OBJEKTORIENTĒTĀ PROGRAMMĒŠANA

28.nodarbība

SVARĪGI PRINCIPI PROGRAMMĒŠANĀ

- Sarežģītu problēmu sadalīt vienkāršākās (un risināt pa vienai):
  - Pēc tam «lielās problēmas» risinājumu salikt no «mazo problēmu» risinājumiem kā no lego klucīšiem.
- Vienreiz izdomāt risinājumu, daudzreiz to izmantot:
  - Ja jau reiz ritenis ir izgudrots, to var pielikt gan velosipēdam, gan automašīnai, gan lidmašīnai.

PROGRAMMĒŠANAS «PARADIGMAS»

Pieeja	Raksturojums
Procedurālā programmēšana	Algoritmu dalīšana apakšprogrammās, funkcijās, metodēs.
Objektorientētā programmēšana	Datu definīciju un to apstrādājošo algoritmu iepakošana kopā (klases jēdziens). Objektu izmantošana.
Ir arī citas	Imperatīvā, deklaratīvā, funkcionālā, notikumu orientētā (u.t.t., šis saraksts attīstās un mainās).

Šīs paradigmas ne vienmēr ir pretrunā, bieži tās papildina viena otru.

«PARADIGMAS» UN PROGRAMMĒŠANAS VALODAS

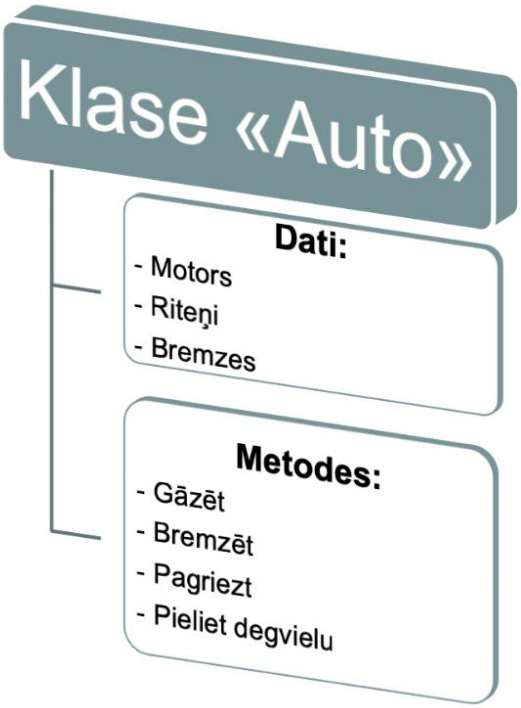
Valoda	Paradigmas
Java	«Stingri» objektorientēta: «everything is object»
C	Procedurāla
C++	Objektorientēta, bet strādā arī gandrīz viss, kas C valodā
C#	Objektorientēta
Visual Basic, Turbo Pascal, PERL, COBOL, ...	Procedurāla, bet pēdējās versijās ir iespēja lietot arī objektus

Jūs varat strādāt «objekt-orientēti» daudzās programmēšanas valodās, katra no tām šo paradigmu interpretē citādi.

