JAVA, Spring, Hibernate (EPAM)
1. forduló

Ismertető a feladathoz

Hát ez az idő is eljött. Te sem gondoltad volna, amikor kiléptél a kocsma / egyetem / főiskola / bármilyen iskola / lakásod (neked ne mondja meg senki hogyan tanuljál!) / szüleid lakása / (bármi, ami illik rád) ajtaján, hogy egyszer eljutsz eddig az elhatározásig. Most már minden tiszta, nincs kérdés a fejedben. Szoftverfejlesztő leszel. Nem is akármilyen. Mindenhez érteni fogsz, és mindenki veled akar majd dolgozni. Mint a filmekben, bármilyen UI-on elkezdesz gépelni, nem baj, ha nincs konzol, de még egy beviteli mező sem a közelben. Gyorsan gépelsz és feltörsz bármilyen rendszert.

Te is tudod, hogy célod eléréséhez a legfontosabb, hogy bejuss egy céghez, ahol csak úgy ragadni fog rád a tudás, és érdekesebbnél érdekesebb feladatokkal várnak. Kézzel fogható tudásod hiányát végtelen lelkesedéssel ellensúlyozod. Ennek az édenkertnek a kapuját a vérfagyasztó szörnyeteg őrzi, amit sokan halkan csak állásinterjúként emlegetnek. Felveszed a legszebb ruhád (fontos a jó benyomás), és nekivágsz.

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 10 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 2 pont

Egy kis csevegés után rögtön a lényegre tér az interjúztató. Te is tudod, hogy egy interjú során a majdani munkádhoz szorosan kapcsolódó, napi szinten felmerülő problémák vannak megfogalmazva kérdés formájában. Így nem lepődsz meg, hogy a Java nyelvvel és ökoszisztémával kapcsolatban próbálnak becsalogatni a szavak erdejébe. Kicsit messziről indulunk, de hát van ez így. Egy félmosoly azért kiül az arcodra: "tudtam, hogy ezt úgyis megkérdezik".

Melyik Java verzió LTS az alábbiak közül?

○ 9
○ 10
● 11
○ 12
○ 13

Magyarázat a megoldáshoz

A felsoroltak közül csak a Java 11-es verzója rendelkezik LTS-sel. Részletes roadmap a különböző Java verziók supportjáról: https://www.oracle.com/java/technologies/java-se-support-roadmap.html

2. feladat 0 / 2 pont

Az alábbi állítások közül melyek igazak egy JAVA osztályra:

Ezzel a kérdéssel kicsit megzavarnak, hiszen korábbi évek alatt felszedett lexikális tudásodon nem tudnak vele fogást találni, de úgy vagy vele, hogy ha megkérdezik, akkor lehet, hogy nem használnak semmiféle IDE-t?

☐ Az öröklődés során a konstruktor és a metódus láthatósága nem változtatható.
☑ A metódusnak a konstruktorral szemben lehet visszatérési értéke.
☐ A konstruktor a metódussal szemben nem lehet private.
☐ A konstruktort a metódussal ellentétben nem lehet túlterhelni.
☑ A konstruktor és a metódus is rendelkezik formális paraméter listával.

Magyarázat a megoldáshoz

- Az öröklődés során a konstruktor és a metódus láthatósága nem változtatható. Ez a mondat hamis, változtatható, amire ügyelni kell azonban, hogy a láthatóságot nem lehet 'szűkíteni'.
- A konstruktor a metódussal szemben nem lehet private. Ez a kijelentés hamis, ugyanis konsturktor is lehet private. Például Singleton pattern-ben találkozhatunk hasonlóval.
- A konstruktort a metódussal ellentétben nem lehet túlterhelni. Ez szintén hamis, bővebben az overloading-ról itt is olvashtasz: https://www.geeksforgeeks.org/constructor-overloading-java/

3. feladat 0 / 3 pont

Az alábbiak közül melyik tulajdonság(ok)nak kell teljesülni az equals metódusra a specifikáció szerint?

Ezen a ponton már semmiben sem vagy biztos. „Valami alkalmazott matematikai projektre jelentkeztem? De hát mondtam a HR-es körön, hogy programozni szeretnék".

☑ reflexív
☐ szálbiztos
☐ additív
☑ konzisztens
☑ tranzitív
☐ aszimmetrikus
☐ induktív

Magyarázat a megoldáshoz

A felsoroltak közül a reflexivitás, a konzisztencia és a tranzitivitás szerepel a specifikációban. A specifikáció megtalálható az alábbi linkeken:
https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html
https://course.ccs.neu.edu/cs5010f17/InterfacesClasses2/equality6.html?

4. feladat 0 / 3 pont

Állítsd sorrendbe a JVM Garbage Collectorokat megjelenési idejük szerint. Melyik a helyes sorrend az alábbiak közül, ha balról jobbra haladva egyre frissebb Garbage Collector-t szeretnénk látni?

Te jó ég, mi történik itt, hát hogy jött ez ide. Mit csinálnak ennél a cégnél, hogy a GC történelemmel is tisztában kell lenni. Mi ez itt, múzeum? Egyáltalán minek nekem GC, kezelem én a memóriát, nekem ne mondja meg senki, mit hogyan csináljak.

○ G1, CMS, Parallel, Serial
○ Az összes Garbage Collector ugyanakkor lett bevezetve.
● Serial, Parallel, CMS, G1
○ Parallel, Serial, CMS, G1
○ Serial, Parallel, G1, CMS

Magyarázat a megoldáshoz

A helyes sorrend Serial, Parallel, CMS, G1 lesz.
https://dzone.com/articles/garbage-collectors-serial-vs-0
https://www.baeldung.com/jvm-garbage-collectors

Vissza a kategóriáimhoz

JAVA, Spring, Hibernate (EPAM)
2. forduló

Ismertető a feladathoz

Egy pillanatig sem volt kérdés, sikerült az interjú. Bejutottál. Junior fejlesztő pozícióba kerültél, és kétség sem fér hozzá, hogy te vagy a cég jövőjének és sikerének a záloga. Ehhez viszont lelkesedésed mellé tapasztalatot is kell párosítani. Első projekteden beleveted magad a kód sűrűjébe. Végre kézzel fogható problémákon dolgozhatsz. Az élet szép és ennél már csak jobb lesz, hiszen a nehezén túl vagy...

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 20 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 3 pont

Az első komolyabb feladat, ami szembe jön veled megtanít a legacy kód szeretetére és arra, hogy vannak esetek, amikor exception-szintű vezérlés és System.out is elfogadható production kódban bizonyos körülmények között.

Egészítsd ki a kódrészletet az alábbi lehetőségek egyikével úgy, hogy garantáltan kiírjuk a 'Helyes válasz'-t!

```
try {
  foo();
} _____ {
  System.out.println("Helyes válasz");
}
```

○ final

○ catch

○ finalize

○ finally

● A felsoroltak közül egyik sem garantálja, hogy mindig lefut a kiíratás

Magyarázat a megoldáshoz

Elsőre a finally tűnne helyes válasznak, de ha a foo() System.exit()-et vagy Runtime.halt()-ot hív (esetleg infinite loopba kerül), nem történik kiíratás.

2. feladat 0 / 3 pont

Szerencsére a legacy kód mellett azért „újdonságokkal” is találkozhatsz az új projekteden. Vagyis hát a csapattársaid úgy hivatkoznak rá, hogy újdonság, hiszen Java 8-ban jelent meg az Optional. „Dehát az 6 éve jött ki, hogy lenne már újdonság” gondolod te. Látszik rajtuk, hogy őket már rendesen megtörte a legacy. Meg is fogadod gyorsan, hogy te sosem leszel ilyen.

Adott az alábbi kódsor:

```
Optional<Integer> possiblyExistingInteger = possiblyExistingIntegerProvi
```

A dokumentációból tudod, hogy a possiblyExistingIntegerProvider.getPossiblyExistingInteger() soha nem ad vissza null-t, de bármikor visszaadhat üres Optional-t. Az alábbiak közül melyik megoldással vagy megoldásokkalgarantálhatod azt, hogy a később hívott possiblyExistingInteger.get() nem dob NoSuchElementException-t?

