Programozás I.

Gyakorló feladat

Éhes cicuskák

Általános követelmények, tudnivalók

- A feladat beadási határideje: 2023.05.28. 22:47:19. Ez szigorú határidő, a Bíró előre megadott időben zár. Pótlásra, javításra a határidő lejárta után nincs lehetőség.
- Az elkészült programot 101 alkalommal lehet benyújtani, a megadott határidőig.
- A feltöltések közül a legmagasabb pontszámú számít a feladat értékelésekor.
- A nem forduló kódok nem kerülnek kiértékelésre. Az előre megírt osztályok, adattagok és metódusok szükségesek a teszteléshez, így azokat kitörölni nem szabad, különben a tesztelés egy része vagy egésze nem elvégezhető.
- Amennyiben egy feltöltött osztályban fordítási hiba van, akkor az az osztály egyáltalán nem tesztelhető.
- Ha végtelen ciklus van a programban, akkor a kiértékelést a Bíró 60 másodperc után megszakítja és az eredmény 0 pont lesz.
- A kész forráskódot tömörítve, zip formátumban kell feltölteni. Ügyelj rá, hogy a feltöltött zipnek a teljes package hierarchiát tükröznie kell. A zipet kitömörítve közvetlenül a default package-ben lévő package-et kell látnunk (prog1). A legtöbb fejlesztői környezet esetén ez azt jelenti, hogy az src mappa tartalmát kell betömörítenünk.
  - Zip készítése: Windowson és Linuxon is lehet a GUI-ban jobb klikkes módszerrel tömörített állományt létrehozni (Windowsban pl. a 7-Zip nevű ingyenes program használatával).
  - Linux terminálon belül például a "zip feladat.zip *.java" paranccsal is elkészíthető a megfelelő állomány.
- A bíró a programot Java 17-es verzióval fordítja és futtatja.
- Az értékelés a megvalósítandó interface-ek metódusain, illetve az egyéb, előre elkészített, de nem implementált függvényeken keresztül történik.
- A kiértékelés során a program egyes részei függnek egymástól, egyes részek a többi nélkül nem megvalósíthatók, nem tesztelhetők, így erre érdemes odafigyelnünk a megoldások elkészítésekor.
- Ebből adódóan, ha a program egy fontosabb részében van hiba, az a program más részeire is hatással lehet, tehát érdemes azt kijavítani először.

- A bíró jelzi, hogy az egyes funkcióknak mely funkciók az előfeltételei. Ha az ikon pirossal jelenik meg, akkor az előfeltételt nem sikerült tökéletesen teljesíteni. Ez nem jelenti azt, hogy ez a részfeladat is 0 pontot ér, csak annyit jelez, hogy lehet, hogy az előfeltétel nem teljesülése miatt nem kapunk rá pontot.
- A kiértékelés részletesen megtekinthető a Bíróban feltöltés után, ahol a rendszer kiírja, hogy mely funkcionalitások értek pontot és melyek nem. Ezen segítség alapján a hibák könnyen javíthatók.
- Ahol a feladat külön nem kéri, referenciákat használjunk és ne hozzunk létre feleslegesen új objektumokat.
- A program elkészítése során semmilyen külső függvénykönyvtár nem használható.
- A Bírós kiértékelő szándékos "kijátszása" esetén (az adott funkciók megvalósítása nélkül egy hibát kihasználva pontszerzés) a kapott pontszám a gyakorlatvezetők által később törölhető.

Kódminőség ellenőrzése

A Bíró a kód minőségét is ellenőrzi, amely rengeteg metrikát magában foglal. Ezeket részletesen a Bíróban lévő riportban megtekinthetjük, illetve mindegyikről rövid leírást olvashatunk, hogy pontosan mit mér és hogyan lehet javítani az adott pontszámon. Számos olyan szempont van, aminél több, mint 100%-ot lehet elérni. A plusz pontszám beleszámít a kódminőségre kapott teljes pontba, így az akár 100%-nál magasabb is lehet. A maximum pontszám szempontonként eltér (de minimum 100, maximum 125 százalék). A kód minősége az alábbi csoportok mentén történik:

- Objektumorientáltság: a programunk mennyire jól valósítja meg az objektumorientáltság elveit, a megoldás mennyire felel meg az objektumorientáltság szemléletmódjának.
- Kódbonyolultság: a programunk mennyire egyszerű és könnyen átlátható. Alapelvként annyi elmondható, hogyha egy osztály vagy metódus túl hosszú, akkor valószínűleg nem jó úton járunk.
- Duplikátumok: a programunk mennyi duplikátumot (klónt, azaz Ctrl+C, Ctrl+V-vel másolt és beillesztett) tartalmaz. A kódismétléseket érdemes elkerülni.
- Warningok: a programunk mennyi warningot tartalmaz
- Antipatternek: a programunk mennyi olyan részt tartalmaz, amely egyenesen ellentmondásba ütközik az objektumorientáltság 1-1 fontos alapelvével
- Dokumentáció: a programunk mennyire (jól) van dokumentálva
- Elnevezések: a programunkban a változók, adattagok, metódusok, osztályok a tanult, Java-ban használatos konvenciók szerint vannak-e elnevezve
- Kódolási stílus: a programunk külalakját ellenőrzi, hogy mennyire van szépen tagolva, a különböző operátorok, zárójelek, stb. mennyire vannak megfelelően használva

- Programkód szervezése: a programunkban vannak-e olyan elemek, amelyek bár a működésre nincsenek hatással, de a program átláthatóságát, olvashatóságát valamilyen szinten rontják.

