# 7. gyakorlat anyaga

## 1. feladat

Készítsük el az `ArrayUtilStructureTest` osztályban leírt szerkezetű programot. Ez egy tömb maximális elemét számítja ki az alábbi módokon. A visszatérési érték mindegyik esetben ugyanaz, a kiszámítás módja azonban eltérő.

- `max()`: a tömb elemeit indexelve járjuk be a tömböt, az eddig érintett rész maximumát számon tartva, és lecserélve, ha az indexelt tömbelem ennél nagyobb
  - Üres tömb maximuma legyen `0`.
  - Nemüres tömbre az érintett rész maximuma az `Integer.MIN_VALUE` értékről induljon.
  - A csere egy egyszerű `if` vizsgálattal történjen.
- `max2()`: hasonlóan, de a csere a `?:` operátorral történjen
- `max3()`: hasonlóan, de a csere a `Math.max()` hívással történjen
- `max4()`: hasonlóan, de a ciklus ne indexelő, hanem bejáró legyen

Írjuk meg az alábbi funkcionális teszteket.

- `maxLength0`: üres tömb maximuma `0`
  - Mind a négy metódust próbáljuk ki.
  - A fentebb említett “Megengedett egyszerűsítés” továbbra is él.
- `maxLength1`: egyelemű tömböt tesztel
  - Paraméterezett tesztelővel próbáljuk ki, ami az egyetlen elem értékét több módon állítja be.
  - Az értékek között szerepeljen `0`, `1`, nagy értékek, negatív értékek.
- `maxLength2`: kételemű tömböt tesztel
  - Paraméterezett tesztelővel próbáljuk ki, ami egy `min` és egy `max` értéket használ.
  - Mind a négy metódust próbáljuk ki úgy is, hogy az értékek `min, max` sorrendben szerepelnek a tömbben, és úgy is, hogy a sorrendjük `max, min`.

A `minMax()` metódus a tömb minimumát és maximumát egy kételemű tömbben adja ki.

- A megvalósítás a maximum/minimum meghatározására a fenti lehetőségek közül bármelyiket használhatja.
- Tipp: szükség lesz az `Integer.MAX_VALUE` értékre.
- Készüljenek ehhez funkcionális tesztek: `minMaxLengthN`, ahol `N=0, 1, 2`.

A `array.util.main.Main.main()` metódus próbálja ki a kódot úgy, hogy a felhasználó kézzel adja meg az adatokat.

- Először az `arrayLenTxt` változóba olvassuk be a tömb hosszát, és alakítsuk számmá az `arrayLen` változóba.
- Ezután készítsünk egy `arrayLen` méretű tömböt.
- Utána olvassunk be `arrayLen` bemenetet, és tároljuk el számként a tömb elemeiben.
- Végül írjuk ki a tömböt magát és a fenti metódusok segítségével meghatározott minimumát/maximumát.
  - A tömbök `System.out.println(array)` módon kiírva nem mutatják meg az elemeiket…
  - … ezért `System.out.println(Arrays.toString(array))` használatával lehet őket kiíratni, ahol az osztály teljes neve `java.util.Arrays`.
  - A `minMax()` adta tömböt is így írjuk ki.

## 2. feladat

Készítsük el a `SectorTimerTestSuite` osztályban leírt szerkezetű programokat. Ezek egy versenyautó köreinek időmérését próbálják megvalósítani, az utolsó kivételével szándékosan rosszul.

Először a `race.car.WrongSectorTimer1` osztályt készítsük el. *Szándékosan rontsuk el úgy*, hogy az adattagjai `public` láthatóságúak legyenek. Próbáljuk ki a következőket.

- A következő tesztesetek mind úgy indulnak, hogy egy `1, 2, 3` tartalmú tömbbel példányosítani kell az osztályt.
  - Kísérletező kedvűeknek: `@BeforeEach` használatával még elegánsabb megoldás adható.
- `seemsGood`: az adattag három elemű tömb a megfelelő értékekkel
- `setArrayElemsBreaksEncapsulation`: a tömb mező egyes értékeit közvetlenül el lehet érni és be lehet állítani; ez a hosszát nem változtatja meg
  - Ez a teszt (és a következő) szokatlan, mert egy **direkt rossz** megvalósítást próbál ki.
  - Éppen ezért akkor zöld (elfogadott), ha **rosszul működik az osztály**.
- `setArrayElemsBreaksEncapsulation`: az adattagban a tömb teljesen lecserélhető, ez megváltoztatja a hosszát és az értékeit is

Próbáljuk megjavítani: készítsük el a `race.car.WrongSectorTimer2` osztályt. Az adattag láthatósága legyen most `private`, de a getterek, setterek és a konstruktor megvalósítása legyen naívan egyszerű.

