**Sorting Arrays**

* Java foloseste **QuickSort** pentru array de date primitive
* Java foloseste **MergeSort** pentru array de obiecte
* Folosim doar **Arrays.sort(array)**
* Evident, daca nu e vorba de date primitive, obiectele trebuie sa implementeze **Comparable**, sau sa folosim:

**Arrays.sort(array,Comparator)**

**Sorting Collections**

* Pentru a sorta colectii, folosim clasa Collections. Atentie! Nu Collection, care e interfata!
* **Collections.sort(List)** – sorteaza elementele din lista. sort() poate sorta doar List. Elementele din lista trebuie sa implementeze Comparable
* **Collections.sort(List,Comparator) –** sortam pe baza la un Comparator, de ex

**Collections.sort(List,Collections.reverseOrder) –** sortam invers. Nu e nicio problema daca itemele din lista implementeaza Comparable si dam si un comparator.In asa caz, e folosit Comparator. Collections.reverseOrder va necesita oricum ca elementele sa implementeze Comparable, dar daca cream noi unul, nu va fi neaparat necesar ca elementele sa implementeze Comparable.

**Arrays Class**

* Toate metodele din aceasta clasa sunt statice si ne pot ajuta sa facem anumite lucruri ce ar fi greu de facut cu tablouri simple.
* Nu putem crea propriu zis obiecte de tip Arrays,, dar putem sa utilizam metodele din ea.
* **Arrays.asList(array)** – returneaza un obiect de tip List bazat pe valorile lui array. E important sa trimitem Arrays de tip obiect, nu date primitive, caci lista contine un sir de obiecte, si un tablou primitiv are date primitive in el, el fiind doar un singur obiect de date primitive. De altfel, va trebui sa definim lista ca de tip int[] de ex, daca e int, si ea va avea doar un element care va si si tabloul de tip int

Integer[] tablou = {1,2,3,4,5};

List <Integer> lista = Arrays.*asList*(tablou);

System.***out***.println(lista);

**int**[] tablou = {1,2,3,4,5};

List <**int** []> lista = Arrays.*asList*(tablou);

System.***out***.println(lista); // returneaza locatia in memorie a tabloulu

* **Arrays.asList(el1,el2,el3,..,eln)** – returneaza un list creat din toate elementele date.
* **Arrays.binarySearch(array, element)** – returneaza pozitia elementului gasit. Daca nu il gaseste, va returna cu semnul minus pozitia unde ar fi trebuit sa fi fost elemental daca s-ar fi gasit in array, dar pozitia va incepe de la -1, nu de la 0. Trebuie de folosit metoda **Arrays.sort(array)** inainte de a o utiliza, de altfel rezultatul va fi unul gresit.
* **Arrays.binarySearch(array, poz1,poz2,element) –** cauta elememtul de la poz1 pana la poz2
* **Arrays.compare(array1,array2) –** nu compara lungimile lor, ci compara intai primele lor elemente(pozitia 0), si daca sunt egale, compara urmatoarele(pozitia 1) si tot asap ana gaseste prima pereche de elemente ce nu sunt egale. Daca array1[x] > array2[x], returneaza 1, in caz contrar -1
* **Arrays.compareUnsigned(array1, array2)** – compara tablouri de date primitive fara a tine cont ca sunt pozitive sau negative.
* **Arrays.copyOf(array, nr)** – returneaza un nou Array ce contine un numar de **nr** elemente, primele fiind cele din array de copiat. Daca array de copiat are mai putine elemente decat nr, noul array returnat va adauga 0 unde nu a avut ce copia(pentru tabluri de tip date primitive) sau null(pentru tablouri de tip obiecte)

**int**[] a = {1,2,3,4,5};

System.***out***.println(Arrays.*toString*(Arrays.*copyOf*(a,8)));

* **Arrays.copyOfRange(array,poz1,poz2) –** copie elementele lui array de la poz1 pana la poz2, fara poz2. Daca poz2 e mai mare ca nr de elemente a lui array, ele iar vor fi inlocuite cu 0 sau null
* **Arrays.toString(array)** – returneaza o reprezentare cu toate elementele din array.
* **Arrays.deepToString(array)** – returneaza o reprezentare la toate elementele din array, dar daca e multidimensional, va arata oricum fiecare element. toString, in caz de un array multidimensional, ar arata doar locatiile fiecarui tablou, dar aceasta metoda arata si elementele fiecarui tablou di tabloul de tablouri.
* **Arrays**.**equals(array1,array2)** – compara fiecare element din arrayuri, si daca gaseste 2 care nu sunt egale, returneaza false, in caz contrar true
* **Arrays.fill(array,valoare)** – umple array de elemente valoare, adica va contine in el doar un elemente pe toate pozitiile.
* **Arrays**.**hasCode(array)** – returneaza un HashCode
* **Arrays**.**mismatch(array1,array2)** – returneaza prima pozitie in care array1 si array2 au elemente ce difera
* **Arrays**.**sort(array,**comparator**(optional)) –** sorteaza array
* **Arrats.parallelSort(array)**
* **Arrays**.**sort(array,poz1,poz2)** – sorteaza array de la poz1 pana la poz2, fara poz2, restul elementelor pastrandu-si pozitia
* **Arrays**.**sort(array,poz1,poz2,comparator)** – sorteaza array de la poz1 pana la poz2 pe baza la comparator, fara poz2, restul elementelor pastrandu-si pozitia
* **Arrays.stream(array) –** returneaza un stream cu toate elementele lui array

