Programozási nyelvek II.: JAVA, 3. gyakorlat

2017. szeptember 25 - 29.

A 3. gyakorlat tematikája

- Osztályok definíciója
- Láthatósági módosítók
- Getterek és setterek
- A `this` pszeudováltozó
- Konstruktorok, túlterhelés

Osztályok definíciója

- programok strukturálására
  - implementációs részletek elrejtése
  - összetartozó fogalmak egységbe zárása (encapsulation)
  - saját típus létrehozása + típuson értelmezett műveletek
  - osztálydefiníció = adattagok + műveletek (metódusok) + egyéb osztályok
- `static` módosító: osztályszintű (adattag / metódus)

Pontok osztálya (Point.java)

Készítsük el a `Point` osztályt (Point.java)!

- Tulajdonságok: x és y koordináta (mindkettő double típusú)
- Művelet: eltolás: dx és dy értékekkel eltolja a pontot (mindkettő double típusú)

Pontok osztálya (Point.java)

```
class Point {

    double x;
    double y;

    void move(double dx, double dy) {
        x += dx;
        y += dy;
    }
}
```

Körök osztálya (Circle.java)

Készítsük el a kört reprezentáló `Circle` osztályt (Circle.java)!

- Tulajdonságok: középpont (`Point`) és sugár (double)
- Műveletek: eltolás és nagyítás
  - eltolás: dx és dy értékekkel eltolja a középpontot (mindkettő double típusú)
  - nagyítás: egy adott számmal szorozza a sugarat (`factor` double típusú)

Körök osztálya (Circle.java)

```
class Circle {

    Point center;
    double radius;

    void move(double dx, double dy) {
        center.move(dx, dy);
    }

    void enlarge(double factor) {
        radius *= factor;
    }
}
```

Láthatóság

Motiváció:

- az objektum állapotát kívülről ne tudjuk tetszőleges módon megváltoztatni
- az osztályokban a metódusok viselkedése a felhasználó számára mindig lényegében egy fekete doboz

Láthatósági módosítók

Minden adattagra és minden metódusra pontosan egy láthatósági (hozzáférési) kategória vonatkozhat, ezért az alább megadott módosítószavak közül pontosan egyet lehet használni minden egyes tag és metódus esetében. A láthatósági kategóriák (módosítószavak) a következők:

- Félnyilvános (package-private): ha nem írunk semmit
  - azonos csomagban definiált osztályok (objektumai) érik csak el
- Nyilvános: `public`
  - más csomagokban definiált osztályok is elérik
  - pl. a főprogramnak is ilyennek kell lennie, hogy futtatható legyen: `public static void main(String[] args)`
- Privát: `private`
  - csak az osztálydefiníción belül érhető el
  - az osztály minden objektuma hozzáfér
- Védett: `protected`
  - a félnyilvános kategória kiterjesztése: azonos csomagban lévő, plusz a leszármazottak (ld. később)

Láthatósági reláció

`public` $\supseteq$ `protected` $\supseteq$ package-private $\supseteq$ `private`

Módosító	osztály	csomag	leszármazott	mindenki
public	igen	igen	igen	igen
protected	igen	igen	igen	nem
nincs (package-private)	igen	igen	nem	nem
private	igen	nem	nem	nem

Getterek és setterek

- Adattagok `private`–tá tétele:
  - ilyenkor az adattaghoz készíthető egy lekérdező ("getter") metódus, amellyel az adattag értékét le tudjuk kérdezni,
  - a lekérdező metódus már bárki által hívható lesz.
- A referenciák esetében: a lekérdező metódus ne közvetlenül magát a referenciát adja vissza, hanem mindig másolja le az általa hivatkozott objektumot! (kivéve, ha az objektum `immutable`, mint pl. a `String` osztály objektumai)
- Adattag kívülről írhatóvá tétele: beállító ("setter") metódust készítünk hozzá. Ennek a metódusnak a feladata, hogy - szükség esetén - ellenőrizze, hogy az új érték megfelelő-e.
- A referenciák esetében a beállító metódus is másolja le a hivatkozott objektumot! (kivéve, ha az objektum `immutable`)

