

# Klase II

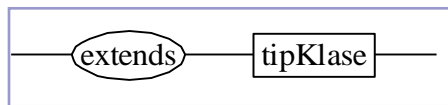
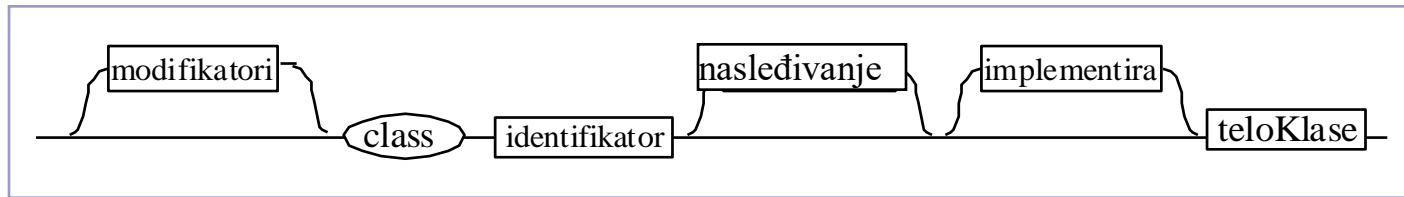
## - inicijalizatori i ugnježdene klase -

### Objektno-orjentisano programiranje 1

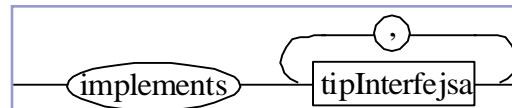


# Deklaracija klase

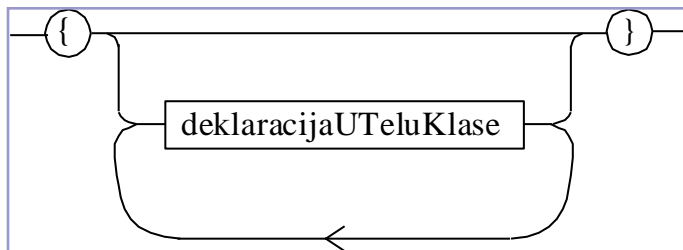
- Deklaracija klase koja nije ugnježdjena može započeti nekim od modifikatora: **public**, **abstract**, **final**, **strictfp**
  - **strictfp – strict floating point** (reprezentacija i operacije sa realnim brojevima po IEEE 754 standardu, ne koriste se mašinski-specifični *extended* formati za međurezultate)



