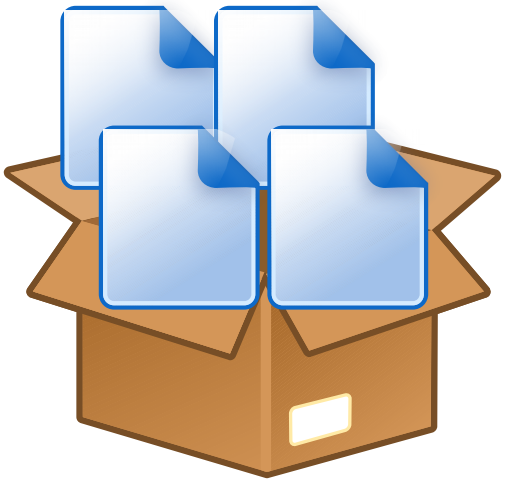
Collection Framework

What is collection?

- **A collection — sometimes called a container — is simply an object that groups multiple elements into a single unit.**
- **Data items that form a natural group.**
  - List<Letters>



Collection of Letters objects

collections

- **Collections are used to store, retrieve, manipulate, and communicate aggregate data.**

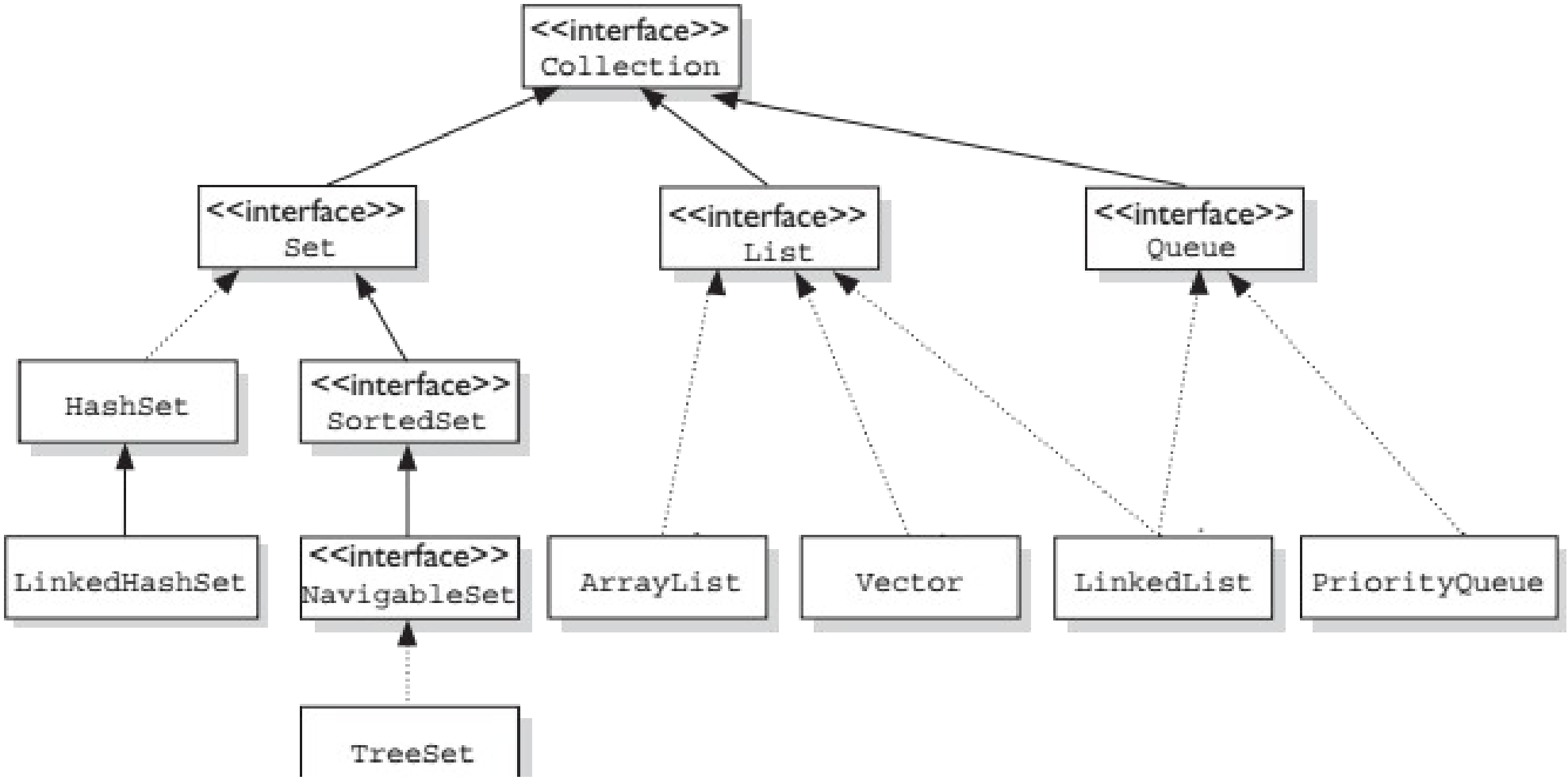
Why to use?

