Általános tudnivalók

Ebben az ismertetésben az osztályok, valamint a minimálisan szükséges metódusok leírásai fognak szerepelni. A feladatmegoldás során fontos betartani az elnevezésekre és típusokra vonatkozó megszorításokat, illetve a szövegek formázási szabályait. Segédfüggvények is létrehozhatók, a feladatban nem megkötött adattagok és elnevezéseik is a feladat megoldójára vannak bízva. Törekedjünk arra, hogy az osztályok belső reprezentációját a lehető legjobban védjük, tehát csak akkor engedjünk meg, és csak olyan hozzáférést, amelyre a feladat felszólít, vagy amit az osztályt használó kódrészlet megkíván!

A beadott megoldásnak működnie kell a mellékelt tesztprogrammal, de ez nem elégséges feltétele az elfogadásnak. Törekedjünk arra, hogy a megírt forráskód kellően általános és újrafelhasználható legyen!

Használható segédanyagok: Java dokumentáció, legfeljebb egy üres lap és toll. Ha bármilyen kérdés, észrevétel felmerül, azt a felügyelőknek kell jelezni, *NEM* a diáktársaknak!

A feladat összefoglaló leírása

A feladat a torpedó játék (egyszerűsített) megvalósítása.

A játékban egy 10x10-es táblán különböző alakú hajókat helyezünk el. Minden játékoshoz tartozik egy játéktábla, amelyen lévő hajókra az adott játékos tüzel. Az a játékos nyer, aki előbb süllyeszti el az összes hajót a tábláján.

A programhoz tartozik egy tesztelő (Test.java, board1.txt és board2.txt), amely az egyes osztályok funkcionalitását teszteli, illetve a várható pontszámot mutatja.

A feladat részletes ismertetése

Index *(5 pont)*

A `battleShip.board.Index` osztály segítségével a játéktábla koordinátáit reprezentáljuk.

- Az `Index` osztálynak két privát adattagja van: a sor jelölésére szolgáló `row` és az oszlop jelölésére szolgáló `column` (mindkét adattag egész típusú). Az osztálynak meg kell valósítania a `Comparable<Index>` interfészt.
- Az `Index` osztály konstruktora publikus, amely két kapott paraméter alapján beállítja a `row` és a `column` adattagokat a megadott értékekre.
- Az osztálynak két lekérdező metódusa van: a `getRow` és a `getColumn`, amelyek segítségével a koordináták lekérdezhetőek.
- Az `Index` osztályban felüldefiniáljuk az `Object`-től örökölt `hashCode` metódust. Megköveteljük, hogy azonos indexekre a `hashCode` azonos értéket adjon vissza. *(1 pont)*
- Az `Object`-től örökölt `equals` metódust is felüldefiniáljuk: két `Index` típusú objektum akkor egyenlő, ha a koordinátáik értéke rendre megegyezik. *(1 pont)*
- Az `Index` osztálynak meg kell valósítania a `Comparable<Index>` interfészt: elsődlegesen a sor, másodlagosan az oszlop szerint rendezünk. *(1 pont)*

- Az `Index` osztályban felüldefiniáljuk az `Object`-től örökölt `toString` metódust. A felüldefiniált `toString` metódus az alábbi szöveget adja vissza: "([sor],[oszlop])", ahol [sor] a sort jelölő koordináta, az [oszlop] pedig az oszlopot jelölő koordináta, például "(3,1)". *(1 pont)*
- Definiáljuk a logikai értéket adó `isNeighbour` függvényt, amely paraméterül egy másik `Index` objektumot vár. A függvény akkor adjon vissza igazat, ha az adott index és a paraméterben megkapott index szomszédosak: egyik közvetlenül a másik fölött, alatt vagy mellett van (az átló nem számít). *(1 pont)*

Response *(1 pont)*

A `battleShip.board.Response` felsorolási típus segítségével a játékban a lövésre érkező lehetséges válaszokat reprezentáljuk.

- Vegyük fel a lehetséges válaszokat (`HIT`, `SUNK`, `MISS`) tartalmazó felsorolási típust! (A játékban a `HIT` fogja jelölni, hogy egy adott lövés célba ért, a `SUNK` pedig azt, hogy ezzel a lövéssel a hajó el is süllyedt, míg a `MISS` azt jelöli, hogy a lövés nem talált el semmit) *(1 pont)*

GameException *(1 pont)*

Hozzunk létre egy saját ellenőrzött kivételosztályt (`battleShip.GameException`), amely majd a játéktábla felépítése során előforduló hibákat jelöli. Legyen egy szöveges paramétert váró konstruktora, amely hívja meg az ősének az ugyanilyen szignatúrájú konstruktorát. Származzon a `RuntimeException` osztályból.

