

Java duomenų tipai

Mindaugas Karpinskas
2017



Duomenų tipai

- Prisiminsime duomenų tipus: programoje apibrėžtus (nuorodos) ir primityvius
- Klasės laukų inicializacija, konstruktoriai, "this()" išraiška.
- Constructors
- **static** modifikatorius

Links

<https://www.thejavaprogrammer.com/cannot-make-static-reference-non-static-method/>

<https://www.sitesbay.com/java/java-static-and-non-static-method>

<http://study.com/academy/lesson/static-vs-non-static-methods-in-java.html>

<https://beginnersbook.com/2015/03/for-loop-in-java-with-example/>

https://www.tutorialspoint.com/java/java_loop_control.htm

<http://www.learnjavaonline.org/en/Loops>

Certified Associate Java® SE 8 Programmer I Study Guide Exam 1Z0-808

- *Declare and initialize variables*
- *Differentiate between object reference variables and primitive variables*
- *Know how to read or write to object fields*
- *Explain an Object's Lifecycle (creation, “dereference by reassignment” and garbage collection)*

Prisiminti primityvius tipus

int, boolean, long, short, byte, char, double, float.

- **int** skaicius = 5;
- **boolean** tiesa = **true**;
- **long** didelisSkaicius = 123123123L;
- **short** pazymys = 10;
- **char** korpusas = 'C';
- **float** someNumber = 32.32F;
- **double** someBigNumber = 64.64;

public static void main

public class PVZ1 {

public static void main(String[] args) {
 // **TODO** Auto-generated method stub

 }

}

Objektas vs Klasė (prisiminkime)

Klasė - šablonas, nuorodos tipas

Objektas - gaminys pagal šablona, egzempliorius

Programuotojo apibrėžiami tipai

Java kalboje nauji tipai apibrėžiami su **class** arba **interface** arba **enum**.

Su **interface**, **enum** susipažinsime vėlesnėse dalyse, o dabar dar kartą aptarsime **class** tipo nuorodas

Sakykime, galime apibrėžti "**MyType**" klasę, kuri bus paskelbta **public**.

**Šis modifikatorius pažymi, kad naujai paskelbtas tipas MyType yra viešas, t.y. prieinamas kitoms klasėms.*

Užduotis: Tipai1

(prisiminkim ir pakartokim)

-
1. Susikurti tipą/klasę vardu: <Vardas>
 - a. Naujas tipas turi turėti kintamąjį: **pranešimas**
 - b. Naujas tipas turi turėti metodą: spausdinkPranesima(); - išspausdina kintamojo reikšmę.
 2. Main metode sukurti Jūsų pasirašytą tipą ir priskirti reikšmę klasės kintamajam.
 3. Main metode iškviesti metodą spausdinkPranesima();

Užduotis: Tipai1

```
package lt.codeacademi.study.tipai;
```

```
public class Mindaugas {
```

```
    String pranesimas;
```

```
    V
```

```
}
```

```
}
```

```
package lt.codeacademi.study.tipai;
```

```
public class Uzduotis01 {
```

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

```
        Mindaugas m = new Mindaugas();
```

```
    }
```

```
}
```

Užduotis: Tipai1

```
package lt.codeacademi.study.tipai;  
  
public class Mindaugas {  
    private String pranesimas;  
  
    void spausdinkPranesima() {  
        System.out.println(pranesimas);  
    }  
}
```

```
package lt.codeacademi.study.tipai;  
  
public class Uzduotis01 {  
  
    public static void main(String[] args) {  
        Mindaugas m = new Mindaugas();  
    }  
}
```

Užduotis: Tipai1

(prisiminkim ir pakartokim)

```
package lt.codeacademi.study.tipai;
```

```
public class Mindaugas {  
    String pranesimas;  
  
    void spausdinkPranesima() {  
        System.out.println(pranesimas);  
    }  
}
```

```
package lt.codeacademi.study.tipai;
```

```
public class Uzduotis01 {  
  
    public static void main(String[] args) {  
        Mindaugas m = new Mindaugas();  
        m.pranesimas = "Mano vardas";  
        m.spausdinkPranesima();  
    }  
}
```