Értékelés

A Bíróban az értékelésnek 3+1 szakasza van.

- Kötelező elemek: ezek az alapvető funkcionalitásokat tesztelik, amelyek nélkül a program többi része nem tesztelhető. A többi teszt futtatásához ennek a résznek tökéletesen kell lefutnia. Ezek a tesztek nem érnek pontot. A minta.zip-ben lévő megoldáskezdemény már teljesíti az összes feltételt, csak arra kell figyelni, hogy el ne rontsuk azokat.

- Funkcionalitás: a kért funkciókat ellenőrzi, hogy azok meg vannak-e valósítva és megfelelően működnek-e.

- Kódminőség: a kód minőségét teszteli a fent leírt, illetve a bíróban látható szempontok mentén.

- Statisztika: a programunkkal kapcsolatos néhány statisztika. A kiértékelés során semmilyen jelentősége nincs, de érdekességképpen megtekinthető.

A feladat végső pontszáma funkcionalitásra és kódminőségre kapott pontszámok összege. A kódminőségre kapott pontszám korrigálásra kerül a funkcionalitásra kapott pontszámmal. Például:

- Ha a funkcionalitás 100%-os, akkor a kódminőségre annyi pont jár, amennyit a bíró kiszámított.

- Ha a funkcionalitás 50%-os (félig van kész a feladat), akkor a kódminőségre kapott pontszám megszorzásra kerül 0.5-tel

- Ha a funkcionalitás 10%-os, akkor a kódminőségre kapott pontszám megszorzásra kerül 0.1-gyel

A feladatra kapott pontszám a riport.html fájlban látható. A bíró felületén az összesítésben (a bíró limitációi miatt) a kapható pontszám 100x-osa jelenik meg, erre érdemes odafigyelni. A bíró kijelzi, hogy a feladatra mennyi a minimum elérendő pont (ahol ebből adódóan szintén egy 100x-os érték látható).

Program elkészítése

A program elkészítéséhez egy elkezdett keretrendszert kell kibővíteni és a hiányzó részeit megvalósítani. A keretrendszer a Bíró-ról letölthető minta.zip néven.

Kicsomagolás után ezt beimportálhatjuk egy általunk kedvelt fejlesztői környezetbe (opcionális). A keretrendszer dokumentálva van: minden package, osztály, adattag és metódus tartalmaz JavaDoc kommenteket, amelyeket a forráskódon belül is megtekinthetünk, azonban célszerű legenerálni belőle a HTML dokumentumot (a kedvenc fejlesztői környezetünkben, vagy parancssorból a javadoc paranccsal) a jobb átláthatóság kedvéért.

Ebben láthatjuk minden osztálynak a leírását, illetve a megvalósítandó függvényeknél minden azzal kapcsolatos tudnivalót. Az esetleges "hiányzó" információkat ezen pdf tartalmazza.

A kiadott osztályok alapján UML diagramot készíthetünk, amely segít megérteni a projekt szerkezetét. Ehhez ha IntelliJ IDEA-ban dolgozunk, kattintsunk jobb klikkel az src mappára, majd ott diagrams -> show diagram. Ha itt egyes helyeken csak package-eket látunk, akkor jobb klikk, majd expand nodes, és akkor az összes package-et is ki fogja fejteni. Ezt addig ismételjük, amíg nem látjuk az összes osztályt. A diagramon alapvetően nem jelennek meg, csak az osztályok. Ha szeretnénk látni az adattagokat, metódusokat és kapcsolatokat, akkor ezt a bal felső sarokban tudjuk szabályozni (fields, constructors, methods és show dependencies) a kis ikonok segítségével.

Ha szeretnénk csak egy adott package-ben lévő osztályokat látni, azt is megtehetjük, ezesetben az adott package-re kell jobb klikkelni az src helyett.

Másik fejlesztői környezetben is nagy valószínűséggel lehet UML diagramot generálni, ennek pontos menetéről az interneten érdemes utánanézni.

Megvalósítással kapcsolatos információk

- Új package-ek szabadon létrehozhatóak, de minden package a meglévő solution package-en belül kell, hogy legyen
- Új osztályok szabadon létrehozhatóak, de minden osztály a meglévő solution package-en belül kell, hogy legyen
- A meglévő osztályokban új adattagok, új metódusok szabadon létrehozhatóak
- A meglévő osztályokban már meglévő adattagokat és metódusok törzsét nem célszerű módosítani, mert az a tesztelés során problémákhoz vezethet és pontveszteséggel járhat.
- A meglévő osztályokban már meglévő metódusok, illetve az interface-ek metódusainak fejlécét megváltoztatni tilos, mert az a tesztelés egy pontján garantáltan fordítási hibához vezet és fordítás hiányában a programot nem lehet értékelni.