Ship *(12 pont)*

A `battleShip.board.Ship` osztály segítségével egy hajót reprezentálunk.

- A `Ship` osztálynak nyilván kell tartania, hogy a hajó melyik területet foglalja el a játéktáblán (ez a gyakorlatban `Index` objektumok csoportját jelenti), és meg kell jegyeznie azt is, hogy a hajónak melyik részét lőtték már ki. (Például a hajó az `(1,2)`-es és `(1,3)`-as pozíción van, amelyből az `(1,2)`-es pozíciót már eltalálták.) Ezen adatok tárolását privát adattaggal/adattagokkal kell megoldani, de az adattag(ok) nevének és típusának megválasztásában a feladatot megoldó szabad kezet kap.
- Definiáljunk egy publikus osztályszintű `isConnected` metódust, amely `Index` objektumok tömbjét várja, és logikai értéket ad vissza. A metódus feladata annak eldöntése, hogy ha a paraméterben kapott `Index` objektumok egy hajó részeit írnák le, akkor az adott hajó tartalmaz-e izolált pozíciót. (Nem akarjuk ezzel a metódussal a sokkal nehezebb feladatot eldönteni, miszerint összefüggő-e a hajó. Többletpontért persze megoldható ez a feladat is!) Egy hajót akkor tekint a metódus Connectednek, ha minden indexére igaz az, hogy az adott index a hajó valamelyik másik indexével szomszédos. (Tehát például ha a hajó az `(1,1)` és az `(1,3)` indexű részekből áll, akkor a hajó nem Connected, mert ez a két index nem szomszédos.) Az indexek sorrendjére viszont semmilyen megkötést nem teszünk. (Tehát, ha a hajó az `(1,1)`, az `(1,3)` és az `(1,2)`indexű részekből áll, akkor Connected, mert minden indexe szomszédos a hajó legalább egy másik indexével.) Feltesszük azt is, hogy egy olyan hajó, amelynek csak egy indexe van, az mindig Connected. *(2 pont)*

- Az osztálynak egy publikus konstruktora legyen, amely paraméterül `Index` objektumok tömbjét várja. Ha a tömbben nincsenek indexek, vagy vannak, de azok nem alkotnak Connected hajót, akkor a konstruktor dobjon egy `GameException` kivételt a következő szöveggel: "Ship is invalid.". Ellenkező esetben viszont az adattagjában/adattagjaiban rögzítse a hajó indexeit. Kezdetben a hajó semelyik részét sem találták még el. *(2 pont)*
- Legyen egy publikus `hasHit` metódus, amely egy `Index` objektumot vár és logikai értéket ad vissza. Ha a kapott index nem része a hajónak, akkor a metódus dobjon egy`GameException` kivételt a következő szöveggel: "Index is invalid.". Ellenkező esetben viszont adja vissza, hogy a hajó adott indexű részét eltalálták-e (`true`), vagy sem (`false`). *(1 pont)*
- Legyen egy publikus `hasSunk` metódus, amelynek nincs paramétere, és logikai értéket ad vissza. A metódus akkor adjon vissza igazat, ha a hajó elsüllyedt, tehát ha már mindegyik pozícióját eltalálták. *(1 pont)*
- Legyen egy publikus `fire` metódus, amely egy `Index` objektumot vár, és `battleShip.board.Response` felsorolási típus egy elemét adja vissza: a metódus "lő" a hajó egy adott pontjára és visszaadja az eredményt. Az eredmény legyen `MISS`, ha a lövés nem találta el a hajó semelyik pontját sem. Az eredmény legyen `HIT`, ha a lövés eltalálta a hajót, de az nem süllyedt el (maradt még olyan pontja, amelyet nem ért találat). A metódus adjon `SUNK` választ, ha a lövés talált, és a hajó el is süllyedt (minden részét eltalálták már). Nem kell külön kezelni azt az esetet, ha a hajó egy olyan részére lőttek, amelyet már korábban is eltaláltak: ez esetben is ugyanúgy adjunk `HIT` vagy `SUNK`választ annak függvényében, hogy maradt-e még találat nélküli pont, vagy sem. Ha volt találat, akkor ne felejtsük el "könyvelni", hogy az adott rész találatot szenvedett! Ha a metódus eredménye `HIT` vagy `SUNK`, akkor a metódus írja is ki az eredményt a képernyőre. *(3 pont)*
- Legyen egy publikus `isInside` metódus, amely négy egész számot vár (`minRow`, `minCol`, `maxRow`, `maxCol`), és logikai értéket ad vissza. A metódus azt ellenőrzi, hogy a paraméterben megadott méretű játéktáblán a hajó rajta van-e. Hamisat kell visszaadni, ha az adott méretű játéktábláról a hajó "lelógna", tehát ha van legalább egy olyan indexe, amelynek vagy a sora nem esik `minRow` és `maxRow` közé, vagy az oszlopa nem esik `minCol` és `maxCol` közé. (az egyenlőséget is elfogadjuk) *(1 pont)*
- Legyen egy publikus `isOverlapped` metódus, amely paraméterül egy másik hajót vár, és logikai értéket ad vissza. A metódus azt ellenőrzi, hogy a két hajónak van-e közös pontja. Ha van legalább egy közös pont, akkor adjon vissza igazat, különben hamisat. *(2 pont)*