- Most már nem elérhető kívülről az adattag. Ez helyes.
- `seemsGood`: mint korábban
- `constructorBreaksEncapsulation`: a `WrongSectorTimer2` példányt egy lokális változóban levő, `1, 2, 3` tartalmú tömb inicializálja, majd írjuk a tömb elemeit
  - Ez a `timer.getSectorTimes()` eredményének hosszát nem változtatja meg.
  - Ellenőrzendő: a `timer.getSectorTimes()` és `timer.getSectorTime()` eredményeként viszont így más eredmény adódik.
- `getterBreaksEncapsulation`: írjuk a `timer.getSectorTimes()[n]` értékeket
  - Ez a `timer.getSectorTimes()` eredményének hosszát nem változtatja meg, de az elemeket igen.
- `setterBreaksEncapsulation`: a setternek egy `4, 5, 6, 7` tartalmú, lokális változóban tárolt tömb átadása, majd a változón keresztül a tömbelemek módosítása
  - Ez a reprezentáció tömb hosszát és értékeit is módosítja.

Most tényleg javítsuk meg az enkapszulációt: `race.car.SectorTimer` osztály.

- A konstruktor, getter, setter egyaránt készítsen másolatot annak során, hogy átveszi/kiadja a reprezentáció tömbjét.
  - Még jobb, ha a konstruktor és a setter egy közös, privát `initSectorTimes` segédfüggvényre hív át.
- A tesztelőben legyen `constructorEncapsulatesWell`, `getterEncapsulatesWell` és `setterEncapsulatesWell` metódus.
  - Ezek ugyanazt próbálják ki, mint a fentiek, és ugyanazokat az `assertX` vizsgálatokat hajtsák végre.
  - Figyeljük meg, hogy az enkapszuláció most jól működik: nem tudunk kívülről ,,betörni'' a reprezentációba.
- A konstruktornak/getternek/setternek legyen olyan változata is, ami a tömb másolását a `System.arraycopy` művelettel hajtja végre.
  - Célszerű aSectorelv: ne találjuk fel a spanyolviaszt, ha elérhető beépített/könyvtári/sztenderd megoldás/eszköz.
  - Mivel egy osztályban nem lehet két, azonos paraméterezésű konstruktor, az egyik kapjon egy extra `boolean` paramétert, amit nem használunk benne.
  - A konstruktor és a setter *vararg* stílusban kapja meg a paramétereit, ne csak egyszerű tömbként.

# 3. feladat

Készítsük el az `ElectionTestSuite` által leírt osztály és felsorolási típust. Egy olyan szavazást írnak le, amelyben jelöltekre lehet szavazatokat leadni.

Az `ElectionTestStructureTest` osztály az elkészítendő tesztek szerkezetét mutatja.

- `noVotes`: ha nincsen egy szavazat sem, az első jelölt győz (ez most `JACK`)
- `singleCandidate`: a megadott számú szavazatot adják le, mindegyiket ugyanarra a jelöltre, aki ezáltal győz
  - Felsorolási típus átadható a tesztelőnek: a metódus paraméterének legyen `Candidate` a típusa, és a CSV részre az enum-érték neve kerüljön.
- `twoCandidatesSameVoteCount`: ketten ugyanannyi szavazatot kapnak, aki az enumban a korábbi elem, az győz
  - Kipróbálandó: bármelyik sorrendben is kapják a szavazatokat, ugyanez az eredmény.
- `candidateVoteCount`: a jelöltek sorban 1, 4, 3 és 2 szavazatot kapnak
  - Kipróbálandó: `getCandidatesWithMoreVotesThan` a helyes eredményt adja a 0, 1, 2, 3 és 4 paraméterre is.
  - Mivel egy `@CsvSource` nem tud nyilvánvaló módon tömböt kapni paraméterként, a hívásokat és az elvárt értékeket sorrendben bele kell írni a metódusba.
    - Nehezebb változat: az elvárt értékeket egy szövegben (így: `'JILL,SAM,MAX'`) kell átadni, és ezekből egy rövid kódrészlet alakítja ki az elvárandó `Candidate` tömböt.
  - Itt nem `assertEquals`, hanem `assertArrayEquals` vizsgálat szükséges.

## 4. feladat

Rajzoljon memóriatérképet (memory map) a következő Java program kommentben jelzett soraihoz. (Másképp: Rajzolja fel a stack és heap pillanatnyi állapotát következő Java program végrehajtása során). A konstruktor paramétereitől tekintsünk el.

Az (5) végrehajtása után mely objektumokat törölheti a szemétgyűjtő?

Main.java:

```
class Foo {
    private int x;

    public Foo(int initX) {
        x = initX;
    }
}

public class Main {
    public static void main(String[] args) {
        int counter = 10;       // (1)

        Foo obj;                // (2)
        obj = new Foo(5);       // (3)

        Foo obj2 = new Foo(7);  // (4)
        obj2 = obj;             // (5)

        // ...
    }
}
```

Írja át `Foo` konstruktorát hogy `initX` helyett `x` legyen a paraméter, tegye egyértelművé a `this` kulcsszóval, hogy melyik `x` azonosítóra hivatkozik.

# Gyakorló feladatok

## 1. gyakorló feladat

Rajzoljon memóriatérképet (memory map) a következő Java programhoz. (Másképp: Rajzolja fel a stack és heap pillanatnyi állapotát következő Java program végrehajtása során).

```
class Foo {
    private int x;
```

```
    public Foo(int x) {
        this.x = x;
    }
}

public class Main {
    public static void main(String[] args) {
        Foo obj = new Foo(0);
        obj = new Foo(10);
        Foo obj2 = obj;
        int i = 1;
        obj2 = new Foo(20);
    }
}
```