Programozás I. Gyakorló feladatsor

SZTE Szoftverfejlesztés Tanszék

2023. tavasz

Általános követelmények, tudnivalók

- A feladat elkészítési határideje: **vasárnap 23:59:59**. Ez szigorú határidő, a Bíró előre megadott időben zár, pótlásra nincs lehetőség.
- A feladatokat számítógép előtt kell megoldani, tetszőleges fejlesztői környezetben, tetszőleges operációs rendszer segítségével.
- Az elkészült programot **20** alkalommal lehet benyújtani, a megadott határidőig.
- Csak a leírásban szereplő osztályokat, metódusokat és adattagokat kell megvalósítani, egyéb dolgokért nem jár plusz ont.
- A feladat megoldása során minden megadott előírást pontosan követni kell! Tehát, ha a feladat leírása szerint egy adattag neve a "elsoFoku", akkor az alábbi elnevezések nem megfelelőek: "elsőFokú", "elsofoku", "elso_foku", "elsőFoq". Ugyanez igaz a metódusok, osztályok elnevezésére is!
- A metódusok esetében a visszatérési típus, a név, módosítók és a paraméterek típusai (és azok sorrendje) kerülnek ellenőrzésre, azonban a paraméterek nevei tetszőlegesek lehetnek.
- Az órán tanult konvenciókat követni kell (getter/setter elnevezés, toString, indentálás, stb). Abban az esetben is, ha ezt a feladat külön nem emeli ki, az ellenőrzés során erre is építünk.
- A nem forduló kódok nem kerülnek kiértékelésre, ezt utólagosan a gyakorlatvezető sem bírálhatja felül. (Hiszen mindenki rendelkezésére áll a saját környezete, ahol fordítani, futtatni tudja a forráskódot, így feltöltés előtt ezt mindenképpen érdemes megnézni!)
- Az adattagok és konstruktorok hiányában garantáltan 0 pontos lesz a kiértékelés, mert ezek minden teszt alapját képezik.
- Ha végtelen ciklus van a programban, akkor ezt a Bíró ki fogja dobni 3 másodperc után (ha többször is meghívásra kerül ilyen metódus, akkor ez többszöri 3 másodperc, összesen akár 2 perc is lehet). Ilyenkor NE kattints még egyszer a *Feltöltés* gombra, mert akkor kifagyhat a Bíró, csak a böngésző újraindításával lehet megoldani a problémát (emellett elveszik 1 feltöltési lehetőség is).
- Kérdés/probléma esetén a gyakorlatvezetők tudnak segítséget nyújtani.
- **A feladat megoldása során a default csomagba dolgozz, majd a kész forrásfájlokat tömörítve, zip formátumban töltsd fel, azonban a zip fájlt tetszőlegesen elnevezheted!**

  - 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 feladatokban az alábbi dolgok az alapértelmezettek (**kivéve**, ha a feladat szövege mást mond)
  - az osztályok láthatósága publikus
  - az egész érték 32 bites
  - a lebegőpontos számok dupla pontosságúak
  - az olyan metódusok void visszatéréssel rendelkeznek, amelyeknél nincs specifikálva visszatérési típus.
  - a metódusok mindenki számára láthatóak
  - az adattagok csak az adott osztályban legyenek elérhetőek

- A *riport.txt* és a fordítási log fájlok megtekinthetőek az alábbi módon:
  1. Az *Eredmények megtekintése* felületen a vizsgálandó próba új lapon való megnyitása
  2. A kapott url formátuma: `https://biro2.inf.u-szeged.hu/Hallg/IB204L/FELADAT/hXXXXXX/4/riport.txt`
  3. Az url-ből visszatörölve a 4-esig (`riport.txt` törlése) megkaphatók a 4-es próbálkozás adatai.

