Programozási nyelvek (BSc, 18) Java 8. gyakorlat

1. feladat

a

Írjon `datastructures.Month` néven felsorolási típust amelyben a hónapok neveit tároljuk.

b

Írjon `datastructures.mutable.SetOfMonths` néven osztályt, amellyel hónapok egy halmazát reprezentáljuk
egy `short` típusú adattagban oly módon, hogy az adattag `i` -ik bitje 1, pontosan akkor, ha az `i` -edik sorszámú hónap benne van a halmazban, 0 máskülönben.

Az osztálynak legyen egy privát konstruktora, amely tárolja a paraméterként kapott, `short`
típusú adatot. Egy `SetOfMonths` objektum konstruálásához legyen az osztálynak egy `empty()`
nevű publikus metódus, amely visszatér egy olyan `SetOfMonths` objektum referenciájával,
amelyben a reprezentáló adattag értéke 0.

Írjon `add()` néven függvényt, amely egy `Month` -t betesz a halmazba. Készítse el az `add()`
egy olyan változatát is, amely a hónapok neveit sztringtömbben fogadja, és a tömbben lévő
hónapokat beteszi a halmazba. Oldja meg, hogy az `add()` függvényeket láncolva is lehessen hívni (azaz ha például `example` egy `SetOfMonths` referenciája, akkor a `example.add(May).add(September).add(October)` hatására `May` , `September` és `October`
mind az `example` halmazba kerüljön.

Írjon `contains()` függvényt, amely egy adott hónapról eldönti, hogy benne van-e a halmazban.

Írjon `remove()` függvényt, amely egy adott hónapot kivesz a halmazból. A `remove()` függvényt
az `add()` függvényhez hasonlóan lehessen láncolni.

Írjon `toString()` függvényt, amely `{hónap1, hónap2, ..., hónapN}` alakban megadja a
halmazban tárolt hónapokat.

Írjon `of()` néven variadikus függvényt, amely paraméterként tetszőleges számú hónapot
fogad paraméterként, majd megkonstruál egy ezen hónapokat tartalmazó `SetOfMonths` objektumot.

Készítsen bináris fájlból beolvasó, valamint bináris fájlba kiíró műveletet az osztályhoz. A kiírás a `short` típusú adattagot írja ki (2 bájt), a beolvasás pedig két bájtot beolvasva töltse fel a `short` típusú adattagot. Egy bináris fájlt írásra így nyithatunk meg:

```
DataOutputStream out = new DataOutputStream( new FileOutputStream( fname )
```

Kiírni egy `short` típusú adatot az `out.writeShort(aShort)` művelettel lehet.
A beolvasás hasonló.

c

Készítse el `datastructures.mutable.SetOfMonths` néven az előző osztály módosíthatatlan
(immutable) változatát. A feladat tehát, hogy az osztályt úgy kell módosítani, hogy minden hozzáféréskor a tárolt objektum nem változhat meg; módosításkor az új adatokkal
új objektum jöjjön létre.

Ebben a megoldásban `empty` egy osztályszintű adattag legyen, amely egy üres `SetOfMonths`
referenciáját tartalmazza.

https://docs.oracle.com/javase/tutorial/essential/concurrency/imstrat.html

2. feladat

Hozza létre az `UnmodifiableStringArray` típust, ami egy `String[]` tömbből módosíthatatlan
tömböt képez.

Az `UnmodifiableStringArray` konstruktorként egy `String[]` paramétert vár, ezen kívül
rendelkezzen az alábbi metódusokkal:

- `empty()` - osztályszintű metódus, egy üres `UnmodifiableStringArray` -t ad vissza
- `get(int index)` - visszaadja az indexnek megfelelő elemet, ha nincs ilyen, `IllegalArgumentException` -t dob
- `find(String str)` - visszaadja a `str` paraméterrel megegyező elemét, ha nincs ilyen, `null` -t
- `contains(String str)` - `boolean` érték, megtalálható-e `str` a tömbben
- `sort()` - visszaadja az értékeket, ABC sorrendbe rendezve
- `getAllItems()` - visszaadja a tárolt értékek tömbjét (vigyázzon arra, hogy a belső állapot ne szökjön ki az osztályból)
- `size()` - a tömb elemeinek száma
- `maxLength()` - a tömb leghosszabb elemének hossza integerként
- `minLength()` - a tömb legrövidebb elemének hossza integerként
- `allLength()` - a tömb elemeinek hossza összegezve

Készitsen teszteket minden metódusra!

