Programiranje 1

Poglavje 6: Razredi in objekti

Luka Fürst

Tabela vs. objekt

tabela	objekt
skupek spremenljivk istega tipa	skupek spremenljivk poljubnih tipov
spremenljivke imajo praviloma enak pomen	spremenljivke imajo praviloma različen pomen, še vedno pa pripadajo isti celoti
spremenljivke = elementi	spremenljivke = atributi
elementi so dostopni prek indeksov (*tabela*[*indeks*])	atributi so dostopni prek imen (*objekt*.*ime*)

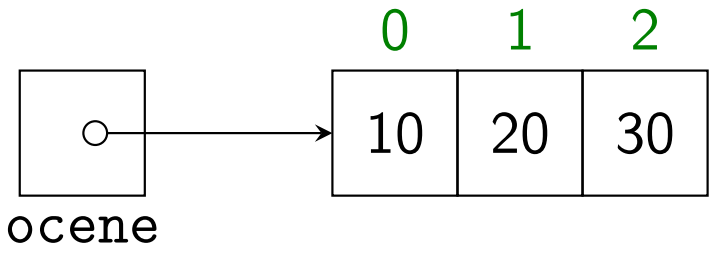
Tabela vs. objekt

tabela	objekt
tip tabele je določen s tipom elementov in številom dimenzij	tip objekta podamo v razredu (v njem navedemo tipe in imena atributov)
tabelo ustvarimo z operatorjem `new`	objekt ustvarimo z operatorjem `new`
spremenljivka tabelaričnega tipa vsebuje kazalec na tabelo	spremenljivka objektnega tipa vsebuje kazalec na objekt

Tabela vs. objekt

- izdelava tabele

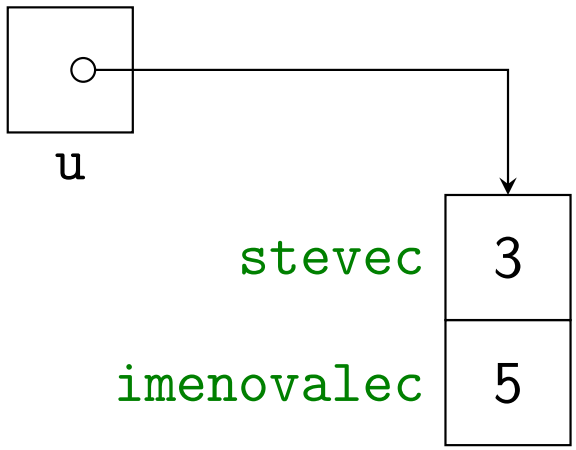
```
int[] ocene = new int[3];
ocene[0] = 10;
ocene[1] = 20;
ocene[2] = 30;
```



- izdelava objekta

```
public class Ulomek {
    int stevec;
    int imenovalec;
}
```

```
Ulomek u = new Ulomek();
u.stevec = 3;
u.imenovalec = 5;
```



Razred

- razred praviloma definiramo v ločeni datoteki
- ime datoteke = ime razreda

Ulomek.java

```
public class Ulomek {
    int stevec;
    int imenovalec;
}
```

TestUlomek.java

```
public class TestUlomek {
    public static void main(String[] args) {
        Ulomek ulomek = new Ulomek();
        ulomek.stevec = 3;
        ulomek.imenovalec = 5;
        System.out.printf("%d/%d%n",
                ulomek.stevec,
                ulomek.imenovalec);
    }
}
```

Primeri razredov

```
public class Ulomek {
    int stevec;
    int imenovalec;
}
```

```
public class KompleksnoStevilo {
    double re;
    double im;
}
```

```
public class Tocka3D {
    double x;
    double y;
    double z;
}
```

Primeri razredov

```
public class Datum {
    int dan;    // npr. 23
    int mesec;  // npr. 7
    int leto;   // npr. 2005
}
```

