1) Első osztály: BankAccount (privát mezők + metódusok)

Cél: class, konstruktor, privát property-k (encapsulation), metódushívás.

Feladat:

1. Hozzon létre egy BankAccount osztályt külön fájlban.
2. Az osztály tárolja privát mezőkben:
    - tulajdonos nevét,
    - egyenleget.
3. Legyen konstruktora, ami beállítja az adatokat (kezdő egyenleg minimum 0).
4. Legyenek publikus metódusok:
    - deposit(összeg): pozitív összeg hozzáadása,
    - withdraw(összeg): pozitív összeg levonása (ne lehessen mínuszba menni),
    - getBalance(), getOwner().
5. Készítsen index.php oldalt űrlappal:
    - tulajdonos,
    - kezdő egyenleg,
    - összeg,
    - művelet (befizetés/kifizetés).
6. A művelet eredményét és az aktuális állapotot táblázatban jelenítse meg.
7. Hibákat kezeljen értelmes üzenetekkel (pl. negatív összeg, nincs fedezet).

---

2) Több osztály együtt: Product + CartItem + Cart

Cél: több osztály együttműködése, objektumok tárolása tömbben.

Feladat:

1. Hozzon létre Product osztályt: id, név, ár (Ft).
2. Hozzon létre CartItem osztályt: egy Product példány + mennyiség + részösszeg számítás.
3. Hozzon létre Cart osztályt, ami:
    - termékeket tud hozzáadni (product + qty),
    - vissza tudja adni az elemeket,

- ki tudja számolni a teljes összeget.

4. Készítsen index.php oldalt:
   - legyen egy „katalógus” 3 termékkel (pl. tömbben példányosítva),
   - űrlap: termék választás + mennyiség,
   - kosár tartalmának megjelenítése táblázatban (termék, egységár, mennyiség, részösszeg, összesen).
5. Validálja a mennyiséget és a termék kiválasztást.

---

3) Static metódusok vs példánymetódusok: TextStats

Cél: statikus metódus, példány metódus, „factory” jellegű metódus.

Feladat:

1. Hozzon létre TextStats osztályt, ami egy szöveget tárol.
2. Legyen benne:
   - példánymetódus karakterek számolására,
   - példánymetódus szavak számolására,
   - statikus metódus, ami tetszőleges szövegre szót számol példányosítás nélkül.
3. Készítsen index.php oldalt textarea-val:
   - beküldés után jelenítse meg a statisztikákat táblázatban,
   - mutassa meg ugyanazt az eredményt statikus metódushívással is.
4. Tegyen bele legalább egy „factory” jellegű metódust (pl. POST-ból készít példányt).

---

4) Öröklődés és polimorfizmus: Shape → Circle/Rectangle

Cél: abstract class, öröklés, felületen keresztüli használat.

Feladat:

1. Hozzon létre egy Shape absztrakt osztályt:
   - legyen neve,
   - legyen absztrakt area() és perimeter() metódusa.
2. Készítsen két leszármazottat:

- Circle (sugárral),
- Rectangle (a és b oldalakkal).

3. Készítsen index.php oldalt:
   - választható alakzat (select),
   - a választott alakzatnak megfelelő input mezők,
   - számítás gomb.
4. Az eredményt a nézet úgy jelenítse meg, hogy csak a Shape felületet használja (terület/kerület), ne if-ekkel számoljon a nézetben.
5. Validálja az adatokat (0 vagy pozitív számok).

---

5) Interface + Dependency Injection: Notifier + RegistrationService

Cél: interface, implementációk, konstruktor-injektálás (DI).

Feladat:

1. Hozzon létre Notifier interface-t egy metódussal: notify(to, subject, body).
2. Készítsen két implementációt:
   - EmailNotifier: demóként fájlba logoljon (pl. data/mails.log),
   - NullNotifier: ne csináljon semmit.
3. Készítsen RegistrationService osztályt, ami a konstruktorban Notifier-t kap (DI).
4. A RegistrationService validáljon (név kötelező, e-mail formátum), és siker esetén hívja a notifier-t.
5. index.php oldalon legyen:
   - regisztrációs űrlap (név + e-mail),
   - select: melyik notifier legyen (email/log vagy null),
   - siker esetén PRG (átirányítás), és írja ki a sikert.
6. Listázza az utolsó 20 logsort, és legyen „log törlése” funkció.