8. gyakorlat anyaga

08. `List`, `Set`, `Map`

Demo

1. demó

Készítsük el a `WorkerScheduleStructureTest` szerint a megadott osztályt. Ennek adattagja leírja, hogy melyik héten kik dolgoznak. A heteket a hívó fél `1` kezdéssel számozza, de belül `0` kezdéssel tároljuk el.

Dolgozókat két `add()` művelettel adunk a munkarendhez.

- Az egyik egy specifikus héthez adja hozzá a második paraméterben megadott dolgozókat.
  - Hozzáadhatók egyenként is, de a `HashSet` osztály `addAll()` metódusával is.
  - Ha a hét még nem szerepel a `weekToWorkers` adattagban, be kell tenni dolgozók nélkül (üres halmazt hozzárendelve).
- A másik több hetet add meg, a dolgozókat mindegyikhez hozzá kell adni.
  - Ez a metódus hívjon át az előzőre.

A másik két metódus megadja, hogy egy adott dolgozó szerepel-e a munkarend egy megadott napján, illetve megadja az összes olyan napot, amelyeken egy adott dolgozó dolgozik.

- Az utóbbihoz használjuk a `HashMap` osztály `entrySet()` metódusát a kulcs-érték párok bejárásához.
  - Ez `Entry<kulcs típusa, érték típusa>` értékeket jár be (`java.util.Map.Entry`).

A következő módon tesztelendő.

- `emptySchedule`: üres munkarendben senki nem dolgozik semelyik napon
  - Itt `assertFalse(<feltétel>)` is használható, ami rövidebb, mint a szintén érvényes `assertEquals(false, <feltétel>)`.
- `schedule`: vegyünk fel néhány dolgozót mindkét `add()` használatával, és aztán próbáljuk ki a két másik metódust.
  - A második változathoz néhány `ArrayList`re van szüksége. Ezek összeállíthatók úgy, hogy minden névre meghívjuk az `add` metódust.
  - Összeállítható így is: `new ArrayList<>(List.of(name1, name2, ...))`. Ebben `List` csomagja szintén `java.util`.
  - A teszt működtethető a szokásos módon `@ParameterizedTest` segítségével is, amivel rövidebb és átfogóbb megoldás adható.

Feladatok

1. feladat

A következő feladatok kódját a `ListUtilStructureTest` szerint kell elkészíteni.

A `divisors()` metódus a paraméterként kapott nemnegatív egész szám osztóit adja vissza egy `ArrayList` adatszerkezetben.

Tesztelendő `0`, `1` és `64` bemenetekre.

2. feladat

A `withSameStartEnd()` metódus szövegeket kap meg `ArrayList` adatszerkezetben, és ezek közül azokat adja vissza egy másik `ArrayList`ben, amelyek nem üresek, nem is `null` értékek, és első és utolsó karakterük megegyezik.

A tesztelés egy üres `ArrayList`tel induljon. Erre a `withSameStartEnd()` hívásnak üres listát kell adnia. Ezután ugyanebben a metódusban sorban adjuk a listához az alábbi szövegeket; minden hozzáadás után tesztelendő, mit ad ki a hívás.

- üres szöveg (a kimenet nem változik meg)
- `null` (a kimenet nem változik meg)
- egyetlen szóközt tartozó szöveg (belekerül a kimenetbe)
- az `x` szöveg (belekerül a kimenetbe)
- üres szöveg (a kimenet nem változik meg)
- a `different start and end?` szöveg (a kimenet nem változik meg)
- az `ends and starts the same` szöveg (belekerül a kimenetbe)

3. feladat

A `maxToFront()` metódus a megkapott lista ábécérendben utolsó elemét helyezze át a lista elejére.

- A listát helyben kell megváltoztatni, ne jöjjön létre új lista.
- Ha a lista `null` vagy üres, nincsen teendő.
- Tipp: `Collections.max()` .
- Tipp: az `ArrayList` osztály megfelelő metódusaival el lehet távolítani a listából a megtalált értéket, majd vissza lehet tenni a `0` indexre.

A következő bemenetek tesztelendők.

- `null`: `assertX` jellegű hívásra nincsen szükség, csak meg kell hívni a metódust
- üres lista
- egyelemű lista
- három szöveget tartalmazó lista: `can, you, succeed`, erre az elvárt kimenet `you, can, succeed`
- három szöveget tartalmazó lista: `-123, 2000, 100`, erre az elvárt kimenet `2000, 100, -123`

4. feladat

A `CharacterStatisticsStructureTest` osztály alapján dolgozzunk.

Az osztály konstruktora egy szöveget kap meg, és az ebben szereplő összes karakterre beteszi annak előfordulási gyakoriságát az adattagba.

- Tipp: a `String` osztály `toCharArray()` metódusa segítségével könnyen elérhetők a szöveg karakterei.
- Tipp: a `HashMap` osztály `getOrDefault()` adattagja is jól használható itt. Oldjuk meg a feladatot a használatával és a használata nélkül is.

A `getCount()` metódus az adott karakter gyakoriságát adja meg.

Szintén elkészítendő egy `toString()` implementáció az osztályhoz, ami a karakterek gyakoriságának szöveges reprezentációját mutatja. A metódus kapja meg az `@Override` annotációt. A gyakoriságok az `entrySet()` sorrendjében jelenjenek meg benne.

Tesztelés: az alábbi szövegekre mind vizsgáljuk meg legalább egy karakter gyakoriságát.

- Kipróbálandó az az eset is, amikor a megadott karakter nem szerepel a szövegben.

A `toString()` kimenete szintén megvizsgálandó.

