

# Constructor & “final” keyword

Mindaugas Karpinskas  
2017



# Links

---

<http://beginnersbook.com/2013/03/constructors-in-java/>

## Rules for creating java constructor

---

here are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

## Types of java constructors

---

There are two types of constructors:

?

?

## Types of java constructors

---

There are two types of constructors:

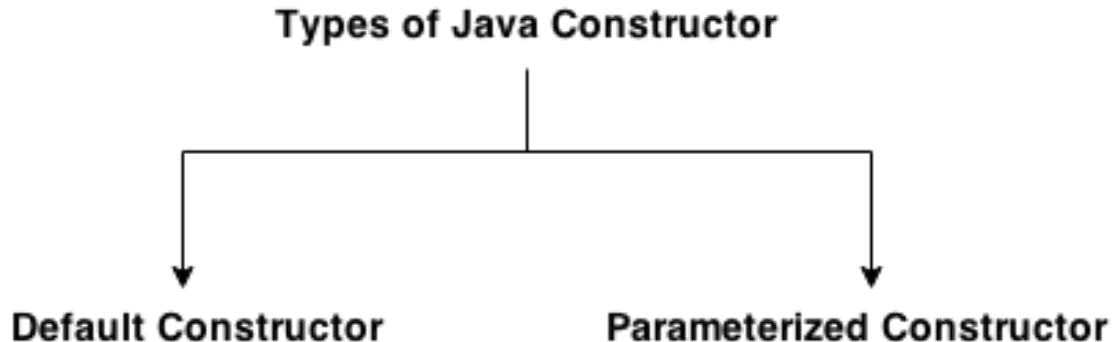
1. Default constructor (no-arg constructor)
2. Parameterized constructor

## Types of java constructors

---

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



## Syntax of default constructor:

---

- `<class_name>(){}`

```
class Bike1 {  
    Bike1() {  
        System.out.println("Bike is created");  
    }  
  
    public static void main(String args[]) {  
        Bike1 b = new Bike1();  
    }  
}
```

## Syntax of default constructor:

---

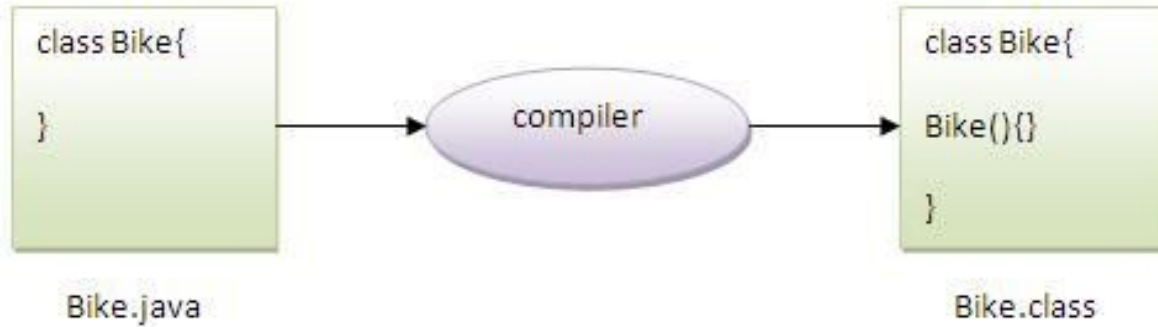
- `<class_name>(){}`

```
public class Bike1 {  
    public Bike1() {  
        System.out.println("Bike is created");  
    }  
  
    public static void main(String args[]) {  
        Bike1 b = new Bike1();  
    }  
}
```



**Rule: If there is no constructor in a class, compiler automatically creates a default constructor.**

---



## Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

```
class Student4 {  
    int id;  
    String name;  
    Student4(int i, String n) {  
        id = i;  
        name = n;  
    }  
    void display() {  
        System.out.println(id + " " + name);  
    }  
    public static void main(String args[]) {  
        Student4 s1 = new Student4(111, "Mindaugas");  
        Student4 s2 = new Student4(222, "Tadas");  
        s1.display();  
        s2.display();  
    }  
}
```

# Constructors in Java

---


```
public class Example2 {  
    private int var;  
    public Example2() {  
        // code for default one  
        var = 10;  
    }  
    public Example2(int num) {  
        // code for parameterized one  
        var = num;  
    }  
    public int getValue() {  
        return var;  
    }  
    public static void main(String args[]) {  
        Example2 obj2 = new Example2();  
        System.out.println("var is: " + obj2.getValue());  
    }  
}
```

