STATIC KINTAMIEJI, METODAI, BLOKAI IR KLASĖS.

3 PASKAITA

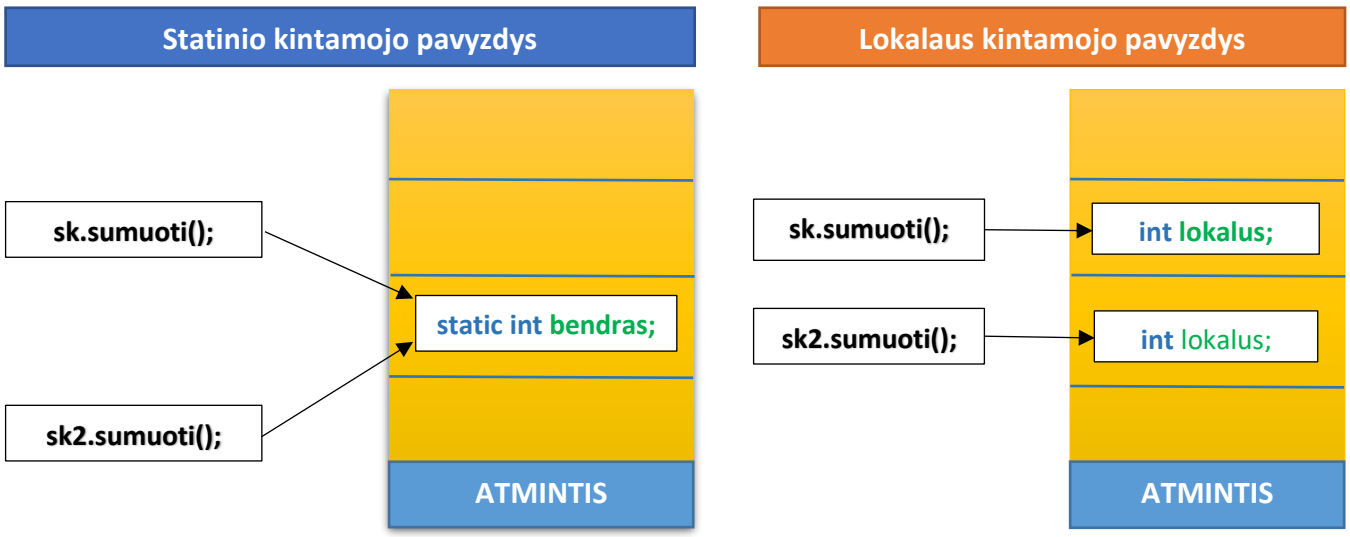
1. STATIC IR LOCAL TIPO KINTAMŲJŲ PALYGINIMAS

Statiniai kintamieji naudojami tuo atveju, kai jums nėra svarbu turėti klasės egzempliorių t.y. kai jūs norite turėti kintamąjį bendrą visiems klasės egzemplioriams, taip yra sutaupoma atmintis, bet ši atmintis yra išskiriama ir naudojama visą programos vykdymo metu, žr. Į žemiau pateiktą programinį kodą. Dešinėje pusėje yra pateikta klasė **Skaiciuokle,** kurioje yra du statiniai kintamieji: *bendras* ir *kiekObjektuSukureme,* taip pat pateikta funkcija sumuoti(), kuri didina reikšmę sumuodama 10 prie lokalus ir bendras kintamųjų, taip pat pateiktas konstruktorius Skaiciuokle(), kuriame yra didinama reikšmė vienetu *kiekObjektuSukureme*. Konstantos apibrėžimas Java kalboje: public static final int KONSTANTA;

