# Java kisokos

*Érettségi programozási feladatokhoz*

Ez a segédlet az **Érettségi Kézikönyv 2.0**-ban szereplő érettségi feladattípusokhoz ad Java-mintákat. **A közös megoldási sémákra épít: beolvasás, adatok tárolása, feldolgozás, majd pontos kiírás.**

**Használat:** válaszd ki a feladathoz illő mintát, nevezd át a változókat, és tartsd meg a feladatban kért fájlneveket és kiírási formátumot.

## 1. A program alapváza

A vizsgafeladatokhoz általában elég egyetlen osztály és a main metódus. A fájl neve egyezzen a **public** osztály nevével; ha a környezet ***Main*** osztályt vár, a **fájlt is *Main.java*** néven mentsd.

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) throws IOException {
        List<String> sorok = Files.readAllLines(
            Paths.get("adat.txt"), StandardCharsets.UTF_8);
        // feldolgozás és kiírás
    }
}
```

## 2. Melyik adatszerkezetet válasszam?

| Feladatban szerepel… | Java-típus | Mikor jó? |
|---|---|---|
| Rögzített számú azonos típusú adat | int[] | A méret előre ismert; gyors indexelés. |
| Bővülő, rendezett adatsor | ArrayList<T> | Megőrzi a sorrendet, ismétlődhetnek az elemek. |
| Csak különböző értékek | HashSet<T> | Egyedi elemek; gyors tagságvizsgálat. |
| Kulcshoz tartozó érték / darabszám | HashMap<K,V> | Gyakoriság, csoportosítás, gyors keresés kulcs alapján. |
| Rács, tábla vagy pálya | int[][] / char[][] | Sor- és oszlopindexekkel bejárható kétdimenziós adat. |

**Megjegyzés:** a gyűjtemények generikus típusában objektumtípust használunk: például ArrayList<Integer>, nem ArrayList<int>.

## 3. Fájlbeolvasás és sorok feldolgozása

A legtöbb vizsgafájl szöveges. A readAllLines UTF-8 kódolással sorlistát ad, így először a sorokat, majd a sor mezőit dolgozd fel.

### Szóközzel elválasztott számok

```
List<Integer> meresek = new ArrayList<>();
for (String sor : sorok) {
    if (!sor.trim().isEmpty()) {
        String[] mezok = sor.trim().split("\\s+");
        for (String mezo : mezok) {
            meresek.add(Integer.parseInt(mezo));
        }
    }
}
```

A \\s+ egy vagy több szóköz karaktert jelent. Az Integer.parseInt szövegből egész számot készít; tizedeshez Double.parseDouble kell.

### Pontosvesszővel tagolt rekord

```
for (String sor : sorok) {
    String[] m = sor.split(";", -1); // üres utolsó mezőt is megtart
    int ev = Integer.parseInt(m[0]);
    String tipus = m[2];
    String leiras = m[3];
    int darab = Integer.parseInt(m[4]);
}
```

**Tipp:** a mezőindexek 0-tól indulnak. Ha a feladat több soron át használja az egy rekordhoz tartozó mezőket, tárold őket saját osztályban, például Konyv(ev, tipus, leiras, darab). Például: 2025. május 19. – KÖNYVKIADÁS, 2025. október 21. - SEBESSÉG.

### Felhasználói adat bekérése

```
Scanner sc = new Scanner(System.in);
System.out.print("Keresett név: ");
String nev = sc.nextLine();
int sorszam = Integer.parseInt(sc.nextLine());
```

**A nextInt() után a következő nextLine() még a sorvéget olvashatja be.** Egyszerű vizsgafeladatnál kényelmes megoldás minden adatot nextLine()-nal olvasni, majd számmá alakítani.

# 4. Alap algoritmusminták

## Összegzés és megszámlálás

```
long osszeg = 0;
int db = 0;
for (int x : meresek) {
    if (x != -1) {          // a hibás mérés nem számít bele
        osszeg += x;
        db++;
    }
}
```

A -1 helyére a feladatban megadott jelölőérték kerül. Nagy összegnél használj **long** változót.

## Maximum és első előfordulási hely

