Objektum orientált programozás
10. gyakorlat
Tömbrendezés, dinamikus tömb, enum

1. Objektum tömbök rendezése és enum

Készítsen saját csomagban Személy osztályt (név, kor adattagokkal), Alkalmazott osztályt a Személyből származtatva (munkahely, fizetés adattagokkal) és EgyetemiAlkalmazott osztályt az Alkalmazottból származtatva (beosztás és osztályszintű alapbér adattaggal; az alapbér 500000). Az osztályokhoz adjuk hozzá a generált konstruktort, getter és setter metódusokat, és a toString metódust.
Az EgyetemiAlkalmazott beosztása legyen enum (PROF, OKTATO, ADMIN). Megvalósítható beágyazott osztályként (static), ill. önálló osztályként. A beosztástól függően a fizetés az alapbér megadott százaléka: professzor – 100, oktató – 50, adminisztrátor – 30. Ezt a konstruktorban kell megadni.

```
//Példányszintű adattag értéke másik példányszintű adattag értékétől függ
//Itt: a fizetés a beosztástól függ
public UniEmployee(String name, int age, String workplace, Position position) {
	super(name, age, workplace, 0);	//a fizetés 0, lent módosítjuk
	this.position = position;
	switch(position) {
		case PROF : setSalary(basesalary); break;
		case OKTATO : setSalary(basesalary*50/100); break;
		case ADMIN : setSalary(basesalary*30/100); break;
	}
}
```

- Készítsen futtatható osztályt, amelyben felvesz 5 egyetemi alkalmazottat, eltárolja dinamikus tömbben és kiírja az összes adatukat (a fizetést a program a beosztás alapján automatikusan számítsa).
- Készítsen statisztikát: hány egyetemi alkalmazott van az egyes beosztásokban.
Algoritmus: *Veszem a következő alkalmazottat. Ha a beosztása megegyezik egy korábban már vizsgált alkalmazottéval, akkor továbblépek a következő alkalmazottra. Ha ez az adott beosztás első előfordulása, akkor létrehozok egy számláló változót 1 értékkel és kiírom a beosztást. Az aktuális indextől kezdve nézem az alkalmazottak beosztását, ha azonos beosztást találok, növelem eggyel a számlálót. A végén kiírom a számláló értékét.*
- Rendezze a tömböt. A rendezéshez használja a *java.util.Arrays* osztály *sort()* metódusát.

Megoldás változatok:

- **Comparable interfész implementálása** (egyszerű, de csak egyféle rendezettség definiálható). Például az EgyetemiAlkalmazott osztály implementálja a Comparable interfészt és életkor szerinti rendezettséget definiál az osztály példányaira:

```
public class UniEmpComp extends Employee implements Comparable<UniEmpComp> {
...
	@Override
	public int compareTo(UniEmpComp o) {
		return this.getAge() - o.getAge();
	}
}
```

Használata UniEmpComp típusú tömb esetén: `Arrays.sort(array);`

- **Comparator interfész implementálása** (többféle rendezettség definiálható). Minden rendezési elvet külön beágyazott static, vagy önálló osztályban definiálunk. Például a névsor szerinti rendezettséget leíró osztály:

```
public class NameSorter implements Comparator<Employee> {
	@Override
	public int compare(Employee e1, Employee e2) {
		return e1.getName().compareTo(e2.getName());
	}
}
```

Használata Employee típusú tömb és önálló rendező osztály esetén:

```
	Arrays.sort(array, new NameSorter());
```

Ha static beágyazott osztályként valósítjuk meg, akkor a beágyazó osztály nevével minősítve érhető el. Pl.: `Employee.NameSorter()`

- **Java8 Comparator** használata (rugalmas, fordított és többszintű rendezettség is megadható) . Az Arrays.sort() metódus második paramétere lehet a rendezettséget leíró Comparator interfész metódushívásokat tartalmazó kifejezés.