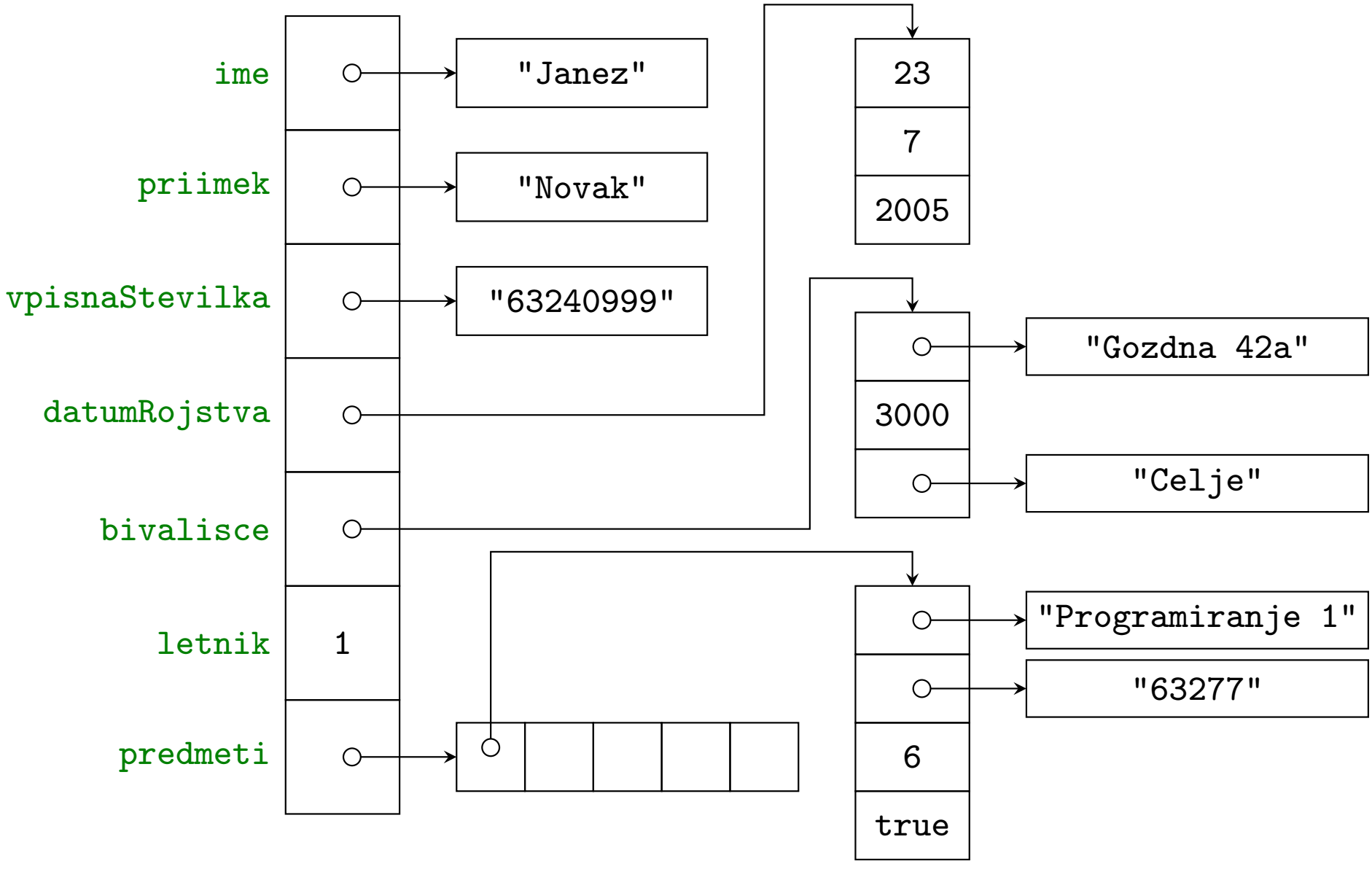
```
public class PostniNaslov {
    String ulicaInHisnaStevilka;  // npr. "Gozdna 42a"
    int postnaStevilka;           // npr. 3000
    String posta;                 // npr. "Celje"
}
```

Primeri razredov

```
public class Predmet {
    String ime;             // npr. "Programiranje 1"
    String sifra;           // npr. "63277"
    int steviloKT;          // npr. 6
    boolean jeObvezni;      // npr. true
}
```

```
public class Student {
    String ime;                 // npr. "Janez"
    String priimek;             // npr. "Novak"
    String vpisnaStevilka;      // npr. "63240999"
    Datum datumRojstva;
    PostniNaslov bivalisce;
    int letnik;                 // npr. 1
    Predmet[] predmeti;
}
```

Primer objekta tipa Student



Razred Cas

- objekt razreda Cas predstavlja nek trenutek v dnevu, izražen z uro in minuto

```
public class Cas {
    int ura;
    int minuta;
}
```

```
public class TestCas {
    public static void main(String[] args) {
        Cas trenutek = new Cas();
        trenutek.ura = 10;
        trenutek.minuta = 35;
        ...
    }
}
```

Dostopna določila

- načelo dobre prakse: atributi naj ne bodo vidni izven razreda!
  - do atributov naj ne bi neposredno dostopali
  - dostop omogočimo prek dobro definiranih metod
  - varovanje konsistentnosti objekta
  - skrivanje podrobnosti implementacije
- dostopno določilo (za element razreda R)
  - določa, od kod lahko dostopamo do elementa
  - `public`: iz kateregakoli razreda
  - `protected`: iz razredov v istem paketu in iz podrazredov razreda R
  - (*brez*): iz razredov v istem paketu
  - `private`: samo iz razreda R
- atribute praviloma deklariramo z dostopnim določilom `private`

Privatni atributi

```
public class Cas {
    private int ura;
    private int minuta;
}
```

```
public class TestCas {
    public static void main(String[] args) {
        Cas trenutek = new Cas();
        trenutek.ura = 10;      // napaka pri prevajanju!
        ...
    }
}
```

Konstruktor

- če so atributi privatni, moramo ponuditi neko drugo možnost njihovega nastavljanja
- konstruktor
  - posebna vrsta metode, ki se kliče ob izdelavi objekta
  - ime konstruktorja je enako imenu razreda
  - konstruktor nima izhodnega tipa
  - v konstruktorju praviloma nastavimo atribute na privzete ali podane vrednosti