1. gyakorló feladat

Készítsen `BinaryCalculator` osztályt `calculate(a, b)` osztályszintű metódussal, amelyet túlterhel
int, short, long, double, float és String tipusú párok kezelésére is!
A `calculate` metódusok hívják meg a `doCalculate(long a, long b)` metódust, amely írja ki bináris alakban:

- `a` számot
- `b` számot
- `a` AND `b` értékét
- `a` OR `b` értékét
- `a` XOR `b` értékét
- NOT `a` értékét
- NOT `b` értékét

Írjon egy `Main` osztályt, amely segítségével leteszteli a `BinaryCalculator calculate` metódusát
minden típusra!

2. gyakorló feladat (*)

a

Készítsen egy játékot, ahol hősöknek kell végigmennie ugyanazokon a pályákon, szörnyeket legyőzve!

A pálya álljon `n` szintből, minden szint `m` mezőből.
A `Map` osztály reprezentálja a pályát, tartalmazza a lehetséges mezők típusát (`Map.Field` felsoroló),
amelyek az alábbiak lehetnek:
`EMPTY, ROCK_MONSTER, PAPER_MONSTER, SCISSORS_MONSTER, JOKER.`

A `Map` osztálynak legyen egy privát, módosíthatatlan adattagja, `Field[][] levels`, amelyet a konstruktorban töltünk fel véletlenszerűen értékekkel.
A konstruktor várjon két integert, amely megadja a szintek számát, illetve az egyes szinteken a mezők számát.
Legyen egy `public Field[] getLevel(int n)` metódusa, amely az adott szint mezőit adja vissza. Figyeljünk a megvalósítás során, hogy a függvény ne járjon adatszivárogtatással!

Készítse el a `Hero` osztályt, ami a hősöket fogja reprezentálni. Minden hősnek van

- neve (módosíthatatlan)
- élete
- aranya
- nyersanyaga
- gyengéje (`Map.Field`, módosíthatatlan)
- szövetségese (`Map.Field`, módosíthatatlan)
- erőssége (`Map.Field`, módosíthatatlan)
- joker cselekvése (`Runnable`, módosíthatatlan)

A joker cselekvése az, amit minden `JOKER` típusú mezőn csinál a hős.
Hozzon létre két konstruktort a `Hero` osztályhoz,
`public Hero(String name, int health, int gold, int resources, Map.Field losingField, Map.Field friendlyField, Map.Field winningField, Runnable jokerAction)`

illetve
`public Hero(String name, int health, int gold, int resources, Map.Field losingField, Map.Field friendlyField, Map.Field winningField)`,
utóbbinál a joker cselekvés nem vált ki semmilyen hatást.

Definiáljon egy `public String toString()` metódust, ami kiírja a hős állapotát!

Hozzon létre egy `public void processField(Map.Field actualField)` metódust, amely lekezeli, hogy az adott mezőn
mi történik a hőssel:

- Ha a gyengéje, veszít egy életet
- Ha a szövetségese, kap egy nyersanyagot
- Ha az erőssége, kap egy aranyat
- Ha a `JOKER` mező, végbehajtja a joker cselekvést

Készítsen egy `Main` osztályt, amely létrehozza a `Map` pályát, majd a `Hero` hősöket
és minden szint után kiírja a hősök aktuális állapotát.
Amennyiben egy hős élete 0, vagy alatta, ne menjen tovább a pályán, kiesett.

b

Biztosítson lehetőséget egy játék aktuális állapotának elmentésére, illetve egy állapot betöltésére!
A mentés és olvasás bináris fájlba/fájlból történjen a `java.io.ObjectInputStream` illetve
`java.io.ObjectOutputStream` segítségével.
A mentés és betöltés csak a hősöket érintse, a pályát ne!