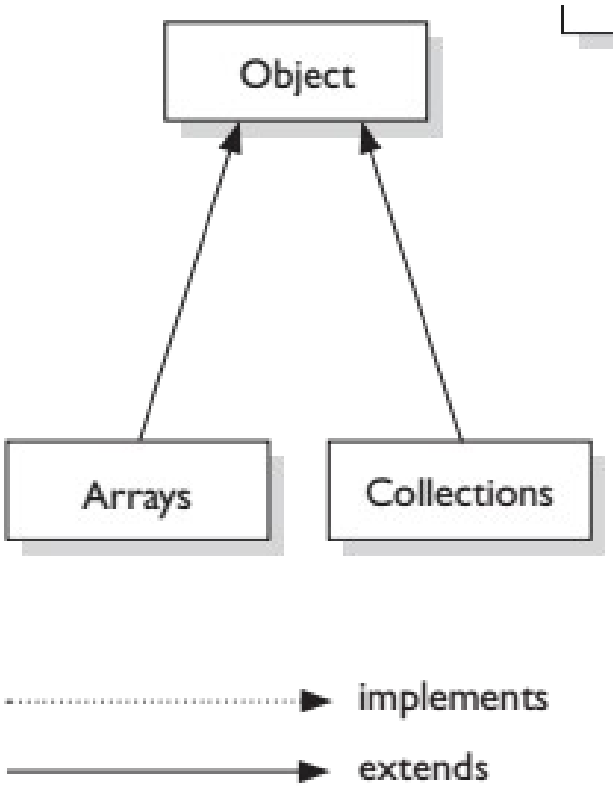
- **Provides high-performance, high-quality implementations of useful data structures and algorithms.**
- **Gives us lists, sets, maps, and queues.**