"MyType" klasė. Tipo apibrėžimas

```
class MyType {  
    public int myDataMember = 4;  
  
    public void myMethodMember() {  
        System.out.println("As esu svarbus narys!");  
        System.out.println("myData=" + myDataMember);  
    }  
}
```

“new”

Klasių tipo inicijuoti kintamieji vadinami **objektais** arba **egzemplioriais**. Kaip ir masyvų atveju, klasės kintamasis tampa **egzemplioriumi**, kai jam išskiriama vieta atmintyje naudojant **new** operatorių.

```
public class MyType {  
    public int myDataMember = 4;  
  
    public void myMethodMember() {  
        System.out.println("As esu svarbus narys!");  
        System.out.println("myData=" + myDataMember);  
    }  
}
```

```
public class RunMe {  
    public static void main(String[] args) {  
        MyType mine = new MyType();  
        int i = mine.myDataMember;  
        mine.myMethodMember();  
        System.out.println(i);  
    }  
}
```

```
> javac RunMe.java  
> java RunMe  
As esu svarbus narys!  
myData=4  
4
```

Klasės elementai pasiekiami per tašką "."

Kintamasis

private String **pranesimas**;

- Kaip iš kitos klasės pakeisti privataus kintamojo reikšmę?
 - Inicijavimo metu priskirti reikšmę
 - Sukurti metodą kurio pagalba keisime jo reikšmę
 - Konstruktorius

constructor

Pavyzdys iliustruoja tris darbo su myType klase elementus

- Sukūrimą **new**
- pasinaudojimą egzemplioriaus kintamaisiais
- pasinaudojimą egzemplioriaus metodais

myMethodMember metodas atspausdins:

```
As esu svarbus narys!  
myData=4
```

Kadangi myDataType yra nuorodos tipo, reikia naudoti **new** operatorių, kuris išskiria egzemplioriui vietą atmintyje. Po **new** yra nurodomas klasės konstruktoriaus pavadinimas, kuris turi sutapti su klasės pavadinimu. **Konstruktorius gali atlikti kokius nori veiksmus; pateiksime myType konstruktoriaus pavyzdį, kuris praneša apie savo sukūrimą.**

Apie inicijavimą pranešantis konstruktorius

```
public class MyType {  
    int myDataMember=0;  
    public MyType() {  
        System.out.println("Esu inicijuojamas!");  
    }  
    public MyType(int val) {  
        System.out.println("Priskiriu myDataMember="+val);  
        myDataMember=val;  
    }  
}
```

MyType konsruktorių panaudojimas

```
public class RunMe {  
    public static void main(String s[]) {  
        MyType instancel=new MyType();  
        MyType instance2=new MyType(100);  
    }  
}
```

Rezultatas turėtų būti:

Esu inicijuojamas!!

Priskiriu myDataType=100

Užduotis: Tipai2

1. *Susikurti tipą vardu: ManoPranešimas*
 - a. *Naujas tipas turi turėti kintamąjį: **pranesimas***
 - b. *Naujas tipas turi turėti metodą: spausdinkPranesima();*
- ~~2. *Main metode sukurti Jūsų pasirašytą tipą ir priskirti reikšmę kintamajam.*~~
- ~~3. *Main metode iškvieisti metodą spausdinkPranesima();*~~
4. Pakeisti užduoties “Tipai1” sukurtą klasę taip, kad **pranesimas** kintamojo reikšmę būtų galima priskirti konstruktoriui

Užduotis: Tipai2

```
package lt.codeacademi.study.tipai;  
public class Mindaugas {  
    String pranesimas;  
  
    void spausdinkPranesima() {  
        System.out.println(pranesimas);  
    }  
}
```

```
package lt.codeacademi.study.tipai;  
  
public class Uzduotis2 {  
  
    public static void main(String[] args) {  
        m.spausdinkPranesima();  
    }  
}
```