Egészítse ki a `Hero` osztályt egy harmadik konstruktorral, amely egy karakter alapján állít be
joker cselekvést a hősnek!

- H esetén növeli a hős életét
- M esetén növeli a hős aranyát
- R esetén növeli a hős nyersanyagát
- Bármi egyéb esetén nem csinál semmit a joker cselekvés

Hozzon létre egy `public Character getJokerActionCharacter()` metódust, amellyel le lehet kérdezni, hogy
az aktuálisan beállított joker cselekvésnek mi a kódja.

A program amennyiben egy argumentummal indult el, feltételezze, hogy az a betöltendő fájl neve
és a hősöket azalapján hozza létre.
Minden szint végén kérdezze meg a felhasználót, hogy szeretné-e kimenteni jelenlegi állását fájlba,
amennyiben nem, a játék folytatódjon, ha igen, a fájlnév megadásával mentse ki a hősök aktuális állapotát
és lépjen ki a programból.

3. gyakorló feladat

Egészítse ki hibakezeléssel és készítsen tesztet a 2/a gyakorló feladathoz! A tesztek írása alatt figyeljen a következők tesztelésére:

- A `Map` osztály szintjei módosíthatatlanok
- Érvénytelen értékek esetén `Exception`
- A játék szabályszerű működésének tesztelése

A játékot feketedoboz-teszteléssel tesztelje!

4. gyakorló feladat (*)

Írjon egy programot, amely szimulálja az emberek vásárlási szokásait világjárvány esetén!
Készítsen egy `Shop` osztályt, ami 3 terméket árul: maszkokat, WC papírt és kézfertőtlenítőt.
A `Shop` osztályban az elérhető termékek mennyisége integerként legyen számontartva.
A bolt, amint új szállítmány érkezik, hírleveleket küld ki vásárlóinak, akik erre feliratkoztak.

Készítsen egy felsoroló típust, `CustomerStatus` néven, a következő értékekkel: `FEELS_SAFE, NEUTRAL, PANICS.`

A `Shop` osztálynak készítsen egy-egy metódust minden termékre, amely paraméterként egy számot vár,
hogy egy vevő mennyit venne az adott termékből, és egy `CustomerStatus` értékkel tér vissza,
jelezve, hogy a vásárlás sikeres volt-e, vagy nem.

- Ha tud annyit venni, amennyit szeretne, a visszatérési érték `FEELS_SAFE`

- Ha valamennyit tud venni, de nem annyit, amennyit akarna, a visszatérési érték `NEUTRAL`
- Ha egyet sem tud venni, a visszatérési érték `PANICS`

A `Shop` mennyisége a termékekből legyen privát, kívülről ne lehessen hozzáférni, 0 alá ne mehessen!

Hozzon létre egy `subscribe(Runnable shop)` metódust a `Shop` osztálynak, ami a kapott lambdát
eltárolja.
Legyen egy `notifyCustomer()` metódus is, ami azokat a lambdákat futtatja le, amik a `subscribe`
használatával feliratkoztak.

Készitsen egy `Simulator` osztályt, amely hozzon létre `n` vásárlót. A vásárlók iratkozzanak fel
a `Shop` hírlevelére, lambdaként átadva, hogy miből mennyit szeretnének vásárolni. Minden vásárló
minden termékből próbáljon egy 1 és 10 közötti mennyiséget vásárolni, `CustomerStatus` -át mindig a
legrosszabbul sikerült vásárlás határozza meg (pl ha 2 termékből sikeresen vásárolt, de egyből nem, állapota `PANICS`).
Ezután a hívjuk meg a `Shop` `notifyCustomer()` metódusát és végül írjuk ki, hogy melyik `CustomerStatus` -ból mennyi van.