<http://beginnersbook.com/2013/03/constructors-in-java/>

# Constructors in Java

---

```
public class Example2 {  
    private int var;  
    public Example2() {  
        this(10);  
        // code for default one  
    }  
    public Example2(int num) {  
        // code for parameterized one  
        var = num;  
    }  
    public int getValue() {  
        return var;  
    }  
    public static void main(String args[]) {  
        Example2 obj2 = new Example2();  
        System.out.println("var is: " + obj2.getValue());  
    }  
}
```



1. Using “**this**” as method **this()** we can re-use existing constructors in same class
2. If in constructor body we are calling another constructor: the other constructor call (super, this)  
  
statement **must** be **first** in constructor body!

# Game

---

```
class Game {  
    int players;  
    Game() {  
        players = new Random().nextInt(4) + 1;  
    }  
  
    Game(int players) {  
        this.players = players;  
    }  
}
```

```
class Game {  
    int players;  
    Game() {  
        this.players = new Random().nextInt(4) + 1;  
        System.out.println("Today will play " + this.players + " players");  
        initPlaygroundFor(this.players);  
        giveBoll();  
    }  
    Game(int players) {  
        this.players = players;  
        System.out.println("Today will play " + this.players + " players");  
        initPlaygroundFor(this.players);  
        giveBoll();  
    }  
    void initPlaygroundFor(int players) {    }  
    void giveBoll() {    }  
}
```

```
class Game {  
    int players;  
    Game() {  
        this.players = new Random().nextInt(4) + 1;  
        System.out.println("Today will play " + this.players + " players");  
        initPlaygroundFor(this.players);  
        giveBoll();  
    }  
    Game(int players) {  
        this.players = players;  
        System.out.println("Today will play " + this.players + " players");  
        initPlaygroundFor(this.players);  
        giveBoll();  
    }  
    void initPlaygroundFor(int players) { }  
    void giveBoll() { }  
}
```

```
class Game {  
    int players;  
    Game() {  
        this.players = new Random().nextInt(4) + 1;  
        this(this.players);  
    }  
    Game(int players) {  
        this.players = players;  
        System.out.println("Today will play " + this.players + " players");  
        initPlaygroundFor(this.players);  
        giveBoll();  
    }  
    private void initPlaygroundFor(int players2) {  
    }  
  
    private void giveBoll() {  
    }  
}
```



```
class Game {  
    int players;  
    Game() {  
        this.players = new Random().nextInt(4) + 1;  
        this(this.players);  
    }  
    Game(int players) {  
        this.players = players;  
        System.out.println("Today will play " + this.players + " players");  
        initPlaygroundFor(this.players);  
        giveBoll();  
    }  
    private void initPlaygroundFor(int players2) {  
    }  
  
    private void giveBoll() {  
    }  
}
```

```
class Game {  
    int players;  
    Game() {  
        this(new Random().nextInt(4) + 1);  
        System.out.println("Playing with random players.");  
    }  
    Game(int players) {  
        this.players = players;  
        System.out.println("Today will play " + this.players + " players");  
        initPlaygroundFor(this.players);  
        giveBoll();  
    }  
    private void initPlaygroundFor(int players2) {  
    }  
  
    private void giveBoll() {  
    }  
}
```

# Konstruktoriai

---

- Klasė turi du konstruktorius.
  - Pirmasis išspausdina tekstą “K1” į konsolę
  - Antrasis taip pat išspausdina “K2” ir perduotą reikšmę į konsolę
- Konstruktoriuje iškviesti kitą konstruktorių

**final**

# final kintamieji

---

- Raktas **final** prie kintamojo reiškia, kad kintamojo reikšmė nekis – ji galutinė
- Kintamajam reikšmę galima priskirti jį deklaruojant
- Jei reikšmę priskiria kompiliatorius, kintamojo vardas rašomas didžiosiomis raidėmis, žodžiai skiriami pabraukimais
- Pvz., final float **MANO\_PI** = 3.14;

# Final for variable

---

**final** modifikatorius nustato, kad kintamojo reikšmė negali būti pakeista. Šis kintamasis iš karto turi būti inicijuojamas ir bet kuris bandymas jį keisti iššauks kompiliavimo klaidą. **final** modifikatorius paprastai naudojamas apibrėžti konstantas.

