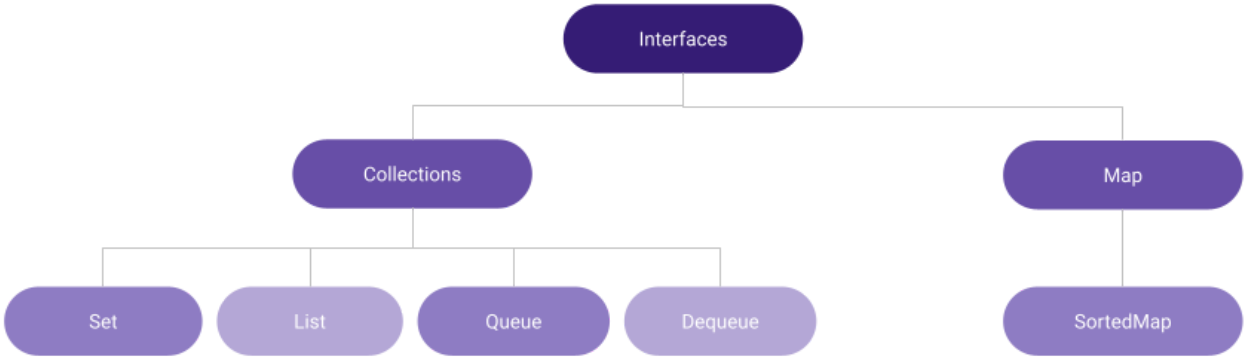
name: Oscar Rodolfo Umaña Linares.

Java Collections.

A collection is an object that represents a group of objects, the collection framework allows us to manipulate these data structures independently of the implementation details.

The Java collection framework consists of interfaces that represent the blueprint of different types of collections such as sets, lists, queue, dequeue and maps; all of these interfaces extend the Iterable<E> interface.



Implementations.

Set Implementations:

A Set is a Collection that cannot contain duplicate elements. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited. Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ. Two Set instances are equal if they contain the same elements.

The most used Set implementations in the Java Platform are: HashSet and TreeSet, most of the times we use the HashSet but these implementations have one key difference, the first is sorted and the second is unsorted.

1. **HashSet:**
    - **Definition:**This class implements the Set interface, backed by a hash table. It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element.
    - **Features:**
        - Allows null value.
        - It is a non-synchronized class.
        - Only contains unique elements.
        - Most useful in the search operations.
    - **When to use it:** It's recommended if we have a collection of objects and we care only about the existence, this implementation is really great if we care about performance.

2. **LinkedhashSet:**

- **Definition:** Hash table and linked list implementation of the Set interface, with predictable iteration order. This implementation differs from HashSet in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is the order in which elements were inserted into the set.
- **Features:**
    - Duplicate values are not allowed in LinkedHashSet.
    - One null element is allowed in LinkedHashSet.
    - It is an ordered collection which is the order in which elements were inserted
- **When to use it:** It's recommended if we want to store unique elements with their insertion order.

3. **TreeSet:**
    - **Definition:** The elements are ordered using their natural ordering, or by a Comparator provided at set creation time, depending on which constructor is used. This implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains).
    - **Features:**
        - Contains unique elements only like HashSet.
        - Access and retrieval times are quite fast.
        - Doesn't allow null elements.
        - Is non synchronized.
        - Maintains ascending order.
    - **When to use it:** It's only recommended if we care about the order of the elements in the set, if not it is better to use the HashSet implementation.

List Implementations:

A List is an ordered Collection. Lists may contain duplicate elements. In addition to the operations inherited from Collection, the List interface includes operations for the following:

- Positional access methods:
    - get()
    - set()
    - addaddAll()
    - remove()
- Search methods:
    - indexOf()
    - lastIndexOf()
- Range-view methods:
    - sublist()