- Szövegek összehasonlításánál az egyezés a pontos egyezést jelenti, azaz ha kis-nagy betűben térnek el, akkor már nem tekinthetők egyenlőnek (pl. a "piros" != "Piros")

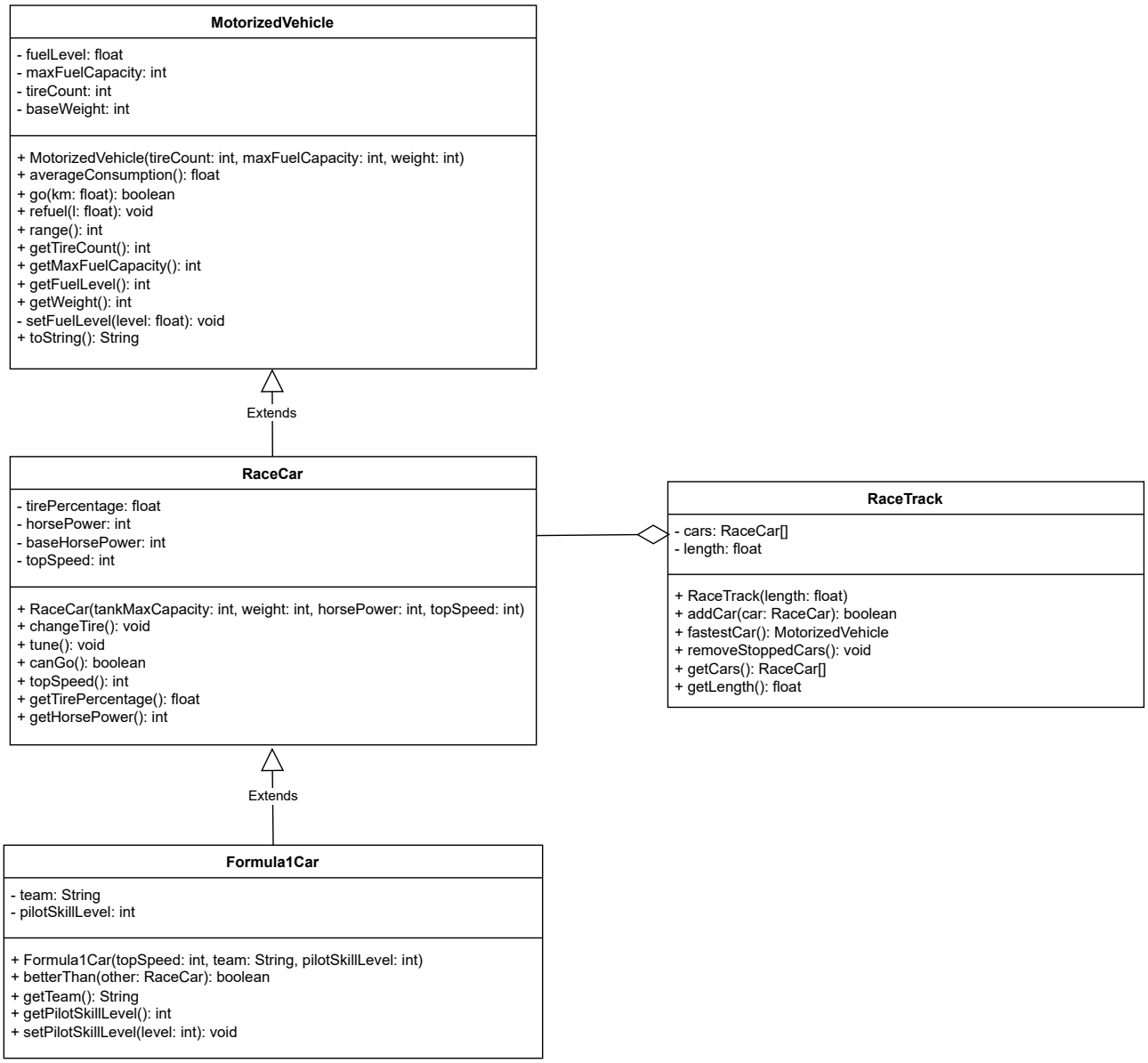
- A leírásokban bemutat példákban a stringek köré rakott idézőjelek nem részei az elvárt kimenetnek, azok csak a string határait jelölik. Például ha az szerepel, hogy a példa bemenetre az elvárt kimenet az, hogy `"3 alma"`, akkor az elvárt kimenet idézőjelek nélkül az `3 alma`, de a szóköz szükséges!