☐
```
possiblyExistingInteger = possiblyExistingInteger == null ? Optional.
```

☑
```
possiblyExistingInteger = possiblyExistingInteger.isPresent() ? possi
```

☐
```
possiblyExistingInteger = possiblyExistingInteger.or(42);
```

☐
```
possiblyExistingInteger = possiblyExistingInteger.map(value -> value
```

☑
```
possiblyExistingInteger = possiblyExistingInteger.or(() -> Optional.o
```

Magyarázat a megoldáshoz

possiblyExistingInteger = possiblyExistingInteger == null ? Optional.of(42) : possiblyExistingInteger;: A feladatban leírtak alapján a possiblyExistingInteger soha nem lesz null, így az empty Optional-t sem cseréljük ki soha.

possiblyExistingInteger = possiblyExistingInteger.isPresent() ? possiblyExistingInteger : Optional.of(42);. Empty possiblyExistingInteger esetén a possiblyExistingInteger.isPresent() hamis, így a possiblyExistingInteger-hez az Optional.of(42)-t rendeljük, egyébként nem változtatunk az értékén, így a változónk biztosan nem lesz üres Optional.

possiblyExistingInteger = possiblyExistingInteger.or(42);: Ez nem fordul le, nincs ilyen metódusa az Optional-nek.

"If a value is present, apply the provided mapping function to it, and if the result is non-null, return an Optional describing the result. Otherwise return an empty Optional."

possiblyExistingInteger = possiblyExistingInteger.map(value -> value == null ? 42 : value);: Empty optional esetén nincs értékünk, ezért nem fut rá a mapping function, így az eredmény egy empty Optional lesz.

possiblyExistingInteger = possiblyExistingInteger.or(() -> Optional.of(42)); Az Optional<T> or(Supplier<? extends Optional<? extends T>> supplier) dokumentációja alapján: If a value is present, returns an Optional describing the value, otherwise returns an Optional produced by the supplying function.

3. feladat 0 / 3 pont

Éppen nyakig vagy a kódban, amikor egy kolléga odagurul hozzád, hogy éppen interjúztatni készül, és mivel te még friss vagy a cégnél, biztos emlékszel tankönyvi definíciókra a Springgel kapcsolatban. Állítása szerint ő csak használni tudja, definiálni nem ezeket a dolgokat.

Az alábbi Springben használt annotációkra vonatkozó állítások közül mely(ek) igaz(ak)?

☐ A @Bean annotáció osztály szintű

☑ A @Component annotáció osztály szintű

☐ @Bean és @Component annotáció használatához szükséges a @Configuration annotáció

☑ Megfelelően konfigurált component scan során a @Service annotációval jelölt osztályok automatikusan bekerülnek Spring context-be

☑ @Bean annotációval szeparálhatjuk a bean deklarációt és a bean implementációt

Magyarázat a megoldáshoz

A @Bean annotáció osztály szintű - Ez a kijelentés hamis, ugyanis az annotáció metódus szintű.
@Bean és @Component annotáció használatához szükséges a @Configuration annotáció - Ez szintén hamis, ugyanis a @Component annotáció használatához nem szükséges valamint a @Bean annotáció is használható @Configuration nélkül.
A többi válasz helyes.

4. feladat 0 / 3 pont

Egy idő után a Junior fejlesztő élete is repetitívvé válik, így bizonyos feladatokat igyekszel „kreatívabban” megoldani, csak hogy szórakoztasd magad.

Az alábbiak közül mely módon lehet Spring beanek életciklusát kontrollálni?

☑ InitalizingBean interface afterPropertiesSet() metódusánakfelülírása

☑ custom destroy() metódus definiálása

☑ @PreDestroy annotáció használata adott metóduson

☑ BeanPostProcessor használatával

☐ A felsoroltak közül egyik sem használható erre

Magyarázat a megoldáshoz

A felsorolt lehetőségek közül mindegyikkel befolyásolható a lifecycle, pontos használatukról itt olvashatsz:
https://www.journaldev.com/2637/spring-bean-life-cycle
https://dzone.com/articles/spring-bean-lifecycle

5. feladat 0 / 3 pont

Amióta csak elkezdtél a Java nyelvvel foglalkozni, erre a pillanatra vártál. Előkerültek a generikus adattípusok. Jól tudod, hogy amióta bevezették őket, az élet egyszerűbb, a kód olvashatóbb és persze a fű is zöldebb, de ahogy megírod, elbizonytalanodsz, és beléd hasít a kérdés.

Melyik kódrészlet fordul le az alábbi változó deklarációval?

```
List<? extends Number> numberList;
```

○
```
numberList.add(1L);
```

○
```
numberList.add((Integer) 1);
```

○
```
numberList.add((Number) 1L);
```

○
```
numberList.add((Object) 1L);
```

● Egyik sem

Magyarázat a megoldáshoz

List<? extends Number> kovariáns adattípus ezért csak olvasható. Az így definiált listákhoz elemet adni nem lehet, viszont olvasni belőlük igen.

Vissza a kategóriáimhoz

JAVA, Spring, Hibernate (EPAM)
3. forduló

Ismertető a feladathoz

Immáron kerek 1 éve vagy a cégnél, ami úgy röppent el, mint az a RuntimeException, ami bedöntötte a céges rendszert órákra (értsd: észrevétlenül) és már úgy érzed, egyre több mindennel találkoztál. Talán eddigi karriered legfontosabb eseménye, hogy részt vettél a karácsonyi csapatépítő bulin is, ahol ünnepélyesen bejelentették az év végi bónuszokat és örömmel fogadtad, hogy lelkesedésed és jó teljesítményed alapján Te is részesültél némi juttatásban. Eddig azt hitted, hogy a bónusz olyan dolog, mint a Yeti meg a Loch Ness-i szörny. Beszélnek róla sokan, de még valójában senki sem látta. Pár sör elfogyasztásán kívül a buli többi része jótékony homályba merül, és az újévi fogadalmak után teljes erőbedobással kezded az új esztendőt.

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 20 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 4 pont

Máris az év első munkanapján sok feladat vár rád, mert kiderült, hogy egy nagy projektet nyert a cég. Mivel már kezdesz hírnevet szerezni magadnak, te is rákerülsz. A projekten nagy hangsúlyt fektetnek az adatbázis tervezésre, így juniorként utána kell nézned pár dolognak.

Egy Hibernate entitás alosztályait a következőképpen szeretnéd relációs adatbázisban tárolni: A közös ős osztály adatait közös táblában, az alosztály specifikus adatokat külön táblában. Az alábbiak közül melyik annotációt alkalmazod az ősosztályon?

○ `@Inheritance(strategy=InheritanceType.SINGLE_TABLE)`

● `@Inheritance(strategy=InheritanceType.JOINED)`

○ `@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)`

Magyarázat a megoldáshoz

`SINGLE_TABLE - Tábla / osztály hierarchia`

`TABLE_PER_CLASS - Tábla / konkrét entitás osztály`

`JOINED - Az alosztály specifikus adatok külön táblába kerülnek es join müvelet segitségével lesznek beolvasva`

2. feladat 0 / 4 pont

A karácsonyi bulin megkérdőjelezhető állapotban megkérdőjelezhető döntést hoztál. Megígérted, hogy segítesz a következő karácsonyi buli szervezésében és előkészületeiben, ahol a terv az, hogy a cég kedveskedni szeretne a dolgozóinak azzal, hogy az összes ott dolgozó gyerekeinek a nevét elhelyezik a karácsonyfán. Mivel még óvodában is Stream volt a jeled, úgy döntesz, hogy ott segítesz, ahol csak tudsz.

Ehhez szükség lenne a nevek megállapítására. Az adatbázisban az adat megtalálható, már csak ki kell nyerni valamilyen formában, de egy sor nagyon hiányzik. Melyik a helyes?