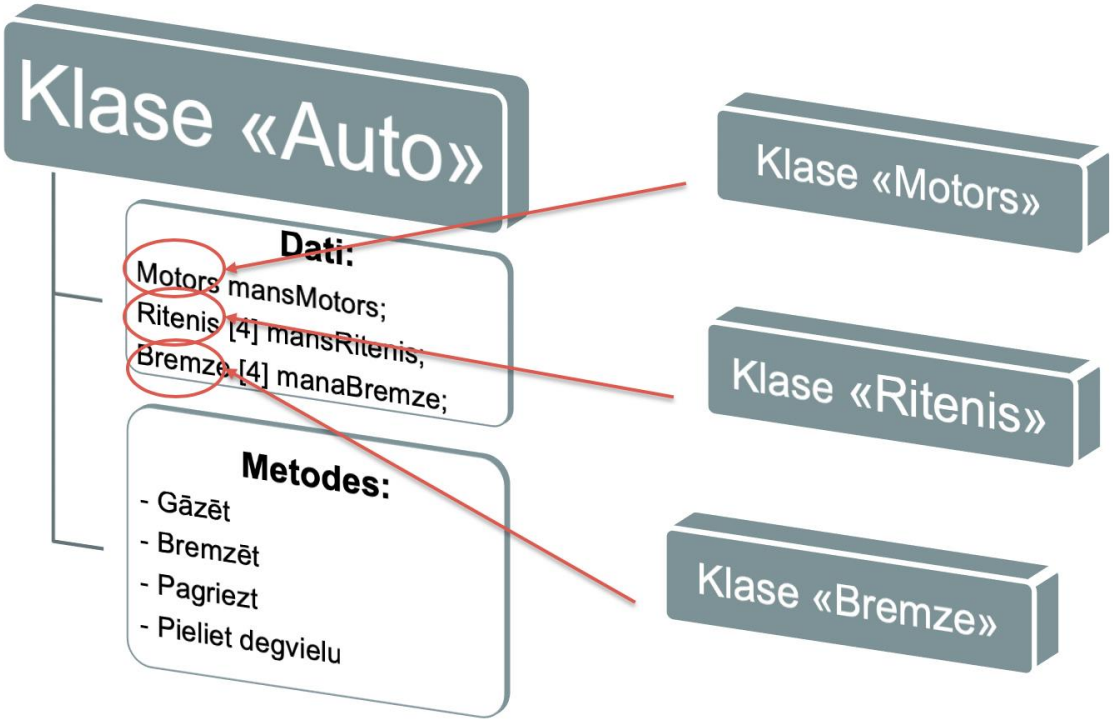
Šai lekcijā runāsim par to, kā to interpretē java.

