Tehtävät 1-3.

Luokka ei ole täydellinen. Esimerkiksi vähennyslasku, kertolasku ja jakolasku puuttuvat.
Metodien kommenteissa mainittu kohdemurtoluku tarkoittaa murtolukua, johon metodin kutsu kohdistetaan.

```
/*************************************************************
 * Murtolukujen (eli rationaalilukujen) luokka. Murtoluku on
 * muotoa osoittaja/nimittäjä missä osoittaja ja nimittäjä ovat
 * kokonaislukuja, ja nimittäjä > 0. Negatiivisessa murtoluvussa
 * osoittaja on < 0. Nolla esitetään muodossa 0/1.
 *************************************************************/

public class Murtoluku
{
    // *********************************************************
    //                  Tietokentät
    // *********************************************************

   /** Osoittaja */
   private int os;

   /** Nimittäjä */
   private int nim;


    // *************************************************************
    //                  Konstruktorit
    // *************************************************************

    /** Luodaan murtoluku, kun osoittaja a ja nimittäjä b on annettu
     *   ja muutetaan esitys standardimuotoon eli nimittäjä on positiivinen.
     *   Myös b voi olla negatiivinen, jolloin suoritetaan lavennus -1:llä.
     *   Luotu murtoluku on supistetussa muodossa, mutta a/b:n ei tarvitse olla.
     *   Alkuehto: Nimittäjän b on oltava nollasta poikkeava.
     */
   public Murtoluku(int a, int b)
   {
      if (b < 0)
         // Jos nimittäjä < 0, niin lavennetaan -1:llä
         { this.os = -a; this.nim = -b; }
      else
         { this.os = a; this.nim = b; }

      // lopuksi kohdemurtoluku supistetaan
      this.supista();
    } // Konstruktori päättyy




   // *********************************************************
   //              Havainnointimetodit
   // *********************************************************

   /** Palauttaa kohdemurtoluvun osoittajan. */
   public int getOs() { return this.os; }

   /** Palauttaa kohdemurtoluvun nimittäjän. */
   public int getNim(){ return this.nim; }


   /** Palauttaa kohdemurtoluvun esityksen merkkijonona (tulostusta varten), esim. 1/2
    *    Alussa ja lopussa välilyönnit.
    */
   public String toString() {return " " + this.os + "/" + this.nim + " ";}
```

```
// ***************************************************************************
//  Murtolukujen väliset laskutoimitukset. Metodit palauttavat aina uuden murtolukuolion ja
//  kohdemurtoluku ei muutu.
// ***************************************************************************

 /**  Murtolukujen yhteenlasku: Funktio luo ja palauttaa kohdemurtoluvun
  *    ja parametrina annetun murtoluvun toinen summan.
  */
public Murtoluku add(Murtoluku toinen)
{
    // Lavennetaan yhteenlaskettavat toistensa nimittäjillä
    // ja lasketaan lavennetut osoittajat yhteen.
    //  Luodaan uusi murtoluku eli kohdeolio ei muutu.
    int summaOs = this.os * toinen.nim + toinen.os* this.nim;
    int summaNim = this.nim * toinen.nim;
    return new Murtoluku(summaOs, summaNim);
 } // add


// ***********************************************************
//                  Muutosmetodit
// ***********************************************************

/** Kohdemurtoluvun supistaminen syt:llä, jos se on > 1. */
private void supista()
{
    if (this.os == 0)
    {
        this.nim = 1; // Nollan esitys on 0/1
    }
    else
    {
        // syt-metodi vaatii sitä, että molemmat parametrit ovat positiivisia.
        // Math.abs palauttaa parametrinsa itseisarvon.
        // Ensin määrätään  osoittajan itseisarvon ja nimittäjän suurin yhteinen tekijä.
        int supistaja = syt(Math.abs(this.os), this.nim); // vain osoittaja voi olla negatiivinen!

        if (supistaja > 1)
        {
            // huom. alla olevat jakolaskut menevät tasan, koska supistaja on syt.
            this.os = this.os / supistaja;
            this.nim = this.nim / supistaja;
        }  // if
     } // else
 } // supista
```

```
    // ************************************************************
    //                    Apumetodi
    // ************************************************************

    /** Palauttaa kokonaislukujen x ja y suurimman yhteisen tekijän.
     *   Alkuehto: x >0 ja y > 0
     */
    private static int syt(int x, int y)
    {
       while (x != y)
       {
          if (x > y)
             x = x - y;
          else
             y = y - x;
        } // while
       return x;
    } // syt
} // luokan Murtoluku loppu
```

Huom. yllä metodin syt määreenä on **static** eli se on ns. luokkametodi eli sen kutsua ei kohdisteta luokan Murtoluku olioon, vaan sitä käytetään 'perinteellisesti ilman oliota'. Sen sijaan supista on esiintymämetodi (koska static puuttuu metodin otsikkoriviltä) eli sen kutsu kohdistetaan olioon, kuten yllä konstruktorissa Murtoluku oleva kutsu `this.supista()`.