Egyetlen kritérium szerinti növekvő rendezés:

```
	Arrays.sort(array, Comparator.comparing(Person::getAge));
```

Egy kritérium szerinti csökkenő rendezés:

```
	Comparator<Employee> compByAge = Comparator.comparing(Person::getAge);
	Arrays.sort(array, compByAge.reversed());
```

Több kritérium szerinti növekvő rendezés:

```
	Arrays.sort(array, Comparator.comparing(Employee::getSalary)
			.thenComparing(Person::getAge)
			.thenComparing(Person::getName));
```

2. Tagosztályok használata

Definiáljon saját csomagban egy Polynomial nevű interfészt.
Metódusai:

- Egy valós számot kap és egy valós számot ad vissza. Szerepe, egy megadott x értékre visszaadni a kiszámolt y-t.
- Paraméter nélküli és egy egészet ad vissza. Szerepe: a polinom fokát visszaadni.

Definiáljon egy másik csomagban LinearPolynomial osztályt, amely egy elsőfokú egyváltozós polinomot reprezentál (képlete: y=ax +b).
Adattagjai: a polinom együtthatói (a és b), valós számok
Konstruktor: két paraméterben megadott értékkel inicializálja az adattagokat
Metódusai:
Implementálja a Polynomial interfész metódusait.
Definiálja felül a toString metódust, amely a polinom képletét adja vissza egy string-ben a konkrét együtthatókkal (Pl.: y=2x-3).

Definiáljon egy futtatható osztályt.
a) Ebben hozzon létre egy osztályszintű metódust, amely paraméterként megkap egy polinomot (Polynomial típusú referenciát) és kiírja a kapott polinom függvénytáblázatát 0 és 1 között 0.1 lépéssel.

b) A main-ben hozzon létre 5 lineáris polinomot beolvasott együtthatókkal és tárolja egy tömbbe. Majd írja ki az összes polinom képletét. Ezután az előző metódus segítségével irassa ki az utolsó polinom függvénytáblázatát.

Definiáljon egy QuadraticPolynomial osztályt, amely egy másodfokú egyváltozós polinomot reprezentál (képlete: $y=ax^2 + bx + c$).
Adattagjai: a polinom együtthatói (a, b és c), valós számok
Konstruktor: három paraméterben megadott értékkel inicializálja az adattagokat.

Metódusai:
Implementálja a Polynomial interfész metódusait.
Definiálja felül a toString metódust, amely a polinom képletét adja vissza egy string-ben a konkrét együtthatókkal (Pl.: $y=2x^2 – 3x + 5$).
Kiszámítja a diszkriminánst ($b^2 – 4ac$).
Visszaadja a megoldás változatokat (2 komplex gyök, 1 valós gyök, 2 valós gyök). A megoldás változatokat enum-ban tárolja!
Kiszámítja az egyes megoldás változatokhoz tartozó gyököket.

Ahhoz, hogy két gyököt vissza tudjon adni, definiálni kell a 2 valós gyököt ill. a 2 komplex gyököt leíró beágyazott osztályokat. A beágyazott osztályokban definiálja felül a toString metódust, ami a két gyököt összefűzve adja vissza.

Egy komplex szám leírásához definiálni kell a Complex osztályt.
Forrás: https://introcs.cs.princeton.edu/java/32class/Complex.java.html

```
public final class Complex {

    private final double re;   // the real part
    private final double im;   // the imaginary part

    // create a new object with the given real and imaginary parts
    public Complex(double real, double imag) {
        re = real;
        im = imag;
    }

    // return a string representation of the invoking Complex object
    public String toString() {
        if (im == 0) return re + "";
        if (re == 0) return im + "i";
        if (im <  0) return re + " - " + (-im) + "i";
        return re + " + " + im + "i";
    }

    // return the real or imaginary part
    public double re() { return re; }
    public double im() { return im; }

}
```

A futtatható osztály main függvényében hozzon létre 5 másodfokú polinomot beolvasott együtthatókkal és tárolja egy tömbbe. Végezzen ellenőrzést az együtthatókra: ha a és b = 0, akkor nem számítunk megoldást; ha a=0, akkor elsőfokú a polinom. Majd írja ki az összes polinom képletét és számítsa ki a megoldásukat.