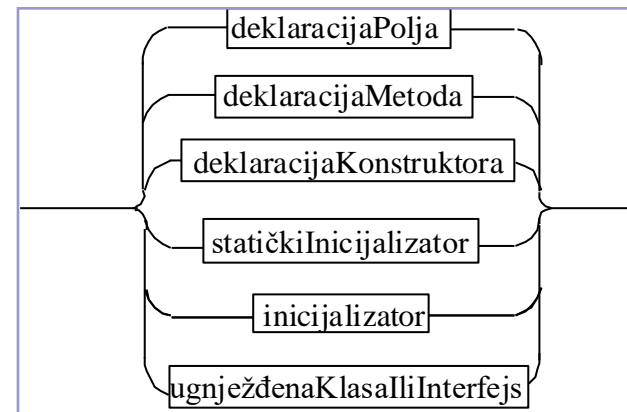
*nasleđivanje*



*implementira*



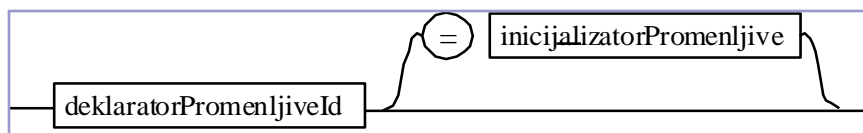
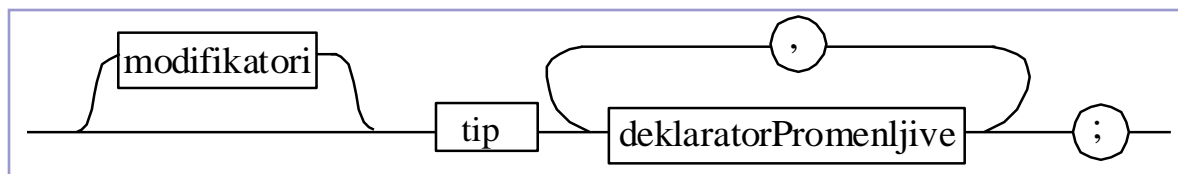
*teloKlase*



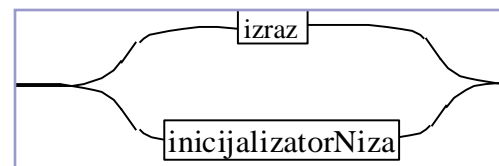
*deklaracijaUTeluKlase*

# Deklaracija klase – polja

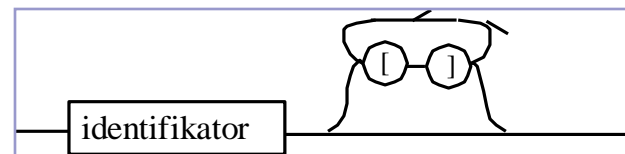
- Za deklaraciju polja mogu biti upotrebljeni modifikatori: **public**, **protected**, **private**, **final**, **static**, **transient**, **volatile**
  - **transient** – polje koje se ne koristi pri serijalizaciji i deserijalizaciji objekata date klase
  - **volatile** – sinhronizovano polje, atomičke operacije (višenitne aplikacije)



*deklaratorPromenljive*



*inicijalizatorPromenljive*



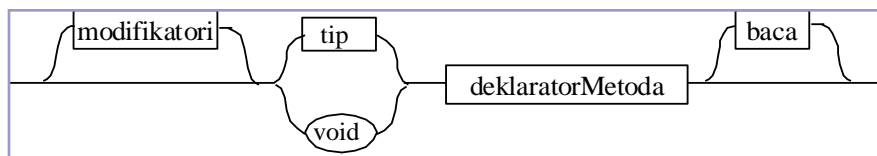
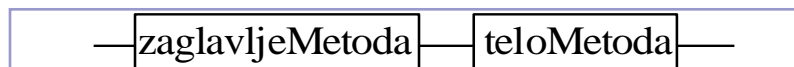
*deklaratorPromenljiveId*

## Primeri

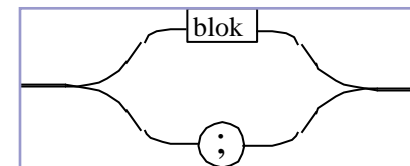
```
protected final int x = 10, z = 9;  
private double[] niz, matrica[];  
volatile public String boja = "zeleno";  
MojaKlasa mk1, mk2 = new MojaKlasa();
```

# Deklaracija klase – metode

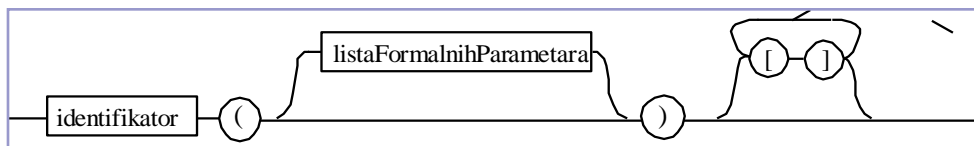
- Zaglavlje metoda može početi navođenjem jednog ili više modifikatora: **public**, **protected**, **private**, **abstract**, **final**, **static**, **synchronized**, **native**, **strictfp**
  - **synchronized** – sinhronizovana metoda (višenitne aplikacije)
  - **strictfp** – striktne floating point operacije po IEEE 754 standardu
  - **native** – apstraktan metod implementiran u nekom drugom jeziku u nekoj od biblioteka koje se dinamički učitavaju