(a getChildren() Stream<Child>, a hasChidlren() pedig Boolean típussal tér vissza)

```
Stream<String> childNames = company
.getEmployees()
<helyes válasz>
.map(Child::getName);
```

☐ `.map(Employee::getChildren)`

☐ `.filter(Employee::hasChildren)`

☑ `.flatMap(Employee::getChildren)`

☐ `.collect(Collectors::toList)`

☑ Egyik válasz sem helyes.

Magyarázat a megoldáshoz

Mivel a getChildren() Stream<Child>-al tér vissza, így ezen a ponton egy Stream-ben Stream-el állunk szemben, flatMap hívása szükséges, hogy a Child objektumokat megkapjuk egy kisimított Stream-ben, amelyeken már a következő .map művelettel dolgozni tudunk.

3. feladat 0 / 4 pont

Az új projekttel jól haladtok és már az 1.0 verziót ki is raktátok production környezetbe, ahol a felhasználok már el is kezdték használni az éles rendszert. Folyamatosan monitorozzátok az alkalmazást, viszont a kollégád, aki az adatbázis dashboard-ért felel, éppen szabadságon van. A vezető fejlesztő hozzád fordul (hiszen Te múltkor nagyon beleástad magad az adatbázisok rejtelmeibe), hogy nézd át, minden rendben van-e, vagy esetleg kell-e finomhangolni a beállításokat. Te örömmel vállalod a kihívást, hiszen már több lekérdezést is megírtál teljesen egyedül (na jó, kis segítséggel), ezért vitán felül jogosan érzed expert-nek magad.

A következőt látod az applikáció adatbázis kapcsolatainak listázásakor. A maximum kapcsolatok száma 5-re, a minimum 3-ra van állítva. Ha érkezik egy új tranzakció, mi fog történni?

```
Connection -> UPDATE -> Mysql
Connection -> IDLE   -> Mysql
Connection -> SELECT -> Mysql
Connection -> SELECT -> Mysql
```

○ Új kapcsolat nyílik az adatbázis felé.

● Felhasznál egy meglévő IDLE kapcsolatot az adatbázis felé.

○ Vár, amíg végez a leggyorsabb tranzakció.

○ Hibát dob és nem tudja végrehajtani a tranzakciót.

Magyarázat a megoldáshoz

Az IDLE azt jelzi, hogy van olyan élő kapcsolat az alkalmazás és az adatbázis között, amely még nem zárult le (pl. nem telt le a timeout) es teljesen szabad, így ha új tranzakció érkezik, akkor az ezen a kapcsolaton kerül végrehajtásra.

4. feladat 0 / 4 pont

Egyik nap, mikor bemész az irodába, azzal fogadnak, hogy nézz rá egy kódrészletre, amit Te írtál még akkor, amikor a céghez kerültél, mert nem teljesen értik a működését, és ki kellene egészíteni egy új funkcióval. Ezen kicsit fel is húzod magad, hiszen íratlan szabály, hogy régebben írt kódot nem illik felhasználni az ember ellen, de mindennek megvan a maga helye és ideje... Ránézel és tökéletesen visszaemlékszel arra a pillanatra, amikor lelkesen a Java verziók újdonságait olvastad és szeretted volna megmutatni hozzáértésedet szinte minden kódsorban. Az az érzés is tisztán feldereng, amikor a kód megírása után hátradőltél, és elégedetten biccentettél. Mondjuk ki, a saját hatásod alá kerültél. Ennek a kódnak sajnos az idő nem tett jót, és így utólag visszatekintve rájössz, hogy túltoltad egy kicsit.

Adott "A" osztály implementálja B és C interfészeket. B és C interfészek a következőképpen definiáltak:

```
public interface B {
    default void m(){
        System.out.print("B.m");
    }
}

public interface C {
    default void m(){
        System.out.print("C.m");
    }
}
```

Válaszd ki a helyes állítást az alábbiak közül:

☐ Az A.m() metódus meghívásakor a következő jelenik meg a kimeneten: "B.m".

☐ Az A.m() metódus meghívásakor a következő jelenik meg a kimeneten: "C.m".

☐ Az A.m() metódus meghívásakor a következő jelenik meg a kimeneten: "B.mC.m".

☑ A kód fordítási hibát jelez, "A" osztálynak implementálnia kell az "m" metódust.

☑ A kód fordítási hibát jelez, "A" osztály nem implementálhat két interfészt azonos default metódussal.

☐ A kód futás idejű hibát jelez, "A" osztálynak implementálnia kell az "m" metódust.

Magyarázat a megoldáshoz

Mivel mind a két intérfeszben van azonos nevű ("m") default metódus és az A osztály mind a kettőt implementálja igy már forditási idöben kiderül, hogy a kettö közül nem eldönthetö melyik implementació kerüljön az "m" metòdus mögé.

5. feladat 0 / 4 pont

Eljött a nagy pillanat, amire régóta vártál. Az egyik senior kollégád odamegy hozzád és megkér, hogy küldj át neki egy interjúztatós feladatot, mert egy hasonló tapasztalatú jelöltet szeretnének felvenni, mint te. Összeszeded minden tudásod, és visszagondolsz a felhőtlen gyermekkorodra, diákévekre, az egyetemre, az első napjaidra a munka világában, arra, hogy mi az, amit maximum az ellenségeidnek kívánnál egy interjún, és materializálod egy feladatba, ami a fejedben van.

Válaszd ki az **IGAZ** állításokat az alábbiak közül!

☐ Egy k elemű tömb n-edik elemének lekérdezése O(n) komplexitású művelet.

☐ Az ArrayDeque példányok szálbiztosak.

☑ Tetszőleges elem törlésében a LinkedList hatékonyabb, mint az ArrayList.

☐ A HashSet iterátora a természetes rendezés szerinti sorrendben adja vissza a tárolt elemeket.

☑ Az a.compareTo(b) hívás egy negatív számot ad vissza, ha b > a.

Magyarázat a megoldáshoz

- Egy k elemű tömb n-edik elemének lekérdezése O(n) komplexitású művelet - hamis, O(1) komplexitású

- Az ArrayDeque példányok szálbiztosak - hamis, Javadoc egyértelműen jelzi, hogy nem szálbiztosak

- Tetszőleges elem törlésében a LinkedList hatékonyabb, mint az ArrayList - igaz, LinkedList törlés: O(1) - ArrayList törlés: O(1)-O(n) attól függően, hogy honnan törlünk

- A HashSet iterátora a természetes rendezés szerinti sorrendben adja vissza a tárolt elemeket - hamis, véletlenszerűen adja vissza a tárolt elemeket

- Az a.compareTo(b) hívás egy negatív számot ad vissza, ha b > a. - igaz, 0-át ha egyeznek es pozitívot ha b < a

Vissza a kategóriáimhoz

JAVA, Spring, Hibernate (EPAM)
4. forduló

Ismertető a feladathoz

3 éve vagy a cégnél. Egyre több dologra látsz rá, a pozíciód megjelölése mellől észrevétlenül eltűnt a „Junior” jelző. Teljes értékű fejlesztővé váltál. 2-3 kisebb projekten is részt vettél, tanultál sok elméleti dolgot, elsajátítottál már 3-4 tervezési mintát gyakorlatban. Még a UI-osok misztikus világába is belekóstoltál már: CSS-t írtál megkérdőjelezhető minőségben (rounded corner és !important voltak a legjobb barátaid, soha nem hagytak cserben). Megbeszélések során már nem csak a szoba dekorációjának tekintenek. Van már hangod, még ha néha rekedtes is, de már próbálgatod hallatni és néha meg is hallják. Elkezdesz nyitni a szoftverfejlesztői társadalom támogatástól duzzadó, bár nyomokban intrikákat is tartalmazó intézménye felé. Szociális életed egyre jobb (mármint végre van), mondhatni szárnyal, és akár még a kollégáiddal is elmész néha egy ebédre, ahol jól is érzed magad. Ki hitte volna, hogy a JVM-en túl is van élet?

