**Sandpile Game**

A feladat egy egyszerű játék implementálása, ahol a játékosnak ki kell jutnia a pálya közepéről, úgy hogy elkerülje az összes ellenséges entitásokat, a férgeket.

A játék pályája 19X19 méretű, ami határait a Position osztály fogja definiálni és a játék entitásai körönként fognak mozogni.

Alapvető elvárás, hogy a structure tesztek fussanak le helyesen. Mindenhol tedd ki az @Override annotációt, ahol lehetséges.

**Előkészületek (8+6T)**

A megfelelő StructureTest-ek alapján készítsd el a Direction enumot és Position osztályt.

**Direction (1 pont)**

A Direction enum csak 4 irányt tartalmaz.

**Position (7 pont)**

A Position az entitások koordinátáinak ábrázolására szolgál.

Amennyiben a Position osztályt a szélső értékeken kívül eső (0 < [koordináta] < 20) értékekkel próbálják példányosítani dobjunk el egy IllegalArgumentException-t Coordinates out of bounds! üzenettel.

* A move() művelet segítségével lehessen egy pozíciót elmozgatni megadott irányba. Visszatérési értéke jelezze a mozgás sikerességét. Akkor sikertelen a mozgás, ha a pozíciót a megadott szélsőértékeken kívül próbálják mozgatni (ebben az esetben ne változzon a pozíció).
  + A move() művelethez tartozik kettő segédfüggvény, ami az irányok koordináta különbségét hivatott kiszámolni. Ezt következő módon készítsük el:
    1. Direction.RIGHT X koordinátát növeli eggyel
    2. Direction.LEFT X koordinátát csökkenti eggyel
    3. Direction.UP Y koordinátát növeli eggyel
    4. Direction.DOWN Y koordinátát csökkenti eggyel
* A turnDistance térjen vissza azzal, hogy hány kör alatt lehet egyik pozícióból a másikra eljutni (a koordináták különbségének összegével).
* A StructureTest hivatkozik EQUALITY\_CHECK-re. Itt csupán az egyenlőség vizsgálatra lesz szükségünk, a hashCode metódus implementálására elegendő egy minimális return 0 is mivel nem fogunk rá hivatkozni.

**Position UnitTest (6 pont)**

Négy teszt eset készüljön el. Ezek sorra a következők legyenek:

1. Konstruktor tesztelésére testConstructor(int x, int y) paraméterezett teszt, ami csak a paraméterekkel létrehoz egy példányt és teszteli, hogy a megfelelő értékekkel jön-e létre. Legalább három féle paraméterezése szerepeljen.
2. Konstruktor hívását hibás, a definiált korlátokon kívüli értékekkel is teszteljük a testConstructorException() metódusban. Ilyenkor győzödjünk meg arról, hogy a megfelelő hibát eldobja a Position konstruktora.
3. A testMove() teszt hozzon létre egy pontot és mozgassa el minden irányba. Vizsgáljuk, hogy helyesen változnak a mozgatás után a koordináták. Ezután a pálya szélére helyezzünk fel egy másik pontot és gyözödjünk meg róla, hogy nem sikerült kikerülnie a pozíciónak a pályáról a kifele mozgatás hatására.
4. A testDistance() teszt során a távolság definiciónk müködését ellenőrízzük. Kettő Position távolságát vizsgáljuk illetve azt is, hogy ugyanaz a pont 0 távolságra van magától.

**InputSource (9 pont)**

Az InputSource egy olyan interface, ami a bemenetek absztrakciós rétegeként szolgál. Három különböző implementációval kell rendelkeznie, ezek sorra:

* RandomInputSource **(3 pont)**: Egy random generátor segítségével mindig véletlenszerűen generálja a következő irányokat. Ez interaktív játékmód esetén a férgek irányítására szolgál majd.
* InteractiveInputSource **(5 pont)** : A paraméter nélküli konstruktora létrehoz egy BufferedReader-t, ami a parancssori bemenetről fogja a irányokat beolvasni. Ez interaktív játékmód esetén a játékos irányítását fogja működtetni. A másik konstruktor a paraméterül kapott Reader-rel hoz létre BufferedReader-t.
  + Feltehetjük, hogy csak jó bemeneteket kap a programunk.
  + A felsorolási típusra való konverzióra pedig az Direction-hoz tartozó valueOf(String str) műveletet érdemes használni.
  + Tipp: A BufferedReader-t így érdemes inicialízálni: this.br = new BufferedReader(new InputStreamReader(System.in)); a nyilvános konstruktorban.
  + A nextDirection a következő sor olvasásával kapott irányt adja vissza.
  + A potenciálisan felmerülő hibát kezeljük le lokálisan a következő üzenet kiíratásával: Reading unsuccessfull! és a defaultDirection mező visszaadásával.
* PredefinedInputSource: **(1 pont)** Ez az osztály csupán leszármazik az InteractiveInputSource-ből és a konstruktorából megkapott fájlnévből készített FileReader-el hívja meg a szülő konstruktorát. Így ez az osztály egy fájlból fogja majd felolvasni a bemeneteit.
  + Az ellenőrzőtt hibát ne kezeljük le lokálisan, hagyjuk továbbterjedni.