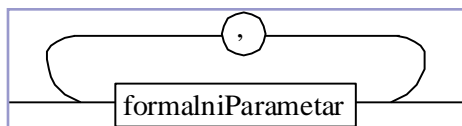
**zaglavljeMetoda**



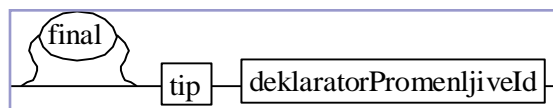
**teloMetoda**



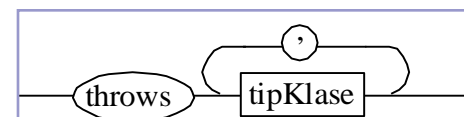
**deklaratorMetoda**



**listaFormalnihParametara**

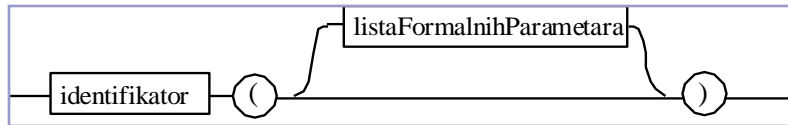


**formalniParametar**

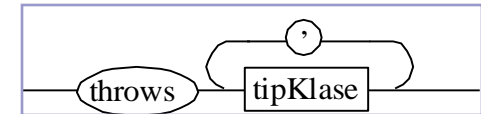


**baca**

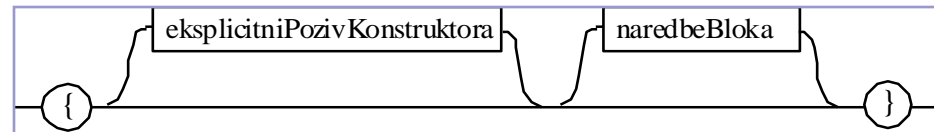
# Deklaracija klase – konstruktori



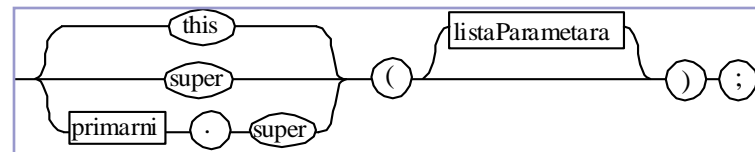
**deklaratorKonstruktora**



**baca**



**teloKonstruktora**



**eksplicitniPozivKonstruktora**

- Eksplicitni poziv konstruktora mora biti ili prva naredba ili kompajler dodaje super poziv bez argumenata
- Primarni izraz kod eksplicitnog poziva konstruktora se koristi kada klasa nasleđuje neku nestatičku ugnježdenu klasu

# Inicijalizacija statičkih atributa klase

- Statički atributi klase se mogu inicijalizovati
  - direktno prilikom deklaracije atributa
  - u statičkom inicijalizatoru (može ih biti više)
- Statički inicijalizator je oblika: **static** blokNaredbi

```
public class Covek {  
    private String id;  
    public Covek(String id) { this.id = id; }  
}
```

```
public class Foo {  
    private static double dvaPi = 2.0 * Math.PI;  
    private static double cexp = Math.Log10(3.4) + dvaPi;  
    private static Covek[] ljudi;  
  
    static {  
        ljudi = new Covek[10];  
        for (int i = 0; i < ljudi.length; i++)  
            ljudi[i] = new Covek("Covek " + i);  
    }  
}
```