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 35 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 6 pont

Refaktorálást kezdeményeztek egy legacy alkalmazásnál, mert már nagyon nehéz karbantartani a kódot és lassan már csak Te értesz hozzá, hiszen akkor íródott, amikor a céghez kerültél. Itt találkozol először azzal a jelenséggel, hogy a refaktorálás leginkább neked fontos. A business másképp látja a dolgokat. Meg amúgy is ki az a business és miért szól bele a dolgaimba, ha nem is ért a kódhoz. Itt már kicsit elkalandoztál, vissza az alapproblémához, bizony, 3 és fél év és legacy lett a kódod. Az egyik junior kollégád odahív, hogy segíts neki buildelni és elindítani az appot.

Adott egy ClassPathXmlApplicationContext-et használó Spring app, a követekző XML konfigrációval:

```
<?xml version="1.0" encoding="UTF-8"?>
 <beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans"
        "http://www.springframework.org/schema/beans/spring-beans.xsd"
        "http://www.springframework.org/schema/context"
        "http://www.springframework.org/schema/context/spring-context.xs
  <context:property-placeholder />
  <bean class=com.epam.fun.SomeClass>
    <constructor-arg-value="${some-value}"/>
  </bean>
 </beans>
```

`Továbbá a következő SomeClass osztállyal:`

```
public class SomeClass {
   private String someValue;

   public SomeClass(String someValue){
     this.someValue=someValue;
   }
}
```

Mivel az appottöbbféle környezetben kellenefuttatni, szeretnéd a SomeClass bean someValue mezőjének értékét egy környezeti változó értéke alapján megadni. Lehetséges ez? Ha igen, mi lehet a környezeti változó neve?

☐ some.value

☐ SOMEVALUE

☑ SOME_VALUE

☑ SOME-VALUE

☑ some-value

☑ Sajnos ez nem lehetséges.

Magyarázat a megoldáshoz

A Spring ClassPathXmlApplicationContext -jében regisztrálva van a SystemEnvironmentPropertySource, amely támogatja a relaxed binding-et.

— Specialization of MapPropertySource designed for use with system environment variables. Compensates for constraints in Bash and other shells that do not allow for variables containing the period character and/or hyphen character; also allows for uppercase variations on property names for more idiomatic shell use. For example, a call to getProperty("foo.bar") will attempt to find a value for the original property or any 'equivalent' property, returning the first found:

—

— - foo.bar - the original name

— - foo_bar - with underscores for periods (if any)

— - FOO.BAR - original, with upper case

— - FOO_BAR - with underscores and upper case

Any hyphen variant of the above would work as well, or even mix dot/hyphen variants.

2. feladat 0 / 5 pont

A refaktorálás során jó pár régi kódrészletet lecseréltél SZERINTED sokkal olvashatóbb funkcionális megfelelőjére. Azért csak szerinted, mert a „vének tanácsa” (azaz a seniorabb kollegák némelyike) finoman utalgat rá, hogy vannak megoldások, amelyek hagyományőrző megközelítéssel is ugyanolyan olvashatóak SZERINTÜK. Ezen a ponton megismerkedtél a generációs szakadékkal, ami az IT szakmában akár pár év is lehet.

Adott egy String-eket tartalmazó (List<String> elements), ebből szeretnéd kiválogatni azokat az elemeket, amelyek nem null-ok. Ezt hagyományosan annak idején így írtad:

```
List<String> nonNullElements = new ArrayList<>();

for (String act : elements) {
  if (act != null) nonNullElements.add(act);
}
```

Az alábbi opciók közül melyik viselkedik ugyanúgy, mint a fenti kód részlet:

☑

```
nonNullElements = elements.stream().filter(Objects::nonNull).collect(
```

☑

```
nonNullElements = elements.stream().filter(act -> act != null).collec
```

☑

```
nonNullElements = elements.stream().map(Optional::ofNullable).flatMap
```

Magyarázat a megoldáshoz

A "hagyományos" megoldások mellett Java 9 óta az Optional::stream-en lehet flatMap-et hívni.

3. feladat 0 / 3 pont

Behívnak egy meeting-re, ahol nagy vita folyik arról, hogy hogyan kellene a refaktorálandó alkalmazás adatbázis szkriptjei között rendet tenni. A jelenlegi megoldást még Te írtad juniorként (ezen a ponton reflektálsz is kicsit, úgy látszik, rendesen megdolgoztál a pénzedért), kézzel futtatják az adatbázison, visszatekintve iszonyat törékeny és csoda, hogy a mai napig még nem omlott össze a production adatbázis szerkezet. A meeting-en megkérdeznek, hogy milyen szebb megoldást alkalmaznál. Gondolatban elmélyülsz a témában (miközben bután nézel ki az fejedből), majd elmosolyodsz, és felajánlasz pár profi megoldást.

Az alábbiak közül melyek használhatóak adatbázis séma evolúció kezelésére a JVM világában?

☐ Lombok

☑ Flyway

☐ Mockito

☐ Swagger

☑ Liquibase

☐ JDBC

Magyarázat a megoldáshoz

- Lombok (kod generalas annotaciokkal): https://projectlombok.org/

- Mockito (mockolo keretrendszer teszteleshez): https://site.mockito.org/

- Swagger (API design es dokumentacio): https://swagger.io/

- JDBC (Java adatbazis kapcsolat): https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

Séma evolúció kezelés:

Liquibase: https://www.liquibase.com/

Flyway: https://flywaydb.org/

4. feladat 0 / 5 pont

Java-ban az öröklődés egy megkerülhetetlen téma, ahol sok kérdés régen is felmerült benned. Egyetem alatt sok dolgot megtanultál, de gyakorlatban csak a jelenlegi cégednél sikerült elsajátítanod. Egy probléma megoldása során rájössz, hogy többek között az öröklődés lenne az egyik legjobb út a működés kiegészítéséhez. Ehhez viszont jó lenne visszaemlékezned az alapokra, ami egyrészt rég volt, másrészt akkor se tanultad meg rendesen, mert pont kihagytad az erről szóló előadást egy előző napi házibuli miatt. Bezzeg a balra dőlő Piros-Fekete forgatási stratégiáinak pszeudo kódját mind tudod fejből, ennek a tudásnak is mennyi hasznát vetted eddig…

Adott két azonos csomagban definiált osztály: A és B, a következőképpen:

```
public class B {
  private void m1(){}
  void m2(){}
  protected void m3(){}
  public void m4(){}
}

public class A extends B {...}
```

A felsorolt "A" osztálybeli metódus felüldefiniálások közül melyek a helyesek?

☐ @Override public void m1(){}

☑ @Override protected void m2(){}

☐ @Override private void m2(){}

☑ @Override public void m3(){}

☐ @Override void m4(){}

Magyarázat a megoldáshoz

@Override public void m1(){} - HAMIS - private láthatóságú metódust nem lehet Override-olni

@Override protected void m2(){} - IGAZ - ha nincs megadva láthatóság az implicit package private es mivel ugyan abban a package-ben van a 2 class igy hatasa megegyezik a kettőnek

@Override private void m2(){} - HAMIS - öröklődésnél a láthatóság nem szűkíthető

@Override public void m3(){} - IGAZ - láthatóság bővítése lehetséges

@Override void m4(){} - HAMIS - implicit package private szűkítést jelent a public-hoz képest, ami oroklodesnel nem megengedett

5. feladat 0 / 4 pont

Egyik reggel felébredsz, de iszonyú fáradt vagy, mert egész este „kódoltál” egy haveroddal egy új programozási nyelven. Lefőzöd a kávét / szisszentesz egy sört / energia italt és felcsapod a laptopot, de annyira zúg a fejed és homályosan látsz, hogy magadban sem vagy biztos.

Melyik sor íródhatott Java-ban az alábbiak közül?

