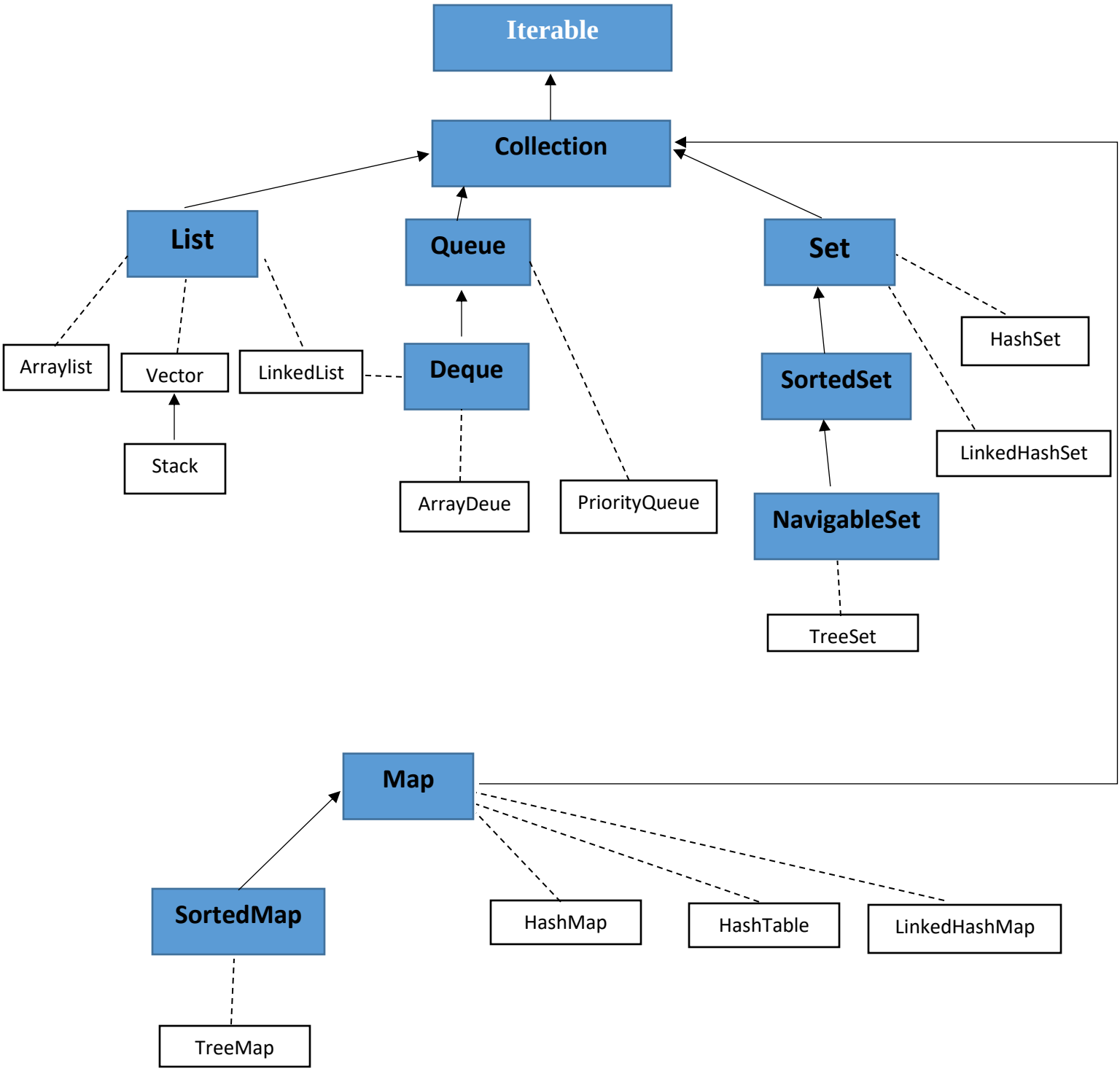
Collection Framework

The collection framework is an architecture made up of interfaces and classes. In simple words, it is like a skeletal structure for components that is ready to use for various programming needs. It also offers different data operations like searching, sorting, insertion, deletion, and manipulation. All of the classes and interfaces of the collection framework are bundled into the **java.util.package**.