- Az elvárt kimeneteknek karakterről karakterre olyan formátumúnak kell lennie, ami a feladatban le van írva (szóközöket és sortöréseket is beleértve).

Jármű osztály (13 pont)

Írj egy `MotorizedVehicle` nevű osztályt, ami egy motorizált járművet reprezentál. Az adattagokat, valamint a szükséges metódusokat a **??**. ábrán láthatjuk. Ügyelj a megfelelő láthatóságokra.

1. ábra. MotorizedVehicle osztálydiagram



Konstruktor (2 pont)

A jármű összes adattagja csak nemnegatív szám lehet, valamint a fuelLevel attribúrum a tank maximális kapacitásánál nem vehet fel nagyobb értéket.
A fuelLevel az üzemanyagtartályban lévő üzemanyag mennyiséget tárolja el, a maxFuelCapacity

tartály maximális kapacitását, a tireCount a kerekeink számát és a baseWeight a járművünk tömegét (üzemanyag nélkül) tárolja el.

Getterek (1 pont)

A getter metódusok értelemszerűen működjenek.

Metódusok (10 pont)

`averageConsumption()` metódus visszaadja, hogy a járművünk 100 km alatt átlagosan hány liter üzemanyagot használ el. Ezt az értéket a következőképpen számoljuk: **(2 pont)**

$$5 * \sqrt{\frac{baseWeight}{1302}}$$

`go()` metódus paraméterben a megtenni kívánt távolságot jelképezi km-ben mérve. Az utazás során az üzemanyagtartájban lévő üzemanyag az átlagfogyasztás szerint csökken. Ha nem áll rendelkezésre elegendő üzemanyag, akkor inkább el sem indulunk a jűrművünkel. **(2 pont)**

`refuel()` metódus paraméterben azt kapja, hogy mennyi üzemanyagot próbálunk beletölteni a járművünkbe. Ez a metódus ennek megfelelően változtatja az üzemanyagszintet. **(1 pont)**

A `range()` metódus azt adja vissza, hogy a tartályban lévő üzemanyagmennyiséggel hány km-t tudunk még megtenni. Egész számot ad vissza, ha belekezdünk egy újabb km-be arra úgy tekintünk, hogy nem tudjuk megtenni. **(2 pont)**

`getWeight()` a járművünk tömegét adja vissza kg-ban mérve. Itt a tartályban lévő üzemanyag tömege is számít.
Esetünkben az üzemanyag sűrűsége: 770 $\frac{kg}{m^3}$ **(2 pont)**

A *toString()* metódus a járművünk adatait adja vissza. Az üzemanyagszintet 1 tizedesjegyig kerekítsük, a végén legyen sortörés.
Az alábbi formátumba legyen:
"*Alapsuly: {baseWeight} Kg, Kerekek szama: {tireCount}, Uzemanyag tartaj: {fuelLevel}/50 l*" **(1 pont)**

Versenyautó osztály (11 pont)

Írj egy `RaceCar` nevezetű osztály, amely a `MotorizedVehicle` osztályból öröklődik.

Konstruktor (1 pont)

A versenyautókról annyit tudunk, hogy 4 kerekük van. A versenyautókról érdemes lehet tudni, hogy hány lóerősek, milyen a kerékállapotuk valamint mennyi a végsebességük. A baseHorsePower az autó eredeti lóerője, a horsePower az autó aktuális lóerője, a tirePercentage a kerekek állapota százalékos formában, a topSpeed pedig a végsebessége. Minden versenyautó legalább 1 lóerős, és a létrehozásuk után új gumikkal rendelkeznek.

Getterek (1 pont)

A getter metódusok értelemszerűen működjenek.