○ var x = { s:String -> s.isEmpty()}

● var x = (Predicate<String>) s-> s.isEmpty();

○ var x = (s:String) => s.isEmpty

○ var x = s => !s.length;

Magyarázat a megoldáshoz

var x = { s:String -> s.isEmpty()} - Kotlin
var x = (Predicate<String>) s-> s.isEmpty(); - Java
var x = (s:String) => s.isEmpty - Scala
var x = s => !s.length; - JavaScript

Vissza a kategóriáimhoz

JAVA, Spring, Hibernate (EPAM)
5. forduló

Ismertető a feladathoz

Telnek-múlnak az évek és valahogy nem akar jönni az a senior pozíció. Pedig te tényleg „fullba” nyomod, de csak nem hullik az elismerés. Egyik nap rendesen össze is vitatkozol a vezető fejlesztővel egy problémán, ami neked egyértelmű, de mégsem tudtok egyezségre jutni. Ez az utolsó csepp, eddig is ott motoszkált a fejedben, de most már nem is kérdés, váltanod kell. Elmész pár interjúra és megpróbálsz új életet kezdeni.

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 40 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 7 pont

Rögtön az első interjún rájössz, hogy a matematika még mindig az állásinterjúk horizontig érő kupolájának tartó oszlopa. Érdekes módon nem tudsz felidézni olyan esetet karriered során, ahol valóban alkalmaznod kellett volna ilyen tudást, de most már tapasztaltabb vagy, jól tudod, hogy az interjú tartalma és a később végzendő munka alacsony korrelációt mutat.

Mely input(ok) esetén **NEM** kerül végtelen ciklusba az alábbi metódus?

```
public int compute(int n) {
    while (n > 1) {
        if (n % 2 == 0) {
            n /= 2;
        }
        else {
            n = 3 * n + 3;
        }
    }
    return n;
}
```

- ☐ 1012
- ☑ 4096
- ☐ 4040
- ☑ 1024
- ☐ 1000
- ☐ 525

Magyarázat a megoldáshoz

A fenti kódrészlet csak 2 hatványokra éri el a while ciklus terminálási feltételét (azaz az 1-et).

2. feladat 0 / 8 pont

Néhány jó hangulatú interjú kör után máris az új munkahelyen találod magad. Meg is lett a jól megérdemelt Senior pozíció. Természetesen nem a pénz miatt csináltad, neked elveid vannak. Rögtön esélyt kapsz a bizonyításra, rád is bíznak egy érdekes feladatot, mert bíznak benne, hogy könnyeden és gyorsan megoldod, hiszen az interjún azt mondtad, hogy Spring expert lettél az elmúlt években, és sokat láttál-hallottál már a témában.

A következő új osztályon dolgozol:

```
@Component
 public class A {
     @Autowired
     private B someSpecificBee;
     // ...
 }
```

Az újdonsült osztályodnak a működéshez szüksége van egy meghatározott *B* példányra, ezt létre is hozod a következőképpen.

```
@Configuration
public class AppConfiguration {
    @Bean
    public B someSpecificBee() {
        return new B("new and exciting");
    }
}
```

Nekifogsz kipróbálni az új kódot, ám meglepve tapasztalod, hogy nem a megfelelő *B* példány került injektálásra *A*-ba. Némi kutatás után megtalálod a következő kódrészletet egy *@Configuration*-nel annotált osztályban.

```
@Bean
@Primary
public B beeUsedSomewhereElse() {
    return new B("old and boring");
}
```

Sajnos ez a bean már használatban van egy *oldBeeConsumer* nevű beanben, ráadásul az *oldBeeConsumer* nem működne jól az újonnan létrehozott *someSpecificBee* bean-nel és az új *A* bean sem lenne használható ezzel a "régi" *B* bean-nel.

Hogyan biztosítható az, hogy az új ***A*** bean-be az új, ***someSpecificBee*** nevű ***B*** bean kerüljön injektálásra anélkül, hogy az ***oldBeeConsumer***-be injektált bean változna?

- ☐ Az *A* osztály *bee* nevű mezőjén lévő *@Autowired* annotáció *@javax.inject.Inject* annotációra történő cseréjével.
- ☐ Az *AppConfiguration* osztály *someSpecificBean* metódusának *@Order(1)* -el történő annotálásával.
- ☐ Az *AppConfiguration* osztály *someSpecificBean* metódusának *@Scope("prototype")* -el történő annotálásával.
- ☑ Az *A* osztály *bee* nevű mezőjének *@Qualifier("someSpecificBee")*-el történő annotálásával.
- ☑ Az *A* osztály *bee* nevű mezőjén lévő *@Autowired* annotáció *@javax.annotation.Resource* annotációra történő cseréjével.

Magyarázat a megoldáshoz

1. A *@Inject* annotációt a fenti példában a Spring azonos módon kezeli az *@Autowired* annotációval.

2. Az *@Order* annotáció segítségével a bean-ek rendezési sorrendjét határozhatjuk meg, nem pedig a precedenciájukat.

3. A *@Scope("prototype")* annotáció lehetővé teszi azt, hogy egy bean definícióval több példányt létre tudjunk hozni. Ez nem teszi egyértelművé azt, hogy melyik helyre milyen bean-eket injektáljunk, így nem oldja meg a problémát.

4. A *@Qualifier("someSpecificBee")* annotáció segítségével adhatjuk meg, hogy az *A.bee* mezőbe csak olyan bean-ek kerülhessenek beinjektálásra, amelyeknek a qualifier értéke "*someSpecificBee*". Mivel a *someSpecificBee* bean létrehozásánál nem adtunk meg qualifier értéket, a Spring a bean nevét veszi figyelembe. Ez a megfelelő beant választja ki injektálásra, és nem befolyásolja a *beeUsedSomewhereElse* bean használatát.

5. A *@Resource* annotáció az *@Autowired*-del szemben nem by-type, hanem by-name szemantikát követ, azaz a bean nevét veszi figyelembe injektáláskor. Amennyiben az injektálni kívánt bean nevét nem határozzuk meg egyértelműen, a Spring a mező nevéből származtatja azt. Mivel a mező neve azonos az injektálni szánt bean nevével, ez a megfelelő beant választja ki injektálásra, ezen felül nem befolyásolja a *beeUsedSomewhereElse* bean használatát.

3. feladat 0 / 5 pont

Spring expert vagy, ez nem is kérdés, de a csúcson csak egy embernek van hely, a trónkövetelőket pedig a helyükre kell tenni, így szinte már vadászod a komplexebb Spring-es feladatokat, végre előkerül egy méltó ellenfél. Itt már kénytelen vagy még mélyebben beleásni magad a Spring rejtelmeibe.

Szeretnéd az alábbi **A** osztályba injektálni a Spring context összes **B** típusú bean-jét úgy, hogy **A** konstruktora a **B** bean-ek listáját kapja meg.

```
@Component
public class A {
    private List<B> bees;

    @Autowired
    public A(List<B> bees) {
        this.bees=bees;
    }
    // ...
}
```

Az alábbiak közül melyik állítás igaz?

- ○ A Spring nem támogatja a Java kollekciók injektálását.
- ● A Spring további konfiguráció nélkül összegyűjti és injektálja az összes *B* bean-t *A*-ba.
- ○ Ahhoz, hogy a fenti kódrészlet az elvártnak megfelelően működjön, mindenképpen létre kell hozni egy *List<B>* típusú bean-t.
- ○ A Spring összegyűjti és injektálja az összes *B* bean-t *A*-ba, de csak akkor, ha az *A* konstruktorában a *List<B>*-t *Collection<B>*-re cseréljük.

Magyarázat a megoldáshoz

Az @Autowired dokumentációja alapján a Spring támogatja adott típusú beanek collectionként való injektálását különösebb konfiguráció nélkül is. A List<B> bean létrehozása is egy működő megoldás lenne, de nincs rá feltétlenül szükségük a kívánt működés elérése érdekében. A Spring dokumentáció @Autowired-re vonatkozó részlete alapján tetszőleges typed collection-t használhatunk, tehát nincs szükség Collection típusra.

https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-autowired-annotation

4. feladat 0 / 10 pont

Tovább gyűjtögeted Spring-es skalpjaidat, a legújabb jelölt egy új projekt, ahol egy komplex feature-t kellene implementálni, de az előző senior fejlesztő Spring Data JPA-t használt az adatbázis kezeléshez és lekérdező metódusok használatával készített lekérdezéseket. Tapasztalatod azt mutatja, hogy minél komplexebb valami és minél valószínűbb, hogy a létező kódot látva feltennéd a „mire gondolhatott a költő” messzire vezető kérdést, annál biztosabb, hogy a témában érintett már nincs a cégnél. Természetesen itt is ez a minta materializálódott. Picit el is gondolkozol, vajon az én kódommal is ez lehet a régi cégnél? Áááá dehogy, én hibátlan időtálló kódot írok... Na de vissza a problémához. Azt hallottad a távozó kollégáról, hogy hihetetlen tehetséges volt, és bármit meg tudott oldani lekérdező metódusok segítségével. Kicsit értetlenül állsz/ülsz a megoldás előtt, de megpróbálod megfejteni a lekérdezés eredményét.

