Tartalomjegyzék

Feladatok tagolása

Már több osztállyal dolgozunk akár egy feladaton belül is. Így egy új tagolást javasolok.

- Eclipse-ben a már létezõ vagy új projectet nyissátok le, és az **src** mappára jobb klikk **New** -> **Package**
- Nevezzétek el és kész is
- Minden összetartozó osztályt egy ilyen package-be rakjatok
- Amikor egy osztály egy **Gyak2** package-ben van azt jelezni kell a fájl elején a következõ módon:

```
package Gyak2;
```

- Package-eket lehet egymásba is ágyazni, késõbb a komolyabb dolgokat majd külön projectben sok package-ben fogjuk tárolni

Bemelegítõ feladatok

Figyeljetek oda, hogy minden osztálynak a saját nevével megyegyezõ nevû fájlban kell lennie. Pl a **Complex** osztálynak a **Complex.java** fájlban kell lennie.

Egészítsétek ki a feladatokat a **//TODO** részeknél. Ez van ahol csak egy parancs, máshol több sor is lehet akár.

1. Complex kiegészítés

```
public class Complex {
    private float realPart_;
```

```
    private float imaginaryPart_;

    public Complex() {
        realPart_ = 0;
        imaginaryPart_ = 0;
    }

    public Complex(float rePart) {
        realPart_ = //TODO
        imaginaryPart_ = //TODO
    }

    public Complex(float rePart, float imPart) {
        //TODO
    }

    public Complex add(Complex other) {
        float rePart = this.realPart_ + other.realPart_;
        float imPart = this.imaginaryPart_ + other.imaginaryPart_;
        Complex retval = //TODO
        return retval;
    }

    public Complex multiply(Complex other) {
              //TODO
    }
}
```

2. Position

A Position reprezentáljon a síkon egy pontot. Ezzel fogjuk az aszteroidák és a játékos ûrhajójának helyét tárolni.

- A **default konstruktor** rakja a (0, 0)-ba a pontot.
- A **Position(float xn, float yn)** konstruktor létrehoz egy Position-t a (xn, yn) ponton.
- A **distance** metódus visszaadja a pont (euklideszi) távolságát egy másik ponttól.
- A **translate** metódus az (xt, yt) vektorral eltolja a Position-t.

```
public class Position {
private float x_;
private float y_;

public Position() {
//TODO
}

public Position(Position other) {
this.x_ = other.x_;
this.y_ = other.y_;
}

public Position(float xn, float yn) {
//TODO
}

public float distance(Position other) {
//TODO
//Math.sqrt-vel lehet gyököt számolni
//viszont double-t ad vissza, ezt így lehet float-á castolni:
//(float)valtozónév
}
```

```
public void translate(float xt, float yt) {
//TODO
}
}
```

Feladatok

Érdemes minden osztályhoz gyártani egy **main**t, hogy kipróbáljátok jól mûködik-e. Ha lesz rá idõnk tanulunk teszt rendszert is, de nem biztos, hogy eljutunk odáig. Ezt lehet az adott osztályba írni, nem kell mindig külön **Main** osztályt gyártani miatta. Itt egy példa a **Complex**re:

```
public static void main(String[] args) {
        Complex comp1 = new Complex(5, 6);
        Complex comp2 = new Complex(4);
        Complex comp3 = comp1.add(comp2);
        System.out.println(comp3);
    }
```

Azt nem ellenõrzi, hogy jól számolt-e, ezt ugye már nehezebb, mivel privátak az adattagjai, de írhatnánk egy kiirató függvényt, akkor már könnyû lenne.