**Sorting With Lambda**

public class Test {  
 public static void main(String[] args) {  
 List<Student> list = new ArrayList<>();  
  
  
 list.add(new Student("Mititiuc","Eduard",20));  
 list.add(new Student("Goncear","Alexandru",18));  
 list.add(new Student("Turcan","Petru",14));  
 Collections.*sort*(list,(o1,o2)->Integer.*compare*(o1.getAge(),o2.getAge()));  
 System.*out*.println(list);  
 }

Asa nici nu ne trebuie Comparator sau Comparable

Sau:

Collections.*sort*(list,Comparator.*comparing*(Student::getAge));

Oferim metoda asupra la care vrem sa se efectueze compararea metodelor

Caci asa lucreaza ea:

public static <T, U extends Comparable<? super U>> Comparator<T> comparing(  
 Function<? super T, ? extends U> keyExtractor)  
{  
 Objects.*requireNonNull*(keyExtractor);  
 return (Comparator<T> & Serializable)  
 (c1, c2) -> keyExtractor.apply(c1).compareTo(keyExtractor.apply(c2));  
}

**(Vezi Lambda Expression din Folderul Java)**

Collections.*sort*(list,Comparator.*comparing*(Student::getAge).reversed());

Collections.*sort*(list,Comparator.*comparing*(Student::getAge).thenComparing(Student::getFirstName));

thenComparing() se foloseste cand nu s-a putut sorta dupa prima metoda daca s-au intalnit valori identice.

**Collections Class**

* Contine doar metode statice
* Pot fi efectuate asupra la Collection
* **Static Methods**:
* addAll(Collection,el1,el2,el3,..,elN)
* reverse()
* emptyEnumeration() – returneaza o Enumeration goala
* emptyIterator() – returneaza un Iterator gol
* emptyMap/NavigableMap()
* emptySet/NavigableSet()
* asLifoQueue(Dequeu) – transforma o deque in queue
* binarySearch(List,element)
* copy(List destination,List list)
* min/max(Collection) – fiecare element trebuie sa implementeze Comparable
* disjoint(Collection,Collection) – returneaza true daca cele 2 collectii nu au nici un element in comun
* min/max(Collection,Comparator)
* shuffle() – modifica ordinea elementelor randomly
* rotate(Collection, size) – deci, va face ca ficare element sa fie impins cu size pozitii. De ex:

1 2 3 4 5 6

rotate(1)

6 1 2 3 4 5

Sau rotate(2) la 1 2 3 4 5 6:

5 6 1 2 3 4

* enumeration(Collection) – returneaza o Enum cu elementele din collectie
* fill(List,Element) – inlocuieste toate elementele cu Element
* frequency(Collection,Element) – returneaza de cate ori Element e prezent in lista
* list(Enumeration) – returneaza un ArrayList ce contine toate elementele din Enumeration in ordinea aparitie lor
* nCopies(int n,T t) – creaza o lista imutabila in care pune elementul t de n ori
* replaceAll(List, old, new) – inlocuieste toate elementele old cu new in lista
* singletonMap(key,value) – returneaza un immutable Map cu key si value specificata
* singleton(T o) – un set immutabil ce are acel element
* singletonList(T o) – o lista imutabila formata doar din o element
* sort(List)
* sort(List, Comparator)
* swap(List, element1, element2) – schimba elementele cu locul
* synchronizedList/Map...(...) – returneaza o noua List/Map care este synchronized
* unmodifiableCollection/List/Set/Map(Collection/List/Set/Map) – returneaza o noua lista ce nu poate fi nicidecum modificata.

**Unmodifiable collections**

* O unmodifiable collection este o colectie asupra careia nu pot fi efectuate metode de adaugare, stergere sau replace. Putem doar sa le accesam
* unmodifiableCollection
* unmodifiableList
* unmodifiableSet
* unmodifiableMap
* unmodifiableNavigableMap
* unmodifiableNavigableSet

**Synchronized Collections**

* Synchronized sunt: Vector, Stack si HashTable
* synchronizedList/Map...(...) – returneaza o noua List/Map care este synchronized . Putem da ca parametru si Collection a noastra si se va crea una noua cu toate elementele din cea data de noi