Oswaldo Giovanni Caceres Samayoa

Java Collection Interface

- **HashShet**
  Collection that uses a hash table for storage. Stores the elements by using a mechanism called hashing containing unique elements only.

  Features
  - Contains unique elements only.
  - Allows null values.
  - Elements are inserted on the basis of their hashcode.
  - Excels at search operations.

  When to use
  Hash set is usually used when you need high-performance operations involving a set of unique data.

- **LinkedHashSet**
  The LinkedHashSet it's a LinkedList implementation of the HashSet. The main difference is that it maintains insertion order.

  Features
  - Contains unique elements only.
  - Allows null values.
  - Elements ordered by insertion.

  When to use
  LinkedHashSet is usually used when you need high-performance operations involving a set of unique data but you need to keep the order in which the data was inserted.

- **TreeSet**
  TreeSet is an implementation of a Set that uses a tree for storage.

  Features
  - Contains unique elements only.
  - Fast access and retrieval operations.
  - Doesn’t allow null elements.
  - Elements are stored in ascending order.

  When to use
  It's better to use when you need a dynamic search tree with frequent read/write operations and it needs to keep order.

- **ArrayList**
  Dynamic array for storing elements. More flexible than the traditional array, there is no size limit, and you can add and remove elements anytime.

  Features
  - Can contain duplicate elements.
  - No size limit. It can grow and shrink dynamically.
  - Maintains insertion order.
  - Allows random access.

  When to use
  If you need more frequent search operations than add and remove operations, as it has O(1) time complexity in those kinds of operations.

- **Vector**
  It's a dynamic array that can grow or shrink its size with fail-fast iterators.

  Features
  - No size limit, grows and shrinks dynamically.
  - It has a thread-safe implementation.
  - Maintains legacy methods not part of the collections framework.

  When to use
  It's recommended to use only the thread-safe implementation. If you don’t need the thread-safety features you should use ArrayList instead as it will perform better.

- **LinkedList**
  Class that uses a doubly linked list to store elements. It provides a linked-list data structure.

  Features
  - Can contain duplicate elements.
  - Maintains insertion order.
  - Fast data manipulation, there is no shifting.
  - It can be used as a list, stack or queue.

  When to use
  When you need to do frequent data manipulation or you need your list to act as a Queue.

- **PriorityQueue**
  Class that provides the facility of using a queue, but it does not order the elements in a FIFO manner. Instead it bases the order on a priority basis.

  Features
  - Doesn't allow null values.
  - Only permits comparable objects.
  - Processes elements based on priority, not insertion order.

  When to use
  When you need to process objects/elements based on the priority, not insertion order.

Java Map Interface

- **HashTable**
  Structure that maps keys to values. When using a HashTable, you specify an object that is used as a key, and a value that you want linked to that key. The key is then hashed and used as the index at which the value is stored within the table.

  Features
  - Contains unique elements.
  - Doesn't allow null key or value
  - Is synchronized.
  - Stores elements as a key/value pair.

  When to use
  When you need a Thread-safe implementation of a map.

- **HashMap**
  Structure that maps keys to values. When using a HashTable, you specify an object that is used as a key, and a value that you want linked to that key. The key is then hashed and used as the index at which the value is stored within the table.

  Features
  - Contains unique elements.
  - Allows null key or value
  - Stores elements as a key/value pair.

  When to use
  When you need to store values with unique keys, as it permits fast search and access operations.

- **LinkedHashMap**
  It's an implementation of the Map interface, with predictable iteration order.

  Features
  - Contains unique elements.
  - Allows null key or value
  - Stores elements as a key/value pair.
  - Maintains insertion order

  When to use
  When you need to store values with unique keys, as it permits fast search and access operations but still need to keep the order in which the elements were inserted.

- **TreeMap**
  Tree based implementation of the Map interface, provides efficient means of storing key-value pairs in sorted order.

  Features
  - Contains unique elements.
  - Doesn't allow null key or value
  - Stores elements as a key/value pair.
  - Maintains ascending order

  When to use
  When you need to store values with unique keys, as it permits fast search and access operations but you need the order of the elements to be sorted.