Iterable Interface:

It is the root of the entire collection hierarchy, which means that every class and interface implements it. The primary function of an iterator is to allow the user to traverse through all of the collection class objects as if they were simple sequence of data items.

Iterator <E> iterator()

This interface specifies the following methods:

Method	Description
hasNext(): boolean	Returns true if there is at least one more element from the collection that can be returned, false otherwise.
next(): E	Returns the next element from the collection
remove(): void	Remove from the collection returned by the last call to next(). This method can be called at least one time for each call to next().

Collection Interface:

It is the foundation upon which the collections framework is built because it must be implemented by any class that defines a collection. It is a generic interface that has this declaration:

Interface Collection <E>

Here, E specifies the type of objects that the collection will hold. It extends the iterable interface. This means that all collections can be cycled through by use of the for-each style for loop.

Method	Description
add(E obj): boolean	Adds obj to the invoking collection. Returns true if obj was added to the collection. Returns false if obj is already a member of the collection and the collections does not allow duplicates.
equals(object obj): boolean	Returns true if the invoking collection and obj are equal. Otherwise, returns false.
remove(object obj): boolean	Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false.
int size()	Returns the number of elements held in the invoking collection.

Object[] toArray()	Returns an array that contains all the elements stored in the invoking collection. The array elements are the copies of the collection elements.

List Interface:

This interface extends from the Collection interface. The elements in a list are ordered like a sequence. It is an ordered collection of objects in which duplicate values can be stored. The user can use the index number to access a particular element in the list, that is to say, the user has complete control over which element is inserted wherein the list.

Method	Description
void add(int index, E obj)	Inserts obj into the invoking list at the index passed in index. Any preexisting elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten.
E get(int index)	Returns the object stored at the specified index within the invoking collection.
ListIterator<E> listiterator()	Returns an iterator to the start of the invoking list.

1. ArrayList

IT is a java class implemented using the List interface. It provides us with dynamic arrays in java. If we declare an array, we need to mention the size, but in ArrayList, it is not needed to mention the size of ArrayList.

2. LinkedList

This class is an implementation of the linked list data structure which is a linear data structure where the elements are not stored in contiguous locations and every elements is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node. It acts as a dynamic array and we do not have to specify the size while creating it. Internally, this is implemented using the doubly linked list data structure. The main difference between a normal linked list and a doubly linked list is that a doubly linked list contains an extra pointer named previous pointer.

3. Vector

It implements a dynamic array which means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index. They are similar to ArrayList, but vector is synchronized and has some legacy methods that the collection framework does not contain.

i. Stack

Java collection framework provides a Stack class that models and implements a stack data structure. The class is based on the basic principle of last in first out. In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek. The class is the subclass of Vector class.

Queue Interface:

The Queue interface is present in java.util package and extends the Collection interface is used to hold the elements about to be processed in FIFO(First In First Out) order. It is an ordered list of objects with its use limited to inserting elements at the end of the list and deleting elements from the start of the list, it follows the FIFO or the First-In-First-Out principle.

Method	Description
boolean add(object)	It is used to insert the specified element into this queue and return true upon success.
boolean offer(object)	It is used to insert the specified element into this queue.
Object remove()	It is used to retrieves and removes the head of this queue.
Object poll()	It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
Object element()	It is used to retrieves, but does not remove, the head of this queue.
Object peek()	It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

PriorityQueue:

PriorityQueue is also class that is defined in the collection framework that gives us a way for processing the objects on the basis of priority. It is already described that the insertion and deletion of objects follows FIFO pattern in the Java queue. However, sometimes the elements of the queue are needed to be processed according to the priority, that's where a PriorityQueue comes into action. The PriorityQueue is based on the priority heap. The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.