Konstruktor

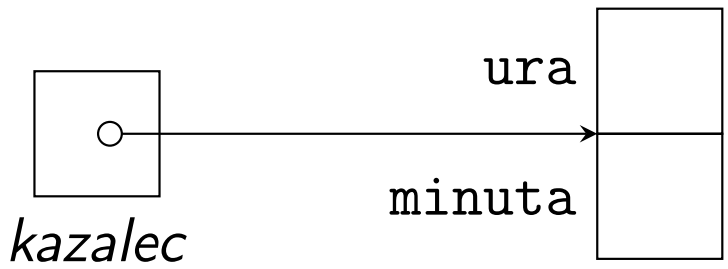
```
public class Cas {
    private int ura;
    private int minuta;

    public Cas(int h, int min) {   // brez izhodnega tipa!
        this.ura = h;
        this.minuta = min;
    }
}
```

```
public class TestCas {
    public static void main(String[] args) {
        Cas trenutek = new Cas(10, 35);
        // trenutek.ura: 10; trenutek.minuta: 35
        ...
    }
}
```

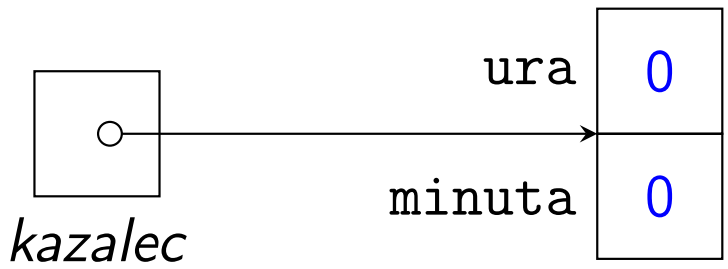
Izdelava in inicializacija objekta

- `Cas trenutek = new Cas(...);`
- operator `new` izdela nov objekt v pomnilniku



Izdelava in inicializacija objekta

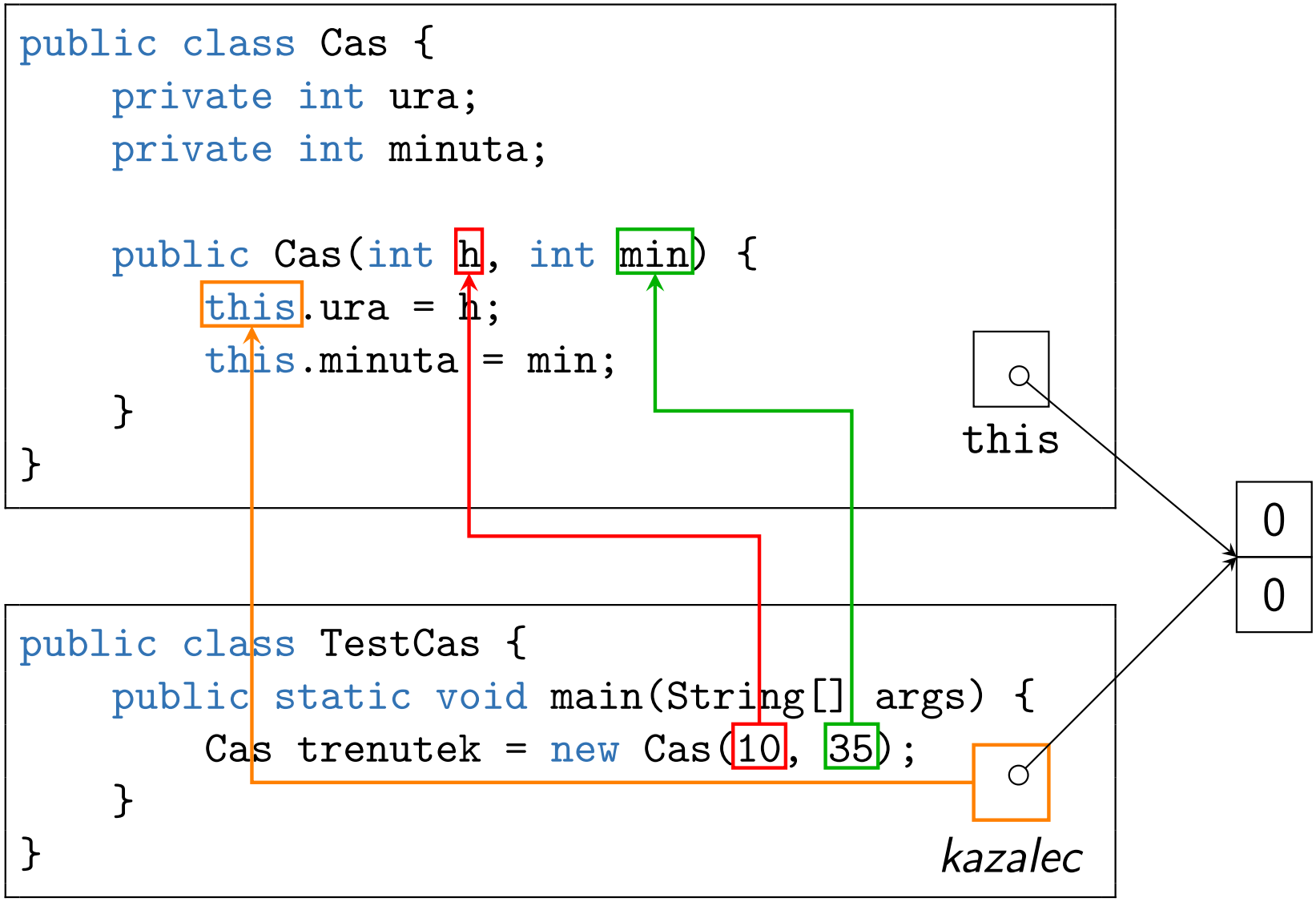
- `Cas trenutek = new Cas(...);`
- operator `new` nastavi atribute na privzete vrednosti