Ohjelman KoeMurtoluku suoritus:

```
Murtoluvut:  1/2  ja  2/3
Summa:  7/6
Sen nimittaja on 6
```

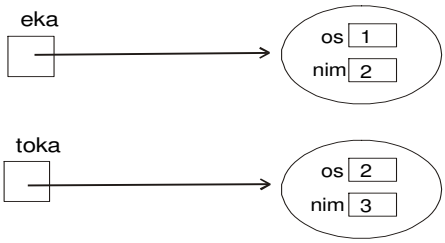
Tarkastellaan vielä kertauksena yksityiskohtaisesti main-metodin lausetta

```
Murtoluku eka = new Murtoluku(os1, nim1);
```

Tällöin tapahtuu seuraavaa (lue tarkkaan!):

- varataan tilaa luokan Murtoluku oliolle (new) ja new:n arvona on viittaus tähän olioon.
- suoritetaan luokan Murtoluku metodi (konstruktori) nimeltä Murtoluku: ensin perustyyppisen muuttujan os1 sisältö kopioidaan parametrin a arvoksi ja nim1 parametrin b arvoksi ja sen jälkeen suoritetaan metodin Murtoluku runko. if –lauseessa annetaan arvot sekä this.os:lle että this.nim:lle. Tässä this tarkoittaa kohdeolioita eli oliota, joka juuri luotiin. Sen jälkeen suoritetaan komento this.supista(); eli kohdistetaan komento supista tähän olioon.
- oikeanpuolen arvona on viittaus luotuun Murtoluku-olioon. Esitellään tyyppiä Murtoluku oleva muuttuja eka, joka saa arvokseen tämän viittauksen.

Murtolukujen eka ja toka luonnin jälkeen muistin tilaa voidaan havainnollistaa seuraavasti:



Huom1. luokan tietokenttiin (esiintymämuuttujiin) os ja nim voidaan viitata suoraan vain sen luokan sisällä, missä ne on esitelty eli siis luokassa Murtoluku (mutta ei muissa luokissa), koska niiden määre on **private**. Muissa luokissa tulee käyttää havainnointimetodeja getOs ja getNim. Myös luokassa Murtoluku voitaisiin käyttää myös havainnointimetodeja getOs ja getNim sekä kohdeolion että parametrin toinen yhteydessä eli voisimme kirjoittaa metodin add myös muodossa:

```
public Murtoluku add(Murtoluku toinen)
{
  // Lavennetaan yhteenlaskettavat toistensa nimittäjillä
  // ja lasketaan lavennetut osoittajat yhteen.
      int summaOs = this.getOs()* toinen.getNim() + toinen.getOs()*this.getNim();
      int summaNim = this.getNim() * toinen.getOs();
      return new Murtoluku(summaOs, summaNim);
 }
```

Huom2. Voisimme myös tehdä metodi tulostaMurtoluku, joka ei palauta merkkijonoa, vaan tulostaa suoraan. Sen käyttö ei olisi kuitenkaan niin joustavaa kuin mitä toString-metodiin perustuva käyttö on. Esimerkiksi jos luokassa Murtoluku on metodi

```
public void tulostaMurtoluku()
{ System.out.print(" " + this.os + "/" + this.nim + " "); }
```

ja emme olisi kirjoittaneet toString-metodia, niin testiluokassa oleva tulostuslause

```
System.out.println("Murtoluvut: " + eka + " ja " + toka);
```

pitäisi tällöin kirjoittaa muodossa:

```
System.out.print("Murtoluvut: ");
eka.tulostaMurtoluku();
System.out.print(" ja ");
toka.tulostaMurtoluku();
System.out.println();
```

Edellisestä nähdään, että metodin tulostaMurtoluku käyttö ei ole aina joustavaa ja näin ollen **käytäntönä** on kirjoittaa luokkaan tulosta-metodin lisäksi oma toString-metodi, koska tällöin myös tulostuksen saa suoritettua helpommin.

