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 realPart) {
        realPart_ = //TODO
        imaginaryPart_ = //TODO
    }

    public Complex(float realPart, float imaginaryPart) {
        //TODO
    }

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

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

2. User

Egészítsétek ki, ahogy értelmes. Érdekesség (és fontos) ez a kettõ nem ekvivalens:

```
password_ == enteredPassword

password_.equals(enteredPassword)
```

A második azt csinálja, amit az elsõre gondolna az ember. Összehasonlítja a 2 Stringet, ha megegyeznek **true**-t, ha különböznek **false**-t ad vissza.

Viszont az elsõ nem ezt csinálja. Hanem azt vizsgálja meg, hogy ugyanazon a memóriacímen vannak-e tárolva, azaz, hogy tényleg megegyeznek-e, nem csak tartalomban. Ez azért van amit már említettem, hogy itt kb minden pointer (referencia) csak rejtetten. Errõl lesz szó következõ elõadáson.

A **passwordCheck** metódus (függvény) kap egy jelszót, ez az amit beírt a felhasználó. Ezt hasonlítsa össze, a már tárolt jelszóval. Ha megegyezik a kettõ, akkor igazat (**true**) különben hamisat (**false**) adjon vissza.

A **setNickName** állítsa át a **nickName_** adattagot a kapott paraméterre **newNickName**-re.

```
public class User {
private String realName_;
private String nickName_;
private String password_;

public User(String realName, String nickName, String password) {
          = realName;
                    nickName_
```

```
            = nickName_
            = password_
}

public boolean passwordCheck(String enteredPassword) {
if (password_.equals(enteredPassword)) {
//TODO
}
                //TODO
}

public void setNickName(String newNickName) {
//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.

3. ComplexVector szorzás

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

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

    public ComplexVector(int dimension) {
        dimension_ = dimension;
        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 függvényt **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. Felhasználó rendszer

Írjatok egy osztályt (ti találjátok ki a nevét, de legyen beszédes és célratörõ, általában a programozásban ez a legnehezebb feladat) ami a korábban megírt **User** objektumokból tud sokat tárolni (tömb). Továbbá valamilyen módon tárol még kommenteket is. Egy komment az üzenetbõl és a készítõjébõl áll, aki egy **User**. Találjátok ki, hogyan tárolnátok a kommenteket, érdemes új osztályt bevezetni rá. A rendszernek magának legyen 2 függvénye, egyik amivel új **User**t lehet felvenni és egy amivel új kommentet lehet írni.

Pár tanács:

- Hogy a tömb lefoglalással ne legyenek gondok, amikor létrejön ez a rendszer foglaljatok le mondjuk 500 **User**nek helyet és egy számlálóval mentsétek, hogy pontosan mennyi is van valóban a rendszerben.
- Az elõbbihez az kell, hogy a **User** osztálynak legyen **default konstruktor**a, csináljatok hát neki.
- A **User** felvétele függvény talán úgy a legegyszerûbb, ha egy bemente van és az a **User** objektum amit be akarunk rakni a tömbbe.
- A komment tárolásánál elég ha beküldõ nevét tároljátok mondjuk itt nem kell teljes **User**t tárolni.
- Mivel kommentbõl is sokat kell tárolni ezért nagyon hasonló módon meg lehet oldani a tárolásukat, mint a **User**ekét.

Bónusz

6. Komment keresõ

Az 5. feladat rendszeréhez írjatok olyan függvényt, ami képes egy adott **User** össze kommentjét kiírni. Legyen mondjuk a bemenete egy **String**, ami a keresendõ **User nickName** adattagja és listázza ki az összes olyan komment tartalmát aminek õ a szerzõje.

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.)