Házi feladat:

1. Készítsen osztályt új csomagban BookWithStyle néven a korábbi könyv osztály leszármazottjaként. Legyen benne `public` beágyazott felsorolás típus (enum) BookStyle néven a könyv stílusának megadásához a következő konstansokkal: CRIME, COOK, OTHER.
Adattagja: BookStyle típusú stílus adattag
Konstruktor: minden adattagot a paraméterben kapott értékekkel inicializálja
Metódusok:

- Definiálja felül a toString metódust úgy, hogy az ősbelihez még fűzze hozzá a stílust.
- Getter metódus a stílus lekérdezésére.

Ugyanebben a csomagban készítsen egy futtható osztályt.
a) A main metódusban hozzon létre egy dinamikus BookWithStyle tömböt és töltse fel adatokkal. Írja ki a könyvek adatait, majd válogassa ki és írja ki a COOK stílusúakat.
b) Legyen egy osztályszintű metódusa, amely megkap egy könyv tömböt (Book) és minden könyv adatát kiírja.
c) Legyen egy osztályszintű metódusa, amely megkap egy BookWithStyle tömböt és egy BookStyle-t és visszaadja azon könyvek tömbjének referenciáját (BookWithStyle tömb), amelyek a megadott stílusúak.
d) A könyveket lehessen rendezni: cím szerint ABC rendben, oldalszám szerint csökkenő sorrendben.

Dinamikus tömb létrehozása, használata:

```
ArrayList<BookWithStyle> alist = new ArrayList<BookWithStyle>();
alist.add(new BookWithStlye(...));        //új elem hozzáadása
alist.remove(3);                          //megadott indexű tömbelem törlése
```

2. Készítsünk Planet enum osztályt (önálló osztályként), amely tartalmazza a bolygókat és az alábbi konstans adataikat (enum konstansok):

	Merkúr	Vénusz	Föld	Mars	Jupiter	Szaturnusz	Uránusz	Neptunusz
Tömeg (kg)	3.303e+23	4.869e+24	5.976e+24	6.421e+23	1.9e+27	5.688e+26	8.686e+25	1.024e+26
Sugár (m)	2.4397e6	6.0518e6	6.37814e6	3.3972e6	7.1492e7	6.0268e7	2.5559e7	2.4746e7

Az osztálynak ezen felül legyen egy osztályszintű private final adattagja: az egyetemes tömegvonzás konstans (G = 6.67300e-11) és legyen konstruktora. Figyelem! Az enum osztály konstruktora legyen private. Ez automatikusan létrehozza az enum konstansokat, de nem hívható (az enum osztályt nem lehet példányosítani).
Az osztály metódusai:
- Írjuk meg az adattagokat lekérdező metódusokat.
- Számítsuk ki a bolygón a tömegvonzást (double): `G * bolygó tömege/(sugár*sugár)`
- Számítsuk ki az egyik bolygón mért tömeget (paraméter) a másik bolygón mérhető tömegre: `paraméterként kapott tömeg * tömegvonzás az adott bolygón`.

Készítsünk futtatható osztályt, amelyben felhasználjuk a Planet osztályt. Beolvasunk egy tömeget. Ezt elosztjuk a bolygóra jellemző tömegvonzással. Ezt a tömeget lehet átszámítani másik bolygón mérhető tömegre. Egy enum típus összes tagjának elérése a `values()` metódussal történik.

```
int earthWeight = readInt();
double mass = earthWeight/Planet.EARTH.surfaceGravity();
for (Planet p : Planet.values()) {
   System.out.printf("Your weight on %s is %f%n", p, p.surfaceWeight(mass));
}
```

Példa forrása: https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html