Which class do you choose when you want _____	Answer (single best type)	Reason
to pick the top zoo map off a stack of maps	`ArrayDeque`	The description is of a last-in, first-out data structure, so you need a stack, which is a type of `Queue`. (`Stack` would also match this description, but it shouldn't be used for new code.)
to sell tickets to people in the order in which they appear in line and tell them their position in line	`LinkedList`	The description is of a first-in, first-out data structure, so you need a queue. You also needed indexes, and `LinkedList` is the only class to match both requirements.
to write down the first names of all of the elephants so that you can tell them to your friend's three-year-old every time she asks. (The elephants do not have unique first names.)	`ArrayList`	Since there are duplicates, you need a list rather than a set. You will be accessing the list more often than updating it, since three-year-olds ask the same question over and over, making an `ArrayList` better than a `LinkedList`. `Vector` and `Stack` aren't used in new code.
to list the unique animals that you want to see at the zoo today	`HashSet`	The keyword in the description is *unique.* When you see "unique," think "set." Since there were no requirements to have a sorted order or to remember the insertion order, you use the most efficient set.
to list the unique animals that you want to see at the zoo today in alphabetical order	`TreeSet`	Since it says "unique," you need a set. This time, you need to sort, so you cannot use a `HashSet`.
to look up animals based on a unique identifier	`HashMap`	Looking up by key should make you think of a map. Since you have no ordering or sorting requirements, you should use the most basic map.

TABLE 3.7 Java Collections Framework types

Type	Can contain duplicate elements?	Elements ordered?	Has keys and values?	Must add/remove in specific order?
`List`	Yes	Yes (by index)	No	No
`Map`	Yes (for values)	No	Yes	No
`Queue`	Yes	Yes (retrieved in defined order)	No	Yes
`Set`	No	No	No	No

TABLE 3.8 Collection attributes

Type	Java Collections Framework interface	Sorted?	Calls hashCode?	Calls compareTo?
ArrayList	List	No	No	No
ArrayDeque	Queue	No	No	No
HashMap	Map	No	Yes	No
HashSet	Set	No	Yes	No
Hashtable	Map	No	Yes	No
LinkedList	List, Queue	No	No	No
Stack	List	No	No	No
TreeMap	Map	Yes	No	Yes
TreeSet	Set	Yes	No	Yes
Vector	List	No	No	No

TABLE 3.11 The basics of the merge and compute methods

Scenario	`merge`	`computeIfAbsent`	`computeIfPresent`
Key already in map	Result of function	No action	Result of function
Key not already in map	Add new value to map	Result of function	No action
Functional Interface used	`BiFunction` (Takes existing value and new value. Returns new value.)	`BiFunction` (Takes key and existing value. Returns new value.)	`Function` (Takes key and returns new value.)

TABLE 3.12 Merge and compute methods when `null`s are involved

Key has	Mapping functions returns	`merge`	`computeIfAbsent`	`computeIfPresent`
`null` value in map	`null`	Remove key from map.	Do not change map.	Do not change map.
`null` value in map	Not `null`	Set key to mapping function result.	Add key to map with mapping function result as value.	Do not change map.
Non-`null` value in map	`null`	Remove key from map.	Do not change map.	Remove key from map.
Non-`null` value in map	Not `null`	Set key to mapping function result.	Do not change map.	Set key to mapping function result.
Key not in map	`null`	Add key to map.	Do not change map.	Do not change map.
Key not in map	Not `null`	Add key to map.	Add key to map with mapping function result as value.	Do not change map.