KLASE APVIENO DATUS UN ALGORITMUS

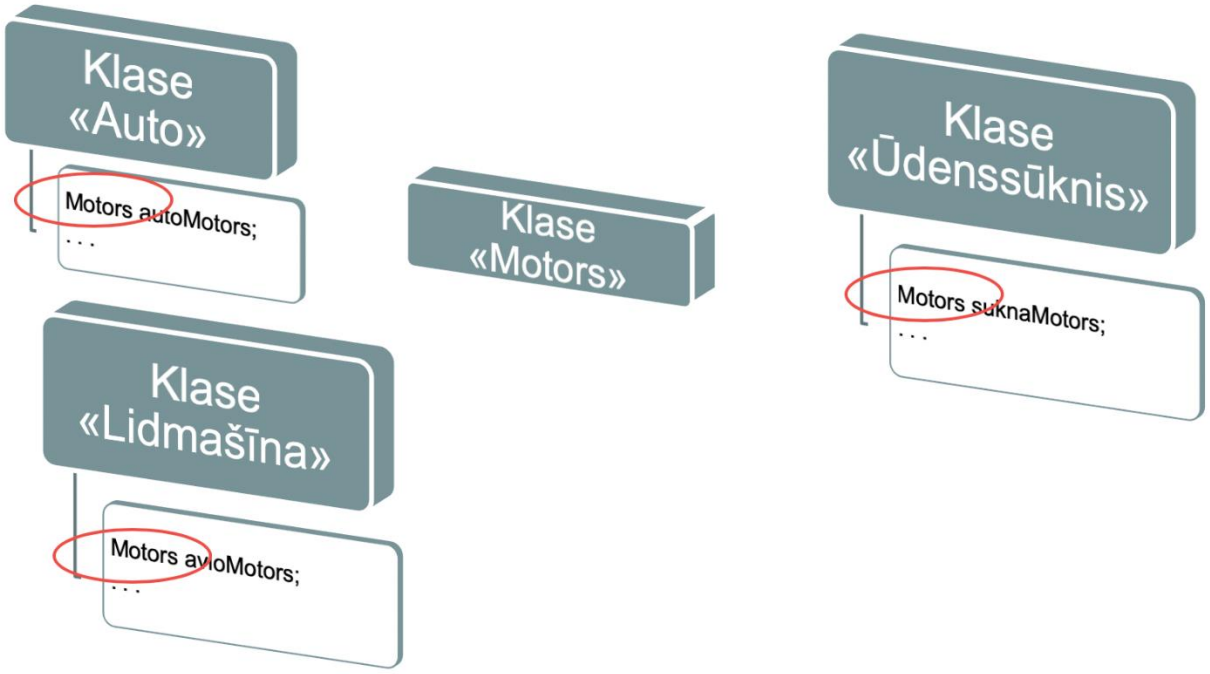
ŠĀDS PAŅĒMIENS INDUSTRIJĀ RADĀS, MODELĒJOT REĀLAS LIETAS



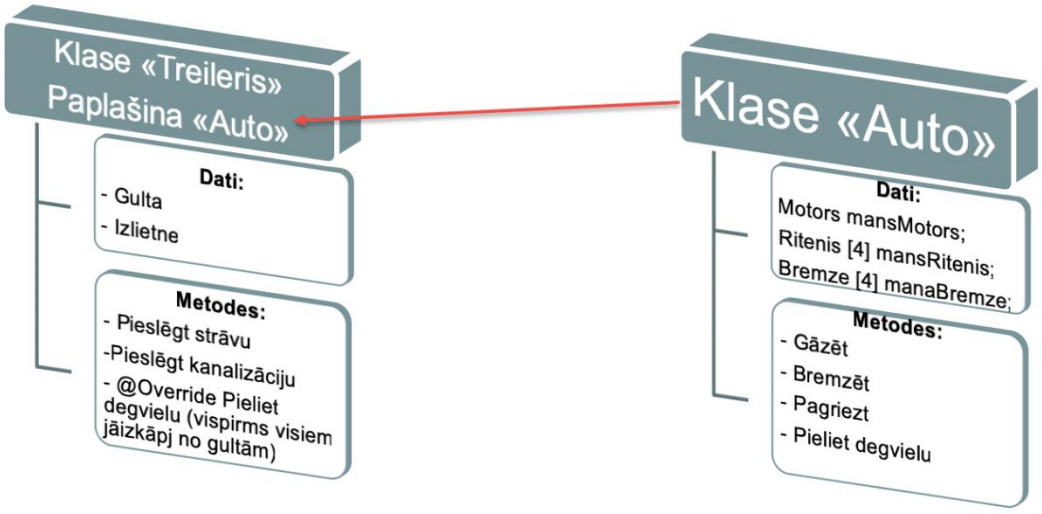
KLASES DATI TIEK BŪVĒTI NO CITĀM KLASĒM KĀ NO LEGO KLUCĪŠIEM