1 pav. matome **static int bendras;** ir **int lokalus;** kintamojo reikšmės pokytį kviečiant funkciją **sk1.sumuoti()** ir **sk2.sumuoti()**, kadangi statiniai kintamieji neturi priklausomybės nuo klasės egzempliorių, jiems yra išskiriama atmintis tik vienam egzemplioriui, kuris yra ***bendras*** = *sk1.sumuoti() + sk2.sumuoti() = 20;*. Naudojant lokalius kintamuosius, kiekvienam klasės egzemplioriui atmintyje yra išskiriama vieta, todėl **lokalus** = sk1.sumuoti() = 10; ir **lokalus** = sk2.sumuoti() = 10. O kam skirtas statinis kintamasis *kiekObjektuSukureme ir kodėl jo reikšmė yra lygi 2? Kaip reikėtų elgtis jeigu norėtume paskaičiuoti kiek objektų sukūrėme naudojant lokalų kintamąjį?*

```
public static void main(String[] args) {

    Skaiciuokle sk = new Skaiciuokle();
    Skaiciuokle sk2 = new Skaiciuokle();

    sk.sumuoti();
    sk2.sumuoti();

    sk.getBendras(); // REIKŠMĖ BENDRAS = 20
    sk.getLokalus(); // REIKŠMĖ LOKALUS = 10
    sk.getKiekObjektuSukureme(); // REIKMŠMĖ
            //KIEKOBJEKTUSUKUREME = 2
}
```

```
public class Skaiciuokle {

    static int bendras;
    int lokalus;
    static int kiekObjektuSukureme;

    public Skaiciuokle() {
        kiekObjektuSukureme++;
    }
    public void sumuoti(){
        bendras += 10;
        lokalus += 10;
    }
```



1. pav. Statinio ir lokalaus kintamojo atvaizdavimas atmintyje.

2. SKIRTUMAS TARP STATINIO IR NESTATINIO METODO

Pagrindinis skirtumas toks jog statinis metodas priklauso tik tai klasei, kurioje jis yra apibrėžtas. Paveldėjus bet, kurią klasę, kurioje yra statinių metodų, visi metodai su raktiniu žodžiu **static** nebus paveldimi. Nereikia kurti klasės objekto / egzemplioriaus, kad jį įvykdyti užtenka nurodyti klasės vardą ir metodo pavadinimą pvz: pieš tai turėjome klasę *Skaiciuokle*, kurioje apibrėžtas metodas **public static void *skaiciuotiSuma*(*int a, int b*)**.

```
public static void skaiciuotiSuma(int a, int b){

    System.err.println("Suma"+a + b);
}
```

Statinių metodų vykdymas kaip ir įprastinės klasės, nurodant nuorodą (Klasės_pav.metodas arba Klasės_pav.kintamasis). Skirtumas toks jog statiniai metodai neturi klasės egzempliorių, todėl jums pakanka nurodyti klasės pavadinimą ir kreipinį į norimą metodą pvz: Skaiciuokle.*skaiciuotiSuma(20, 45)*. Kaip matome žemiau pateiktas kodo fragmentas, kuriame matome, kad įprastinio metodo iškvietimas yra galimas tik tuo atveju, kai yra sukuriamas objektas / klasės egzempliorius Skaiciuokle sk = **new** Skaiciuokle(); tik tada mes galyme vykdyti sk.sumuoti();

```
public static void main(String[] args) {

    Skaiciuokle.skaiciuotiSuma(20, 45); // A + B REZULTATAS = 65

    Skaiciuokle sk = new Skaiciuokle();
    Skaiciuokle sk2 = new Skaiciuokle();

    sk.sumuoti();
    sk2.sumuoti();
```

Taip pat reikėtų žinoti jog **static** metodai gali vykdyti tik statinius kintamuosius ir metodus. Pateiktas kodo fragmentas iš klasės *Skaiciuokle* ir **static** metodo ***pakeltiKvadratuSuma***() realizacija, kuriame negalime iškviesti nestatinių (angl. **non-static**) metodų ir kintamųjų, priešingai su nestatiniais (angl. non-static) metodais, kuriuose galima vykdyti **static** ir **non-static** metodus. Norint iškviesti nestatinį metodą ar kintamąjį yra tik vienas būdas sukurti klasės objektą / egzempliorių, tada galėsite naudoti lokalius metodus ir kintamuosius.

```
public static void skaiciuotiSuma(int a, int b) {

    System.err.println("Suma: " + (a + b));
}
public static void pakeltiKvadratuSuma() {
    Skaiciuokle skaiciuokle = new Skaiciuokle();
    skaiciuokle.getBendras();
    skaiciuotiSuma(32, 65);
}
public int getBendras() {
    System.err.println("Bendras" + bendras);
    return bendras;
}
```