A this pszeudováltozó

- Predefinit név.
- Az osztálydefiníción belül a példánymetódusokban `this` névvel hivatkozhatunk az aktuális objektumra.
- A `static` metódusokban a `this` nem használható.
- A `this.xxx`–hez általában nem kell a minősítés, néha azonban szükség lehet rá (pl. ha a metódus paramétere ugyanolyan nevű, mint az adattag)
- A `this` paraméterként is átadható.

Pontok és Körök osztálya (Point.java, Circle.java)

Készítsünk lekérdező és beállító műveleteket (metódusokat), amelyekkel le tudjuk kérdezni / be tudjuk állítani a pontok és a körök attribútumait (egy pont x és y koordinátáját, valamint egy kör középpontját és sugarát)! A metódusok legyenek publikusak! (Az adattagok nem publikusak!)
Használjuk a `this` pszeudováltozót!

Pontok osztálya (Point.java)

```
public class Point {

    private double x;
    private double y;

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }
    ...
```

Pontok osztálya (Point.java) (folyt.)

```
    ...
    public void setX(double x) {
        this.x = x;
    }

    public void setY(double y) {
        this.y = y;
    }

    public void move(double dx, double dy) {
        x += dx;
        y += dy;
    }

}
```

Körök osztálya (Circle.java)

```
public class Circle {

    private Point center;
    private double radius;

    public Point getCenter() {
        Point result = new Point();
        result.setX(center.getX());
        result.setY(center.getY());
        return result;
    }

    public void setCenter(Point center) {
        this.center = new Point();
        this.center.setX(center.getX());
        this.center.setY(center.getY());
    }
    ...
```

Körök osztálya (Circle.java)(folyt.)

```
    ...
    public double getRadius() {
        return radius;
    }

    public void setRadius(double r) {
        if (r < 0.0) r = 0.0;
        radius = r;
    }

    public void move(double dx, double dy) {
        center.move(dx, dy);
    }

    public void enlarge(double factor) {
        radius *= factor;
    }
}
```

Főprogram (PointCircleTest.java)

Készítsünk tesztprogramot a pontok és körök osztályához! Hozzunk létre egy `Point` és egy `Circle` objektumot! Kérdezzük le és állítsuk be a pont koordinátáit! Toljuk el a pontot megadott értékkel! Legyen a pont a kör középpontja, állítsuk be a sugarát! Toljuk el, nagyítsuk a kört! Kérdezzük le a középpontját és a sugarát!

Főprogram (PointCircleTest.java)

```
public class PointCircleTest{

  public static void main(String[] args) {

    Point p = new Point();
    System.out.println(p.getX());
    System.out.println(p.getY());

    p.setX(1.23);
    p.setY(2.56);
    System.out.println(p.getX());
    System.out.println(p.getY());

    p.move(1.22,1.33);
    System.out.println(p.getX());
    System.out.println(p.getY());
    ...
```

Főprogram (PointCircleTest.java)(folyt.)

```
    ...
    Circle c = new Circle();

    c.setCenter(p);
    c.setRadius(3.21);
    System.out.println(c.getRadius());

    c.move(1.22,1.33);
    System.out.println(c.getCenter().getX());
    System.out.println(c.getCenter().getY());

    c.enlarge(2);
    System.out.println(c.getCenter().getX());
    System.out.println(c.getCenter().getY());
    System.out.println(c.getRadius());

  }
}
```

Konstruktorok

- Programkód, ami a példányosításkor "automatikusan" végrehajtódik.
- A konstruktor neve megegyezik az osztály nevével.
- A konstruktor paramétereket vehet át.
- Több (különböző szignatúrájú) konstruktor is lehet egy osztályban (túlterhelés, ld. később).
- Csak példányosításkor hajtódhat végre (`new` mellett).
- A konstruktor mindig eljárás, de konstruktor esetében szintaxishiba kiírni a `void`-ot!

