Test č. 2, úloha č. 1

Napíšte generickú triedu `ComparableArrayList<E extends Comparable<E>>` v nepomenovanom balíku, ktorá bude rozširovať triedu `ArrayList<E>` z balíka `java.util` a súčasne implementovať rozhranie `Comparable<List<E>>`. Pôjde o „vylepšenú“ verziu dynamických polí navzájom porovnateľných prvkov – tieto polia bude možné porovnávať s ľubovoľným zoznamom prvkov rovnakého typu. Pri porovnávaní zoznamov sa bude používať variant lexikografického usporiadania opísaný nižšie.

Vaša trieda by mala poskytovať tri konštruktory, ktoré sa budú správať *rovnako* ako príslušné konštruktory triedy `ArrayList<E>`:

- Konštruktor `public ComparableArrayList()`, ktorý vytvorí prázdne pole.
- Konštruktor `public ComparableArrayList(int initialCapacity)`, ktorý vytvorí prázdne pole, pričom jeho kapacitu (objem interne alokovanej pamäte) nastaví podľa argumentu `initialCapacity`.
- Konštruktor `public ComparableArrayList(Collection<? extends E> c)`, ktorý vytvorí zoznam obsahujúci prvky zoskupenia `c`.

Navyše by vaša trieda mala poskytovať metódu `public int compareTo(List<E> o)` realizujúcu porovnanie danej inštancie vašej triedy so zoznamom `o`. Toto porovnanie by sa malo diať vzhľadom na lexikografické usporiadanie na zoznamoch typu `E`, ktoré je definované podobným spôsobom ako prirodzené usporiadanie na reťazcoch typu `String`. Dané dva zoznamy `a`, `b` sa teda pri tomto usporiadaní porovnávajú nasledovne:

(*i*) Ak existuje aspoň jeden index `i` z rozmedzí oboch zoznamov taký, že prvok `a.get(i)` sa nerovná prvku `b.get(i)`, uvažujme najmenší taký index `m`. Zoznam `a` je potom menší resp. väčší ako zoznam `b` práve vtedy, keď je prvok `a.get(m)` vzhľadom na prirodzené usporiadanie na `E` menší resp. väčší ako prvok `b.get(m)`.

(*ii*) Ak takýto index neexistuje a zoznamy nie sú rovnakej dĺžky, vyhlási sa za väčší v lexikografickom usporiadaní ten dlhší z nich.

(*iii*) Ak nenastane ani jeden z predchádzajúcich dvoch prípadov, musia sa zoznamy rovnať.

Ak je argumentom metódy `compareTo` referencia `null`, vyhodí sa výnimka `NullPointerException`. V opačnom prípade metóda realizuje porovnanie danej inštancie triedy `ComparableArrayList` so zoznamom `o` z jej argumentu vzhľadom na vyššie opísané lexikografické usporiadanie na zoznamoch. Výstupom metódy je teda záporné číslo, nula, resp. kladné číslo podľa toho, či je zoznam, pre ktorý sa metóda volá, menší, rovnaký, alebo väčší ako zoznam `o`. Volanie metódy `compareTo` by nemalo spôsobiť žiadnu zmenu v porovnávaných zoznamoch.

Na testovač odovzdávajte súbor `ComparableArrayList.java` obsahujúci zdrojový kód vašej triedy. Rešpektujte konvencie jazyka Java a zásady objektovo orientovaného programovania. Tam, kde je to vhodné, použite anotáciu `@Override`.

Príklad 1. Nech `a` je inštancia typu `ComparableArrayList<Integer>` reprezentujúca dynamické pole `[1, 2, 3, 4, 5, 6, 7]`. Nech je ďalej `o` inštancia typu `LinkedList<Integer>` reprezentujúca zoznam `[1, 2, 3, 5, 4, 7, 6]`. Keďže je zoznam `a` vzhľadom na lexikografické usporiadanie menší ako zoznam `o`, bude výstupom metódy `a.compareTo(o)` záporné celé číslo.

Príklad 2. Nech `a` je inštancia typu `ComparableArrayList<String>` reprezentujúca dynamické pole `[a, bb, ccc, dddd, eeeee, ffffff]`. Nech `o` je inštancia typu `LinkedList<String>` reprezentujúca zoznam `[a, bb, ccc, dddd, eeeee]`. Keďže je zoznam `a` vzhľadom na lexikografické usporiadanie väčší ako zoznam `o`, bude výstupom metódy `a.compareTo(o)` kladné celé číslo.