Izdelava in inicializacija objekta

- pokliče se konstruktor
- vrednosti argumentov se skopirajo v parametre konstruktorja
- kazalec na pravkar ustvarjeni objekt se skopira v spremenljivko `this`

Izdelava in inicializacija objekta



Izdelava in inicializacija objekta

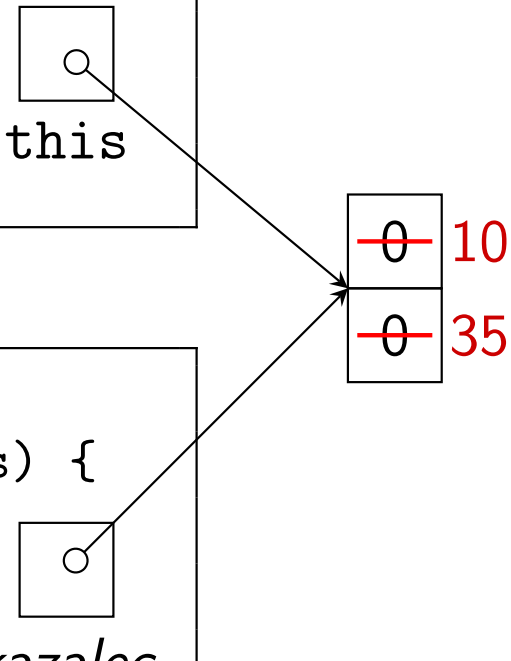
- konstruktor se izvrši
- konstruktor nastavi atributa `ura` in `minuta` pravkar ustvarjenega objekta

Izdelava in inicializacija objekta

```
public class Cas {
    private int ura;
    private int minuta;

    public Cas(int h, int min) {
        this.ura = h;    // 10
        this.minuta = min;    // 35
    }
}
```

```
public class TestCas {
    public static void main(String[] args) {
        Cas trenutek = new Cas(10, 35);
    }
}
```



Izdelava in inicializacija objekta

- rezultat izraza

  `new Razred(...)`

  je kazalec na izdelani objekt

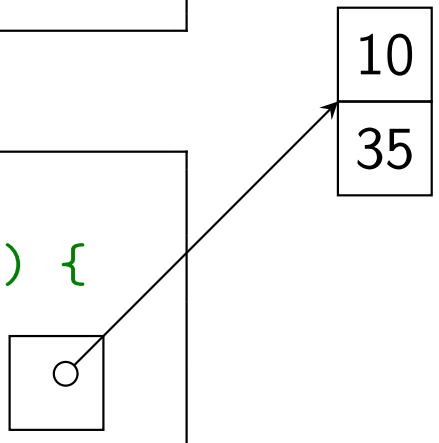
Izdelava in inicializacija objekta

```
public class Cas {
    private int ura;
    private int minuta;

    public Cas(int h, int min) {
        this.ura = h;    // 10
        this.minuta = min;    // 35
    }
}
```

```
public class TestCas {
    public static void main(String[] args) {
        Cas trenutek = new Cas(10, 35);
    }

}
```



Metode

- metode določajo, kaj lahko z objektom počnemo
- primeri
  - vračanje vrednosti posameznih atributov
  - nastavljanje posameznih atributov
  - izpis vrednosti atributov
  - vračanje niza, ki podaja vsebino objekta
  - izdelava novega objekta na podlagi obstoječega
  - primerjava objektov
  - ...

Primer metode

- napišimo metodo, ki podanemu času prišteje *h* ur in *min* minut
- metoda potrebuje
  - objekt tipa `Cas`, nad katerim naj deluje
  - število *h*
  - število *min*
- metoda mora biti definirana znotraj razreda `Cas`, saj sicer ne more dostopati do atributov
- prva ideja

```
public static void pristejStatic(
                Cas cas, int h, int min)
```

Prva ideja: statična metoda

- definicija (razred `Cas`)

```
public static void pristejStatic(Cas cas, int h, int min) {
    int noviCas = 60 * (cas.ura + h) + (cas.minuta + min);
    noviCas = (noviCas % 1440 + 1440) % 1440;
    cas.ura = noviCas / 60;
    cas.minuta = noviCas % 60;
}
```