Konstruktorok

- Módosítók közül csak a láthatósági kategóriát leírók használhatók (egyéb módosítók, pl. a `static` vagy a `final`, nem).
- A törzs olyan, mint egy void visszatérési értékű metódusé, a paraméter nélküli `return`–t használhatjuk.
- Bevett szokás, hogy ugyanazokat a neveket használhatjuk konstruktor formális paraméternek, mint a példányváltozóknak (ekkor az adattagot a `this`-en keresztül érjük el).
- Egy konstruktor meghívhat egy másik konstruktort is, `this` névvel: az effajta konstruktormeghívás csak az első utasítás lehet!

Konstruktorok

- A láthatósági kategóriák vonatkoznak a konstruktorokra is.
- A konstruktor használata: `new Osztálynév` után paraméterek megadása zárójelben.
- Aktuális argumentumok a konstruktornak: ezek döntik el, hogy melyik konstruktor hívódik meg.
- Ha nem írtunk konstruktort, akkor automatikusan (és csak ekkor) létrejön egy ilyen, a fordítóprogram beilleszt a kódba egy ún. default konstruktort, amely paraméter nélküli és üres törzsű. Pl. mintha a `Pointba` bekerülne egy ilyen:

```
public Point(){}
```

- Minden default konstruktor paraméter nélküli, de nem minden paraméter nélküli konstruktor default.

Túlterhelés (overloading)

- Az osztályok metódusainak és konstruktorainak lehet többféle paraméterezése.
- Ilyenkor egyetlen névhez társítunk többféle viselkedést (több metódust nevezünk el ugyanazzal a névvel).
- A megfelelő változatot a paraméterlista alapján választjuk ki.
- A metódusokat a visszatérési értékre nem lehet túlterhelni.

Konstruktorok készítése, túlterhelés

A `Point` és `Circle` osztályokhoz készítsünk konstruktorokat! A `Point` osztályhoz csak egyet, amelynek a koordinátákat lehet átadni. A `Circle` osztályhoz hármat:

- amelynek a sugár mellett egy `Point` objektumot,
- illetve a középpont koordinátáit lehet átadni,
- valamint egy paraméter nélküli konstruktort.

Pontok osztálya (Point.java)

```
public class Point {

    private double x;
    private double y;

    public Point(double x, double y) {
        this.x=x;
        this.y=y;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }
    ...
```

Pontok osztálya (Point.java) (folyt.)

```
    ...
    public void setX(double x) {
        this.x = x;
    }

    public void setY(double y) {
        this.y = y;
    }

    public void move(double dx, double dy) {
        x += dx;
        y += dy;
    }

}
```

Körök osztálya (Circle.java)

```
public class Circle {

    private Point center;
    private double radius;

    public Circle(Point p, double radius) {
        center = new Point(p.getX(),p.getY());
        this.radius = radius;
    }

    public Circle(double x, double y) {
        center = new Point(x,y);
    }
    ...
```

Körök osztálya (Circle.java)(folyt.)

```
...
public Circle() {
    //a double-double parameteru konstruktor hivasa
    this(0.0,0.0);
}

public Point getCenter() {
    return new Point(center.getX(),center.getY());
}

public void setCenter(Point center) {
    this.center = new Point(center.getX(),center.getY());
}
```

Körök osztálya (Circle.java)(folyt.)

```
    ...
    public double getRadius() {
        return radius;
    }

    public void setRadius(double r) {
        if (r < 0.0) r = 0.0;
        radius = r;
    }

    public void move(double dx, double dy) {
        center.move(dx, dy);
    }

    public void enlarge(double factor) {
        radius *= factor;
    }
}
```

