Data Structures Covered in FPP

Collection	Description
ArrayList	An indexed sequence that grows and shrinks dynamically
LinkedList	An ordered sequence that allows efficient insertion and removal at any location
ArrayDeque	A double-ended queue that is implemented as a circular array
HashTable	A data structure that stores key/value associations. Similar to HashMap
HashMap	A data structure that stores key/value associations
TreeMap	A map in which the keys are sorted
HashSet	An unordered collection that rejects duplicates
TreeSet	A sorted set

Data Structures NOT Covered in FPP

Collection	Description
EnumSet	A set of enumerated type values
LinkedHashSet	A set that remembers the order in which elements were inserted
PriorityQueue	A collection that allows efficient removal of the smallest element
EnumMap	A map in which the keys belong to an enumerated type
LinkedHashMap	A map that remembers the order in which entries were added
WeakHashMap	A map with values that can be reclaimed by the garbage collector if they are not used elsewhere
IdentityHashMap	A map with keys that are compared by ==, not equals

Basic interfaces of Java Collections Framework

Collection	Description
Collection	The root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn't provide any direct implementations of this interface.
List	An ordered collection that can contain duplicate elements. Some of its implementations: AbstractList, ArrayList, LinkedList, SortedList, Stack, Vector
Map	An object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.
Set	A collection that cannot contain duplicate elements. This interface models the mathematical set abstraction and is used to represent sets, such as the deck of cards.

Guidelines for using Data Structures

Array List

Use: when main need is for a list with random access reads, infrequent adds beyond initial capacity (or maximum number of list elements is known in advance), or the list only needs to occasionally be sorted

Avoid: when many adds expected, but number of elements unpredictable, or inserts/deletes need to maintain some ordering by key.

Linked List

Use: when insertions and deletions are frequent, and/or many elements need to be added, but total number is unknown in advance

Avoid: when there is a need for repeated access to the nth element (random access) as in sorting

Binary Search Tree

Use: when data needs to be maintained in sorted order by key for frequent searches

Avoid: when the extra benefit of keeping data in sorted order is not needed and rapid read access (e.g., iteration or random access) is needed

Hash Table

Use: when random access to objects is needed but array indexing is not practical (possible index range is too large)

Avoid: when an ordering (possibly unrelated to keys(maybe by data input)) of data must be preserved, or iterating through values in the table is frequent

Set

Use: when elements do not need to be kept in a special order and searches for elements in the set are infrequent (do not need to be rapid)

Avoid: when an ordering of data must be preserved or searches for elements in the set are frequent (instead use TreeSet, which uses a binary search tree)

Stack

Use when you need a 'Last In First Out' data structure (LIFO).

Queue

Use when you need a 'First In First Out' data structure (FIFO).

Standard Exam Topics

Question 1: Data Structures:

Be able to create your own Linked List, ArrayList, Stack, Queue (with Nodes or with arrays, not relying on Java library data structures).

Be able to use the Java library classes LinkedList, ArrayList, HashMap, and use a LinkedList as a stack or queue. You should also be able to use the interfaces List and Map. For this type of question, you would need to solve a problem involving multiple data structures.

Question 2: OO (Object Oriented)

Be able to perform a computation or collect information using polymorphism.