# Inicijalizacija statičkih atributa klase

- **Prilikom direktne inicijalizacije statičkog atributa i u statičkom inicijalizatoru možemo koristiti samo statička polja i metode**

```
public class Bar {  
    private int x = 5, y = 3;  
  
    // ne prolazi kompajliranje!  
    private static int z = x + 1;  
  
    // ne prolazi kompajliranje!  
    private static int w = m() + 2;  
  
    // ne prolazi kompajliranje!  
    static {  
        // y i m nisu statički elementi!  
        y = 23;  
        w = m() + y;  
    }  
  
    private int m() { return 2; }  
}
```

# Statički inicijalizatori

- Statičkih inicijalizatora može biti više, izvršavaju se u redosledu u kojem su navedeni

```
public class Bar {  
    private static int x, y;  
  
    static {  
        System.out.println("Inicijalizujem x");  
        x = 1;  
    }  
  
    static {  
        System.out.println("Inicijalizujem y");  
        y = 2;  
    }  
  
    public static void main(String[] args) {  
        System.out.println(x + ", " + y);  
    }  
}
```



- **Statički inicijalizatori se izvršavaju tačno jednom (pri prvom referenciranju klase, kada se *class* fajl učitava u memoriju JVM)**

```
public class Bar {  
    private static int x;  
    private int y;  
  
    public Bar(int y) {  
        System.out.println("Konstruktor");  
        this.y = y;  
    }  
  
    static {  
        System.out.println("Inicijalizujem x");  
        x = 1;  
    }  
  
    public int getY() { return y; }  
  
    public static void main(String[] args) {  
        for (int i = 0; i < 3; i++) {  
            Bar b = new Bar(i);  
            System.out.println(b.getY());  
        }  
    }  
}
```

## Output

```
Inicijalizujem x  
Konstruktor  
0  
Konstruktor  
1  
Konstruktor  
2
```

- **Nestatički atributi klase se mogu inicijalizovati direktno, u konstruktoru i u nestatičkim inicijalizatorima**
- Nestatički inicijalizatori su blok naredbe bez zaglavlja

```
public class FooBaz {  
    private int x, y, z;  
  
    {  
        System.out.println("Inicijalizator 1");  
        x = 10;  
    }  
  
    public FooBaz() {  
        System.out.println("Konstruktor");  
        y = 20;  
    }  
  
    {  
        System.out.println("Inicijalizator 2");  
        z = 30;  
    }  
  
    public static void main(String[] args) {  
        new FooBaz();  
    }  
}
```

## Output

Inicijalizator 1  
Inicijalizator 2  
Konstruktor

# Inicijalizatori – rezime

- **Statički inicijalizatori služe sa inicijalizaciju kompleksnih statičkih atributa klase**
- **Sa nestatičkim inicijalizatorima ne treba preterivati**
- **Pristup delovima klase**
  - Statički inicijalizator može pristupati samo statičkim elementima klase
  - Nestatički inicijalizator može pristupati i nestatičkim i statičkim elementima klase
- **Izvršavanje**
  - Statički inicijalizator se izvršava samo jednom, prvi prvom referenciranju klase kada se ona učitava u memoriju JVM
  - Nestatički inicijalizator se izvršava pri svakom instanciranju klase
- **Inicijalizatori se izvršavaju u redosledu u kom su navedeni**
- **Nestatički inicijalizatori se izvršavaju pre konstruktora klase, a nakon što se završi izvršavanje konstruktora iz bazne klase**

# Ugnježdene klase i tipovi

- **Ugnježdena klasa** – klasa definisana unutar neke druge klase
  - imamo potrebu za pomoćnom klasom koja se ne koristi van te klase
    - povećava se enkapsulacija i čitljivost koda
  - kada je klasi prirodno mesto unutar neke druge klase
    - npr. imamo klase koje definišu različite vrste kontejnera, tip elementa za dati kontejner definišemo unutar odgovarajuće klase
- **Ugnježdene statičke i ugnježdene nestatičke (unutrašnje) klase**
  - razlika se ogleda u instanciranju van spoljašnje klase i mogućnostima pristupa elementima spoljašnje klase
- Drugi referencijalni tipovi (interfejsi i nabrojivi tipovi) se takođe mogu definisati unutar neke klase i **implicitno su statički ugnježdjeni tipovi**
- **Dve specijalne vrste unutrašnjih (ugnježdenih nestatičkih) klasa**
  - **Lokalna klasa** – klasa deklarisanu u nekom bloku naredbi
  - **Anonimna klasa** – *singleton* klasa bez imena koja se definiše prilikom instanciranja *singleton* objekta