Főprogram (PointCircleTest.java)

```
public class PointCircleTest{

  public static void main(String[] args) {

    Point p = new Point(1.22,1.33);

    System.out.println(p.getX());
    System.out.println(p.getY());
    p.setX(1.23);
    p.setY(2.56);
    System.out.println(p.getX());
    System.out.println(p.getY());
    p.move(1.22,1.33);
    System.out.println(p.getX());
    System.out.println(p.getY());
    ...
```

Főprogram (PointCircleTest.java) (folyt.)

```
...
Circle c = new Circle();

c.setCenter(p);
c.setRadius(3.21);
System.out.println(c.getRadius());
c.move(1.22,1.33);
System.out.println(c.getCenter().getX());
System.out.println(c.getCenter().getY());

c.enlarge(2);

System.out.println(c.getCenter().getX());
System.out.println(c.getCenter().getY());
System.out.println(c.getRadius());
...
```

Főprogram (PointCircleTest.java) (folyt.)

```
...
Point q = new Point(1.1, 2.2);

System.out.println(q.getX());
System.out.println(q.getY());

Circle a = new Circle(q, 3.4);

System.out.println(a.getCenter().getX());
System.out.println(a.getCenter().getY());
System.out.println(a.getRadius());
...
```

Főprogram (PointCircleTest.java) (folyt.)

```
    ...
    Circle b = new Circle(1.3,4.5);

    System.out.println(b.getCenter().getX());
    System.out.println(b.getCenter().getY());
    System.out.println(b.getRadius());

    Circle d = new Circle();

    System.out.println(d.getCenter().getX());
    System.out.println(d.getCenter().getY());
    System.out.println(d.getRadius());
    ...
  }
}
```

Objektumok szöveges reprezentációja (Point.java)

Készítsünk a `Point` osztályba egy szöveges visszatérési értékű, paraméter nélküli, `toString` nevű publikus metódust, amely visszaadja egy pont szöveges reprezentációját.

```
...
public String toString() {
    return "(" + x + "," + y + ")";
}
...
```

Tömbök használata

- tömb deklarálása: `típus[] tömb;`
  - a `típus` lehet primitív típus vagy osztály, de akár tömb is
- tömb létrehozása: `tömb = new típus[méret];`
  - a tömb elemei "null-szerű" értékkel töltődnek fel
- tömb létrehozása és kezdőértékkel történő beállítása:
  `tömb =` $\{elem_1, elem_2, \ldots, elem_n\}$;
- tömb elemeinek száma: `tömb.length`
- `i`. elem elérése: `tömb[i]`
- tömb bejárása:
  - léptető ciklussal:
    `for( int i = 0; i < tömb.length; ++i ) {`
    ahol `i` sorban felveszi a `tömb` indexeit
  - iteráló ciklussal:
    `for( típus t : tömb ) {`
    ahol `t` sorban felveszi a `tömb` elemeit

Tömegközéppont meghatározása (PointCircleTest.java)