Továbbá mindig emlékezzetek arra, hogy a cél, hogy minél kisebb egységekre szedjük szét a dolgokat. Így ha pl egy függvény mûködéséhez kell abszolút érték függvény, írd meg külön és használd, ne az adott függvénybe írd bele az abszolút érték megvalósítását. (Bár ez elérhetõ a Math.abs-ként is.)

```
public static void main(String[] args) {
           = newComplex(5, 1);
           = newComplex(2, 1);

            = c1.add(c2);sum
             = c1.multiply(c2);

System.out.println(mult.realPart_);
System.out.println(mult.imaginaryPart_);
}

public static void main(String[] args) {
Position p1 = new Position();
Position p2 = new Position(4f, 6f);
   translate(1f,p2.1f);
float dist = p1.distance(p2);
System.out.println(dist);
}
```

3. ComplexVector szorzás

Írjátok meg a **ComplexVector** skaláris szorzatát. Ne felejtsétek el, hogy használhatjátok a **Complex** osztály **multiply** függvényét.

A **clone** metódus lemásolja az adott objektumot, ebben az esetben a tömböt.

```
public class ComplexVector {
    private Complex[] coords_;
    private int dimension_;

    public ComplexVector(int dim) {
        dimension_ = dim;
```

```
        coords_ = new Complex[dimension_];
    }

    public ComplexVector(ComplexVector other) {
        this.coords_ = other.coords_.clone();
        this.dimension_ = other.dimension_;
    }

    public ComplexVector add(ComplexVector other) {
        ComplexVector retval = new ComplexVector(this.dimension_);
        for (int i = 0; i < retval.coords_.length; i++) {
            retval.coords_[i] = this.coords_[i].add(other.coords_[i]);
        }
        return retval;
    }
}
```

4. Kiírás

Írjatok kiíró metódust **print** néven a **Complex** és **ComplexVector** osztályokhoz. Nézzen ki valahogy olvashatóan. Majd ezekkel már jól tudtok tesztelni a **main**ekben.

4.5. Szerkezet változtatás

Írjátok át a **Complex** osztályt, hogy ne két **float**ban tárolja az adatokat, hanem egy 2 elemû **float** tömbben. Majd értelemszerûen a konstruktorokat és függvényeket is írjátok át. Ha ez kész örüljetek, hogy a **ComplexVector** még mindig teljesen jól mûködik. Pedig valójában már õ is megváltozott ezzel.

5. Asteroids

Írjatok egy osztályt **Asteroid** névvel, mely reprezentálni fogja a játékban az aszteroidákat. A következõ (privát) adattagjai legyenek, találjatok ki nekik jó változónevet:

- pozíció: használjátok a korábban megírt **Position**-t
- méret: egész szám

A következõ metódusokat kellene megírni hozzá:

- **konstruktorok**
  - default
  - **Position** és **int**-bõl
  - két **float** és egy **int**-bõl
- **move**:
  - nem ad vissza semmit
  - kap két **float**-ot **x** és **y**-t
  - eltolja az aszteroida pozícióját az (x, y) vektorral.
- **destroy**:
  - egy aszteroida tömböt ad vissza
  - nem kap semmit
  - az elõadáson látotthoz hasonló aszteroida szétesést valósítja meg, azaz létrehoz annyi új eggyel kisebb aszteroidát amekkora a mérete és visszaadja ezek tömbjét. Igaz ez nem a legjobb megoldás, de most legyenek mind az eredeti aszteroida pozícióján.

Bónusz

6. AsteroidContainer

Írjatok egy osztályt ami **Asteroid** objektumokat tud tárolni. Írjatok hozzá konstruktort, ami bekéri, hogy hányat kell tudnia tárolni. Írjátok meg a **push** metódusát, ami egy új Asteroid-ot rak bele, az elsõ üres helyre. Valamint a **get** metódusát, ami egy **int**-et kap és visszaadja az annyiadik Asteroid-ot. Majd módosítsátok az **Asteroid** osztályt, hogy ilyet adjon vissza.

7. Complex mûveletek

Írjátok meg az osztás mûveletet a **Complex** osztályhoz, nyugodtan írjatok hozzá segéd függvényeket, a lényeg hogy minél apróbb részekre bontsátok a programokat. Továbbá írjátok meg a vektoriális szorzatát a **ComplexVector** osztálynak. (Feltételezhetitek, hogy 3 dimenziós vektorokra kell csak mûködnie.)