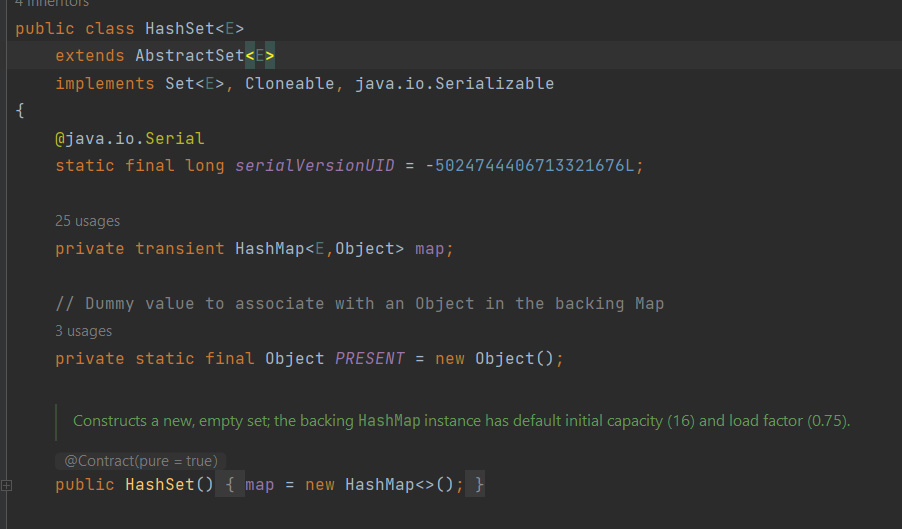


* Deci, anume si astea sunt diferentele intre Set si List:

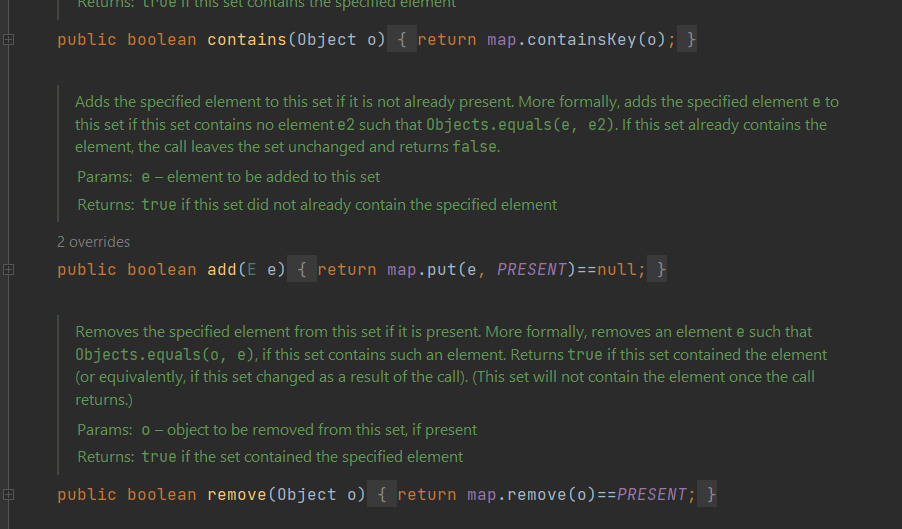
1. In List itemele sunt stocate intr-o ordine particulara, in Set nu
2. In List putem stoca iteme duplicate, in Set nu.

**HashSet**

* Implementeaza interfata Set
* Foloseste hash function si un array
* hash function tot calculeaza index pentru un element in array
* Initial Capacity = 16
* Putem folosi for each deoarece Set e o collection
* Tot foloseste red black tree pentru collisions, deci complexitatea in medie e O(logN)
* Poate fi inserat si null
* Functioneaza pe baza logicii de la HashTable
* **Defapt, el foloseste un HashMap<> object pentru a stoca elementele, deci desi e o structura de date separata, foloseste HashMap, alta structura, pentru a pastra elementele sale.**
* HashMap are nevoie de key value, dar HashSet insereaza doar valoarea trimisa de noi ca key, si value va fi un Object object , deci toate itemele inserate in acest HashMap vor avea doar keys diferite, dar acelasi object ca value









* E comod de a folosi HashMap intern deoarece HashMap deja are implementata logica de a asigura ca keys sa fie mereu unice, si e rapid.
* El nu compara daca 2 obiecte sunt identice doar dupa referinta, ci si dupa componenta lor, folosind hashCode() si equals(). Deci, daca adaugam gen 2 liste diferite ca obiecte, dar cu iteme identice, inclusiv ca ordine, va fi adaugata doar o lista
* Asa cum implementeaza Collection, implementeaza si Iterable, deci poate fi folosit in for each direct, caci care metoda iterator()
* Daca adaugam un element care deja exista, acesta pur si simplu va fi ignorat
* HashSet e mai lent ca HashMap
* **Constructori**:

HashSet() – initial capacity = 16 load factory = 0.75

HashSet(Collection)

HashSet(InitialCapacity)

HashSet(InitialCapacity, LoadFactory)

* **Metode**:
* removeAll(Collection) – sterge toate itemele ce sunt prezente in hashset si collection
* retainAll(Collection) – retine doar elementele din hashset care sunt prezente si in Collection. Restul sunt eliminate.

....

**LinkedHashSet**

* Extinde HashSet
* Este exact ca HashSet, doar ca memoreaza ordinea de inserare a elementelor.
* **Constructori**:

LinkedHashSet() – initial capacity = 16 load factory = 0.75

Linked HashSet(Collection)

LinkedLinkedHashSet(InitialCapacity)

HashSet(InitialCapacity, LoadFactory)

**TreeSet**

* Implementeaza interfata SortedSet
* Tot implementeaza RedBlack Tree
* Nu accepta null
* E bun pentru a stoca cantitati mari de date ce trebuie sortate si sa fie unice.
* **Acesta foloseste intern un TreeMap, si fiecare obiect adaugat de noi va fi key, insa ca value va pune un obiect la toate keys.**
* **Constructors**:
* TreeSet()
* TreeSet(Comparator)
* TreeSet(Collection)
* TreeSet(SortedSet)
* **Methods**:
* ceiling(Item) – returneaza primul element mai mare ca Item sau egal cu el
* floor(Item) – returneaza cel mai mare element care e mai mic sau egal cu Item
* comparator() – returneaza comparatorul folosit
* descendingIterator()
* descendingSet()
* first() – returneaza primul element, cel mai mic adica
* last() – returneaza ultimul element, cel mai mare adica
* headSet(Item) – returneaza elementele care sunt mai mici ca Item intr-un nou TreeSet
* tailSet(Item) – returneaza elementele care sunt mai mari ca Item intr-un nou TreeSet
* higher(Item) – returneaza primul element care e strict mai mare
* lower(Item) – returneaza primul element care e strict mai mic
* pollFirst() – returneaza si elimina primul element
* pollLast() – returneaza si elimina ultimul element
* subsSet(Object From,Object To) – returneaza un TreeSet ce contine elementele de la From(inclusiv) pana la To(exclusiv)

**Sets Comparisons**