- klic (razred `TestCas`)

```
Cas trenutek = new Cas(10, 35);  // trenutek: 10:35
Cas.pristejStatic(trenutek, 2, 50);    // trenutek: 13:25
```

- ker se metoda nahaja v drugem razredu, jo moramo poklicati kot `Razred.metoda(...)`

Boljša ideja: nestatična metoda

- `static` lahko izpustimo in metodo definiramo takole ...

```
public void pristej(int h, int min) {   // brez static
    int noviCas = 60 * (this.ura + h) + (this.minuta + min);
    noviCas = (noviCas % 1440 + 1440) % 1440;
    this.ura = noviCas / 60;
    this.minuta = noviCas % 60;
}
```

- ... in pokličemo takole ...

```
Cas trenutek = new Cas(10, 35);
trenutek.pristej(2, 50);   // objekt.metoda(...)
```

- kazalec `trenutek` v klicatelju se skopira v kazalec `this` v klicani metodi

Statična metoda

```
public class Razred {
    ...
    public static T metoda(T1 p1, T2 p2, ..., Tn pn) {
        ...
    }
}
```

- pokličemo jo kot $Razred.metoda(a_1, a_2, \ldots, a_n)$
- *Razred* lahko izpustimo, če sta klicatelj in klicana metoda v istem razredu
- argumenti se skopirajo v parametre

  $p_1 = a_1$

  $p_2 = a_2$

  $\ldots$

  $p_n = a_n$

Nestatična metoda

```
public class Razred {
    ...
    public T metoda(T1 p1, T2 p2, ..., Tn pn) {
        ...
    }
}
```

- pokličemo jo kot *objekt*.*metoda*(a_1, a_2, ..., a_n)
- *objekt* se obnaša kot dodaten argument
- argumenti se skopirajo v parametre

  `this` = *objekt*

  $p_1 = a_1$

  $p_2 = a_2$

  ...

  $p_n = a_n$

Nestatična metoda

- *objekt* v klicu *objekt*`.`*metoda*`(...)` določa razred, v katerem bo prevajalnik poiskal metodo

```
Cas trenutek = new Cas(10, 35);
trenutek.pristej(2, 50);
```

- ker je spremenljivka `trenutek` tipa `Cas`, se bo poklicala metoda `pristej` iz razreda `Cas`

Metode razreda Cas

- metodi za vračanje vrednosti atributov (»getterja«)

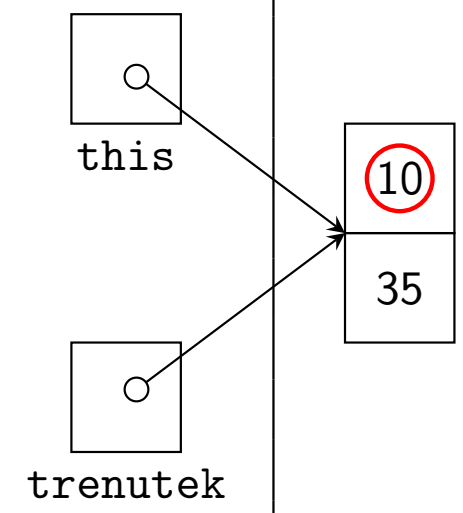
```
public class Cas {
    ...
    public int vrniUro() {
        return this.ura;
    }

    public int vrniMinuto() {
        return this.minuta;
    }
}
```

Primer klica

```
public class Cas {
    ...
    public int vrniUro() {
        return this.ura;
    }
}

public class TestCas {
    public static void main(String[] args) {
        Cas trenutek = new Cas(10, 35);
        System.out.println(trenutek.vrniUro());
    }
}
```



Metode razreda Cas

- metodi za nastavljanje vrednosti atributov (»setterja«)

```
public class Cas {
    ...
    public void nastaviUro(int h) {
        this.ura = h;
    }

    public void nastaviMinuto(int min) {
        this.minuta = min;
    }
}
```

Spremenljivost objektov

- setterji imajo potencialne stranske učinke

```
Cas a = new Cas(10, 35);
Cas b = a;
a.nastaviUro(20);
System.out.println(a.vrniUro());    // 20
System.out.println(b.vrniUro());    // 20 (!)
```

- priporočljivo je, da so objekti nespremenljivi
- če objekti niso preveliki (in če je to smiselno),
  - se izogibamo vsem metodam, ki spreminjajo atribute
  - raje ustvarimo nov objekt

Metode razreda Cas

- nespremenljiva alternativa metodi `pristej`

```
public Cas plus(int h, int min) {
    int noviCas = 60 * (this.ura + h) + (this.minuta + min);
    noviCas = (noviCas % 1440 + 1440) % 1440;
    int novaUra = noviCas / 60;
    int novaMinuta = noviCas % 60;
    return new Cas(novaUra, novaMinuta);
}
```

- primer klica

```
Cas trenutek = new Cas(10, 35);
Cas kasneje = trenutek.plus(2, 50);
// trenutek: 10:35
// kasneje: 13:25
```