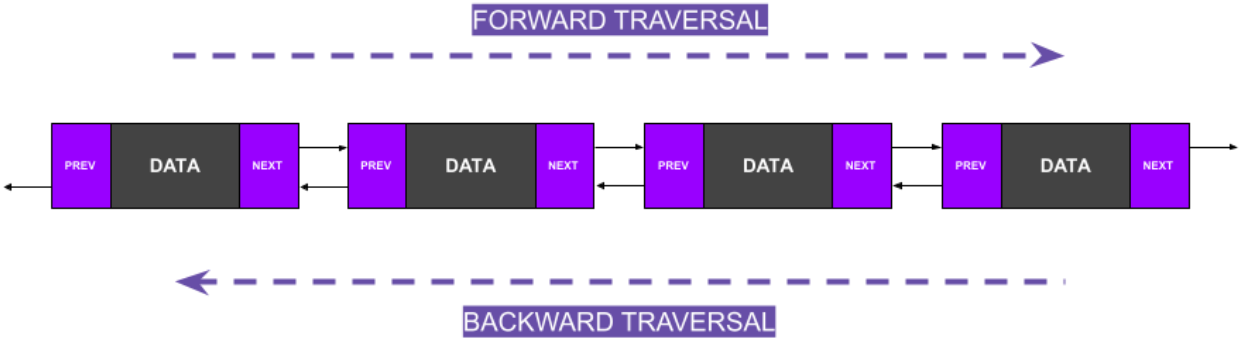
The Java platform contains two general-purpose List implementations. ArrayList, which is usually the better-performing implementation, and LinkedList which offers better performance under certain circumstances.

1. **ArrayList:**

- **Definition:** Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)
- **Features:**
    - An ArrayList automatically expands as data is added.
    - Access to any element of an ArrayList is O(1). Insertions and deletions are O(N).
    - An ArrayList has methods for inserting, deleting, and searching.
    - An ArrayList can be traversed using a foreach loop, iterators, or indexes.
- **When to use it:** It's recommended to use ArrayList if searching is more frequent than add and remove operation. The LinkedList provides constant time for add and remove operations.

2. **Vector:**
    - **Definition:** The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.
    - **Features:**
        - Vector is synchronized.
        - maintains an insertion order
        - it is rarely used in a non-thread environment
        - The Iterators returned by the Vector class are fail-fast. In the case of concurrent modification, it fails.
        - Vector implements a dynamic array.
    - **When to use it:** Vector are used in concurrent environments, it has very similar usages to the ArrayList
3. **LinkedList:**
    - **Definition:** All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

      Note that this implementation is not synchronized. If multiple threads access a linked list concurrently, and at least one of the threads modifies the list structurally, it must be synchronized externally.
    - **Features:**
        - Vector is synchronized.
        - maintains an insertion order
        - it is rarely used in a non-thread environment
        - The Iterators returned by the Vector class are fail-fast. In the case of concurrent modification, it fails.
        - Vector implements a dynamic array.
    - **When to use it:** LinkedList are used when we will be manipulating data frequently because add and remove operations are more effective but if we need to search an specific position these operations can be time consuming because we will need to traverse the list until we find the value.



Queue Implementations:

A collection designed for holding elements prior to processing. Besides basic Collection operations, queues provide additional insertion, extraction, and inspection operations. Each of these methods exists in two forms: one throws an exception if the operation fails, the other returns a special value.

1. **PriorityQueue:**
    - **Definition:** An unbounded priority queue based on a priority heap. The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.
    - **Features:**
        - Each item has some priority associated with it.
        - An item with the highest priority is moved at the front and deleted first.
        - If two elements share the same priority value, then the priority queue follows the first-in-first-out principle for queue operation.
    - **When to use it:** Priority queues are used to select the next process to run, ensuring high-priority tasks run before low-priority ones. It is also applied for load balancing, and interrupt handling.