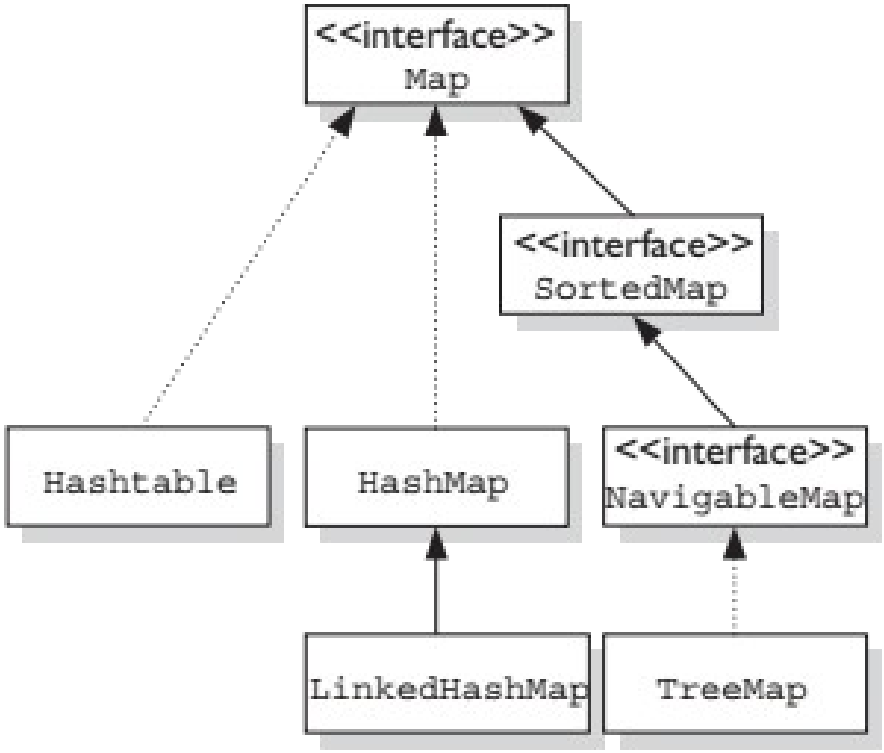
Collection Framework

- **collection (lowercase c) represents any of the data structures**
  - in which objects are stored and iterated over.
- **Collection (capital C),is java.util.Collection interface**
  - from which Set, List, and Queue extend. (no direct implementations of Collection.)
- **Collections (capital C and ends with s) is the java.util.Collections class**
  - that holds a pile of static utility methods for use with collections.

Collection Framework




Object
Arrays
Collections
implements
extends


<<interface>>
Map
<<interface>>
SortedMap
Hashtable
HashMap
<<interface>>
NavigableMap
LinkedHashMap
TreeMap

Major Collection Types

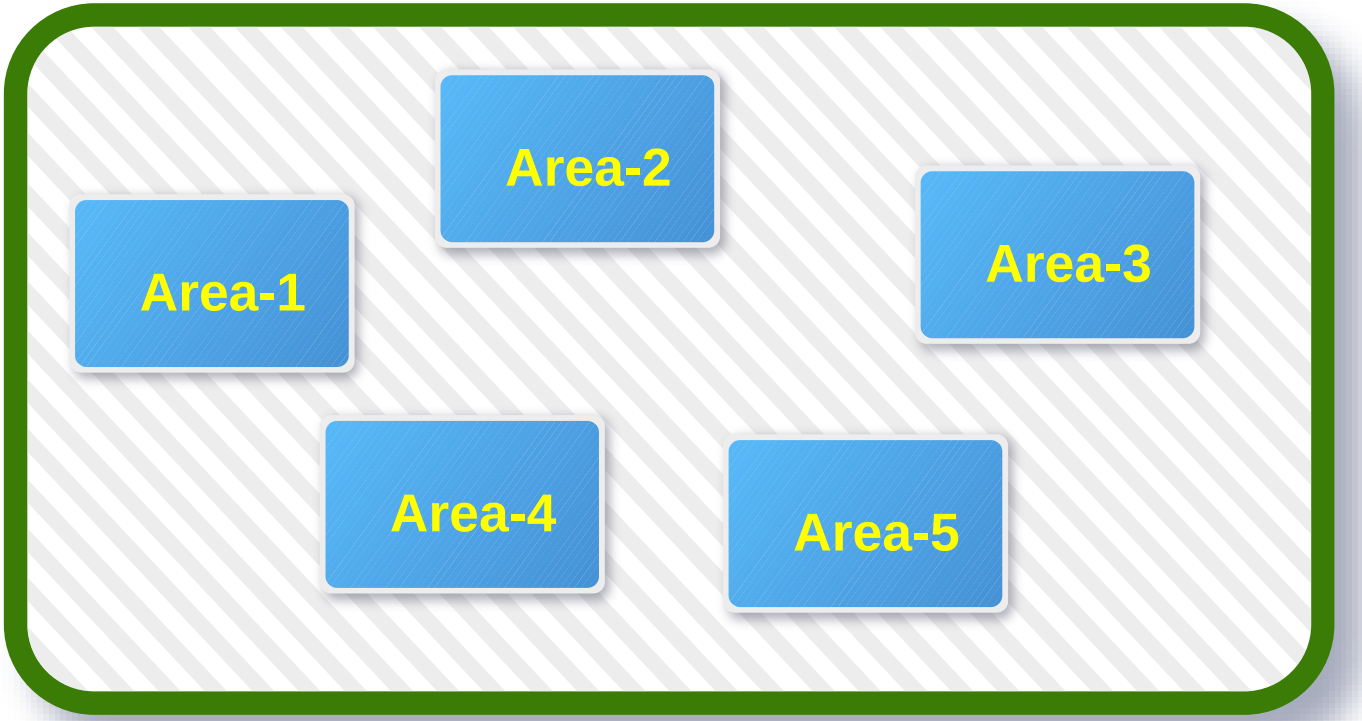
- **Lists:**
  - Lists of things (classes that implement List)
  - Exa- salesman's route of selling products