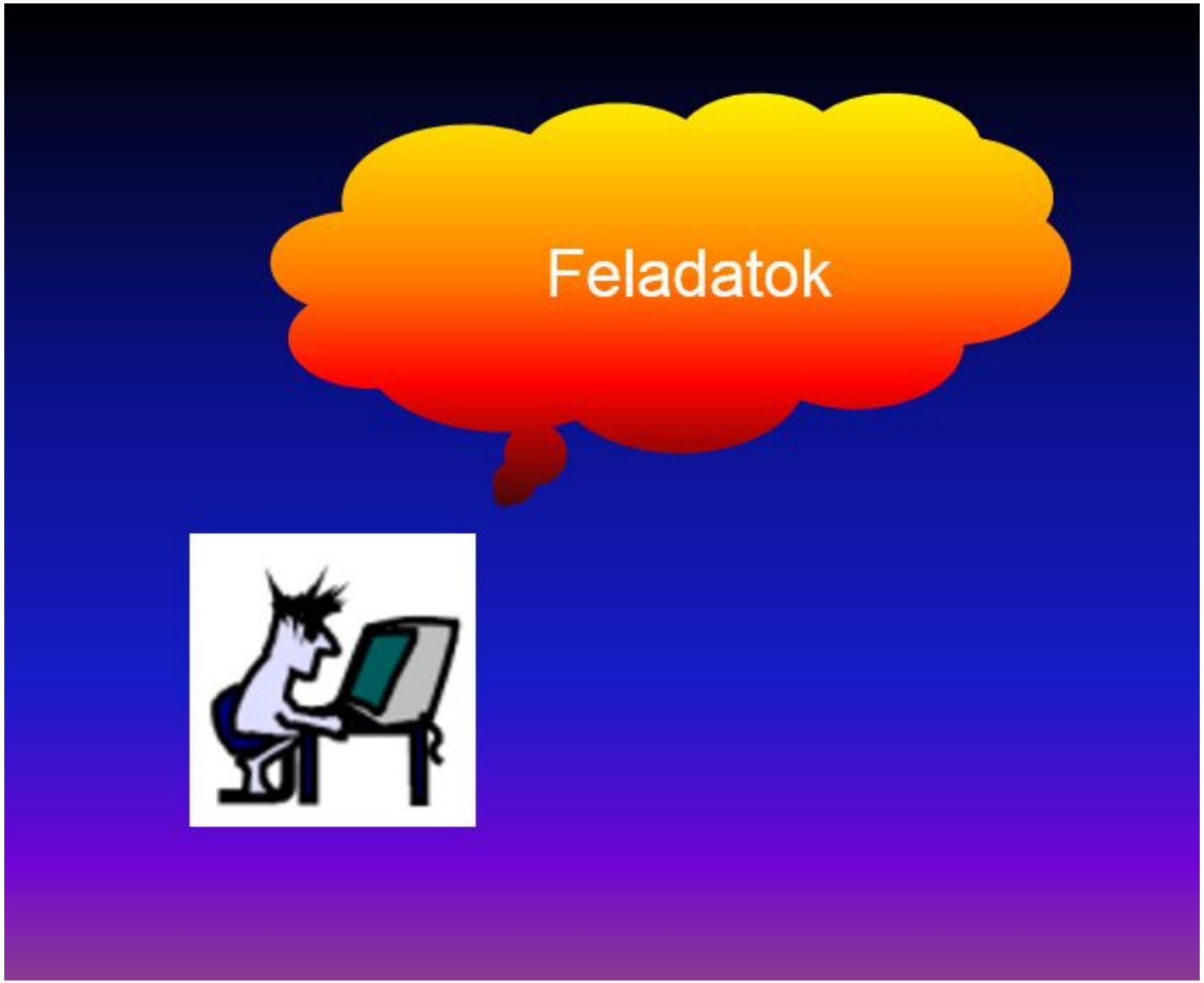
- Készítsünk egy `centerOfMass` nevű statikus metódust a `PointCircleTest` osztályba. A metódus paraméterül egy pontokból álló tömböt kap. Feladata pedig, hogy kiszámítsa a kapott pontok tömegközéppontját (ami szintén egy pont), majd az eredményt visszaadja.
- Készítsünk egy `centerOfMassTest` nevű statikus metódust a `PointCircleTest` osztályba. A metódus hozzon létre két pontokból álló tömböt és (az előző metódus segítségével) mindkettőnek számítsa ki a középpontját, majd az eredményt írja ki.
- A `PointCircleTest` osztály `main` metódusából hívjuk meg a `centerOfMassTest` metódust.