Metode razreda Cas

- metoda za izpis vrednosti atributov

```
public void izpisi() {
    System.out.printf("%d:%02d",
                      this.ura, this.minuta);
}
```

- metoda za vračanje niza, ki podaja vsebino objekta

```
public String toString() {
    return String.format("%d:%02d",
                         this.ura, this.minuta);
}
```

- metoda `String.format` je podobna `System.out.printf`, le da izdelanega niza ne izpiše, ampak ga vrne

Metode razreda Cas

- metoda za preverjanje enakosti dveh objektov
- sprejme dva objekta
  - enega preko `this`
  - enega v preko parametra v oklepaju (`drugi`)

```
public boolean jeEnakKot(Cas drugi) {
    return this.ura == drugi.ura &&
           this.minuta == drugi.minuta;
}
```

- primer klica

```
Cas malica = new Cas(10, 0);
Cas kosilo = new Cas(13, 0);
System.out.println(malica.jeEnakKot(kosilo));
```

Metode razreda Cas

- razlika med časom `this` in časom `drugi` v minutah

```
public int razlikaVMin(Cas drugi) {
    return (this.ura - drugi.ura) * 60 +
           (this.minuta - drugi.minuta);
}
```

- primer klica

```
Cas malica = new Cas(10, 0);
Cas kosilo = new Cas(13, 0);
System.out.println(malica.razlikaVMin(kosilo)); // -180
```

Metode razreda Cas

- še dve metodi za primerjavo objektov tipa Cas

```
public boolean jeManjsiOd(Cas drugi) {
    return this.ura < drugi.ura ||
        (this.ura == drugi.ura &&
         this.minuta < drugi.minuta);
    // alternativa:
    // return this.razlikaVMin(drugi) < 0;
}

public boolean jeManjsiAliEnakOd(Cas drugi) {
    return this.jeManjsiOd(drugi) ||
           this.jeEnakKot(drugi);
}
```

Uporaba razreda `Cas`

- program, ki prebere . . .
  - uro in minuto prve dnevne vožnje avtobusa
  - uro in minuto zaključka dnevnih voženj
  - razmak v minutah
- . . . in izpiše vozni red avtobusa

Uporaba razreda Cas

```
public class VozniRed {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int uraZac = sc.nextInt();
        int minutaZac = sc.nextInt();
        int uraKon = sc.nextInt();
        int minutaKon = sc.nextInt();
        int interval = sc.nextInt();

        Cas cas = new Cas(uraZac, minutaZac);
        Cas casKon = new Cas(uraKon, minutaKon);
        while (cas.jeManjsiAliEnakOd(casKon)) {
            System.out.println(cas.toString());
            cas = cas.plus(0, interval);
        }
    }
}
```

Razred `Cas`

- razred `Cas` omogoča elegantno delo s časom
- namesto da bi se ukvarjali z urami in minutami, obravnavamo čas kot celoto
- z vidika uporabnika razreda `Cas` je vseeno, kako so posamezne metode implementirane
- če so atributi privatni, lahko enostavno uporabimo drugačno predstavitev objektov
  - npr. s številom minut od časa 0:00 namesto z uro in minuto
- vsebina metod se spremeni, glave in funkcionalnosti pa ostanejo enake

Statični atributi

- atributi, ki niso vezani na posamezne objekte, ampak pripadajo celotnemu razredu
- npr. atribut `zapis12`, ki določa, ali naj se časi izpisujejo v 24-urni obliki (`false`) ali 12-urni obliki z AM in PM (`true`)

Statični atributi

```
public class Cas {
    ...
    private static boolean zapis12 = false;
    ...
    public static void nastaviZapis12(boolean da) {
        zapis12 = da;   // ne this.zapis12!
    }
    ...
    public String toString() {
        if (zapis12) {   // ne this.zapis12!
            String pripona = (this.ura < 12) ? ("AM") : ("PM");
            int h = (this.ura + 11) % 12 + 1;
            return String.format("%d:%02d %s",
                                 h, this.minuta, pripona);
        }
        return String.format("%d:%02d", this.ura, this.minuta);
    }
    ...
}
```

Statični atributi

- objekti tipa `Cas` imajo še vedno samo po dva atributa
  - `ura`
  - `minuta`
- statični atribut `zapis12` je shranjen v posebnem prostoru, ki pripada razredu kot celoti

Razred `VektorInt`

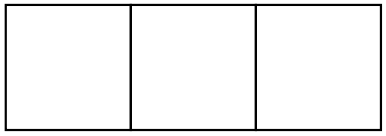
- vektor se obnaša kot »raztegljiva« tabela
- kapaciteta se po potrebi povečuje
- omogoča vstavljanje in odstranjevanje elementov na poljubnem mestu
- na podoben način deluje razred `ArrayList` v paketu `java.util`
- pomanjkljivost: razred ne preverja veljavnosti indeksov
  - razred `ArrayList` v takih primerih sproži izjemo

Atributa

- `int[] elementi`
  - tabela, ki hrani elemente
- `int stElementov`
  - dejansko število elementov v tabeli
  - `stElementov <= elementi.length`
- tabelo ustvarimo z določeno začetno kapaciteto
- ko se napolni, ustarimo novo, večjo tabelo in vanjo skopiramo elemente