Huom3. Edellä supista on **private**-metodi, joten sitä ei voi käyttää luokan Murtoluku ulkopuolella ja ei siis esim. luokassa KoeMurtoluku. Tässä on ajatuksena, että sille ei ole tarvetta luokan ulkopuolella, koska luokan kaikki metodit tuottavat aina supistetun murtoluvun. Se voitaisiin myös varustaa määreellä public ja näin tuleekin tehdä, jos sitä halutaan käyttää luokan ulkopuolella.

Tarkastellaan lopuksi vaihtoehtoista tapaa esittää add-metodi, jossa murtoluvut annetaan parametreina. Tämä on static metodi (luokkametodi), joka ei siis käsittele kohdeoliota lainkaan, vaan on 'perinteellinen' aliohjelma.

```
/** Murtolukujen yhteenlasku: Funktio luo ja palauttaa uuden murtoluvun, joka on parametreina
 *    annettujen murtolukujen m1 ja m2 summa.
 */
   public static Murtoluku add2(Murtoluku m1, Murtoluku m2)
   {
      // Lavennetaan yhteenlaskettavat m1 ja m2 toistensa nimittäjillä
      // ja lasketaan lavennetut osoittajat yhteen.

      int summaOs = m1.os * m2.nim + m2.os* m1.nim;
      int summaNim = m1.nim * m2.nim;
      return new Murtoluku(summaOs, summaNim);
    } // add
```

Kun edellä kirjoitettiin

```
Murtoluku s = eka.add(toka);
```

niin nyt tulee kirjoittaa

```
Murtoluku s = Murtoluku.add2(eka,toka);
```

Metodi add2 ei noudata OO-periaatetta, koska metodin kutsua ei kohdisteta murtolukuun, vaan se on luokan Murtoluku luokkametodi, jolloin kumpikin murtoluku annetaan metodille parametrina. Huomaa lisäksi, että koska add2 on luokkametodi (sen määreenä on static), tulee sitä kutsua muista luokista käsin muodossa

<luokan nimi, jossa metodi on määritelty>.<metodin kutsu>.

Tämä asia on selostettu seikkaperäisesti opiskeluoppaan s. 57-58, joka tulee lukea huolellisesti.

Ensimmäisestä kutsusta `eka.add(toka)` ei näe heti, että kohdeolio (eka) ei muutu, kun taas jälkimmäisessä on selvempää että eka ja toka eivät muutu. Tässä voidaan kuitenkin pohtia sitä kuinka hyvin OO-ajattelu sopii murtolukujen maailmaan, joskin tämäkin on jossain määrin makukysymys.

Jos metodi add on olemassa, niin metodi add2 voidaan toteuttaa sen avulla ja lisäksi ne voidaan nimetä samaksi (metodien ylikuormitus):

```
public static Murtoluku add(Murtoluku m1, Murtoluku m2)
{
   return m1.add(m2);
 } // add
```

Tehtävä 4.

Nämä sijoitetaan luokkaan Murtoluku havainnointimetodien alle. Metodit perustuvat siihen tosiseikkaan, että ehto x/y<a/b on ekvivalentti ehdon x*b<y*a kanssa.

```
/** Palauttaa totuusarvon siitä onko kohdemurtoluku pienempi kuin parametri m */
public boolean less(Murtoluku m)
{
	return this.os * m.nim < this.nim * m.os;
}

 /** Palauttaa totuusarvon siitä onko kohdemurtoluku sama kuin parametri m */
 public boolean equals(Murtoluku m)
 {
	return this.os * m.nim == this.nim * m.os;
 }
```

Tässä olisi voitu suorittaa jakolaskut ja kirjoittaa ensimmäisessä:

```
return (double) this.os / (double) this.nim < (double) m.os / (double) m.nim;
```

Miksi tämä ei kuitenkaan ole hyvä ratkaisu?

Näitä metodeja voitaisiin testata esim. lauseilla:

```
System.out.println(eka.less(toka));
System.out.println(toka.less(eka));
System.out.println(eka.equals(toka));
```

Tehtävä 5-6.