- A kimenetek az alábbiakhoz hasonlóak lesznek, de a karakterek sorrendje függhet a rendszertől.
- A kimenetben a szóközöknek is betűre pontosan egyezniük kell.

A szövegek a következők:

- üres szöveg: üres kimenet
- `aaaaaaaa`: `a(8)`
- `HgFeDcBa`: `a(1) B(1) c(1) D(1) e(1) F(1) g(1) H(1)`
- `a?!_#@{}`: `@(1) a(1) !(1) #(1) {(1) }(1) ?(1) _(1)`
- `Hello world!`: `(1) !(1) r(1) d(1) e(1) w(1) H(1) l(3) o(2)`
  - Itt az első karakter a szóköz ().

09. Sablon

Feladatok

1. feladat

Készítsen a `MultiSetStructureTest` szerint olyan osztályt, amely `E` típusú elemek multiplicitásos halmazát reprezenzálja. Ennek adattagja leírja, hogy milyen elemből hány darab van a halmazban. Ezt az adatszerkezetet zsáknak (bag) is szokták nevezni.

Az osztály konstruktora a paraméterül kapott elemekkel tölti fel kezdetben a zsákot.

Elemeket az `add()` művelettel adhatunk a zsákhoz. Ha az elem még nem volt benne a zsákban, kerüljön bele `1` multiplictással, ha pedig már benne volt, akkor nőjön meg eggyel a multiplicitása.

A `getCount()` a paraméter multiplicitását kérdezi le. Ha ez nem szerepel a zsákban, legyen `0` az eredmény.

Az `intersect()` két zsák metszetét állítja elő. Ebben azok az elemek lesznek benne, amelyek mindkettőben megtalálhatóak, ennek a multiplicitása a kisebbik. A metódus egyik eredeti zsákot se változtassa meg, az eredmény egy új `MultiSet` objektum legyen.

A `countExcept()` megadja a zsákban található összes elem darabszámát (multiplicitással együtt), de a paraméter elemeit nem veszi figyelembe.

A következő módon tesztelendő:

- `multiSetInteger`, `multiSetString`: a névben megadott típusú zsák
  - Adjunk hozzá pár elemet az `add()` metódussal, illetve a konstruktorban, és vizsgáljuk meg a `getCount()` eredményét.
- `intersect`, `countExcept`: hozzunk létre két zsákot, és próbáljuk ki rajtuk az `intersect()` és `countExcept()` függvényeket

2. feladat

Az `OrganiserStructureTest` alapján készítsünk olyan osztályt, amely adatokat tárol sorban az `elems` adattagjában. A másik adattag pozíciópárokat (cseréket) tartalmaz.

A konstruktor a megkapott adatok alapján feltölti az `elems` adattagot. Kezdetben cserék nincsenek beállítva.

- A konstruktor kódjára figyelmeztetést fog adni a fordítóprogram. Mivel a megvalósításban most biztosan nem fog veszélyes kód szerepelni, a konstruktor megkaphatja a `@SafeVarargs` annotációt.

A `get()` metódusnak legyen kétféle paraméterezése.

- Az egyik az `elems` egy érvényes indexét veszi át, és visszaadja az indexelt elemet. Feltételezhető, hogy a paraméter érvényes.
- A másik változat mindegyik eltárolt elemet visszaadja egy listában.
  - Vigyázzunk, hogy ne szivárogjon ki az osztály belső állapota!

Az `addSwap()` művelet egy egész számokból álló párt ad a `swaps` adattaghoz.

- Mivel a Javának nincsen beépített pár/rendezett n-es adatszerkezete, az adattag `Map.Entry` típusú elemekből áll.
  - Ez `Integer` típusparamétereket kap, mert típusparaméterek nem lehetnek primitívek.
- A Map.entry(from, to) hívással készíthető ilyen "pár".

A `swap()` segédmetódus cserélje meg a két megadott indexű elemet az `elems` adattagban.

- Ez a metódus ne látszódjon az osztályon kívül.
- Tipp: átmeneti változó tárolja el az egyik értéket.

A `runSwaps()` sorban végrehajtja a `swaps` listában leírt cseréket.

- Az egyes cserék végrehajtásához a `swap()` hívandó meg.

Az osztály szöveges reprezentációja ilyen alakú legyen: `[1 3 5 7 9 ]` vagy `[d a c b e ]`.

3. feladat

A `RangedStackSuite` alapján készítsünk vermet ábrázoló osztályt, amelybe egyszerre több elem tehető be/vehető ki.

Lehessen üresen is elkészíteni, és másoló konstruktorral is. Ez utóbbi egy másik `RangedStack` példányhoz hasonló tartalmú vermet készít el, de az adattagjaik ne egy közös listára mutassanak (aliasing).

A következő módon tesztelendő.

- `empty`: készüljön el egy üres `RangedStack` és egy másolata
  - Mindkét esetben: akárhány elemet veszünk ki (pl. `0`, `1` vagy `100` elemet egyszerre), a kimenet üres lista.
- `testString`, `testInteger`: mindkét típusra készüljön el `6` elemből egy `RangedStack` (pl. egész típus esetén a `2`, `4`, `6`, `8`, `10`, `12` elemekből), és legyen egy másolata is
  - Az eredetiből vegyünk ki sorban `1`, majd `2`, majd `3`, majd `100` elemet. Az elvárt kimenetek sorban `[12]`, `[10, 8]`, `[6, 4, 2]` és az üres lista.
  - A másolatból vegyünk ki `100` elemet, majd megint `100` elemet. Az első lépés elvárt kimenete `[12, 10, 8, 6, 4, 2]`, a másodiké egy üres listaí.