Mit csinál az alábbi lekérdező metódus (Spring Data JPA query method)?

```
List<Person> findFirst5DistinctPersonByFirstNameStartingWithAndLastNameI
```

- ○ Hibás a metódus (futási hiba történik).
- ● Visszaadja az első 5 entitást firstName alapján rendezve csökkenő sorrendben, ismétlődések nélkül, szűrve a firstName elejét, valamint a paraméterrel nem egyező lastName-t, az age nagyobb vagy egyenlő a megadott paraméternél, kis és nagybetűvel is keresve.
- ○ Visszaad 5 entitást firstName alapján rendezve növekvő sorrendben, kisbetűkkel, ismétlődések nélkül, szűrve a firstName elejét, valamint a paraméterrel nem egyező lastName-t, az age nagyobb vagy egyenlő a megadott paraméternél.
- ○ Visszaadja az első 5 entitást firstName alapján rendezve csökkenő sorrendben, ismétlődések nélkül, a firstName és lastName-ben is megkeresve a 2 string paramétert, az age nagyobb vagy egyenlő a megadott paraméternél, kis és nagybetűvel is keresve.

Magyarázat a megoldáshoz

A Spring Data JPA query buildere a tényleges adatbázisnak megfelelően valami hasonló lekérdezésre fogja a fenti metódus nevet lefordítani:

```
SELECT DISTINCT * FROM Person WHERE FirstName ILIKE '?1%' AND Las
```

Vissza a kategóriáimhoz

JAVA, Spring, Hibernate (EPAM)
6. forduló

Ismertető a feladathoz

Az idő múlásával ahogy szépen gyűlik a tapasztalat, ritkul a haj, fárad a derék, mennek tönkre az ízületek és romlik a szem, (fel is merül benned, hogy indokolt lenne egy veszélyességi pótlék ehhez a munkához) egyre erősebb lesz benned a nosztalgia és a „régen minden jobb volt” érzés. Már nem tudsz ellenállni az anyahajó ismerős hívó kürtjének. Visszamész a korábbi cégedhez. Természetesen már vezető fejlesztő pozícióba, hiszen te már sokat látott harcedzett megmondó ember vagy. Az ifjonci tüzet és tökéletességre törekvést szép lassan kezdi leváltani a praktikum és az „annyit csinálunk meg amennyit kértek” hozzáállás. Természetesen a minőségből továbbra sem engedsz, de már nem te hallgatod mások háborús történeteit, nagy ütközeteit a business-el, hanem te mesélsz a fiataloknak. Mivel tapasztalatod széles körű és tudásod folyamatosan bővíted már nincs nagyon olyan dolog, amivel ne tudnál foglalkozni. Amikor elbukik valaki, rögtön hozzád fordul mindenki segítségért.

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 50 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 10 pont

Egyszer csak azon kapod magad, hogy vitába keveredtél egy Juniorabb fejlesztővel. Mindenáron ZonedDateTime-ot erőltet. Nem akarja megérteni, hogy attól, hogy valami újabb még nem feltétlenül kell használni. A régi dolgok is tudnak hasznosak lenni. Ezen azért picit elgondolkozol, dejavu kerít hatalmába. Ez már egyszer mintha megtörtént volna veled csak akkor az érvelés túloldalán voltál. Úgy érzed, akkor is neked volt igazad, meg most is. Ez valami quantum igazság lesz... Azért természetesen lemész vitapartered szintjére és elkezdődik az érvelés.

Az alábbi állítások mindegyike igaz a java.util.Date és java.time.ZonedDateTime osztályok valamelyikére. Melyek azok, amik csak a java.util.Date-re igazak?

☑ Interfész szinten mindig a rendszer időzónáját használja.

☐ Szálbiztos.

☑ Az időt milliszekundum részletességgel tárolja.

☑ A hónapok nullától indexeltek.

☐ A kapcsolódó naptár nem változtatható az objektumban.

Magyarázat a megoldáshoz

A java.util.Date implementációs szinten implicit UTC időzónát használ, de interfész szinten mindig az aktuális rendszer időzónát fogja figyelembe venni (pl. dátum beállítása, toString).

Mivel a ZonedDate objektum immutable, ezért technikailag abban sem lehet megváltoztatni a naptárt, csak ha újat hozunk létre.

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Date.html
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/ZonedDateTime.html

2. feladat 0 / 6 pont

Érettségedet mi sem bizonyítja jobban, mint hogy megértetted azt, hogy bár bármit meg tudsz te egyedül csinálni, mégis a cég számára a globális optimum azzal érhető el, hogy a munkához úgy állsz hozzá, hogy más is hozzáférjen. A te idődnél kevés értékesebb dolog van a környéken, így fontos, hogy olyasmivel töltsd, aminek komoly hatása van a projektekre / cégre / gazdaságra / világra... mint például a megfelelő GC használata egy újonnan induló projekten.

Melyik GC illik egy alacsony válaszidejű online webszolgáltatáshoz Java 14-es JVM-mel?

☐ Paralell GC

☐ CMS GC

☑ Z GC

☑ G1 GC

☐ Epsilon GC

Magyarázat a megoldáshoz

A Parallel GC inkább átmenő teljesítményre (throughput) optimalizált GC, ezért a Stop The World (STW) miatti szünet magas lehet normál működés közben is, ami negatív hatással van a válaszidőre.

A CMS GC válaszidőre (latency) optimalizált GC, de java 14-ben már nem elérhető.

A ZGC kifejezetten alacsony STW-re optimalizált GC, ezért kifejezetten jó választás egy alacsony válaszidejű webszolgáltatás számára.

A G1 GC-ben az STW miatti szünet ideje korlátozható (bizonyos mértékig), ezért ideális lehet egy alacsony válaszidejű alkalmazás számára.

Az Epsilon GC gyakorlatilag egy tényleges munkát nem végző (noop) GC, ami egyrészt teljesítmény referenciaként használható, vagy olyan esetekben, ahol a memória elfogyása nem probléma vagy nem valószínű. Webszolgáltatásoknál a memória többnyire erősen limitált, ezért ez nem ideális választás.

3. feladat 0 / 8 pont

Azt te is elismered, hogy vannak dolgok, amiket te sem tudsz hirtelen fejből, ezért hosszú évek alatt a stackoverflow koronázatlan királyává váltál. Mindent megtalálsz és mindenre válaszolsz. Sajnos vannak kérdések, amikre nem könnyű rákeresni és inkább tapasztalatra meg intuícióra kell támaszkodj. Mint például a következő:

Melyik kód részlet használhatja a legkevesebb threadet az alábbiak közül?

○

```
@GetMapping("/getAll")
public List<User> getAll() {
    List<User> users = userRepository.getAllUsers();
    return users;
}
```

○

```
@GetMapping("/getAll")
public Flux<User> getAll() {
    List<User> users = userRepository.getAllUsers();
    return Flux.fromIterable(users);
}
```

○

```
@GetMapping("/getAll")
public List<User> getAll() {
    Flux<User> users = userRepository.getAllUsers();
    return users.collectList().block();
}
```

●

```
@GetMapping("/getAll")
public Flux<User> getAll() {
    Flux<User> users = userRepository.getAllUsers();
    return users;
}
```

Magyarázat a megoldáshoz

Egyedül az a Flux<User>-t visszaadó kontroller metódus, ami a repositorytól is Fluxként kapja a Usereket end-to-end reaktív megoldás, így az használja a legkevesebb szálat.

https://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html

4. feladat 0 / 10 pont

Nincs mit szépíteni rajta, nincs hozzád méltó feladat. Nemes egyszerűséggel látod a mátrixot (és ő is visszanéz rád). Lassan már az IDE-re se lesz szükséged. Amúgy is a fejedben már egy teljes értékű JVM fut így a következő feladatot be se kell gépelned, ránézésre pontosan tudod mi az eredmény.