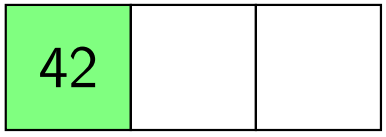
Dodajanje elementov v tabelo

- recimo, da je začetna kapaciteta enaka 3



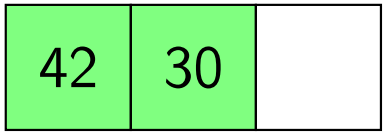
`stElementov: 0`

- dodamo element 42



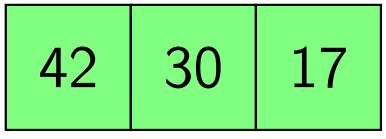
`stElementov: 1`

- dodamo element 30



`stElementov: 2`

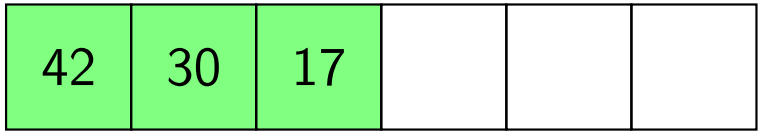
- dodamo element 17



`stElementov: 3`

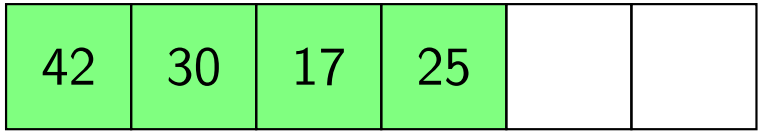
Dodajanje elementov v tabelo

- dodamo element 25
  - kapaciteta je zapolnjena
  - ustvarimo novo, večjo tabelo
  - elemente iz stare tabele skopiramo v novo



`stElementov: 3`

  - dodamo element



`stElementov: 4`

Deklaracije atributov

```
public class VektorInt {
    private static final int ZACETNA_KAPACITETA = 10;

    private int[] elementi;
    private int stElementov;
    ...
}
```

- določilo `final` označuje konstanto
  - atribut, ki ga ne moremo spreminjati
- konstanta pripada celotnemu razredu, ne posameznim objektom
  - zato jo deklariramo z določilo `static`
- `KONSTANTE_PIŠEMO_TAKOLE`

Konstruktor

- izdela tabelo elementov z dolžino `ZACETNA_KAPACITETA`

```
public VektorInt() {
    this.elementi = new int[ZACETNA_KAPACITETA];
    this.stElementov = 0;  // odveč, a poveča jasnost
}
```

Metode steviloElementov, vrni in nastavi

- metoda steviloElementov vrne dejansko število elementov

```
public int steviloElementov() {
    return this.stElementov;
}
```

- metoda vrni vrne element na podanem indeksu

```
public int vrni(int indeks) {
    return this.elementi[indeks];
}
```

- metoda nastavi nastavi element na podanem indeksu

```
public void nastavi(int indeks, int vrednost) {
    this.elementi[indeks] = vrednost;
}
```

Metoda dodaj

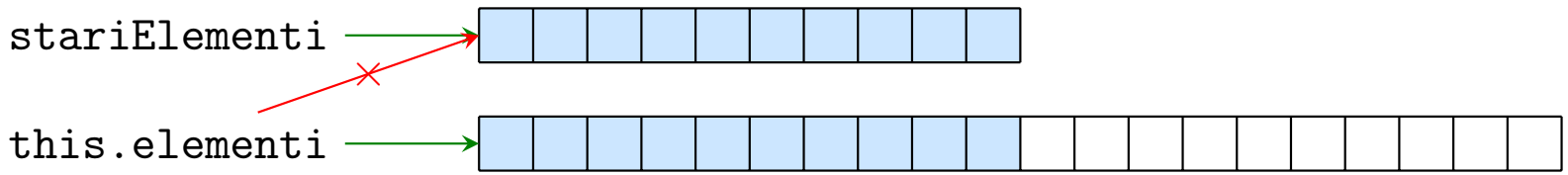
- doda element na konec tabele (na indeks `stElementov`)

```
public void dodaj(int vrednost) {
    this.poPotrebiPovecaj();
    this.elementi[this.stElementov] = vrednost;
    this.stElementov++;
}
```

Metoda poPotrebiPovecaj

- pomožna metoda poPotrebiPovecaj ustvari novo tabelo, če je trenutna pretesna

```
private void poPotrebiPovecaj() {
    if (this.stElementov >= this.elementi.length) {
        int[] stariElementi = this.elementi;
        this.elementi = new int[2 * stariElementi.length];
        for (int i = 0; i < this.stElementov; i++) {
            this.elementi[i] = stariElementi[i];
        }
    }
}
```



Metoda vstavi

- vstavi element na podani indeks
- elemente od podanega indeksa naprej prestavi za eno mesto v desno

```
public void vstavi(int indeks, int vrednost) {
    this.poPotrebiPovecaj();
    for (int i = this.stElementov - 1; i >= indeks; i--) {
        this.elementi[i + 1] = this.elementi[i];
    }
    this.elementi[indeks] = vrednost;
    this.stElementov++;
}
```

