Órai Feladat

Határidő:

110 perc

1. Egyszerű osztály – Car

Készíts egy `Car` osztályt, és haladj lépésenként:

a) Hozd létre az osztályt, és add meg az attribútumokat: `brand`, `model`, `year`.
b) Készíts hozzá egy `__init__()` konstruktort, ami ezeket beállítja.
c) Adj hozzá egy `start()` metódust, ami kiírja:
`"A(z) <brand> <model> beindult."`
d) Csinálj két példányt (pl. egy Toyota és egy Ford), és hívd meg mindkettőn a `start()` metódust.
e) Írj egy `__str__()` metódust, ami visszaadja olvasható formában a kocsi adatait, pl. `"2020 Toyota Corolla"`.
f) Mentsd a két autó adatait egy `cars.txt` fájlba soronként (használd a `__str__()` kimenetét).
g) Gondolkodós rész: írj egy `from_file()` osztálymetódust, ami beolvassa a `cars.txt` tartalmát, és minden sorból létrehoz egy `Car` objektumot (tehát dinamikusan építsd fel az objektumokat fájlból).

---

2. Öröklés – ElectricCar

Fejleszd tovább az előző `Car`-t:

a) Hozd létre az `ElectricCar` osztályt, ami örökli a `Car`-t.
b) Adj hozzá egy új mezőt: `battery_capacity` (kWh-ban).
c) Írd felül a `start()` metódust úgy, hogy kiírja:
`"A(z) <brand> <model> hangtalanul indul."`
d) Példányosíts egy elektromos autót és egy simát, majd hívd meg mindkettő `start()` metódusát, hogy lásd a különbséget.
e) Adj hozzá egy `charge()` metódust, ami kiírja:
`"A(z) <brand> <model> töltése folyamatban..."`
f) Exportáld az elektromos autók adatait egy `electric_cars.json` fájlba (lista formában, JSON objektumként).
g) Gondolkodós rész: írj egy metódust, ami visszaadja, hány órába telne a teljes töltés, ha a töltő teljesítménye `charger_kw` paraméterként érkezik. Figyelj, hogy ne osztás nullával történjen, és kerekítsd két tizedesre.

---

3. Adatelrejtés – BankAccount

Most gyakoroljuk az enkapszulációt:

a) Hozd létre a `BankAccount` osztályt, benne egy privát `__balance` attribútummal, amit a konstruktorban 0-ra állítasz.
b) Írj egy `deposit(amount)` metódust, ami hozzáadja az összeget az egyenleghez.
c) Írj egy `withdraw(amount)` metódust, ami levonja az összeget, de csak akkor, ha van elég pénz. Ha nincs, írja ki:

`"Nincs elég egyenleg a művelethez."`
d) Adj hozzá egy `get_balance()` metódust, ami visszaadja az aktuális egyenleget.
e) Kezeld az invalid értékeket (pl. negatív befizetés/levétel).
f) Írj egy `save_to_csv(filename)` metódust, ami a számlaadatokat menti `csv` fájlba (`account_number`, `balance`).
g) Gondolkodós rész: készíts egy `load_from_csv(filename)` osztálymetódust, ami több sort beolvasva több `BankAccount` objektumot hoz létre.
Ha duplikált számlaszám van, kezeld úgy, hogy a későbbi felülírja a korábbit.

---

4. Polimorfizmus – Shape

Polimorfizmus egy klasszikus példán:

a) Hozd létre a `Shape` alaposztályt egy `area()` metódussal, ami `NotImplementedError`-t dob.
b) Készíts egy `Rectangle` osztályt (örököl a `Shape`-ből), ami `width` és `height` alapján számolja ki a területet.
c) Készíts egy `Circle` osztályt (szintén `Shape`-ből), ami `radius` alapján számolja ki a területet.
d) Hozz létre egy listát különböző `Shape` objektumokkal (néhány téglalap, néhány kör), és járd be őket egy ciklussal, mindegyiken hívd meg az `area()` metódust.
e) Adj a `Shape`-hez egy opcionális `name` attribútumot, és írd ki: `"A(z) {name} területe: {area}"`.
f) Írj egy `shapes_to_json(filename)` függvényt, ami a különböző alakzatokat és azok területét elmenti JSON formátumban.
g) Gondolkodós rész: olvass vissza a JSON-ból, és automatikusan hozd létre újra a megfelelő `Rectangle` vagy `Circle` objektumokat.
A programnak tudnia kell, hogy melyik típus melyik (tehát `type` mezőt is ments el).