# Ugnježdene klase i tipovi

```
public class FooBar {  
    // ugnježdena nestatička klasa  
    public class Foo {  
        ...  
    }  
  
    // ugnježdena statička klasa  
    public static class Bar {  
        ...  
    }  
  
    // ugnježdjeni interfejs (implicitno static)  
    public interface Baz {  
        ...  
    }  
  
    // ugnježdjeni nabrojiv tip (implicitno static)  
    public enum Fuz {  
        ...  
    }  
}
```

# Ugnježdene statičke klase

- **Statički elementi klase su nezavisni od instanci klase**  
→ **statičke ugnježdene klase se mogu instancirati nezavisno od instanci spoljne klase**
- Statičku ugnježdenu klasu van spoljne klase referenciramo njenim punim (kanoničkim) imenom
  - **ako je B statička ugnježdena klasa definisana u klasi A tada je njeno puno (kanoničko) ime A.B**

```
public class Spoljasnja {  
    public static class Ugnjezdena {  
        public Ugnjezdena(int x) {  
            ...  
        }  
    }  
  
    private Ugnjezdena ug = new Ugnjezdena(42);  
}  
  
public class FooBar {  
    private Spoljasnja.Ugnjezdena ug = new Spoljasnja.Ugnjezdena(24);  
}
```

# Puno i relativno ime ugnježdene statičke klase

```
public class A {  
    public static class B {  
        public static class C {  
            public static class D {  
                public static class E {}  
  
                private E e1 = new E();  
                private A.B.C.D.E e2 = new A.B.C.D.E();  
            }  
  
            private D.E e = new D.E();  
        }  
  
        private C.D.E e = new C.D.E();  
    }  
  
    private B.C.D.E e = new B.C.D.E();  
}  
  
public class FooBar {  
    private A a = new A();  
    private A.B b = new A.B();  
    private A.B.C c = new A.B.C();  
    private A.B.C.D d = new A.B.C.D();  
    private A.B.C.D.E e = new A.B.C.D.E();  
}
```

# Vidljivost ugnježdenih klasa i tipova

- Ako je ugnježdena statička klasa **javna** tada je ona vidljiva onoliko koliko je vidljiva njena spoljašnja klasa
- Ako je ugnježdena statička klasa **privatna** tada ona nije vidljiva van spoljašnje klase
- Ako je ugnježdena statička klasa deklarirana sa modifikatorom ***protected*** tada je ona vidljiva
  - U paketu **najviše spoljašnje klase** ako su sve njene spoljašnje klase javne ili deklarirane bez modifikatora pristupa
  - U klasama izvedenim iz spoljašnje klase
- **Ista pravila važe i za ugnježdene nestatičke klase, ugnježdene interfejse i ugnježdene nabrojive tipove**



# Ugnježdene statičke klase

- **Ugnježdene statičke klase mogu pristupati samo statičkim elementima spoljašnje klase (uključiv i privatne)**

```
public class FooBar {  
    private int x;  
    private static int y;  
  
    public static class Foo {  
        public void m() {  
            // ok  
            y = 42;  
  
            // ne prolazi kompajliranje  
            // x = 42;  
        }  
    }  
}
```