Kiegészítendő osztályok

A keretrendszer tartalmaz számos olyan osztályt, amelyben vannak nem implementált metódusok. Ezeket úgy tudjuk felismerni, hogy a törzse egy utasítást tartalmaz:
throw new NotImplementedException();
Az ilyen megvalósítandó metódusok:

- prog1.gyakorlo.cats.solution.Human.Human
- prog1.gyakorlo.cats.solution.Human.getName
- prog1.gyakorlo.cats.solution.Human.remainingMoney
- prog1.gyakorlo.cats.solution.Human.buyFood
- prog1.gyakorlo.cats.solution.Human.feedCats
- prog1.gyakorlo.cats.solution.animals.DomesticCat.DomesticCat
- prog1.gyakorlo.cats.solution.animals.DomesticCat.DomesticCat
- prog1.gyakorlo.cats.solution.animals.DomesticCat.foodRequirement
- prog1.gyakorlo.cats.solution.animals.DomesticCat.isHappy
- prog1.gyakorlo.cats.solution.animals.DomesticCat.makeKitten
- prog1.gyakorlo.cats.solution.animals.DomesticCat.toString

Megvalósítandó interface-ek

A keretrendszer tartalmaz számos interface-t, amelyet jelenleg egy osztály sem implementál. A feladat ezen interface-ekhez olyan osztályt készíteni, amelyek azokat implementálják, és természetesen az összes metódusnak értelmes implementációt adni.

Amennyiben egy interface-t csak egy osztály implementálja, akkor a Bíró automatikusan megtalálja azt. Amennyiben több implementáció is tartozik hozzá, akkor használjuk a @Testable annotációt azon az osztályon, amelyiket a bíróval teszteltetni szeretnénk. Ha nem használjuk (vagy több osztályra is rátesszük), akkor a tesztelésre kerülő osztály véletlenszerűen kerül kiválasztásra.

Az ilyen interface-ek:

- prog1.gyakorlo.cats.solution.places.Garden
- prog1.gyakorlo.cats.solution.places.Supermarket

Kötelező elemek

Az alábbi szempontok kötelezőek, ezek nélkül a program nem tesztelhető.

- Az eredetileg megadott interface-eknek meg kell maradniuk
- Az eredetileg megadott konstruktorok fejléceinek változatlannak kell maradniuk
- A Garden és Supermarket interface-eket megvalósító osztályoknak rendelkezniük kell default konstruktorral

Feladatleírás

Bevezetés

Az öreg kovács nagyon szereti a macskákat, így néhányat tart is a kertjében (mármint nem a kezében). Természetesen ez azt is jelenti, hogy ezekről a macskákról gondoskodnia kell, etetni kell őket. A dobozos macskaeledelt a helyi szupermarketből szerzi be. Nyilván változó, hogy melyik macska mennyit eszik, és nem kell mindegyiknek egy teljes dobozzal odaadni, így az akár több macskának is elegendő.

Természetesen komoly koordinációt igényel ez a feladat, mivel ügyelni kell rá, hogy mindig legyen otthon elég macskaeledel, illetve az sem előnyös, ha az öreg kovács pénze elfogy, mert akkor nem tudja megetetni szegény cicáit, akik ettől elszomorodnak.

Vannak ám nála gazdagabb emberek is, akiknek több kertjük is van, rájuk is gondolnunk kell, nehogy elhanyagoltnak érezzék magukat.

Készítsünk programot a fent leírt rendszer leírására, amely az alábbi komponenseket tartalmazza: bolt, macska, ember és kert.

Bolt

A boltot valószínűleg nem kell bemutatni senkinek. Ő itt bolt.

A boltban termékeket árulnak, amelyeket a beszállító hoz a boltba. Néha új termékekkel bővül a paletta. A termékeket a vásárlók vásárolják meg, akik ezért pénzzel fizetnek. A termékek ára a külső tényezők hatására változhat.

Macska

A macska egy aranyos állat, és mint minden élőlénynek, neki is ételre van szüksége. Ha gyakran etetik, akkor nagyon boldog jószág lesz. Ha két egymáshoz közel álló, azonos korú macska találkozik egymással, annak eredménye lehet egy kiscica.

Ember

Az emberek pénzből élnek. A néhány egyéb kiadás mellett azonban a leglényegesebb dolog, amire a pénzüket költik, az a macskaeledel, azaz a "Cat food", amelyet mindenki a kedvenc boltjában vásárol meg.

A megvásárolt macskaeledelből már meg tudja etetni a saját kertjeiben lévő macskákat.

Kert

Azt már tudjuk, hogy a kertekben lehetnek macskák (továbbá bármikor érkezhetnek újak is),viszont minden kertnek van gazdája. A kertek akár gazdát is cserélhetnek, vagy ha elhagyatott volt, akkor az államtól lehet megvásárolni (az államot (és az államat sem) a jelenlegi rendszerben nem modellezzük). Egy kertnek csak egy tulajdonosa lehet, viszont egy ember rendelkezhet több kerttel is.