Konstantos dažniausiai nurodomos tokiu modifikatorių rinkiniu:

1. **public** - viešai preinamas;
2. **static** - globalus visoms klasėms ir objektams;
3. **final** - nekintamas.

```
class Constants {
```

```
    public static final int ETA = 64;
```

```
}
```

# final kintamasis

---

Galima deklaruoti **final** kintamąjį ir jam iš karto priskirti reikšmę.

```
class F {  
    final int j = 10;  
    // kiti kintamieji <...>  
    F() {  
        // kodas ...  
    }  
    F(int i) {  
        // kodas .  
    }  
    F(Object o) {  
        // kodas ...  
    }  
    // kiti konstruktoriai ir metodai  
    <...>  
}
```

# Tuščias **final** kintamasis

---

Galima deklaruoti tuščią **final** kintamąjį, o jam reikšmę priskirti **kiekviename** konstruktoriuje

```
class F {  
    final int j;  
    F() {  
        this.j = 1;  
    }  
    F(int i) {  
        this.j = i;  
    }  
    F(Object o) {  
        this.j = 1;  
    }  
}
```



# “Final” Uzduotis1

---

Sukurti klasę **TestFinal** su dviem kintamaisiais. Klasė taip pat turi turėti metoda **priskirk** ir konstruktorių, kuriuose kintamiesiems priskiriamos perduotos reikšmės ir išspausdina objekto kintamųjų reikšmes.

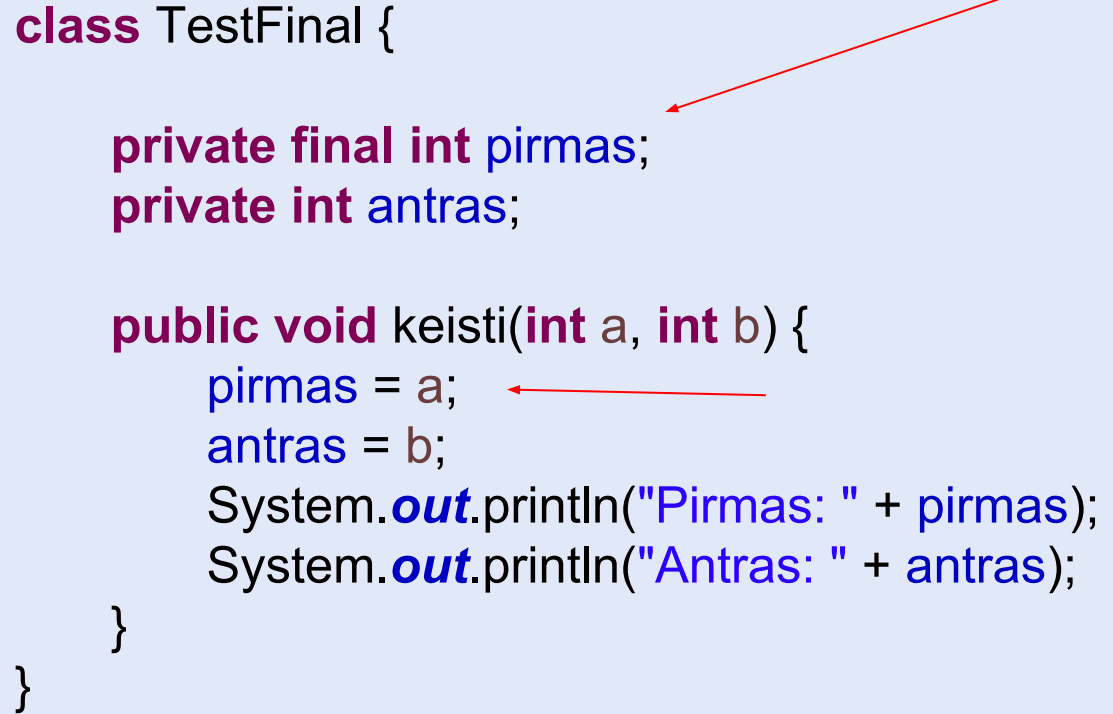
1. Main metode sukurti keleta **TestFinal** objektų ir jiems iškviesti metoda **priskirk**
2. Paskelbti vieną iš kintamųjų **final**.