# Unutrašnje (ugnježdene nestatičke) klase

- **Instance unutrašnje klase su vezane za instance spoljašnje klase**
  - Unutrašnje klase se van spoljašnje klase instanciraju **pozivajući operator new nad objektom spoljašnje klase**

```
public class Outer {  
    public class Inner {  
        public void hello(String x) { S.o.p(x); }  
    }  
  
    public void m() {  
        Inner i = new Inner();  
        i.hello("Outer.m()");  
    }  
}
```

```
public class OuterInner {  
    public static void main(String[] args) {  
        Outer o = new Outer();  
        o.m();  
        Outer.Inner i = o.new Inner();  
        i.hello("OuterInner.main()");  
    }  
}
```

```
public class SA {  
    public static class SB {  
        public static class SC {}  
    }  
}
```

```
public class A {  
    public class B {  
        public class C {}  
    }  
}
```

```
public class FooBar {  
    public void m() {  
        SA.SB sb = new SA.SB();  
        A.B    b = new A().new B();  
  
        SA.SB.SC cs = new SA.SB.SC();  
        A.B.C    c2 = new A().new B().new C();  
    }  
}
```

# Unutrašnje (ugnježdene nestatičke) klase

- **Instance unutrašnje klase su vezane za instance spoljašnje klase**
  - Unutrašnje klase mogu pristupati svim elementima spoljašnje klase (uključiv i privatne)
  - **Podsetnik:** ugnježdene statičke samo statičkim elementima spoljašnje

```
public class A {  
    private int x = 10;  
    private static int y = 20;  
  
    public void m() {}  
  
    public class B {  
        private int z = x + y + 10;  
  
        public void meth() {  
            m();  
        }  
    }  
}
```

# Unutrašnje (ugnježdene nestatičke) klase

- Unutrašnja klasa može deklarirati atribut/metod istog imena kao atribut/metod iz spoljašnje klase (**shadowing**)

- “Zasjenjenom” identifikatoru iz spoljašnje klase možemo pristupiti koristeći **kvalifikovani this izraz** oblika **ImeKlase.this**
- **C.this** je referenca na attribute/metode spoljašnje klase **C**

```
public class A {  
    private int x = 10;  
  
    public class B {  
        private int x = 20;  
  
        public class C {  
            private int x = A.this.x + B.this.x;  
        }  
    }  
}
```

- **Kvalifikovani super izraz C.super** je referenca na attribute/metode bazne klase spoljašnje klase **C**

# Unutrašnje (ugnježdene nestatičke) klase

- U unutrašnjoj klasi nije moguće definisati statički metod
- U unutrašnjoj klasi nije moguće definisati statičko polje osim ukoliko to nije statičko i finalno polje (konstanta)
  - Kod ugnjeđenih statičkih klasa nemamo ova ograničenja

```
public class A {  
    public class B {  
        // ok, prolazi kompajliranje  
        private static final int x = 4;  
  
        // ne prolazi kompajliranje  
        // private static int y = 3;  
  
        // ne prolazi kompajliranje  
        // public static void m() {}  
    }  
}
```

```
public class ListaBrojeva {  
    private class Cvor {  
        int info;  
        Cvor veza;  
    }  
}
```

```
// nema smisla da objekte klase Cvor instanciram van klase ListaBrojeva  
// jedan cvor ne moze biti u vise lista (tacno jedan veza pokazivac)  
// Klase ListaBrojeva i Cvor su u relaciji KOMPOZICIJE
```

```
    public void dodaj(int broj) {  
        // ovde radim new Cvor(...)  
    }  
}
```

```
public class RadnaOrganizacija {  
    public static class Radnik { ... }  
    private ArrayList<Radnik> listaRadnika;
```

```
// ima smisla da objekte klase Radnik instanciram van spoljašnje klase  
// jedan Radnik moze biti u vise radnih organizacija (AGREGACIJA)
```

```
    public void dodajRadnik(Radnik r) {  
        // new RadnaOrganizacija.Radnik(...) van klase RadnaOrganizacija  
    }  
}
```