Duplicates are allowed

Major Collection Types

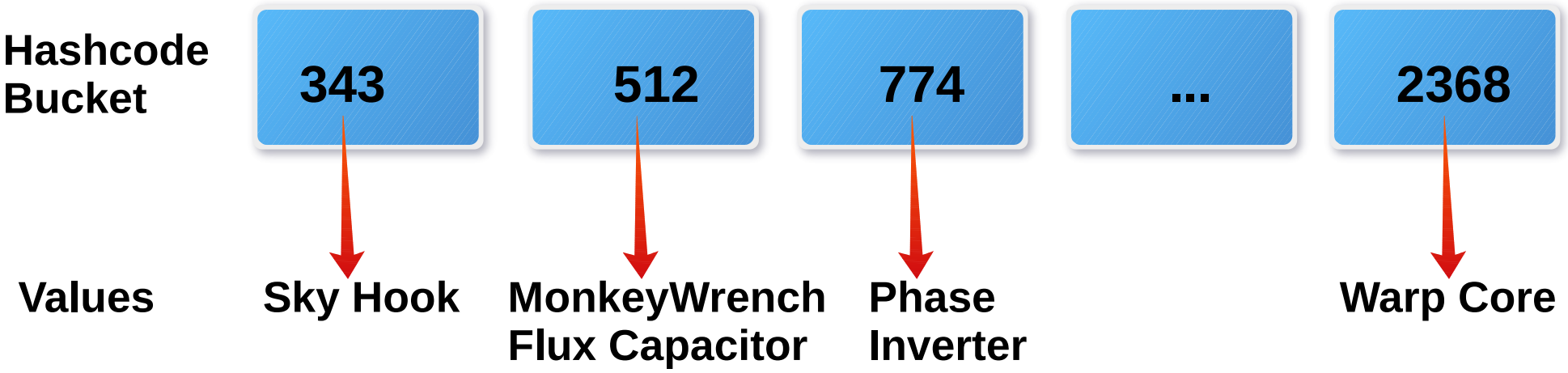
- **Sets:**
  - Unique things (classes that implement Set)
  - Salesman's territory for selling products

Duplicates are not allowed



Major Collection Types

- **Maps:-**
  - Things with a unique ID (classes that implement Map)
  - salesman's products (Keys-product IDs)



Major Collection Types

- **Queues**
  - Things arranged by the order in which they are to be processed

Collection Terminology

- **Iteration**
  - walking through the elements one after another, starting from the first element.

Collection Terminology

- **Ordered**
  - When a collection is ordered, it means you can iterate through the collection in a specific (not random) order.
- **A Hashtable- collection is not ordered.**
- **An ArrayList- ordered by the elements' index position**
  - can insert an element at a specific index position.
- **LinkedHashSet- ordered by insertion.**
  - the last element inserted is the last element in the LinkedHashSet.

Collection Terminology

- **Sorted**
  - A sorted collection means that the order in the collection is determined according to some rule or rules, known as the sort order.
- **Sorting is done based on properties of the objects themselves.**
- **Sort order (including natural order) is not the same as ordering by insertion, access, or index.**
- **Also called the natural order.**
  - String objects- alphabetical.
  - Integer objects- numeric.
  - Other objects- using *Comparable* or *Comparator* interfaces.
-

List Interface

- **A List cares about the index.**
- **It has set of methods related to the index like get(int index), indexOf(Object o), add(int index, Object obj), etc.**
- **All three List implementations are ordered by index position.**
  - ArrayList
  - Vector
  - LinkedList

ArrayList

- **ArrayList- a growable array.**
- **It gives you fast iteration and fast random access.**
- **It is an ordered collection (by index), but not sorted.**
- **It implements RandomAccess interface—supports fast (generally constant time) random access."**
- **Choose this**
  - fast iteration but
  - Not a lot of insertion and deletion.