Užduotis: Tipai2


```
package It.codeacademi.study.tipai;  
public class Mindaugas {  
    String pranesimas;  
  
    public Mindaugas(String string) {  
        pranesimas = string;  
    }  
  
    void spausdinkPranesima() {  
        System.out.println(pranesimas);  
    }  
}
```

```
package It.codeacademi.study.tipai;  
  
public class Uzduotis2 {  
  
    public static void main(String[] args) {  
        Mindaugas m =  
            new Mindaugas("Mano vardas");  
        m.spausdinkPranesima();  
    }  
}
```

Užduotis: Tipai2

```
package It.codeacademi.study.tipai;

public class Mindaugas {
    String pranesimas;
    public Mindaugas() {
    }
    public Mindaugas(String string) {
        pranesimas = string;
    }
    void spausdinkPranesima() {
        System.out.println(pranesimas);
    }
}
```



```
package It.codeacademi.study.tipai;

public class Uzduotis2 {

    public static void main(String[] args) {
        Mindaugas m = new Mindaugas();
        M.pranesimas = "Mano vardas";
        m.spausdinkPranesima();
    }

}
```

Constructors

To create an instance of a class, all you have to do is write **new** before it. For example:

```
Random r = new Random();
```

IŠ:

Certified Associate Java Study Guide

Key points to note about the constructor:

1. The name of the constructor matches the name of the class

```
public class Chick {  
    public Chick() {  
        System.out.println("in constructor");  
    }  
}
```

2. There's no return type

```
public void Chick() {  
    // NOT A CONSTRUCTOR *  
}
```

*It's a regular method that won't be called when you write `new Chick()` .

The “main” purpose of a constructor is to initialize fields

```
public class Chicken {  
    int numEggs = 0; // initialize on line  
    String name = "Duke";  
    public Chicken() {  
        // empty block  
    }  
}
```

```
public class Chicken {  
    int numEggs = 0; // initialize on line  
    String name = "Duke";  
}
```

```
public class Chicken {  
    int numEggs; // defining fields  
    String name;  
    public Chicken() {  
        // initialize in constructor  
        numEggs = 0;  
        name = "Duke";  
    }  
}
```

Užduotis: Tipai3

1. Sukurti klasę kuri turi
 - a. lauką: **private** *'reikšmė'*
 - b. du konstruktorius (be parametro is su parametru):
 - i. kontr. be parametro, laukui nustato reikšmę ??
 - ii. konstruktrius su parametru, laukui nustato parametro perduotą reikšmę:
 - c. metodą: parodyk
2. **main** metode patestuoti kaip viskas veikia
3. Konstruktoriuose išspausdinti tekstą: “Su parametru” arba “Be parametro”.

ConstructorTask

```
class ConstructorTask3 {  
    private char value;  
    public ConstructorTask3() {  
        value = 'Y';  
        System.out.println("Without arguments");  
    }  
    public ConstructorTask3(char value) {  
        this.value = value;  
        System.out.println("With argument 'value'");  
    }  
    public void show() {  
        System.out.println("My value: " + value);  
    }  
}
```

static

Klasės “grįžtam” OOP

Klasėje yra talpinami kintamieji ir metodai.

*Klasės yra Java objektinio programavimo pagrindas. Kiekviena klasė apibrėžia naują programuotojo sukurtą tipą, kuriame yra naudojami kintamieji ir apibrėžiami metodai. Klasė apibrėžia tipą, o to tipo (klasės) kintamuosius vadiname objektais. Objektai skiriasi nuo įprastų kintamųjų tuo, kad juose apibrėžiami **metodai aprašantys kintamųjų keitimo taisykles**. Neobjektinių programavimo kalbų paprogramės neturi lankstaus mechanizmo, kaip reguliuoti paprogramės kintamųjų keitimą kitoms paprogramėms.*

Modifikatoriaus “**static**”

- ****Statiniai metodai yra analogiški į C kalbos funkcijas function*
- Juos galima iškviesti nesukūrus objekto
- Statiniai metodai negali persidengti

Metodai su **static** žyme yra laikomi algoritmais, kuriems nėra reikalingi objekto duomenys ir metodai. Statiniams metodams ir kintamiesiems paskiriama statinė atmintis.