# final

```
class TestFinal {  
  
    private int pirmas;  
    private int antras;  
  
    public void keisti(int a, int b) {  
        pirmas = a;  
        antras = b;  
        System.out.println("Pirmas: " + pirmas);  
        System.out.println("Antras: " + antras);  
    }  
}
```

# final

```
class TestFinal {  
  
    private final int pirmas;  
    private int antras;  
  
    public void keisti(int a, int b) {  
        pirmas = a;  
        antras = b;  
        System.out.println("Pirmas: " + pirmas);  
        System.out.println("Antras: " + antras);  
    }  
}
```



The diagram illustrates the effect of the `final` keyword in Java. A red arrow points from the word `final` in `private final int pirmas;` to the variable `pirmas`, indicating that its value cannot be changed after it is assigned. Another red arrow points from the word `final` in `public void keisti(int a, int b) {` to the method signature, indicating that the method itself cannot be overridden in a subclass.

# final

```
class TestFinal {  
  
    private final int pirmas = 1;  
    private int antras;  
  
    public void keisti(int a, int b) {  
        // pirmas = a;  
        antras = b;  
        System.out.println("Pirmas: " + pirmas);  
        System.out.println("Antras: " + antras);  
    }  
}
```

# final

```
class TestFinal {  
  
    private final int pirmas;  
    private int antras;  
  
    public TestFinal() {  
        pirmas = 1;  
    }  
  
    public void keisti(int a, int b) {  
        // pirmas = a;  
        antras = b;  
        System.out.println("Pirmas: " + pirmas);  
        System.out.println("Antras: " + antras);  
    }  
}
```

# final

```
class TestFinal {  
  
    private final int pirmas;  
    private int antras;  
  
    public TestFinal(int a) {  
        pirmas = a;  
    }  
  
    public void keisti(int a, int b) {  
        // pirmas = a;  
        antras = b;  
        System.out.println("Pirmas: " + pirmas);  
        System.out.println("Antras: " + antras);  
    }  
}
```

# final

```
class TestFinal {  
    private final int pirmas;  
    private int antras;  
    public TestFinal(int a) {  
        pirmas = a;  
    }  
    public TestFinal() {  
        this(1);  
    }  
    public void keisti(int a, int b) {  
        // pirmas = a;  
        antras = b;  
        System.out.println("Pirmas: " + pirmas);  
        System.out.println("Antras: " + antras);  
    }  
}
```

# static final Uzduotis2

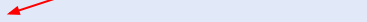
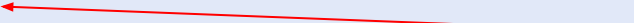
---

Sukurti kalse **StaticTestFinal** su dviem **static** kintamaisiais. Klasė taip pat turi tūrėti metoda **priskirk** kuriame kintamiesiams priskiriamos perdotos reikšmės ir išspausdina objekto kintamųjų reikšmes.

1. Main metode sukurti keleta **StaticTestFinal** objektų ir jiems iškviesti metoda **priskirk**
2. Paskelbti vieną iš kintamųjų **final**.



# static final

```
class StaticTestFinal {  
    private static final int pirmas;   
    private static int antras;  
    public StaticTestFinal(int a) {  
        pirmas = a;   
    }  
    public StaticTestFinal() {  
        this(1);  
    }  
    public void keisti(int a, int b) {  
        // pirmas = a;  
        antras = b;  
        System.out.println("Pirmas: " + pirmas);  
        System.out.println("Antras: " + antras);  
    }  
}
```

# static final

```
class StaticTestFinal {  
    private static final int PIRMAS = 1;  
    private static int antras;  
  
    public StaticTestFinal(int a) {  
        // pirmas = a;  
    }  
    public StaticTestFinal() {  
        this(1);  
    }  
    public void keisti(int a, int b) {  
        // pirmas = a;  
        antras = b;  
        System.out.println("Pirmas: " + PIRMAS);  
        System.out.println("Antras: " + antras);  
    }  
}
```

# final kintamasis (pvz2)

```
class P {  
    int v = 10;  
}
```

```
class D {  
    final P p1 = new P();  
    final P p2 = new P();  
}
```

```
public static void main(String[] args) {  
    D d = new D();  
    // negalima  
    // d.p1=new P(); d.p2=d.p1;  
  
    // galima  
    d.p1.v = 11;  
}
```

Jei kintamasis ne primityviojo, o objektinio tipo, negalima jam priskirti kitą objektą, tačiau patį objektą modifikuoti galima

# static final kintamasis

---

Raktų pora **static final** rodo, kad tai konstanta visiems klasės objektams

```
static final float MANO_PI = 3.14f;
```

# “Final” Uzduotis3

---

Užduotis:

Metodas/Funcija apskaičiuojanti apskritimo perimetrą (parametras `float` spindulys)