Vector

- Vector- old stuff
- It is basically the same as an ArrayList, but Vector methods are synchronized for thread safety.
- Use it when thread safety is needed otherwise it may hit performance.
- It also implements RandomAccess interface.

LinkedList

- Ordered by index position, like ArrayList, except that the elements are doubly linked to one another.
- More methods for adding and removing from the beginning or end.
- An easy choice for implementing a stack or queue.
- Iteration is more slowly than an ArrayList, but it's a good choice when you need fast insertion and deletion.
- Also java.util.Queue interface supporting the common queue methods peek(),poll(), and offer().

Set Interface

- A Set cares about uniqueness—it doesn't allow duplicates.
- equals() method determines whether two objects are identical (in which case, only one can be in the set).

HashSet

- A HashSet is an unsorted, unordered Set.
- It uses the hashcode of the object being inserted, so the more efficient hashCode() implementation, the better access performance.
- Use this class when you want a collection with no duplicates and you don't care about iteration order.

LinkedHashSet

- **LinkedHashSet- ordered version of HashSet that maintains a doubly linked List across all elements.**
- **Use this when you care about the iteration order.**
- **It let us iterate through the elements in the order in which they were inserted.**

TreeSet

- The TreeSet is one of two sorted collections (other TreeMap).
- It uses a Red-Black tree structure.
- Elements will be in ascending order, according to natural order.
- Use a Comparator for custom order rule.

Map Interface

- **A Map cares about unique identifiers.**
- **You map a unique key (the ID) to a specific value, where both the key and the value are objects.**
- **With Map**
  - Search for a value based on the key,
  - Or a collection of just the values
  - Or a collection of just the keys.
- **equals() method determines whether two keys are the same or different.**

HashMap

- **HashMap- An unsorted, unordered Map.**
- **Use it when you don't care about the order when you iterate through Map.**
- **Keys are based on the hashcode, so the more efficient your hashCode() implementation, the better access performance.**
- **HashMap allows**
  - one null key &
  - multiple null values

Hashtable

- Hashtable- old stuff
- Vector is a synchronized counterpart to ArrayList, Hashtable is the synchronized counterpart to HashMap.
- Hashtable doesn't allow anything that's null.

LinkedHashMap

- LinkedHashMap maintains insertion order (or, optionally, access order).
- Somewhat slower than HashMap for adding and removing elements, but faster iteration.

TreeMap

- **TreeMap- is a sorted Map.**
- **Sorted by natural order still custom sort order (via a Comparator).**
- **TreeMap implements NavigableMap.**

Queue Interface

- A Queue holds a list of "to-dos," or things to be processed in some way.
- Typically FIFO (first-in,first-out),but others also like LIFO, priority.
- Standard Collection methods + methods to add and subtract elements and review queue elements.

PriorityQueue

- LinkedList class implements the Queue interface, basic queues can be handled with a LinkedList.
- The purpose of a PriorityQueue is to create a "priority-in, priority out" queue as opposed to a typical FIFO queue.
- A PriorityQueue's elements are ordered either by natural ordering (in which case the elements that are sorted first will be accessed first) or according to a Comparator.
- In either case, the elements ordering represents their relative priority.

Collection summary

Class	Map	Set	List	Ordered	Sorted
`HashMap`	X			No	No
`Hashtable`	X			No	No
`TreeMap`	X			Sorted	By *natural order* or custom comparison rules
`LinkedHashMap`	X			By insertion order or last access order	No
`HashSet`		X		No	No
`TreeSet`		X		Sorted	By *natural order* or custom comparison rules
`LinkedHashSet`		X		By insertion order	No
`ArrayList`			X	By index	No
`Vector`			X	By index	No
`LinkedList`			X	By index	No
`PriorityQueue`				Sorted	By to-do order

The Comparable Interface

- used by the Collections.sort() method and the java.util.Arrays.sort() method to sort Lists and arrays of objects.
-

Implementing Comparable Interface

- **Class must implement a single method, compareTo().**
  - **int x = thisObject.compareTo(anotherObject);**
- **The compareTo() method returns an int with the following characteristics:**
  - **Negative If thisObject < anotherObject**
  - **Zero If thisObject == anotherObject**
  - **Positive If thisObject > anotherObject**