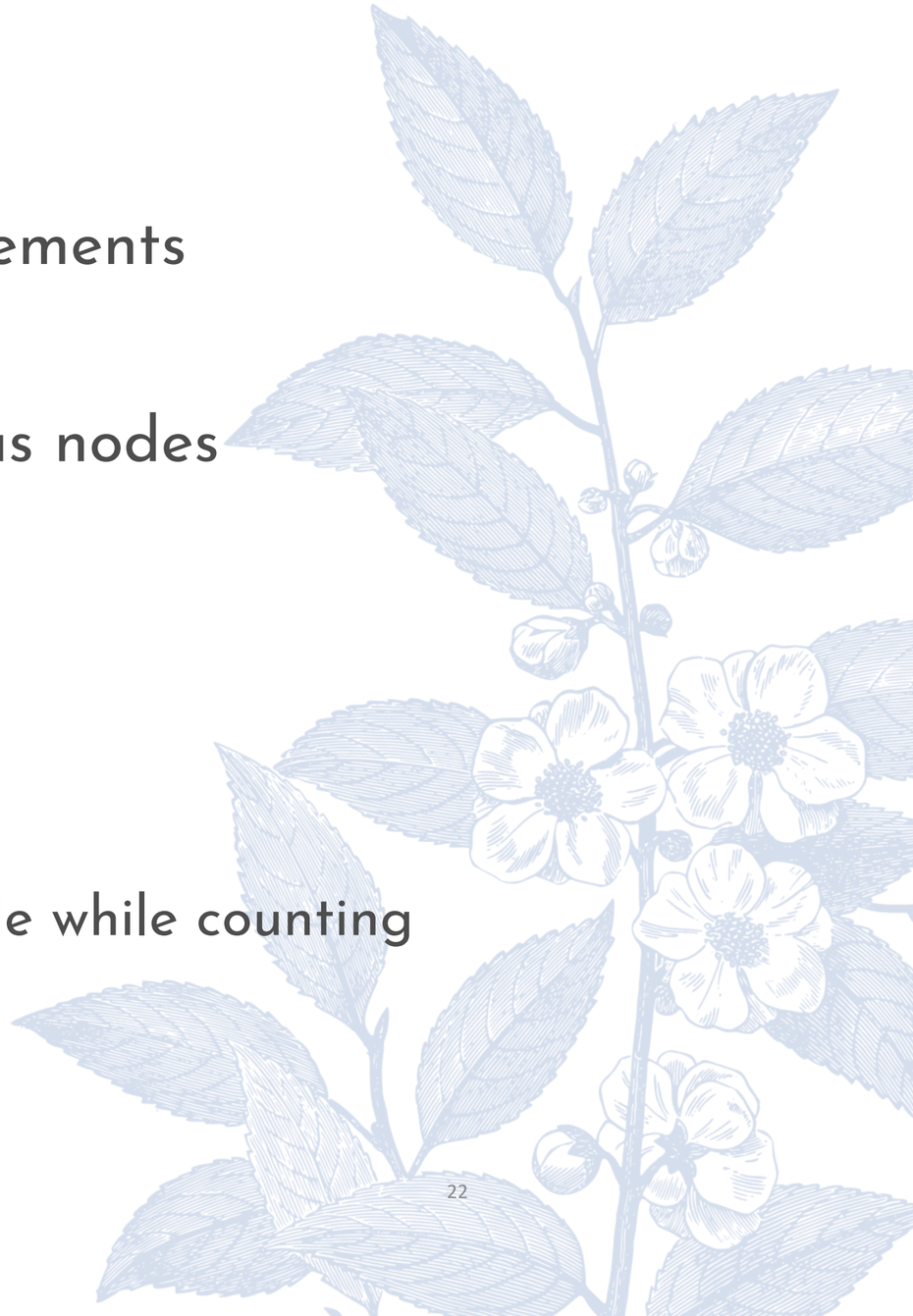
```
public ArrayList(int initialCapacity) {
    if (initialCapacity > 0) {
        this.elementData = new Object[initialCapacity];
    } else if (initialCapacity == 0) {
        this.elementData = EMPTY_ELEMENTDATA;
    } else {
        throw new IllegalArgumentException("Illegal Capacity: "+
                                           initialCapacity);
    }
}
```