Tömegközéppont meghatározása (PointCircleTest.java)

```
...
public static Point centerOfMass(Point[] points) {
    double cx = 0.0;
    double cy = 0.0;

    for(Point p : points) {
        cx += p.getX();
        cy += p.getY();
    }

    cx /= points.length;
    cy /= points.length;

    return new Point(cx, cy);
}
...
```

Tömegközéppont meghatározása (PointCircleTest.java)

```
...
public static void centerOfMassTest(){
    Point[] points1 = new Point[3];
    points1[0] = new Point(2.0,4.0);
    points1[1] = new Point(3.0,6.0);
    points1[2] = new Point(1.0,2.0);

    Point[] points2 =
        { new Point(1.0,2.0)
        , new Point(2.0,4.0)
        , new Point(3.0,6.0)
        };

    System.out.println("centerOfMass(points1)=="
        + centerOfMass(points1));
    System.out.println("centerOfMass(points2)=="
        + centerOfMass(points2));
}
...
```


Feladatok

Hibajavítás (Kangaroo.java, KangarooDemo.java)

- A `Kangaroo` osztály egy kengurut reprezentál.
- Az osztálynak két adattagja van, az egyik egy szöveges típusú, a kenguru nevének, a másik egész típusú és az életkorának az eltárolására szolgál.
- Az osztálynak két konstruktora van. Az első egy szöveges típusú nevet és egy egész típusú életkort kap paraméterként és beállítja a megfelelő adattagokat. A második konstruktor egy egész típusú értéket kap és kiírja a kenguru lábainak számát.
- Az osztály rendelkezik egy `display` nevű metódussal is, egy szöveges típusú országnevet kap paraméterül, és kiírja a kenguru nevét, lakóhelyét (az országot), majd eggyel megnöveli az életkorát és az új életkort is kiírja.

Pontok és körök osztálya (Point.java, Circle.java, PointCircleTest.java)

Egészítsük ki a `Point` és a `Circle` osztályát az alábbi műveletekkel:

- a `Point` osztályban definiáljunk egy metódust, amely megadja az objektum távolságát egy, a paramétereként átadott `Point` objektumtól;
- a `Point` osztályban definiáljunk egy statikus metódust, amely két pont távolságát adja meg;
- a `Circle` osztályban is készítsük el a `toString` metódust;
- a kör területének kiszámítása;

Pontok és Körök osztálya (Point.java, Circle.java, PointCircleTest.java)

- a `Circle` osztályhoz írjunk egy műveletet, amely eldönti, hogy a paramétereként megadott `Point` objektum illeszkedik–e a körvonalra – egy adott tűréshatáron belül;
- a `Circle` osztályhoz írjunk egy műveletet, amely eldönti, hogy a paramétereként megadott `Point` objektum a `Circle` belsejében van–e.

Készítsünk főprogramot is! Oldjuk meg a feladatot csomagok nélkül és csomagokkal is (rakjuk a `Circle` osztályt a `geo`, a `Point` osztályt a `utils.basics`, a főprogramot pedig a `main` csomagba)!

Bináris fa osztály (IntTree.java, IntTreeTest.java)

Készítsünk egy `utils.IntTree` osztályt! A `utils.IntTree` osztály tulajdonképpen egész számokat rendezetten tároló bináris fa osztály. A bináris fa értékek olyan láncolata, ahol minden értéknek nulla, egy vagy két rákövetkezője lehet, és csak nulla vagy egy megelőzője. A láncolást az `IntTree` objektumokra vonatkozó referenciákkal oldjuk meg, ezek az adott fabeli csomópont bal és jobb oldali részfáira fognak mutatni. A csomópontokban ezenkívül még egy `int` értéket is eltárolunk. Ha nincs megelőző, akkor a fa gyökeréről beszélünk.