Board *(8 pont)*

A `battleShip.board.Board` osztály segítségével a játéktáblát reprezentáljuk.

- A `Board` osztálynak két konstans publikus egész típusú adattagja van: a `maxRow`, ami a legnagyobb indexű sort jelöli, és a `maxCol`, ami a legnagyobb indexű oszlopot jelöli. (A sorokat és az oszlopokat 1-től indexeljük). Az osztálynak ezen kívül legyen egy privát `ships` nevű `Ship` objektumokat tartalmazó `ArrayList` adattagja is, amely a táblán lévő hajókat tárolja.
- A `Board` osztály konstruktora publikus, és két egész paramétert vár: a legnagyobb sor-indexet és a legnagyobb oszlop-indexet. A konstruktor a kapott paramétereket eltárolja és az `ArrayList` típusú adattagot üres tömbként inicializálja. *(1 pont)*
- Legyen egy publikus `isPlaceable` metódus, amely egy hajót vár paraméterül, és logikai értéket ad vissza. A metódus azt ellenőrzi, hogy a paraméterben megadott hajó elhelyezhető-e a játéktáblán. A hajó akkor helyezhető el a táblán, ha nem lóg le róla, és

nem ütközik (nincsen közös pontja) egyetlen már korábban a táblán lévő hajóval sem.*(2 pont)*
- Legyen egy publikus, visszatérési érték nélküli `addShip` metódus, amely egy hajót vár paraméterül. Ha a hajó elhelyezhető a táblán, akkor felveszi a hajók tömbjébe, ellenkező esetben pedig a metódus dobjon egy `GameException` kivételt a következő szöveggel: "Not all ships could be placed on the board.". *(1 pont)*
- Legyen egy publikus `fire` metódus, amely egy `Index` objektumot vár, és `battleShip.board.Response` felsorolási típus egy elemét adja vissza: a metódus "lő" a tábla hajóira és visszaadja az eredményt. Az eredmény legyen `MISS`, ha a lövés nem találta el semelyik hajó semelyik pontját sem (a tábla minden hajójára az adott indexszel meghívott`fire` metódus `MISS` értékkel tért vissza). Az eredmény legyen `HIT`, ha a lövés eltalálta valamelyik hajót, de az nem süllyedt el. A metódus adjon `SUNK` választ, ha a lövés eltalálta valamelyik hajót, és az el is süllyedt. Ha a metódus eredménye `MISS`, akkor a metódus írja is ki az eredményt a képernyőre. *(3 pont)*
- Legyen egy publikus `allSunk` metódus, amelynek nincs paramétere, és logikai értéket ad vissza. A metódus akkor adjon vissza igazat, ha a táblán lévő összes hajó elsüllyedt. *(1 pont)*

Player *(2 pont)*

Valósítsuk meg a `battleShip.Player` absztrakt osztályt, amely egy játékost valósít meg.