Entity **(3 pont)**

Az Entity osztály egy sablona lesz a mozgó osztályainknak (Player, Worm), amely nem példányosítható. Ennek egy InputSource és egy Position tipusú mezöt kell eltárolnia. (A konstruktor első két paramétere a Position két koordinátája.) A move() művelet a position mező ugyanilyen move() metódusát hívja meg az InputSource interface által meghatározott nextDirection-ból származó iránnyal.

* A Player és a Worm osztályok származzanak le az Entity osztályból és csak használják az ősosztályának konstruktorát.

**Maga a játék (13 Point + 11T)**

A SandPileGame fogja a játék logikáját implementálni. Ez alapján az ő mezői fogják a játék összes állapotát tárolni. Lehet létrehozni segédfüggvényeket, amelyekkel elkerüljük a kódismétlést abban az esetben is, ha nem kéri a structure teszt.

A tesztelhetőség miatt kétféle módon lehessen létrehozni a SandPileGame osztályt.

A Player minden esetben a pálya közepéről, a (10, 10)-es koordinátákból induljon.

A játékban nem jelent problémát, ha két féreg ugyanazon a mezőre kerül, ennek elkerülésével nem kell külön foglalkozni.

**GameState enum**

A játék lehetséges állapotait tartalmazza. A SandPileGame kontruktorai állítsák be a játék állapotát GameState.GAME\_ONGOING-ra.

**Teszt konstruktor (3 pont)**

* A három paraméteres konstruktor hozzon létre megfelelő számú Worm objektumot, amelyek paraméterként kapott pozicíóra helyez el. Hozzon létre megfelelő módon Player osztályt a neki megfelelő inputtal. A játék állapotát állítsa GAME\_ONGOING-ra.

**Interaktív konstruktor (4 pont)**

* Az egy paraméteres konstruktor 2-nél kevesebb féregszám esetén váltson ki IllegalArgumentException kivételt a következő üzenettel: "At least 2 worms required!"; inicializálja a random generátort; a player mezőt állítsa InteractiveInputSource használatával és a megfelelő mezővel; a férgek legyenek RandomInputSource használatával olyan módon lehelyezve, hogy legalább 3 távolságra legyenek a játékostól; a játék állapotát állítsa GAME\_ONGOING-ra.
* A randomPosition két random koordinátával generált Position-nel térjen vissza.
* A wormsGotPlayer meghatározza, hogy a férgekre rálépett-e a játékos.

**Turn (6 pont)**

A turn() metódus a következőket tegye:

* Amennyiben a játék state-je nem GAME\_ONGOING, térjen vissza magával a state-tel.
* Különben mozgassa a férgeket, amennyiben ezután ráléptek a játékosra, akkor állítsuk be a PLAYER\_LOST állapotot.
* Ha nem léptek rá a játékosra a férgek, akkor mozgassa a játékost
* A játékos akkor nyer (PLAYER\_WON állapot beállítása) ha a lépésével kilépne a játéktábláról (ehhez használjuk a visszatérési értékeket).
* Amennyiben rálépett féregre, veszítsen (PLAYER\_LOST állapot beállítása).
* A függvény minden esetben térjen vissza a state mező értékével.

**Tesztelés (11 pont)**

**SandPileGameTest**

Kettő darab teszt esetet hozzunk létre a SandPileGameTest osztályban. Mindkét esetben vizsgáljunk meg, hogy a játék a várt eredménnyel ért véget illetve annyi kör alatt ér véget, amennyire számítunk. Ellenőrizzük mindkét esetben, hogy miután vége lett a játéknak a játékállapoton ne változtazzon a turn() művelet hatására.

Minden PredefinedInputSource létrehozásakor hagyjuk továbbterjedni a hibát.

Tipp: Minden Entity számára külön szöveges fájlba helyezzük el a mozgásait úgy, hogy a fájl minden sorában pontosan egy Direction szerepel csak.

Az elkészült .txt fájlokat is mindenképp töltsd fel a megoldásod mellé.

**Tesztek**

* Az első teszt esetben a játékos egyenes uton kimegy a pályáról. A férgek eddig oda-vissza mozogjanak.
* A második teszt esetben pedig a játékos mozogjon oda-vissza vízszintesen és pár kör lejátszás után kapja el az egyik féreg, pl. legyen féreg a játékossal egy sorban és menjen az irányába. Ellenőrizzük, hogy az első két körben még tart a játék. Ezt követő lépés után a játék állapota álljon be PLAYER\_LOST-ra.

**Letöltés**

Minden szükséges (megengedett) fájl az elméleti Canvasben, a Fájlok alatt található.

* A Java API dokumentáció.
* A JUnit dokumentációja.
* junit5-checkthat-demo.zip és junit5-all.jar
* [A feladathoz a strukturális/suite tesztelő innen tölthető le.Linkek egy külső oldalra](http://exam.inf.elte.hu/zh.zip)
  + Ha a link nem működne, akkor ki lehet másolni, és a böngésző címsorába közvetlenül lehet bemásolni.
  + A böngésző mindenféle indiszkrét kérdést tehet fel: rákérdezhet, hogy megtartjuk-e a fájlt, jelezheti, hogy potenciálisan veszélyes stb.