8. Datové struktury s klíčem, indexem a hashované

Datové struktůry s klíčem

```
- HashMap
- LinkedHashMap
- TreeMap
- HashTable
```

HashMap

```
Do mapy se ukládá dvojice objektů: klíč a k němu přiřazená hodnota.
Často se mapa používá i v situaci, kdy chceme zajistit rychlý přístup k
prvkům seznamu dle klíče – např. můžeme vytvořit mapu, kde klíčem bude
číslo účtu a hodnotou bude instance třídy Ucet.
Použití map má jedno logické omezení – v mapách nemohou být duplicitní
klíče.
Umožňuje uložit null jako jeden klíč a null jako hodnoty.
Nesezařená a nezachovává pořadí.
Pro vyhledávání a vkládání je časová složitost O(1).
Není thread-safe
- put(K key, V value)
- get(Object key)
- remove(Object key)
- ContainsValue(Object value)
- ContainsKey(Object key)
```

LinkedHashMap

```
Dědí od HashMap.Také dovoluje jeden null jako klíč a null jako hodnoty.
Zachovává pořádí vložení.
Použití když záleží na pořadí prvků podle vložení.
Pro vyhledávání a vkládání je časová složitost O(1).
Nejlépe z ní vytvořit objekt SynchronizedHashMap kvůli thread-safe.
Metody se shodují s HashMap
```

TreeMap

```
Implementuje rozhraní Map a SortedMap.
Nepovoluje null hodnoty.
Řadí klíče podle jejich přirozené hodnoty.
Pro vyhledávání a vkládání je časová složitost O(logN).
Není thread-safe.
Metody uvedené u HashMapy se shodují a jsou některé přidané:
- firstEntry()
- firstKey()
- lastEntry()
- lastKey()

Nově možnost se lépe navigovat v mapě pomocí lowerEntry() and higherEntry()
```

HashTable

```
Nejhorší časová složitost O(n), z dvou důvodů:
- Jestliže je moc elementů nahashováno na jeden klíč, pohled do tohoto
klíče může zabrat O(n) času
- Jakmile HashTable přejde svou optimalizaci musí se přehashovat(udělat
vetší tabulku a překopírovat každý prvek do tabulky)
Je thread-safe za cenu pomalého operovaní.
```

Indexy

```
- Array
- ArrayList
- LinkedList
```

Array

```
Jedná se o nejprimitivnější datovou struktůru.Může být
jednorozměrné,dvourozměrné a vícerozměrné.
Jedná se o struktůru do které lze snadno vkládat hodnoty stejného typu, pod
podmínkou definování velikosti pole.Velikost jde změnit v Javě a v php, ale
není to výsada u programovacích jazyků.První index má hodnotu 0.Struktura
je na haldě o velikosti 4,8,16,...
Ukazatel na pole ukazuje na index 0, číslo napsané do [] určí o kolik se má
ukazatel posunout.
Plusy
- Pro uložení počtu předem daných prvků je nejefektivnější
- Umožňuje přímo ukládat primitivní datové typy, u ostatních struktur nutno
převést na objekty.
- je možné vytvářet jednorozměrná i vícerozměrná pole.
Mínusy:
- Pole má pouze k dispozici metody třídy Object
- Nutnost určit počet prvků
- Problém neexistence prvků
```

ArrayList

```
Implementace List rozhraní.Implementuje všechny možné operace a je
generický včetně null.Tato třídá umožnuje metody na manipulaci velikosti
pole, které je vnitřně používané na ukládání listu.
Size, isEmpty, get,set, iterator, listIterator běží v konstatním čase.
Add operaci běží v amortizovaním konstatním čase.To znamená přidávání n
elementu potřebuje O(n) času.Ostatní operace běží v linearním čase.
Konstatní faktor je nízky oproti implementaci LinkedListu.
Každá aplikace může zvýšit kapacitu ArrayListu  než bude přidávat velké
množství pomocí ensureCapacity operace.
```

LinkedList

```
Implementace rozhraní Listu pomocí dvojitého Linked listu s pomocí
Deque.Implementuje generika i null.
Každá operaci s indexem půjde k indexu bud od začátku nebo konce podle toho
co je blíže.
Tato kolekce není synchronized.
Iterátor vyhozený touto kolekcí jsou fail-fast, jesltliže list je
strukturálně modifikován v jakémkoliv čase po vytvoření iterátoru, v
jakékoliv možnosti mimo iterator metodu add-remove, iterátor vyhodí
`ConcurrentModificationException`
```

Hash

```
- HashSet
- TreeSet
```

Souhrn

```
Výhody hashovaných kolekcí oproti listu je rychlejší najití prvku, list
musí iterovat (procházet jednu za jednou), ale u hashe se spočítá hashcode
poté se půjde do bucketu a tam se pokusí najít stejný hashcode. Ze stejného
důvodu je rychlejší i Remove()
List má rychlejší přídání prvku, u hashe se musí vypočítat hashcode a
bucket, list ho prostě přidá nakonec
```

HashSet

```
Jeho počáteční velikost je 16 a když přeskočí load factor o 0.75 tak se jeho velikost zdvojnásobí.
Implementuje rozhraní Set založené na HashMap instanci.Není jisté uspořádání při iteračním průchodu.
Nabízí také konstatní výkonost při normálních operacích add,remove,size,contains, při hashovací funkci mezi buckety.Iterační čas náleží součtu HashSetové velikosti.
Není thread-safe.
Iterátor vyhozený touto kolekcí jsou fail-fast, jesltliže list je strukturálně modifikován v jakémkoliv čase po vytvoření iterátoru, v jakékoliv možnosti mimo iterator metodu add-remove, iterátor vyhodí `ConcurrentModificationException`
```

TreeSet

```
Uspořádaný NavigableSet založený na TreeMap.Uspořádanost je pomocí jejich přirozené vlastnosti nebo Comparatoru.
Tato implementace garantuje log(n) časovou složitost na základní operace(add,remove,contains)
Není thread-safe.
Nabízí metody jako first(),last(),headSet(),tailSet()
```

HashSet vs TreeSet

	HashSet	**Treeset**
Rychlost	Rychlejší za cenu neuspořádanostiO(1)	pomalejší O(log (n)
Null povolen	null objekty	nedovoluje jelikož používá compareTo
Internal	HashMap	NavigableMap
Porovnání	equals()	compareTo()
Uspořádání	Neuspořádané	Uspořádané pomocí Comparable nebo Comparator
Heterogenní objekty	dovoluje	Nedovoluje vyhodí classCastException