Tämäkään luokka ei ole metodien suhteen täydellinen.

```
public class Kellonaika
{

// ***********************************************************
//                   Tietokentät
// ***********************************************************

/** Kellonajan tunnit 0…23 */
private int tunnit;


/** Kellonajan minuutit 0…59 */
private int minuutit;


/** Kellonajan sekunnit 0…59 */
private int sekunnit;


// ***********************************************************
//                   Konstruktorit:
// ***********************************************************

/**  Uuden kellonaikaolion luonti; aika annetaan parametreina.
 *  Alkuehto: 0 <= h < 24, 0 <= min < 60 ja 0 <= sek < 60. */
  public Kellonaika(int h, int min, int sek)

/**  Uuden kellonaikaolion 0:0:0 luonti */
public Kellonaika()


// ***************************************************************************
//                    Havainnointimetodit:
// ***************************************************************************

/** Palauttaa tunnit */
public int getTunnit()


/** Palauttaa minuutit */
public int getMinuutit()


/** Palautta sekunnit */
public int getSekunnit()


/** Kellonajan tulostus niin, että tunnit, minuutit ja sekunnit tulostuvat omille riveillään ja
 *   rivin  alussa on teksti Tunnit, Minuutit tai Sekunnit.
 */
public void tulostaKellonaika()

/** Palauttaa kellonajan String-tyyppisenä muodossa tunnit:minuutit:sekunnit; esim. 22:58:1 */
public String toString()
```

```
// ***********************************************************************
//            Muutosmetodit:
// ***********************************************************************

/** Kellonajan asetus parametrina annettuun aikaan eli olemassa olevan kellonaikaolion muutos.
 *   Alkuehto: 0 <= h < 24, 0 <= min < 60 ja 0 <= sek < 60.
 */
  public void setAika(int h, int min, int sek)

/** Kellonajan sekuntien asetus parametrina annettuun määrään sek.
 *   Alkuehto: 0 <= sek < 60.
 */
  public void setSekunnit(int sek)

/** Kellonajan minuuttien asetus parametrina annettuun määrään min.
 *   Alkuehto: 0 <= min < 60.
 */
  public void setMinuutit(int min)

/** Kellonajan tuntien asetus parametrina annettuun määrään h.
 *   Alkuehto: 0 <= h < 24.
 */
  public void setTunnit(int h)


// ***********************************************************************
//      Kellonaikojen laskutoimitukset, muunnokset ja vertailut
// ***********************************************************************

/** Muuttaa kohdeolion ajan sekunneiksi ja palauttaa sen */
public int muunnaSekunneiksi()

/** Palauttaa uuden Kellonaika-olion, joka vastaa parametrin sek arvoa.
 *   Luotu olio toteuttaa konstruktorin AE:n. */
public static Kellonaika muunnaSekunnitKellonajaksi(int sek)

/**  Parametrina annettu kellonaika lisäys lisätään kohdeaikaan. Lisäys annetaan Kellonaika-
 *    tyyppisenä oliona eli kyseessä on kellonaikojen yhteenlasku. Tunnit on aina <24 eli tarpeen
 *    mukaan 'vaihdetaan vuorokautta'
 */
public void lisääAikaMuuttamallaKohdeOliota(Kellonaika lisäys)
// Ensin lasketaan yhteen sekunnit. Jos tulos ylittää 60, niin lisätään
// minuutteihin ykkönen ja sekunneiksi jää summa-60.
// Vastaavasti menetellään minuuttien ja tuntien kohdalla


/** Luo uusi Kellonaika-olio, joka on parametrina annetun ja kohdeolion summa, ja palauta se. */
public Kellonaika lisääAika(Kellonaika lisäys)
//  Metodi lisääAika on sama kuin edellinen metodi, mutta nyt kohdeolio ei muutu, vaan metodi
//  palauttaa uuden Kellonaika-olion.


/**  Luo uusi Kelloaika-olio, joka on kohdeolion ja parametrina annetun ajan erotus, ja palauta se.
 *   Alkuehto: kohdeaika tulee olla vähintään yhtä suuri kuin parametrina annettu aika.
 */
public Kellonaika erotus(Kellonaika toinen)
// Tässä muutetaan ensin kohdeaika ja parametrina annettu aika kumpikin
// sekunneiksi, jonka jälkeen niiden erotus muunnetaan
// tunneiksi, minuuteiksi ja sekunneiksi ja lopuksi luodaan vastaava Kellonaika-olio

/** Luo sellainen uusi Kellonaika-olio, joka saadaan kun kohdeolioon lisätään parametrina annettu
sekuntimäärä sek. */
public Kellonaika lisääSekunnit(int sek)

// ja muita metodeja …

} // luokan Kellonaika loppu.
```