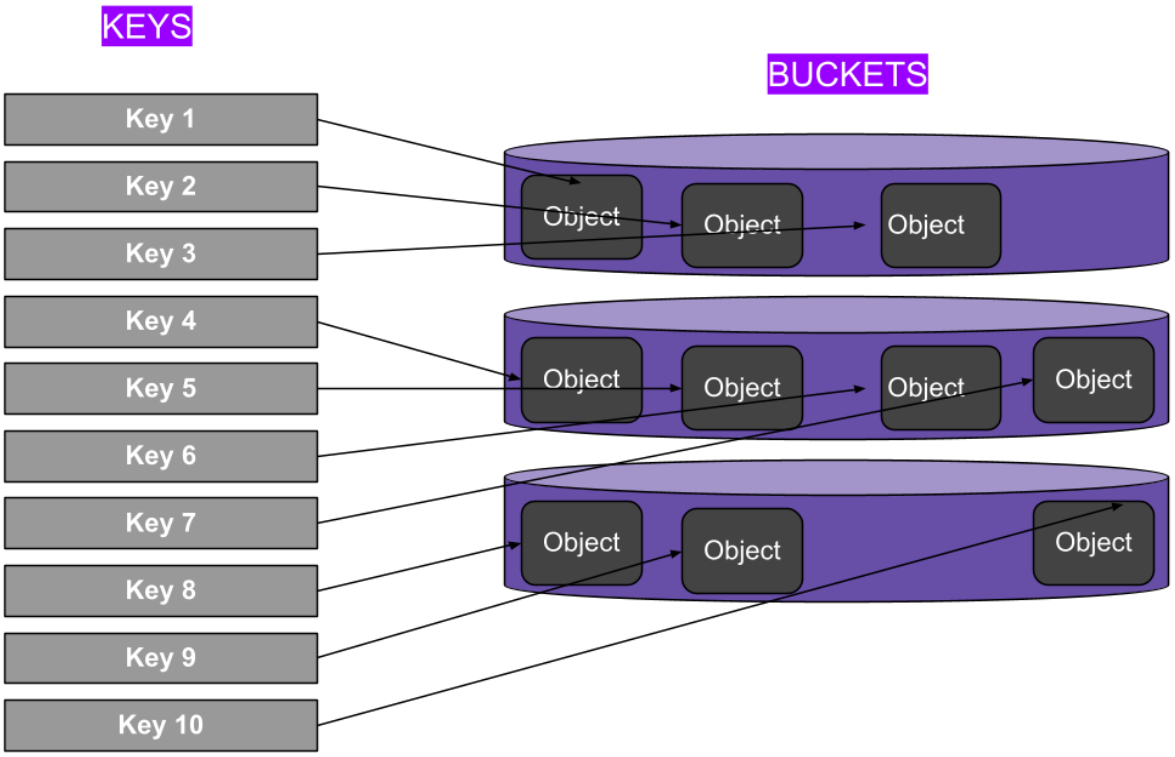
Map Implementations:

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. This interface takes the place of the Dictionary class, which was a totally abstract class rather than an interface. The Map interface provides three collection views, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings.

1. **HashTable:**
    - **Definition:** This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.
    - **Features:**
        - It is similar to HashMap, but it is synchronized while HashMap is not synchronized.
        - It does not accept null key or value.
        - It does not accept duplicate keys.
        - It stores key-value pairs in a hash table data structure which internally maintains an array of lists.

- **When to use it:** They are used to implement associative arrays (arrays whose indices are arbitrary strings or other complicated objects.

2. **HashMap:**
   - **Definition:** This implementation provides all of the optional map operations, and permits null values and the null key. The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls. This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.
   - **Features:**
     - HashMap cannot contain duplicate keys.
     - HashMap allows multiple null values but only one null key.
     - HashMap is an unordered collection. ...
     - HashMap is not thread-safe. ...
     - A value can be retrieved only using the associated key.
     - HashMap stores only object references.
   - **When to use it:** when unique keys are available for the data we want to store. We should use it when searching for items based on a key and quick access time is an important requirement. We should avoid using HashMap when it is important to maintain the same order of items in a collection.



3. **Treemap:**
   - **Definition:** 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.
     This implementation provides guaranteed log(n) time cost for the containsKey, get, put and remove operations.
   - **Features:**
     - Java TreeMap contains only unique elements.

    - Java TreeMap cannot have a null key but can have multiple null values.
    - Java TreeMap is non synchronized.
    - Java TreeMap maintains ascending order.
- **When to use it:** Treemaps are often used for sales data, as they capture relative sizes of data categories, allowing for quick perception of the items that are large contributors to each category.