3. STATINIS BLOKAS VIETOJE STATINIO KONSTRUKTORIAUS

Kaikurios iš programavimo kalbų turi statinius konstruktorius, kadangi Java kalboje šis veiksmas nėra leistinas, todėl kad vaikinės klasės negali paveldėti static konstruktorių arba metodų, šiai problemai spręsti yra naudojami statiniai blokai, žemiau pateiktas pavyzdys kaip yra aprašomas statinis blokas, taip pat reikia žinoti jog šis blokas yra įkraunamas į atmintį ir bus įvykdomas visada pirmas tada kai tik bus sukurtas klasės objektas, kurioje yra apibrėžtas šis statinis blokas.

```
public class Skaiciuokle {

    static {
        System.out.println("Statinis blokas įvykdytas");
    }
    public static final int KONSTANTA = 59545;
    static int bendras;
    int lokalus;
    static int kiekObjektuSukureme;

    public Skaiciuokle() {
        kiekObjektuSukureme++;
        System.out.println("Konstruktorius egzempliorių skaičius: " + kiekObjektuSukureme);
    }
```

Statinių blokų naudojimas yra reikalingas tuo atveju kai jus norite pakeisti statinių kintamųjų reikšmes pagal nutylėjimą (angl. default values). Šis **static** blokas yra įvykdomas tada kai pvz: turime klasę *Skaiciuokle* yra įkraunama į atmintį, pavyzdžiui bus sukuriamas naujas objektas *Skaiciuokle skaiciuokle = **new** Skaiciuokle();* static blokas bus įvykdomas pirmas prieš konstruktorių vykdymą, žvelgiant į pavyzdį visada bus išspausdinimas tekstas „Statinis blokas įvykdytas“, tada tik bus pereinama prie konstruktoriaus vykdymo.

Statiniai blokai yra įvykdomi tik vieną kartą kai Java virtuali mašina įkrauna klasę į atmintį, lyginant su konstruktoriumi, kuris yra vykdomas kiekvieną kartą kai yra sukuriamas klasės egzempliorius. Kiekviena klasė gali turėti ir keletą statinių blokų, kurių seka priklauso nuo eiliškumo kaip juos surašysite, Java virtuali mašina įkrauna į atmintį apjungus juos tarpusavyje kaip vieną statinį bloką. Šie blokai taip pat vadinami inicijavimo blokai (angl. initialization blocks).

```
public class Skaiciuokle {

    static {
        System.out.println("Statinis blokas įvykdytas");
    }

    static {
        System.out.println("Įvykdytas antras");
    }

    static {
        System.out.println("Įvykdytas trecias");
    }
```

4. STATINIS IMPORTAVIMAS IR PANAUDOJIMAS

Statinis importavimas yra naudojamas tuo atveju kai, norime naudoti statinius narius: metodus, kintamuosius, konstantas, nenurodant klasės pavadinimo tiesiog kviečiant nuorodant konkretų pavadinimą kaip žemiau pateiktame pavyzdyje matome statinius narius importuotus iš klasės *Skaiciuokle.*

```
import static static_modifikatorius.Skaiciuokle.*;
import static static_modifikatorius.Skaiciuokle.kiekObjektuSukureme;
import static static_modifikatorius.Skaiciuokle.skaiciuotiSuma;

public class StatinisImportavimas {

    void skaiciuoti(){
        System.out.println("Statinis importas:"+KONSTANTA);
        skaiciuotiSuma(99, 99);
        kiekObjektuSukureme += 10;
    }

}
```

Pasiekti visus statinius narius iš klasės *Klases_pav*:

- **import static** package-pavadinimas.Klases_pav.*;

Pasiekti konkretų nurodytą statinį kintamąjį:

- **import static** package-pavadinimas.statinis-kintamasis;

Pasiekti konkretų nurodytą statinį metodą:

- **import static** package-pavadinimas.statinis-metodas;