Pvz.: Sąskaita

Sukurti klasę Sąskaita, joje apsirašyti du laukus: sumą ir sąskaitos numerį.

Mums įdomu, kiek tokio tipo objektų bus sukurta? Tokio tipo uždaviniams galima pasitelkti static klasės kintamąjį į pagalbą.

Realizacija Java kalboje

```
public class Account {  
    // atributai  
    private int amount;  
    private int number;  
    private static int num_of_accounts;  
    //konstruktorius  
    public Account() {  
        this.number = num_of_accounts;  
        num_of_accounts++;  
    }  
    public void deposit(int amount) {  
        this.amount += amount;  
    }  
    public void withdraw(int amount) {  
        this.amount -= amount;  
    }  
    public int getAmount() {  
        return this.amount;  
    }  
    public int getNumber() {  
        return this.amount;  
    }  
    public void setAmount(int a) {  
        this.amount = a;  
    }  
}}
```

Statiniai atributai ir metodai

- Bendri visai klasei
- Bendri visiems klasės objektams/egzemplioriams
- Pasiekama ne per objektą, o per klasę
 - Pvz. `Account.createAccount();`
 - Pvz. `Account.num_of_accounts;`
- Naudojama globaliems duomenims bei veiksmams aprašyti

Statinių ir dinaminių narių palyginimas

Daugeliu atvejų apibrėždami metodus mes nevartojome modifikatoriaus **static**. Šiuo atveju metodas interpretuojamas **dinaminiu** (pagal nutylėjimą)

static (2)

Statiniai metodai ir kintamieji kviečiami tiesiog rašant pastovų klasės pavadinimą

- Pvz. `Account.createAccount();`

Dinaminis metodas ir kintamasis kviečiamas naudojant klasės egzemplioriaus pavadinimą

- Pvz.:
 - `Account account = new Account();`
 - `Int amount = account.getAmount();`