# Nasleđivanje ugnježđenih klasa

- **Ugnježdene klase (i statičke i nestatičke) se mogu nasleđivati**
  - U praksi ih uglavnom ne nasleđujemo, a gotovo nikad van spoljašnje klase
- Nasleđivanje ugnježdene statičke klase se realizuje kao nasleđivanje običnih klasa (koristimo puno ime ako je nasleđivanje van spoljašnje klase)
- **Kod nasleđivanja unutrašnje klase van spoljašnje klase moramo pozvati njen konstruktor nad vezanim objektom spoljašnje klase**

```
public class A {  
    public A(int x) {}  
  
    public class B {  
        public B(char c) {}  
    }  
}  
  
class C extends A.B {  
    public C(A a, char c) {  
        a.super(c);  
    }  
}
```



# Lokalne klase

- **Klase definisane u bilo kom bloku programskom koda**
- Mogu se instancirati u onom bloku u kom su definisane

```
public class Foo {  
    public void method() {  
        class Loc1 { ... }  
  
        for (int i = 0; i < 10; i++) {  
            class Loc2 { ... }  
  
            if (i % 3 == 0) {  
                class Loc3 { ... }  
            }  
        }  
  
        // ne prolazi kompajliranje!  
        // Loc2 l2 = new Loc2();  
    }  
}
```

# Lokalne klase

```
public int saberi(String prostIzraz) {  
    int plusIndex = prostIzraz.indexOf("+");  
    if (plusIndex == -1)  
        throw new IllegalArgumentException("Izraz nije ok");  
    String sab1 = prostIzraz.substring(0, plusIndex).trim();  
    String sab2 = prostIzraz.substring(plusIndex + 1).trim();  
  
    class Par {  
        private int prvi, drugi;  
  
        public Par(String prvi, String drugi) {  
            this.prvi = Integer.parseInt(prvi);  
            this.drugi = Integer.parseInt(drugi);  
        }  
  
        public int saberi() { return prvi + drugi; }  
    }  
  
    Par p = new Par(sab1, sab2);  
    return p.saberu();  
}
```

# Lokalne klase

- Lokalne klase mogu biti deklarirane sa modifikatorima **abstract**, **final**, **strictfp** ili **bez modifikatora**
- Lokalne klase ne mogu biti privatne, javne ili statičke i atributi/metodi lokalne klase ne mogu biti statički (osim statičkih final polja)
- Lokalne klase mogu pristupati atributima i vidljivim lokalnim varijablama koje su **final** (konstante) ili **efektivno final** (ne menjaju vrednost nakon inicijalizacije)

```
class Foo {  
    private final int x = 2;  
  
    public void method() {  
        final int y = 5;  
  
        class Loc1 {  
            private int z = x + y;  
        }  
    }  
}
```

# Anonimne klase

- *Singleton* klasa – klasa koja ima tačno jednu instancu
- Anonimna klasa – *singleton* klasa bez imena
- Anonimne klase možemo definisati u bilo kom bloku programskog koda i prilikom direktne inicijalizacije atributa
- Anonimna klasa se definiše prilikom kreiranja *singleton* objekta
  - Definicija anonimne klase se daje iza operatora `new`
- Anonimna klasa ili nasleđuje neku klasu `C` ili implementira neki interfejs `I`
  - `C c = new C(parametri-konstruktor-C) telo-anonimne-klase`
  - `I i = new I() telo-anonimne-klase`
- Anonimna klasa nema ime → ne može da ima konstruktor

# Anonimne klase

```
public abstract class Akcija {
    protected String ime;

    public Akcija(String ime) {
        this.ime = ime;
    }

    public abstract void akcija();
}

public class AnonimnaKlasa {
    public static void main(String[] args) {
        Akcija p = new Akcija("pozdrav") {
            public void akcija() {
                System.out.println("Akcija: " + ime);
            }
        };

        p.akcija();
    }
}
```