Vienas iš pavyzdžių būtų Math klasė skirta matematiniams skaičiavimams kadangi rašant tam tikro pobūdžio programas gali prireikti šio statinio importavimo dėl dažno matematinių funkcijų naudojimo, kad nerikėtų kartoti klasės vardo. Bet reikia atsiminti, kad jūsų kodo skaitomumas gali pasidaryti sudėtingesnis kai naudosite **import static.**

```
public class Matematiniai_Skaiciavimai {

    void saknis(){

        Math.sqrt(16);
    }
}
```

```
import static java.lang.Math.*;
public class Matematiniai_Skaiciavimai {

    void saknis(){
        sqrt(16);
    }
}
```

Taigi reikėtu atkreipti dėmesį į tai ar tikrai verta jums naudoti **import static** , priklausomai nuo kreipinių skaičiaus į klasėje realizuotus klasės narius, su kuriais dirbsime.

5. STATIC TIPO KLASĖS STRUKTŪRA

Java programavimo kalboje yra tik vieno tipo static klasės, jos vadinamos vidinėmis (angl. Inner). Žemiau esančiame kodo fragmente galite matyti jos realizacija:

```
public class Pagrindine {
    int id = 50;
    void spausdintiID(){
        // Objekto kurimas kitose klasėse
        // jeigu leisime pasiekti iš išorės
        Pagrindine.Vidine vidine = new Pagrindine.Vidine();
        vidine.grazinti_id();
        // Tik tuo atveju kai yra kviečiame Tėvynėje klasėje
        Vidine v = new Vidine();
        v.grazinti_id();
    }
    static class Vidine{
        int id = 90;
        public int grazinti_id(){return id;}
    }
}
```

Pagrindine klasė turinti id lauką, kurio negalės pasiekti vidinė klasė, kadangi kaip ir įprasta static metodų aprašyme vidinėm klasėm galioja tos pačios taisyklės, kad negalite pasiekti *Pagrindine* klasės narių, kurie yra nestatiniai neturint klasės objekto egzemplioriaus.

6. STATIC TIPO KLASĖS TAIKYMAS KURIANT SINGLETON KLASĘ

Singleton klasės tikslas yra kontroliuoti objekto sukūrimą, apriboti objektų skaičių iki vieno klasės egzemplioriaus. Singleton klasės dažnai naudojamos kontroliuoti resursus prisijungimui prie duomenų bazės, resursų kontrolė. Kadangi Singleton klasė turi tik viena egzempliorių, taigi kiekvienas objekto laukas bus įvykdomas vieną kartą kaip ir static lauko atveju.
Pavyzdžiui, jus turite licencija vienam prisijungimui prie duomenų bazės, Singleton klasė užtikrins tai, kad tikrai būtų atliktas vienas prisijungimas vienu metu.

```
public class SingletonKlase {

    private SingletonKlase() {
    }

    public static SingletonKlase getInstance() {
        return SingletonKlaseHolder.INSTANCE;
    }
    private static class SingletonKlaseHolder {

        private static final SingletonKlase INSTANCE = new SingletonKlase();
    }
}
```

SAVIKONTROLĖS UŽDAVINIAI

1. Kas yra static ir lokalus kintamasis ir koks skirtumas tarp jų? Sukurti klasę ***Savikontrole_Static***, kurioje panaudotumėte **static** kintamuosius.
2. Kokie yra pagrindiniai skirtumai tarp statinio ir nestatinio metodo? Prieš tai sukurtą klasę papildyti static metodais.
3. Kas yra statinis blokas ir kokiam tikslui jis yra naudojamas? ***Savikontrole_Static*** klasėje sukurti keletą prasmingų statinių blokų.
4. Kas yra **import static** ir kada jį patartina naudoti? Toliau pildome savo klasę prijungdami, kurią nors biblioteką, kurioje jūs naudotumėte static importą išskyrūs Math biblioteką.
5. Kokios klasės gali būti paskelbtos static ir kokia jų paskirtis? Toliau tobuliname savo programą prijungdami naują klasę ***Studentas***, kurioje būtų realizuota statinė klasė.
6. Kas yra Singleton ir kada jos yra naudojamos? Sukurti vieną prasmingą Singleton klasę.
7. Visas programos testavimo scenarijus privalo būti aprašytas pagrindiniame faile, kuriame yra pagrindinis metodas **main();**