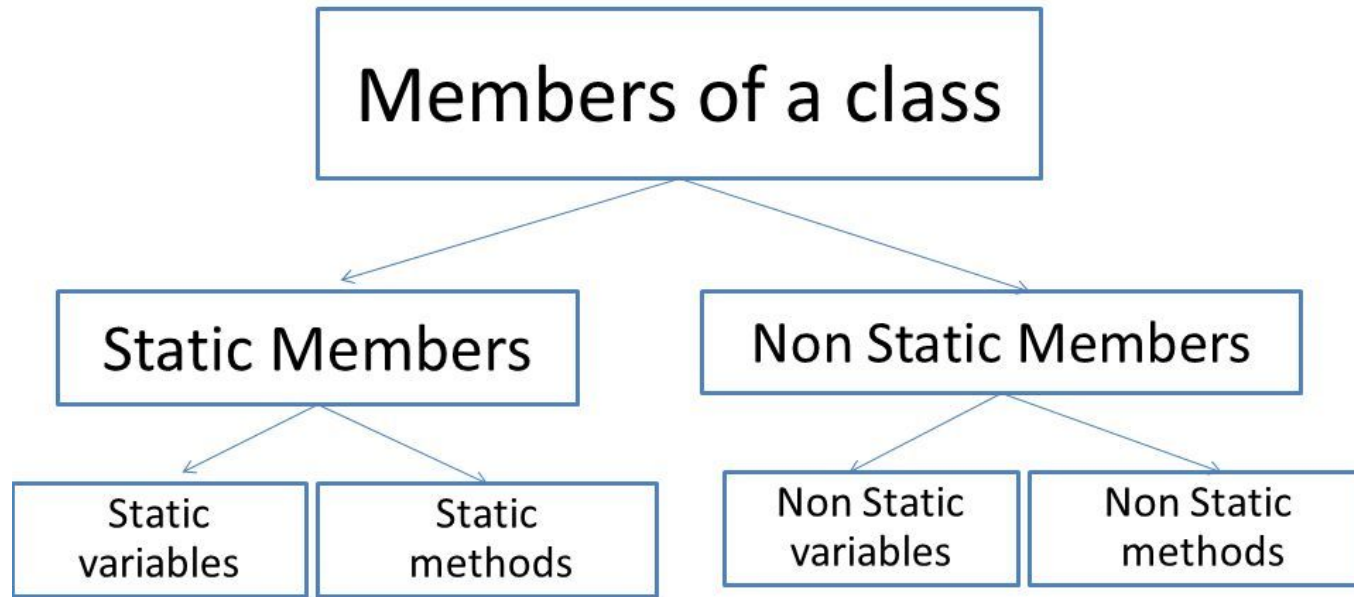


Kadangi ta pati klasė gali turėti daug egzempliorių skirtingais pavadinimais, tai ir to paties metodo ar kintamojo kvietimai atrodys skirtingai.

- `account1.getAmount(); ... account2.getAmount();`

Statinio ir dinaminio modifikatoriaus palyginimas

Metodo tipas	Modifikatorius	Sintaksė
Dinaminis	Nerašomas (nutylimas dinaminis)	<code><objektas>.<metodo_vardas></code> <code>(<parametrų_sąrašas>)</code>
Statinis	<code>static</code>	<code><clasės_vardas>.<metodo_vardas></code> <code>(<parametrų_sąrašas>)</code>



- **Static** variables are shared across all the objects of a class
 - There is only *one copy*
- **Non-Static** variables are not shared
 - There is a *separate copy for each individual live object*.
- Static variables **cannot** be declared **within** a method.

Static Class Members

- *Static fields* and *static methods* do not belong to a single instance of a class.
- To invoke a static method or use a static field, the class name, rather than the instance name, is used.
- Example:

```
double val = Math.sqrt(25.0);
```



Class name

Static method

Difference Between Non-static and Static Method

```
class A {  
    void fun1() {  
        System.out.println("Hello I am Non-Static");  
    }  
    static void fun2() {  
        System.out.println("Hello I am Static"); }  
}  
class Person {  
    public static void main(String args[]) {  
        A obj=new A();  
        obj.fun1(); // Call non static method  
        A.fun2(); // Call static method  
    }  
}
```

Output is:

Hello I am Non-Static
Hello I am Static



DemoClass.java

```
package com;
```

```
public class DemoClass {
```

```
    void print() {
```

```
        System.out.println("Hello World");
```


```
    }
```

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

```
        print();
```

```
    }
```

```
}
```

 Cannot make a static reference to the non-static method print() from the type DemoClass

1 quick fix available:

 [Change modifier of 'print\(\)' to 'static'](#)

Press 'F2' for focus

Užduotis: Tipai4

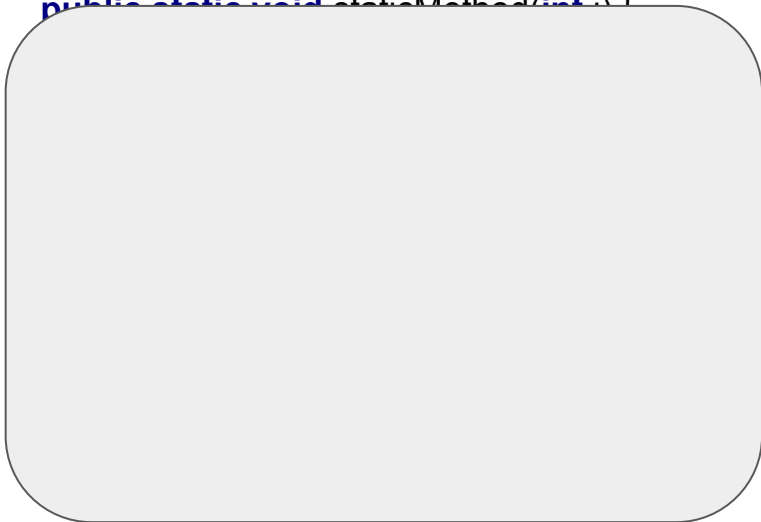
1. Reikalingos dvi klasės
2. Vienoje iš klasių:
 - a. Sukurkime ne-static int kintamąjį
 - b. Sukurkime ne-static metodą `setValue(int i);` su parametru `int i`;
 - i. kuris išspausdina `i` reikšmę
 - ii. ir priskiria ją objekto kintamajam
 - c. Sukurkime static metodą `staticMethod(int j);` su parametru `int j`;
 - i. kuris išspausdina `j` reikšmę
 - ii. ir *priskiria `j` reikšmę objekto kintamajam
3. Kitoje klasėje
 - a. Sukuriamas main metodas
 - i. Kuris pabandys iškviesti pirmos klasės: `staticMethod(int j)` ; su reikšme 10
 - ii. Ir metodą `setValue(int i);` su reikšme 10

