**What is Set?**

*A set is an efficient data structure for storing and processing nonduplicate elements.*

*We can create a set using one of its three concrete classes:* **HashSet***,*

**LinkedHashSet***, or* **TreeSet***.* The **Set** interface extends the **Collection** interface. It does not introduce new methods or constants, but it stipulates that an instance of **Set** contains no duplicate elements. The concrete classes that implement **Set** must ensure that no duplicate elements can be added to the set.

**What is HashSet?**

The **HashSet** class is a concrete class that implements **Set**. You can create an empty *hash set* using its no-arg constructor or create a hash set from an existing collection. By default, the initial capacity is **16** and the load factor is **0.75**. A **HashSet** can be used to store *duplicate-free* elements.

**Describe LinkedHashSet:**

**LinkedHashSet** extends **HashSet** with a linked-list implementation that supports an ordering of the elements in the set. The elements in a **HashSet** are not ordered, but the elements in a **LinkedHashSet** can be retrieved in the order in which they were inserted into the set. A **LinkedHashSet** can be created by using one of its four constructors.These constructors are similar to the constructors for **HashSet**.

**Describe TreeSet:**

**SortedSet** is a subinterface of **Set**, which guarantees that the elements in the set are sorted. Additionally, it provides the methods **first()** and **last()** for returning the first and last elements in the set, and **headSet(toElement)** and **tailSet(fromElement)** for returning a portion of the set whose elements are less than **toElement** and greater than or equal to **fromElement**, respectively.

**Differences between Sets and Lists:**

|  |  |
| --- | --- |
| Sets | Lists |
| Sets are more efficient than lists for storing non-duplicate elements. | Lists are useful for accessing elements through the index. |
| Sets don’t support indexing. | Lists accessed through index. |
| The elements in the sets are unordered. |  |
| Sets use a foreach loop. |  |

**What is Maps?**

A *map* is a container object that stores a collection of key/value pairs. It enables fast retrieval, deletion, and updating of the pair through the key. A map stores the values along with the keys. The keys are like indexes. In **List**, the indexes are integers. In **Map**, the keys can be any objects. A map cannot contain duplicate keys. Each key maps to one value. A key and its corresponding value form an entry stored in a map. There are three types of maps: **HashMap**, **LinkedHashMap**, and **TreeMap**. The **Map** interface provides the methods for querying, updating, and obtaining a collection

of values and a set of keys.