Metoda odstrani

- odstrani element na podanem indeksu
- elemente desno od podanega indeksa prestavi za eno mesto v levo

```
public void odstrani(int indeks) {
    for (int i = indeks; i < this.stElementov - 1; i++) {
        this.elementi[i] = this.elementi[i + 1];
    }
    this.stElementov--;
}
```

Metoda toString

- vrne vsebino v obliki niza $[e_0, e_1, \ldots, e_{n-1}]$

```
public String toString() {
    String str = "[";
    for (int i = 0; i < this.stElementov; i++) {
        if (i > 0) {
            str += ", ";
        }
        str += this.elementi[i];
    }
    str += "]";
    return str;
}
```

Metoda toString

- nizi so nespremenljivi
- vsaka operacija `str1 + str2` ustvari nov niz
- boljša rešitev: razred `StringBuilder`

```
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("[");
    for (int i = 0; i < this.stElementov; i++) {
        if (i > 0) {
            sb.append(", ");
        }
        sb.append(this.elementi[i]);
    }
    sb.append("]");
    return sb.toString();
}
```

Tipi elementov vektorja

- z minimalnimi popravki lahko spremenimo tip elementov vektorja
- npr. razred `VektorString` vsebuje atribut `String[] elementi`
- obstajajo tudi splošnejše rešitve
  - dedovanje
  - generiki

Uporaba vektorja

- na vhodu je podano število otrok (n), število besed izštevanke (k) in imena posameznih otrok
- otroci stojijo v ravni vrsti
- $n - 1$ krogov izštevanke
- v vsakem krogu izpade k-ti otrok (če pri štetju pridemo do konca vrste, se vrnemo na začetek)
- napišimo program, ki po vrsti izpiše imena izpadlih otrok

Uporaba vektorja

- imena otrok hranimo v objektu tipa `VektorString`

```
import java.util.Scanner;

public class Izstevanka {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int stOtrok = sc.nextInt();
        int stBesed = sc.nextInt();

        // preberi imena otrok v vektor
        VektorString otroci = new VektorString();
        for (int i = 0; i < stOtrok; i++) {
            otroci.dodaj(sc.next());
        }
        ...  // se nadaljuje
    }
}
```

Uporaba vektorja

```
public class Izstevanka {
    public static void main(String[] args) {
        ...
        // število krogov izštevanke
        int stKrogov = stOtrok - 1;

        // simuliraj izštevanko
        for (int krog = 1; krog <= stKrogov; krog++) {
            // ugotovi, kdo izpade
            int ixIzpadlega = (stBesed - 1) % stOtrok;
            System.out.println(otroci.vrni(ixIzpadlega));

            // izloči izpadlega
            otroci.odstrani(ixIzpadlega);
            stOtrok--;
        }
    }
}
```

Metode z istim imenom

- podpis metode = ime metode + seznam tipov parametrov
  - `public static void f(int a, double b)` $\rightarrow$ `f(int, double)`
  - izhodni tip ni del podpisa
- v javi lahko imamo več metod z istim imenom, a različnim podpisom
- prevajalnik bo na podlagi tipov argumentov poklical ustrezno metodo

Metode z istim imenom

```
public static void f(int a) {...}
public static void f(double a) {...}
public static void f(int a, double b) {...}
public static void f(double a, int b) {...}

public static void main(String[] args) {
    f(3);        // pokliče prvo metodo
    f(3.0);      // pokliče drugo metodo
    f(3, 4.0);   // pokliče tretjo metodo
    f(3, 4);     // napaka pri prevajanju (dvoumnost)
}
```

Konstruktorji z istim imenom

- razred lahko ima več konstruktorjev, če se ti razlikujejo po podpisu

```
public VektorInt() {
    this.elementi = new int[ZACETNA_KAPACITETA];
    this.stElementov = 0;
}

public VektorInt(int kapaciteta) {
    this.elementi = new int[kapaciteta];
    this.stElementov = 0;
}
```

Klic drugega konstruktorja istega razreda

- konstruktorji se med seboj lahko kličejo

```
public VektorInt() {
    // pokliče drugi konstruktor z argumentom
    // ZACETNA_KAPACITETA
    this(ZACETNA_KAPACITETA);
}
```

- klic `this(...)` je veljaven samo v konstruktorju
- uporabimo ga lahko samo takoj na začetku definicije konstruktorja

Privzeti konstruktor

- če ne definiramo nobenega konstruktorja, potem prevajalnik doda privzeti konstruktor
  - konstruktor brez parametrov in s praznim telesom
  - atributi obdržijo privzete vrednosti, ki jih je nastavil operator `new`
- privzeti konstruktor obstaja samo v primeru, če ne definiramo lastnega

Privzeti konstruktor

```
public class A {
}

public class B {
    public B(int n) {...}
}

public class Test {
    public static void main(String[] args) {
        A a = new A();    // v redu
        B b = new B();    // napaka pri prevajanju
    }
}
```