Statinių ir dinaminių metodų palyginimas. PVZ

```
public class StaticVsDynamic {
```

```
    int i=0;
```

```
    public static void staticMethod(int i) {
```



```
+k);
```

```
}}
```

// Pavyzdžio klasėje apibrėžiamas ir statinis ir dinaminis metodai. Statinis metodas "nežino" apie setInt, returnInt ir i reikšmes. Pasižiūrėkime, kaip atrodo dinaminio ir statinio metodo panaudojimas

Statinių ir dinaminių metodų palyginimas. PVZ

```
public class StaticVsDynamic {  
    int i=0;  
    public static void staticMethod(int j) {  
        System.out.println("Statinis metodas");  
        System.out.println("j="+j);  
    }  
    //dinaminis metodas  
    public void setInt(int k) {  
        i=k;  
        System.out.println("Priskiriame i reiksme "+k);  
    }  
    public int returnInt() {  
        return i;  
    }  
}
```

// Pavyzdžio klasėje apibrėžiamas ir statinis ir dinaminis metodai. Statinis metodas "nežino" apie setInt, returnInt ir i reikšmes. Pasižiūrėkime, kaip atrodo dinaminio ir statinio metodo panaudojimas

Dinaminio ir statinio metodo iškvietimas

```
public class RunMe {  
    public static void main(String S[]) {  
  
        StaticVsDynamic.staticMethod(10);  
        //nereikia naudoti konstruktoriaus iškviečiant statinį metodą  
        StaticVsDynamic a = new StaticVsDynamic();  
        //prieš kviečiant dinaminį metodą pirma jį reikia inicijuoti  
        a.setInt(10);  
        System.out.println("a.i = " + a.returnInt());  
    }  
}
```

Statinis modifikatorius naudojamas ir klasės kintamiesiems. Statinių kintamųjų iškvietimo sintaksė analogiška statinių metodų iškvietimui.

<klasės_vardas>.<kintamojo_vardas>

static privalumai

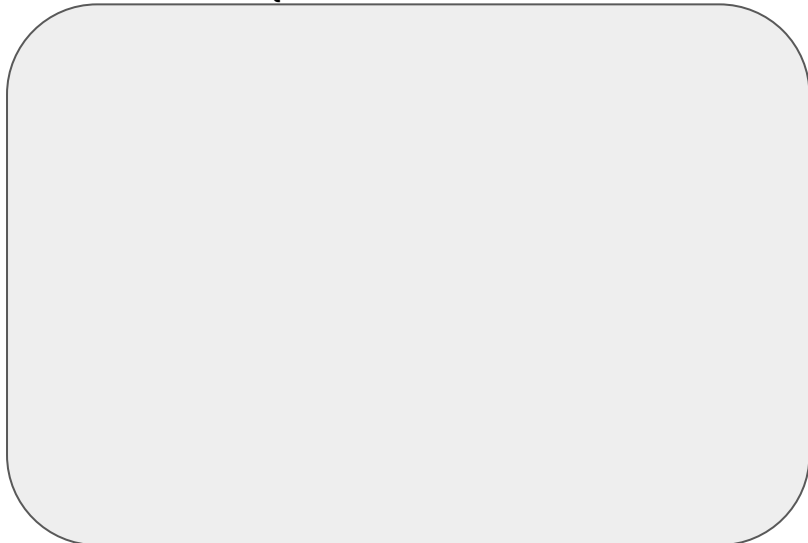
- Visi kintamieji ir metodai turi būti apibrėžti klasėje, statinis modifikatorius pažymi tuos metodus ir kintamuosius, kurie nepriklauso nuo egzemplioriaus.
- Kuriant naują egzempliorių neišskiriama nauja vieta statiniams klasės metodams ir kintamiesiems ir jie visi prieinami pagal fiksuotą klasės pavadinimą.
- *Statiniai kintamieji analogiškai kitose kalbose naudojamiems global kintamiesiems, skirtumas tik tas, kad jie prieinami tik žinant ir panaudojant klasės pavadinimą.*
 - Tai yra Java kalbos privalumas, nes nereikia jaudintis, kad kažkokioje programoje koks nors global kintamojo vardas jau buvo panaudotas ir skirtingos esmės kelių kintamųjų pavadinimas vienu vardu iškels sunkiai aptinkamos programavimo klaidos problemą.