ArrayList methods

- ✓The indexed get and set methods of the List interface are appropriate to use since ArrayLists are backed by an array
  - Object get(int index)
  - Object set(int index, Object element)
- ✓Indexed add and remove are provided, but can be costly if used frequently
  - void add(int index, Object element)
  - Object remove(int index)
- ✓May want to resize in one shot if adding many elements
  - void ensureCapacity(int minCapacity)

LinkedList overview

- ✓LinkedList does not use an array to store its elements
- ✓Stores each element in a node
- ✓Each node stores a link to the next and previous nodes
- ✓Insertion and removal are inexpensive
  - just update the links in the surrounding nodes
- ✓Linear traversal is inexpensive
- ✓Random access is expensive
  - Start from beginning or end and traverse each node while counting



LinkedList methods

- ✓The list is sequential, so access it that way
  - ListIterator listIterator()
- ✓ListIterator knows about position
  - use add() from ListIterator to add at a position
  - use remove() from ListIterator to remove at a position
- ✓LinkedList knows a few things too
  - void addFirst(Object o), void addLast(Object o)
  - Object getFirst(), Object getLast()
  - Object removeFirst(), Object removeLast()

Set Interface

- ✓Same methods as Collection
  - different contract - no duplicate entries
- ✓Defines two fundamental methods
  - boolean add(Object o) - reject duplicates
  - Iterator iterator()
- ✓Provides an Iterator to step through the elements in the Set
  - No guaranteed order in the basic Set interface
  - There is a SortedSet interface that extends Set

HashSet

- ✓Find and add elements very quickly
  - uses hashing implementation in HashMap
- ✓Hashing uses an array of linked lists
  - The hashCode() is used to index into the array
  - Then equals() is used to determine if element is in the (short) list of elements at that index
- ✓No order imposed on elements
- ✓The hashCode() method and the equals() method must be compatible
  - if two objects are equal, they must have the same hashCode() value

TreeSet

- ✓Elements can be inserted in any order
- ✓The TreeSet stores them in order
  - Red-Black Trees out of Cormen-Leiserson-Rivest
- ✓An iterator always presents them in order
- ✓Default order is defined by natural order
  - objects implement the Comparable interface
  - TreeSet uses compareTo(Object o) to sort
- ✓Can use a different Comparator
  - provide Comparator to the TreeSet constructor

Map Interface

- ✓Stores key/value pairs
- ✓Maps from the key to the value
- ✓Keys are unique
  - a single key only appears once in the Map
  - a key can map to only one value
- ✓Values do not have to be unique

Map methods

Object put(Object key, Object value)

Object get(Object key)

Object remove(Object key)

boolean containsKey(Object key)

boolean containsValue(Object value)

int size()

boolean isEmpty()

Map views

- ✓A means of iterating over the keys and values in a Map
- ✓Set keySet()
  - returns the Set of keys contained in the Map
- ✓Collection values()
  - returns the Collection of values contained in the Map. This Collection is not a Set, as multiple keys can map to the same value.
- ✓Set entrySet()
  - returns the Set of key-value pairs contained in the Map. The Map interface provides a small nested interface called Map.Entry that is the type of the elements in this Set.

HashMap and TreeMap

- ✓HashMap
  - The keys are a set - unique, unordered
  - Fast
- ✓TreeMap
  - The keys are a set - unique, ordered
  - Same options for ordering as a TreeSet
    - Natural order (Comparable, compareTo(Object))
    - Special order (Comparator, compare(Object, Object))

Bulk Operations

- ✓In addition to the basic operations, a Collection may provide "bulk" operations

```
boolean containsAll(Collection c);
boolean addAll(Collection c);
boolean removeAll(Collection c);
boolean retainAll(Collection c);
void clear();
Object[] toArray();
Object[] toArray(Object a[]);
```

Utilities

- ✓The Collections class provides a number of static methods for fundamental algorithms
- ✓Most operate on Lists, some on all Collections
  - Sort, Search, Shuffle
  - Reverse, fill, copy
  - Min, max
- ✓Wrappers
  - synchronized Collections, Lists, Sets, etc
  - unmodifiable Collections, Lists, Sets, etc