Metódusok (9 pont)

Definiáljuk felül az ősosztály `go()` metódusát! Mivel a kerekek kopnak ha megyünk az autóval, ezért ezt a viselkedést is tegyük bele. Minden egyes megtett km után a kerekek 0.01%-al használódnak el. (tipp: a super.metodusNev()-el meg tudjuk hívni az ősben létrehozott metódust) **(3 pont)**

`changeTire()` metódus jelképezi az autó gumiainak a cseréjét. Csak akkor tudjuk lecserélni a gumit, ha a gumi minősége nem nagyobb mint 20%. **(1 pont)**

`tune()` metódus jelképezi az autó tuningolását. Minden egyes tuning során 8%-al növeljük meg az autónk lóerejét. Matematikailag helyes kerekítést használjunk. Ez az érték nem haladhatja meg a kezdeti lóerő 130%-át. **(2 pont)**

`canGo()` metódus azt adja vissza, hogy az autónk tud-e menni. Egy autó csak akkor tud menni, ha a kerékminősége nagyobb mint 0-a és a tartályban van üzemanyag. **(1 pont)**

`topSpeed()` metódus azt adja vissza, hogy az autónknak mennyi a jelenlegi végsebessége. A végsebesség függ attól, hogy hány százalékkal nőtt meg a lóerőnk. Vagyis az eredeti végsebesség szorzata a lóerőnövekedés gyökével.
(pl. a lóerő növekedés 8% akkor: $topSpeed * \sqrt{1.08}$) **(1 pont)**

Definiáljuk felül a *toString()* metódust. Használjuk az alábbi formát: "*Ez egy {horsePower} loeros versenyauto, aminek {vegsebesseg} km/h a vegsebessege es jelenleg {tirePercentage} %-os gumik vannak rajta.*" **(1 pont)**

Forma1 autó osztály (6 pont)

Írj egy `Formula1Car` nevezetű osztály, amely a `RaceCar` osztályból öröklődik.

Konstruktor (1 pont)

A forma1-es autókról elég sokmindent tudunk. Az üzemanyagtartályuk 140 l-es a tömegük 789 kg, valamint a motoruk 1000 lóerős. Ezentúl minden forma1-es autónak tartozik egy csapathoz, és az autó pilótájának is van egy képességi szintje. Ezeket rendre a team és pilotSkillLevel attribútomokban tároljuk. A pilóta képessége 0 és 10 közötti érték lehet.

Getterek (1 pont)

A getter metódusok értelemszerűen működjenek.

Metódusok (4 pont)

Definiáljuk felül a `tune()` metódust. Mivel ezek nagyon kifinomult járművek, ezért ezeket nem lehet tovább tuningolni. A metódus meghívására ne történjen semmi. **(1 pont)**

`betterThan()` metódus azt adja vissza, hogy a paraméterben kapott másik versenyautónál jobb-e. Csak akkor vagyunk jobbak ha távolabbra tudunk menni és gyorsabbak is vagyunk a másik autónál. **(2 pont)**

Definiáljuk felül a *toString()* metódust. Használjuk az alábbi formát: Vezetesi szint a pilóta képességétől függ: 0-3 rossz 4-6 atlagos 7-10 kivalo "*Ez egy {team} csapathoz tartozo forma1-es auto, aminek {topSpeed}d a vegsebessege es {vezetesiSzin} a pilotaja.*" **(1 pont)**

Versenypálya osztály (6 pont)

Készítsd el a `RaceTrack` nevezetű osztály.

Konstruktor (1 pont)

Egy versenypályáról csak a hosszát tároljuk el a length változóban. Ezentúl a pályán lehetnek versenyautók is. Az osztály inicializálásakor hozzunk létre egy 20 elemű tömböt ami `RaceCar`-típusú objektumokat tárol.

Getterek (1 pont)

A getter metódusok értelemszerűen működjenek.

Metódusok (4 pont)

Az `addCar()` metódus hozzáadja a tömbhöz a paraméterben kapott autót. Egy objektumot csak egyszer lehessen hozzáadni. Visszatérési érték az az, hogy sikerült-e hozzáadni az autót. **(3 pont)**

`fastestCar()` metódus visszaadja a leggyorsabb autót, ami a pályán van. **(2 pont)**

`removeStoppedCars()` metódus eltávolítja azokat az autókat a pályáról amelyek már nem tudnak menni. **(3 pont)**

Jó munkát!