```
int max = adatok.get(0);
int maxIndex = 0;
for (int i = 1; i < adatok.size(); i++) {
    if (adatok.get(i) > max) { // > esetén döntetlenben az első marad
        max = adatok.get(i);
        maxIndex = i;
    }
}
int sorszam = maxIndex + 1; // csak ha a feladat 1-től számoz
```

Ha lehet üres a lista, a 0. elem elérése előtt ellenőrizd a méretét. Ha hibajelölő érték nem lehet maximum, azt a ciklusban hagyd ki.

## Keresés: van-e, és hol az első?

```
int i = 0;
while (i < adatok.size() && adatok.get(i) != keresett) {
    i++;
}
boolean van = i < adatok.size();
if (van) {
    System.out.println("Első hely: " + (i + 1));
}
```

## Feltételes darabszám

```
int megfelelo = 0;
for (Konyv k : konyvek) {
    if (k.ev == keresettEv && k.darab >= alsoHatar) {
        megfelelo++;
    }
}
```

**Gondolkodási minta:** először írd le pontosan a feltételt, majd a ciklusban minden rekordra egyszer vizsgáld meg.

# 5. Gyakoriság, egyediség és rendezés

## Darabszámok Map (szótár) segítségével

```
Map<String, Integer> gyakorisag = new HashMap<>();
for (String nev : nevek) {
    int regi = gyakorisag.getOrDefault(nev, 0);
    gyakorisag.put(nev, regi + 1);
}
int db = gyakorisag.getOrDefault("Benedek Elek", 0);
```

Csoportos összegzéshez az érték lehet például egy összeg vagy egy külön számlálókat tartalmazó objektum. Ha a kulcsok rendezett sorrendje is kell, használhatsz **TreeMap**-et.

## Egyedi értékek Set (halmaz) segítségével

```
Set<String> egyedi = new HashSet<>();
for (String nev : nevek) {
    egyedi.add(nev); // ugyanaz az érték csak egyszer marad
}
System.out.println("Különböző nevek: " + egyedi.size());
```

## Rendezés

```
Collections.sort(szamok); // növekvő
Collections.sort(szamok, Collections.reverseOrder()); // csökkenő

// Saját rekordok: először év, utána cím szerint
konyvek.sort(Comparator
    .comparingInt((Konyv k) -> k.ev)
    .thenComparing(k -> k.leiras));
```

A rendezés a listát helyben módosítja. Ha az eredeti sorrend később kell, előtte készíts másolatot.

## Szövegkezelés

| Cél | Java-minta |
|---|---|
| Szövegek összehasonlítása | a.equals(b) \| a.equalsIgnoreCase(b) |
| Részszöveg keresése | sor.contains("kulcsszó") |
| Karakter bejárása | for (int i = 0; i < s.length(); i++) { char c = s.charAt(i); } |
| Ismételt szövegépítés | StringBuilder sb = new StringBuilder(); sb.append(c); |

**Fontos:** a **String** esetén az == két objektum azonosságát vizsgálja; tartalom szerinti összehasonlításhoz equals() kell.

# 6. Mátrixok és kimeneti fájl

## Kétdimenziós tömb bejárása

```
for (int sor = 0; sor < tabla.length; sor++) {
    for (int oszlop = 0; oszlop < tabla[sor].length; oszlop++) {
        int ertek = tabla[sor][oszlop];
        // feldolgozás
    }
}
```

Szomszédos mező vizsgálatakor előbb ellenőrizd a határokat: 0 <= sor < magasság és 0 <= oszlop < szélesség.

## Eredmény fájlba írása