Užduotis: Tipai5

1. Reikalingos dvi klasės
2. Vienoje iš klasių:
 - a. Sukurkime static int kintamąjį *ir ne-static int kintamąjį (2kint.)* **public**
 - b. Sukurkime konstruktorių;
 - i. kuris išspausdina kintamųjų reikšmes
 - ii. ir padidina jas vienetu
 - c. Sukurkime static metodą isvalyti();
 - i. kuris išspausdina statinio kintamojo reikšmę
 - ii. ir priskiria jam 0
3. Kitoje klasėje
 - a. Sukuriamas main metodas
 - i. Sukuria 5 pirmos klasės objektus/egzempliorius
 - ii. Išspausdina pirmos klasės statinio kintamojo reikšmę,
 - iii. Iškviečia statinį metodą isvalyti();
 - iv. Ir vėl išspausdina pirmos klasės statinio kintamojo reikšmę

Užduotis: Tipai5

```
class Uzduotis5 {
```



```
}
```

```
public class RunUzduotis4 {
```

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



```
}
```

```
}
```

Užduotis: Tipai5

```
class Uzduotis5 {  
    static int count;  
    int a;  
    Uzduotis4() {  
        count++; a ++;  
        System.out.println("count reikšmė: "  
                               + count);  
        System.out.println("a reikšmė: "  
                               + a);  
    }  
}
```

```
public class RunUzduotis5 {  
  
    public static void main(String[] args) {  
        new Uzduotis4();  
        new Uzduotis4();  
        new Uzduotis4();  
        new Uzduotis4();  
        new Uzduotis4();  
    }  
}
```

Užduotis: Tipai5

```
class Uzduotis5 {  
    static int count;  
    Uzduotis5() {  
        count++;  
        System.out.println("Kintamojo reikšmė: "  
                               + count);  
    }  
  
    static void išvalyk() {  
        System.out.println("Kintamojo reikšmė: "  
                               + count);  
        count = 0;  
    }  
}
```

```
public class RunUzduotis5 {  
  
    public static void main(String[] args) {  
        new Uzduotis5();  
        new Uzduotis5();  
        new Uzduotis5();  
        new Uzduotis5();  
        new Uzduotis5();  
  
        System.out.println(  
            "Kintamojo reikšmė prieš išvalymą: "  
                + Uzduotis4.count);  
  
        Uzduotis5.išvalyk();  
        System.out.println(  
            "Kintamojo reikšmė po išvalymo: "  
                + Uzduotis4.count);  
    }  
}
```

Example of Static and non-Static Method

```
class A {  
    void fun1() {  
        System.out.println("Hello I am Non-Static");  
    }  
    static void fun2() {  
        System.out.println("Hello I am Static");  
    }  
}  
  
class Person {  
    public static void main(String args[]) {  
        A oa = new A();  
        oa.fun1();           // non static method  
        A.fun2();           // static method  
    }  
}
```

end

What is one of the characteristics of a static method in Java?

1. A static method belongs to the class
2. A static method belongs to an instance of the class
3. A static method can access an instance variable without creating an instance of a class
4. A static method can access a static variable by creating an instance of a class

Which of the following is NOT true about the main method?

1. The main method is a static method
2. The main method is an instance method
3. The main method is called when the program starts without having to create an instance of the class
4. The main method can access another static method without creating an instance of the class