Tekintsük az alábbi kódrészletet:

```
// ...
int increment = 42;
Function<Integer, Integer> increaseFunction = this.createIncreaseFunctio
increment += 2;
int secretCode = increaseFunction.apply(10);
// ...
```

Illetve az innen hívott**createIncreaseFunction** metódust:

```
public Function<Integer, Integer> createIncreaseFunction(int inc){
  return value -> value + inc / 2;
}
```

Mi a secretCode változó értéke a kódrészlet lefutása után?

○ Ez a kód nem fordítható, hiszen az increment változónak final-nek vagy effectively final-nek kellene lennie.

○ A kód ugyan fordul, de futás közben ExecutionException-t dob, hiszen megváltoztattuk az increment változó értékét.

● 31

○ 32

Magyarázat a megoldáshoz

Igaz ugyan, hogy a lambdák csak final vagy effectively final szabad változókat (és mezőket, etc.)használhatnak, ám a Java csak érték szerinti paraméterádatást támogat (pass-by-value), így a fenti lambdában használt inc paraméter inkább tekinthető az increment változó egy másolatának. Ennek megfelelően egyfelől azinceffectively final, hiszen sehol nem módosítjuk az értékét (így az első válasz helytelen), másfelől nincs rá hatással az, hogy az incrementet hogyan módosítjuk (így a 4. válasz helytelen). Nem final vagy effectively final változó használata lambdában fordítási idejű hiba, így a 2. válasz is helytelen.

Vissza a kategóriáimhoz

JAVA, Spring, Hibernate (EPAM)
7. forduló

Ismertető a feladathoz

Miután a cég vezetése is belátta, hogy nincs nálad jobb fejlesztő a vidéken megkaptad a pozíciók Szent Grálját, te lettél a CTO. Úgy gondolod, mindent elértél, amit akartál, de valami még sincs rendben. Minden olyan furcsa, kódot hetek óta még review formájában sem láttál, bár azért még te interjúztatsz és természetesen a konferenciákon is te képviseled a céget az előadók sorában. Utazni és tanulni továbbra is nagyon szeretsz. Azért az IntelliJ / Ecilipse / Notepad++ / (Insert Your IDE Here) helyét az Excel és az Outlook kezdi átvenni az életedben. Amikor két meeting között van pár perced elmélázol azon, hogy mostanában a technológiák mellett ugyan olyan fontos azt tudni, hogy arra a kérdésre, hogy valami kész van-e, nincs bináris válasz. A kész van és a nincs kész között egy folytonos megoldáshalmaz helyezkedik el, amiben a „kész van” valójában benne sincs.

A hideg verejték ül ki a homlokodra amikor rájössz, hogy a legutóbbi nagy vitában a fejlesztőkkel te a business-el értettél egyet. Érzed, hogy közeleg az egzisztenciális krízis, meg is vetted a megjelenés napján a legújabb Nvidia kártyát (IT-s körökben ez az, ami a legközelebb van a kapuzárási pánik társadalmilag elismert manifesztációjához, a cabrio-hoz), bár kérdés, hogy mennyi időd lesz valójában használni. Nem baj, majd csilingelhet a BitCoin helyette.

Tekintettel arra, hogy egy választ sem rögzítettél az alábbi feladatlapon, ebben a fordulóban a kitöltésére rendelkezésre álló idő teljes egésze, azaz 55 perc került rögzítésre mint megoldáshoz felhasznált idő.

1. feladat 0 / 15 pont

Munkád domináns része az, hogy másoknak segítesz. Ez kiváló lehetőség arra, hogy megismerd a fiatalokat és ők is lássanak téged. A seniorok viccesen azért elmondják, hogy nem kell félni tőled jó szándékú vagy, csak ne etessenek mert az emésztésed már nem a régi. „Ha békén hagyjátok és nem néztek a szemébe akkor tovább áll majd magától.” Valaki az adatbázis tervezésnél kér egy kis segítséget. A szemed felcsillan. Vissza az alapokhoz, amikor még minden más volt.

Az alábbi osztály definíciók milyen adatbázis tábla struktúrát eredményeznek (MySQL)?

```
@MappedSuperClass
public abstract class MainEntity {
  @Id
  @GeneratedValue(strategy = AUTO)
  private Long id;
}

@Entity
public abstract class PersonEntity extends MainEntity {
  private Integer age;
}

@Entity
@Inheritance(strategy = Inheritance.Type.TABLE_PER_CLASS)
public class Person extends PersonEntity {
  private String firstName;
  private String lastName;
}

@Entity
public class WorkerPerson extends Person {
  private String job;
}
```

● 1 db tábla jön létre: person_entity

○ 2 db tábla jön létre: person_entity, worker_person

○ 2 db tábla jön létre: person, worker_person

○ 3 db tábla jön létre: person_entity, person, worker_person

Magyarázat a megoldáshoz

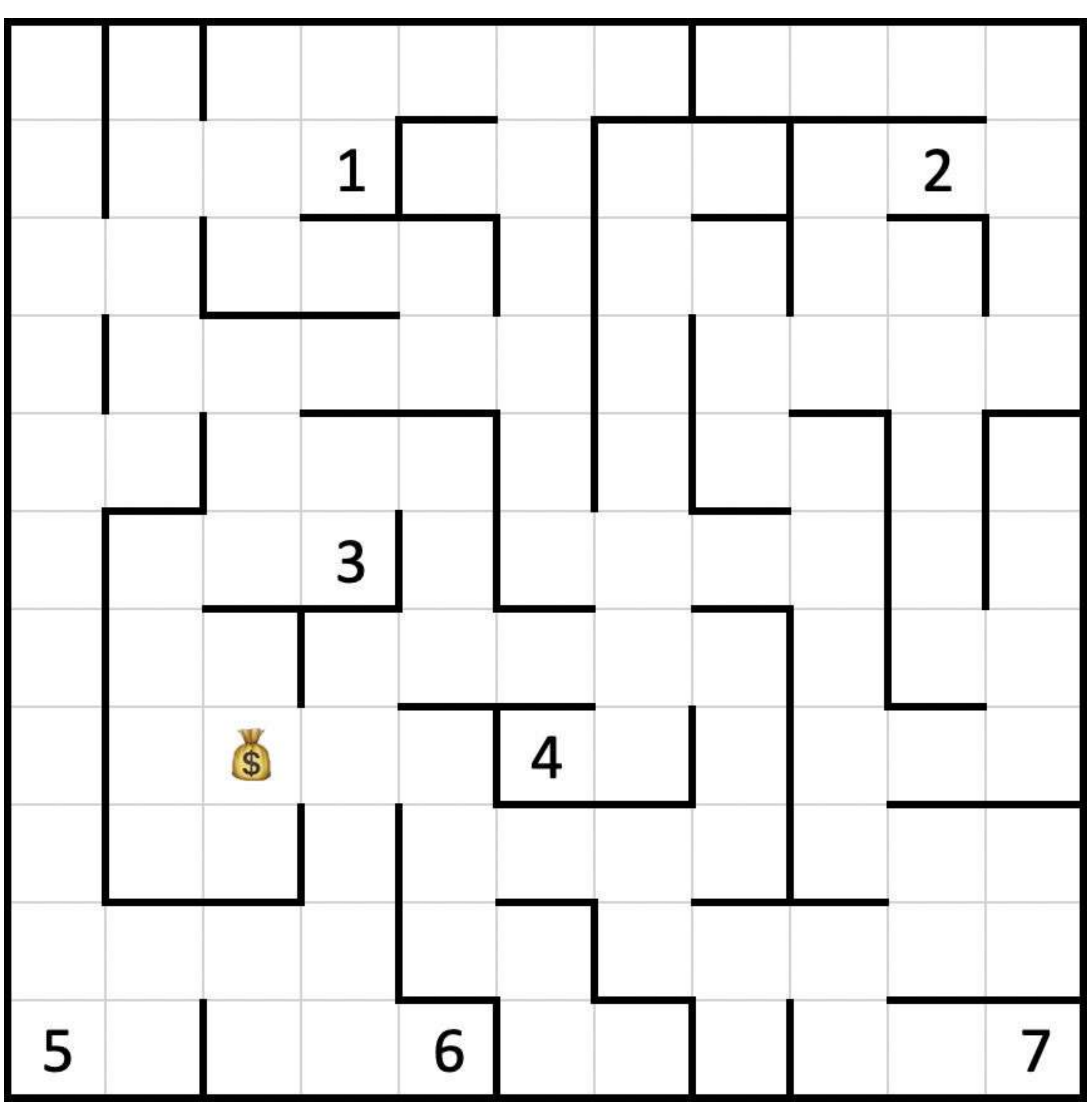
A @MappedSuperclass annotáció miatt a MainEntity osztályból nem jön létre külön tábla, csak a leszármazott osztályaiból.