```
try (BufferedWriter ki = Files.newBufferedWriter(
        Paths.get("eredmeny.txt"), StandardCharsets.UTF_8)) {
    for (int i = 0; i < eredmenyek.size(); i++) {
        ki.write(eredmenyek.get(i));
        ki.newLine();
    }
}
```

**Szükséges importok:** java.io.BufferedWriter, java.nio.charset.StandardCharsets, java.nio.file.Files és java.nio.file.Paths. A try-with-resources a blokk végén automatikusan lezárja a fájlt.

## Kimenet a konzolra

```
System.out.println("3. feladat");
System.out.printf("Összesen: %d%n", osszeg);
System.out.printf("Év\tDarab\tPéldányszám%n");
```

A feladat mintáját kövesd: sorszám, megnevezés, sorrend, elválasztók és új sorok. Táblázatos kimenetnél a tabulátor (%\t) segít az oszlopok elválasztásában.

## Az útvonal a program munkamappájához képest értendő

A Paths.get("adat.txt") nem feltétlenül a .java fájl mappáját jelenti. Futtatáskor az IDE vagy a vizsgakörnyezet munkamappájából indul. Ellenőrizd, hogy a bemeneti fájl neve és helye egyezik-e a feladat előírásával.

# 7. Feladattípusok a Java minták mögött

Az oldalon szereplő konkrét vizsgafeladatok jól mutatják, hogyan kombinálhatók az alapminták:

| Feladat | Visszatérő Java-megoldási séma |
|---|---|
| Forgalomszámlálás (2025. október, közép) | Hibás mérés kihagyása; összegzés és óránkénti csoportosítás; maximumhoz időpont tárolása; döntetlennél első előfordulás. |
| Befőzés (2024. október, közép) | Lista bejárása; legnagyobb érték és 1-től számozott hely megadása; összegzett kapacitás összehasonlítása a bekért mennyiséggel. |
| Könyvkiadás (2025. május, emelt) | Pontosvesszős sorok feldarabolása; szövegre keresés; maximum; év és eredet szerinti darabszám- és összegstatisztika; fájlba írt táblázat. |

A bonyolultabb feladatok is ezekből építkeznek. Rácsrajznál és útvonalnál kétdimenziós tömb és állapotfrissítés kell; időbeosztásnál csoportosítás vagy rendezés; szövegfeladatnál karakterenkénti bejárás és StringBuilder.

# 8. Gyakori hibák és gyors ellenőrzés

| Hiba | Ellenőrzés |
|---|---|
| 0-tól induló index és 1-től kért sorszám összekeverése | Kiíráskor kell-e +1? |
| String összehasonlítása == jellel | Tartalmat hasonlítok? Használjam az equals()-t. |
| Egész osztás miatt elvesző tizedesrész | Kell-e double osztó vagy (double) átalakítás? |
| Hibajelölő bekerül az összegbe vagy a maximumba | Minden feladatnál külön kezelem a sentinel értéket? |
| Döntetlen maximum esetén rossz rekord marad | Az első kell? Akkor frissíts csak > esetén, ne >= jellel. |
| Túl kicsi adattípus vagy üres lista | Elég az int? Üres adatsornál biztonságos az induló érték? |
| Eltérő kiírás vagy fájlnév | Összevetettem a kimenetet a mintával karakterről karakterre? |

## Megoldási sorrend

- Azonosítsd a fájlformátumot, az elválasztót és a hibás vagy opcionális adatokat.
- Válassz egyszerű adatszerkezetet; tárold együtt az egy rekordhoz tartozó mezőket.
- Minden részfeladathoz külön algoritmust írj, a közös bejárásokat csak akkor vond össze, ha olvasható marad.
- Próbáld ki a mintával, majd gondolj üres, egy elemű, szélsőértékű és ismétlődő adatra.
- Végül ellenőrizd az első/utolsó indexet, sorszámozást, fájlnevet és kimeneti formátumot.

## A kisokoshoz felhasznált oldalak

- ÉK2.0 Programozás – feladatgyűjtemény megoldásokkal
- Forgalomszámlálás – középszint, 2025. október
- Befőzés – középszint, 2024. október
- Könyvkiadás – emelt szint, 2025. május