Collection framework

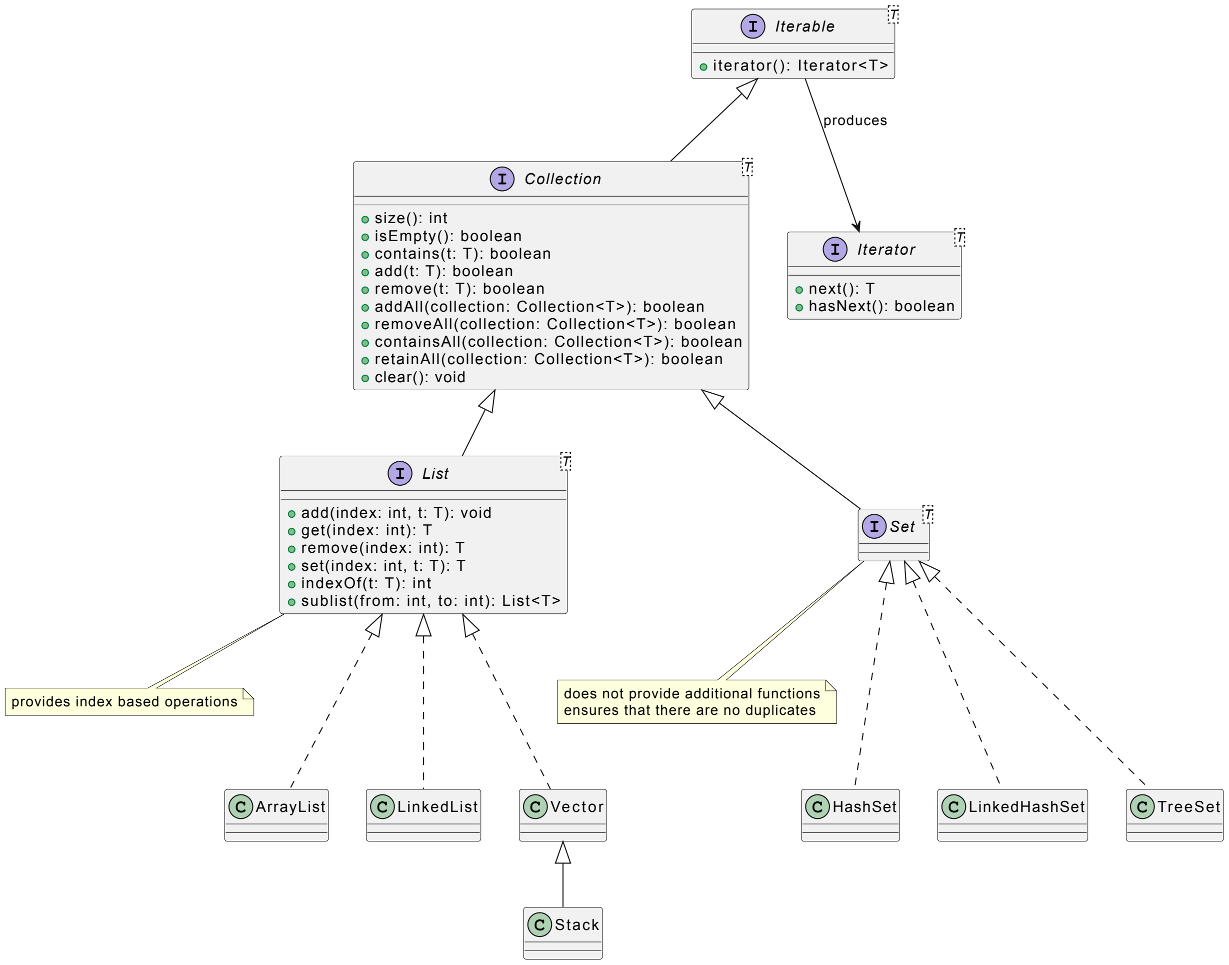
Problems with Arrays

- occupy memory continuously (for large arrays, there may not be sufficient memory)
- inserting an element in any random index is very difficult
    - we have to shift the other elements to make space for the new element
- deleting an element in a random index is difficult
    - we have to shift the elements next to the deleted element to remove the vacuum created
- fixed in size (although the size can be determined at the runtime)
    - what if we want add more elements than the current size?
    - what if we do not have many values, but the array is too big? - waste of space?

Collection framework

- Java 1.0 had class to tackle the above issues
    - `java.util.Vector`
    - `java.util.Dictionary`
    - `java.util.Hashtable`
- Java 1.2 introduced many interfaces and implementation classes to provide more flexibility to the developer
    - developer can choose from a variety of collection classes, based on specific needs
        - need a collection allows a sorted access
        - need a collection that has no duplicates
        - need a collection that is faster while insert/delete at random index
        - need a collection that is faster while accessing at random index
        - just need a collection
    - The older classes such as `Vector`, `Hashtable`, `Dictionary`, `Enumeration` etc were re-written to suit the new collection framework
- Java 1.5 introduced the concept of generics
    - special syntax with `<T>`
    - allows collections to be homogeneous
- Java 1.5 also introduced the `enhanced for loop` or also known as `for-each loop`, which can be used for any object of `Iterable`, which produces `Iterator`
- Java 1.8 introduced the concept of `streams` (not to be confused with IO streams)
    - stream of data flowing from source to the collectors
- Java 1.8 also introduced arrow functions (lambda expressions), that are heavily used along with stream functions

Collection hierarchy (JDK 1.2 onwards)


Iterable
T
+ iterator(): Iterator<T>
produces
Iterator
T
+ next(): T
+ hasNext(): boolean
Collection
T
+ size(): int
+ isEmpty(): boolean
+ contains(t: T): boolean
+ add(t: T): boolean
+ remove(t: T): boolean
+ addAll(collection: Collection<T>): boolean
+ removeAll(collection: Collection<T>): boolean
+ containsAll(collection: Collection<T>): boolean
+ retainAll(collection: Collection<T>): boolean
+ clear(): void
List
T
+ add(index: int, t: T): void
+ get(index: int): T
+ remove(index: int): T
+ set(index: int, t: T): T
+ indexOf(t: T): int
+ sublist(from: int, to: int): List<T>
provides index based operations
Set
T
does not provide additional functions
ensures that there are no duplicates
ArrayList
LinkedList
Vector
Stack
HashSet
LinkedHashSet
TreeSet