A Person osztályon beállított TABLE_PER_CLASS öröklődési stratégia a Person leszármazott osztályaira vonatkozik, így nem befolyásolja a példában szereplő osztályokból létrejövő táblákat.

A PersonEntity osztályon nincs megadva öröklődési stratégia, ezért az alapértelmezett stratégia érvényes: SINGLE_TABLE.

Így egy tábla jön létre (person), ez fogja tartalmazni a tárolt Person és WorkerPerson példányokat is.

2. feladat 0 / 20 pont



Végül hosszas keresgélés után rájössz, hogy nem a cél volt a lényeg, hanem az odavezető út. A kihívások elfogytak, a munka meg kezd repetitív lenni. Úgy érzed, hogy kell még keresned magadnak egy legyőzendő főszörnyet / egy megmászandó Everestet. A semmiből kerül elő ez a labirintus, ami gyönyörű szinonimája életednek és annak, hogy hol tartasz. Ha végre kijutsz innen, új kihívások után nézel. A paradicsom végül is szépen piroslana a kertben, uborka receptet is már régen ki akartad próbálni és úgy hallottad a Józsinak kiváló kovásza van, ha abból tudnál szerezni egy kicsit végre süthetnéd a saját kenyered, de finom is lenne... „Távolba meredő könnyes tekintete a naplementébe réved, majd megmutatja kisiskolás gyerekének, hogy hogyan is kell ciklust írni. A gyermek majd felrobban az izgatottságtól, amikor lefut a kód, programozó akar lenni, ha nagy lesz....” (csapó / the end)

A labirintus megszámozott cellái közül melyik(ek)ből indulva jutunk el a kincshez, ha a lent megadott kódrészlet alapján közlekedünk?

```
public void move() {
  if (canStepLeft() && canStepRight()) {
    stepLeft();
  }
  else if (canStepDown() && !canStepLeft()) {
    stepDown();
  }
  else if (canStepRight()) {
    stepRight();
  }
  else if (canStepUp() && !canStepDown()) {
    stepUp();
  }
}
```

☐ 1

☐ 2

☑ 3

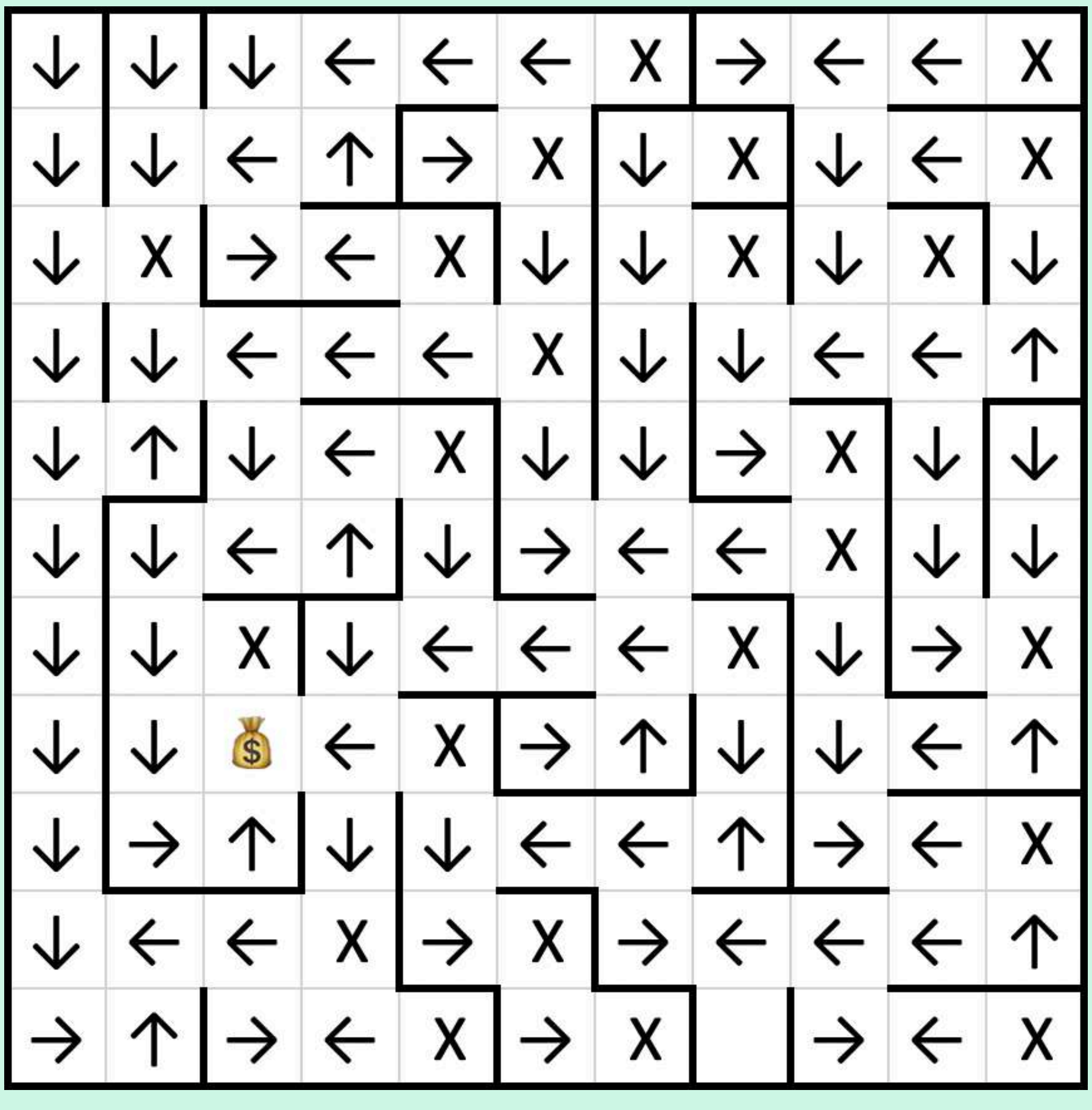
☑ 4

☐ 5

☐ 6

☐ 7

Magyarázat a megoldáshoz



↓	↓	↓	←	←	←	X	→	←	←	X
↓	↓	←	↑	→	X	↓	X	↓	←	X
↓	X	→	←	X	↓	↓	X	↓	X	↓
↓	↓	←	←	←	X	↓	↓	←	←	↑
↓	↑	↓	←	X	↓	↓	→	X	↓	↓
↓	↓	←	↑	↓	→	←	←	X	↓	↓
↓	↓	X	↓	←	←	←	X	↓	→	X
↓	↓	💰	←	X	→	↑	↓	↓	←	↑
↓	→	↑	↓	↓	←	←	↑	→	←	X
↓	←	←	X	→	X	→	←	←	←	↑
→	↑	→	←	X	→	X		→	←	X

Az ábrán látható, hogy merre lépünk tovább az egyes mezőkről a megadott algoritmus alapján.

Vissza a kategóriáimhoz