- A `Player` osztálynak legyen két protected adattagja: egy `Board` típusú, `board` nevű és egy szöveges típusú, `name` nevű. Előbbi azt a játéktáblát tartalmazza, amire az adott játékos lő, utóbbi pedig a játékos nevét.
- A `Player` osztály konstruktora publikus, amely két kapott paraméter alapján beállítja a `board` és a `name` adattagokat a megadott értékekre.
- Legyen egy nyilvános `getBoard` metódus, amivel a táblát lehet lekérdezni.
- Az osztálynak legyen egy absztrakt publikus `makeIndex` nevű metódusa, amely egy `Index` típusú objektumot ad vissza. A metódus a leszármazottban azt fogja majd visszaadni, hogy a játékos melyik mezőre szeretne lőni.
- Legyen egy publikus `fire` metódus, amely logikai értéket ad vissza. A metódus a `makeIndex` meghívásával "kiválasztja", hogy a játékos hova szeretne lőni, majd ezt ki is írja a képernyőre. Ezután elvégzi a lövést a játékos táblájára, és visszaadja, hogy a játékos megnyerte-e a játékot. A játékos akkor nyerte meg a játékot, ha elsüllyesztette a tábláján az összes hajót (nyerés esetén egy üzenetet is írjon ki erről a képernyőre).

ManualPlayer *(1 pont)*

Valósítsuk meg a `battleShip.ManualPlayer` osztályt, amely legyen a `Player` osztály leszármazottja.

- Az osztály konstruktora publikus, amely két kapott paraméter alapján beállítja a `board` és a `name` adattagokat a megadott értékekre (a szülő konstruktorának meghívásával).
- Írjuk meg a `makeIndex` metódust: billentyűzetről kérjük be a sor- és az oszlopkoordinátákat, majd adjuk vissza (a helyességet nem kell ellenőrizni: feltehetjük, hogy mindig megfelelő nagyságú számot adnak meg).

AutomaticPlayer *(1 pont)*

Valósítsuk meg a `battleShip.AutomaticPlayer` osztályt, amely legyen a `Player` osztály leszármazottja.

- Az osztály konstruktora publikus, csak egy paramétert vár: a játéktáblát. A konstruktor meghívja a szülő konstruktorát a táblával, és "Automatic" névvel.
- Írjuk meg a `makeIndex` metódust: érvényes tartományba eső, véletlenszámokból álló Indexet adjon vissza (a tábla legkisebb indexe (1,1), a legnagyobb indexeket pedig a tábla adattagjai tárolják).

Game *(5 pont)*

- A `Game` osztályban legyen egy nyilvános, osztályszintű, `readBoard` nevű metódus, amely egy fájlból olvassa fel a játéktábla hajóit, és a hajókkal feltöltött táblát adja vissza. A metódus egy szöveges paramétert vár: a fájl nevét. A metódus elsőként hozza létre a játéktáblát úgy, hogy mind a maximális sor-index, mind a maximális oszlop-index 10 legyen. Ezután a megadott fájlból olvassa be a hajókat, és helyezze el a táblán, majd adja vissza a táblát. A fájl szerkezete a következő: a fájl minden sora egy hajót reprezentál; a hajókat pedig az indexeik felsorolásával adjuk meg. Az indexek vesszővel vannak elválasztva, egy index pedig a sor-koordinátából és a vesszővel elválasztott oszlop-koordinátából áll. Tehát például az `1,2,1,3,1,4` sor azt a hajót adja meg, amely az `(1,2)`, az `(1,3)` és az `(1,4)` koordinátán van. A fájl helyességét nem kell ellenőrizni, feltételezhetjük, hogy az adatok helyesek (minden sor páros számú, veszővel elválasztott számokat tartalmaz; az általuk elkészített hajó Connected, nem lóg le a tábláról, és a hajók nem fedik egymást). *(3 pont)*
- A `Game` osztálynak legyen két `protected` láthatóságú adattagja: a két játékos (`player1` és `player2`, amelyek `Player` típusúak).
- A `Game` osztály konstruktora nyilvános, és két szöveges paramétert vár: a játékosok neveit. A konstruktor létrehoz két táblát: az egyiket a `board1.txt` fájl alapján, a másikat a`board2.txt` fájl alapján. Ezután a játékosnevek és a táblák alapján létrehoz két `ManualPlayer` objektumot (`player1` és `player1` adattagok). *(1 pont)*
- Legyen egy `play` nevű, nyilvános, visszatérési érték és paraméter nélküli metódus, amelyben a két játékos felváltva lő, mindaddig, amíg valamelyik meg nem nyeri a játékot.*(1 pont)*

Pontozás

A tesztelő által adott pontszám csak becslésnek tekinthető, a gyakorlatvezető levonhat pontokat, vagy adhat részpontokat.

0 - 10: *elégtelen (1)*
11 - 17: *elégséges (2)*
18 - 23: *közepes (3)*
24 - 29: *jó (4)*
30 - 36: *jeles (5)*

Jó munkát! :-)