Bináris fa osztály (IntTree.java, IntTreeTest.java)

Az `IntTree` osztálynak a következő műveletei vannak:

- `insert()`: egy int értéket kap és beilleszti a fába. Ha a beszúrandó elem kisebb, mint a fa gyökerében található elem, akkor a bal részfába (rekurzió) illeszti be. Ha nincs még bal részfánk, akkor ezzel az elemmel hozzuk létre a gyökerében. Ha a beszúrandó elem nagyobb, vagy egyenlő, mint a gyökérben levő, akkor ugyanezt végezzük el a jobb részfára!
- `contains()`: eldönti, hogy a paramétereként megadott elem megtalálható–e a fában? Igaz értéket ad vissza, ha megtalálható az elem, különben hamisat. (rekurzív metódus)
- `konstruktor`: az `int` értékek egy tömbjét beszúrja a fába
- `konstruktor`: egy elemből egy olyan fát hoz létre, amelyben csak egyetlen gyökérelem van (és nincsenek rákövetkezői).

Bináris fa osztály (IntTree.java, IntTreeTest.java)

- `toArray()`: egy `int` értékeket tartalmazó tömbben visszaadja azokat az értékeket, amelyeket a fában tárolunk. Ha szeretnénk kihasználni, hogy a beszúrást rendezetten végeztük, akkor érdemes ezt a tömböt úgy felépíteni, hogy először a bal részfa elemeit vesszük, aztán a gyökérben levő elemet, majd a jobb részfa elemeit. Az így összeállított tömb is rendezett lesz (inorder bejárás). (rekurzív metódus)
- `toString()`: egy fa szöveges alakját adja meg. (csak a tárolt elemeket jelenítjük meg egy felsorolásban).
- `equalsTo()`: eldönti az objektumról és a paraméterként megadott változóról, hogy a kettő megegyezik–e. Két bináris fát akkor tekintünk egyenlőnek, ha ugyanazokat az elemeket tartalmazzák.

Készítsünk főprogramot is, amely teszteli ezen műveleteket!

Egész értékek dinamikus listája (`IntList.java`, `IntListTest.java`)

Készítsünk egy `utils.IntList` osztályt (egész értékek tömbösített, dinamikus tárolására alkalmas osztály).
Készítsünk az osztályhoz két konstruktort:

- egy olyat, amelynek nincs paramétere (a tömb nem tartalmaz semmit),
- egy másikat, amelyben megadjuk egy egész értékeket tároló tömbben, hogy mit tartalmazzon kezdetben!

Egész értékek dinamikus listája (`IntList.java`, `IntListTest.java`)

Az osztály műveletei:

- `add()`: egy új elem hozzáadása a tömb végére,
- `add()`: elem beszúrása az adott indexű helyre,
- `concat()`: egy másik IntList tartalmának hozzáfűzése az aktuálishoz,
- `get()`: az első elem lekérdezése,
- `get()`: adott indexű elem lekérdezése,
- `set()`: adott indexű elem beállítása, amennyiben van olyan indexű elem,
- `remove()`: az első elem törlése,
- `remove()`: adott indexű elem törlése,

Egész értékek dinamikus listája (`IntList.java`, `IntListTest.java`)

- `indexOf()`: elem indexének megkeresése, ha nem található, akkor -1–et adjon vissza,
- `size()`: a jelenleg tárolt elemek száma,
- `clear()`: az összes elem törlése,
- `toArray()`: az osztályban tárolt értékek egyetlen tömbként történő visszaadása,
- `toString()`: szöveggé alakítás.

Egész értékek dinamikus listája (`IntList.java`, `IntListTest.java`)

Továbbá legyen egy `fromString()` nevű osztályszintű metódusa, amellyel szövegből tudjuk beolvasni a tárolni kívánt elemeket. Az értékeket ebben a beolvasni kívánt szöveges reprezentációban egyszerűen szóközökkel választjuk el. Ellenőrizzük azt is, hogy a szövegben valóban számok szerepelnek! Ha ez nem teljesül és nem tudjuk beolvasni az összes elemet, akkor adjunk vissza null referenciát! A beolvasás során használjuk a `java.util.Scanner` osztályt!