SET

* Non ammette elementi duplicati e non ha una nozione di sequenza o di posizione

BIANRY HEAP

* Rappresentato tramite un array di oggetti di tipo **Comparable<>**

DIZIONARIO

* Formati da: elemento | chiave

HASH SET

* Utilizzo dei SET 🡺 Lista 🡺 richiede un solo parametro
* Utilizza una tabella hash
* **Ricerca, inserimento e cancellazione in tempo costante**

TREE SET

* Utilizzo dei SET 🡺 Lista 🡺 richiede un solo parametro
* Albero red-black
* Gli elementi sono ordinati 🡺 presenza di un comparatore

MAPPA < K,V>

* Formato da: **chiave ricerca | valore**
* Il valore viene cercato tramite la **chiave**
* Non è iterabile 🡺 usare il metodo **Map.entrySet()** 🡺 all’interno del ciclo *for*

HASH MAP

* Utilizzo di una mappa
* Gli elementi vengono inseriti con hashing

SORTED MAP

* Estende e specializza Map
* Ordinamento naturale degli elementi

TREE MAP

* Utilizzo di una mappa
* Alloca la mappa in un red-black ordinato in base alle chiavi

BINARY NODE

* Composto da:
  + *Left* 🡺 tipo BinaryNode
  + *Right* 🡺 tipo BinaryNode
  + *Data* 🡺 tipo E ( generico )
* *Left* e *right* a loro volta contengono degli oggetti BINARY NODE facendo si che ci sia una concatenazione
* Utilizzo spesso **chiamate ricorsive**

LINKED BINARY TREE

* Contiene la dimensione dell’albero
* Serve da “contenitore” del binary tree
  + Il binary tree vero e proprio è formato da BINARY NODE

BINARY SEARCH TREE

* Traduzione 🡺 *Albero di ricerca binaria*
* È composto da *BTSNode*
  + Left, right, element
  + Parent 🡺 peculiarità rispetto al binaryNode
* Gli elementi sono ordinati