## Funcija apskaičiuojanti apskritimo perimetrą (parametras r)

---

```
public class Perimetras {  
    static final float MANO_PI = 3.14f;  
  
    public static void main(String[] args) {  
  
        System.out.println("Apskritimo r = 10 perimetras: "  
                             + apskritimoPerimetras(10f));  
    }  
  
    static float apskritimoPerimetras(float spindulys) {  
        return ????????????;  
    }  
}
```

## Funcija apskaičiuojanti apskritimo perimetrą (parametras r)

---

```
public class Perimetras {  
    static final float MANO_PI = 3.14f;  
  
    public static void main(String[] args) {  
  
        System.out.println("Apskritimo r = 10 perimetras: "  
                             + apskritimoPerimetras(10f));  
    }  
  
    static float apskritimoPerimetras(float spindulys) {  
        return 2 * MANO_PI * spindulys;  
    }  
}
```

# Uzduotis4

---

- Sukurit klasę **StudentData**. Klasė turi turėti laukus: studento id, vardas ir metai.
- Laukai turi būti nepasiekiami kitoms klasėms.
- Laukai: studento identifikatorius ir metai - turėtu būti apsaugoti nuo keitimo/modifikavimo (uždrausta keisti jų reikšmes)
- Klasė turi turėti:
  - metodus kurie grąžina laukų reikšmes “getters”,
  - metodą kuris išspausdina klasėje saugomą informaciją,
  - galimybę pakeisti vardą.

Main metode sukurti keleta objektų/egzempliorių. Pabandyti pasiekti laukus, pakeisti jų reikšmes, pakeisti vardą.



# StudentData

```
class StudentData {  
    private final int stuID;  
    private String stuName;  
    private final int stuAge;  
    StudentData(int num1, int num2) {  
        // Parameterized constructor  
        stuID = num1;  
        stuAge = num2;  
    }  
    StudentData(int num1, String str, int num2) {  
        // Parameterized constructor  
        stuID = num1;  
        stuName = str;  
        stuAge = num2;  
    }  
    // Getter and setter methods  
    public int getStuID() {  
        return stuID; }  
    public String getStuName() {  
        return stuName; }  
  
    public void setStuName(String stuName) {  
        this.stuName = stuName; }  
  
    public int getStuAge() {  
        return stuAge; }  
}
```

## TestOverloading

```
class TestOverloading {  
    public static void main(String args[]) {  
        // This object creation would call the default constructor  
        StudentData myobj = new StudentData(1, 25);  
        myobj.setStuName("Laura Ytè");  
        System.out.println("Student Name is: " + myobj.getStuName());  
        System.out.println("Student Age is: " + myobj.getStuAge());  
        System.out.println("Student ID is: " + myobj.getStuID());  
  
        myobj.setStuName("Laura lené");  
        System.out.println("Student Name is: " + myobj.getStuName());  
        System.out.println("Student Age is: " + myobj.getStuAge());  
        System.out.println("Student ID is: " + myobj.getStuID());  
  
        StudentData myobj2 = new StudentData(2, "Mindaugas Karpinskas", 26);  
        System.out.println("Student Name is: " + myobj2.getStuName());  
        System.out.println("Student Age is: " + myobj2.getStuAge());  
        System.out.println("Student ID is: " + myobj2.getStuID());  
    }  
}
```

1th object

2th object

# *Iš projekto: Uzduotis Finansai \*\*\*\**

---

Pajamulrasas[] pajamos = new Pajamulrasas[100];

Islaidulrasas[] islaidos = new Islaidulrasas[100];

Panaudojant **final**.

# Uzduotis5

---

Reikalingos dvi papildomos klasės: Mašina, KūroBakas;

Klasė mašina turi lauką **kuroBakas**. O KūroBakas turi lauką likutis ir metoda papildyti.

Klasė mašina turi metodą važiuoti(); - jį iškvietus kūro likutis sumažėja 10 (jei yra kūro).

Patestuoti main metode kaip viskas veikia.

Kurie lauka galētu būti **final**?

# Klausimai

---

?

# final metodas

---

- Jei metodas yra **final**, jo negalima perrašyti paveldėtose klasėse
- Privatūs metodai ir taip yra final net nerašant raktinio žodžio
- Jei final parašysime prieklases, tai iš jos negalima paveldėti

```
class A {  
    final int metodas() {  
        return 5;  
    }  
}  
class B extends A {  
    // perrašymas negalimas  
    int metodas() {  
        return 6;  
    }  
}
```