MĒRĶIS: IZDOMĀT VIENREIZ, LIETOT DAUDZREIZ



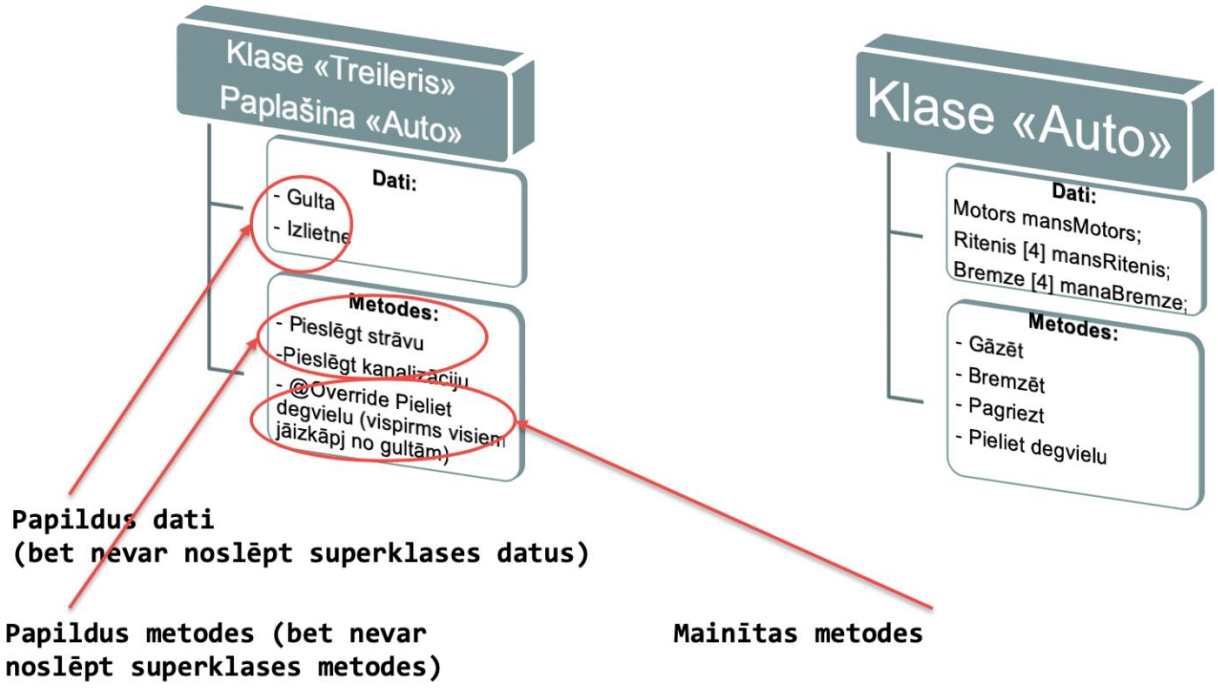
KLAŠU PAPLAŠINĀŠANA I



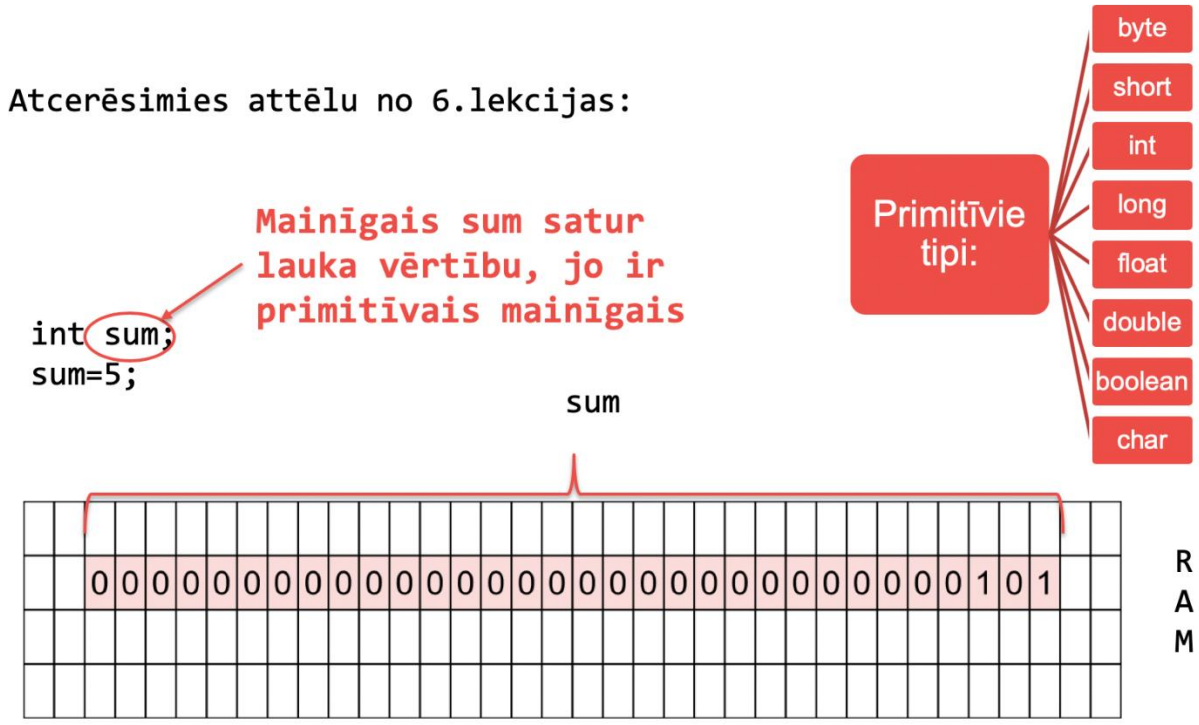

```
Class Treileris extends Auto;
```

Superklase

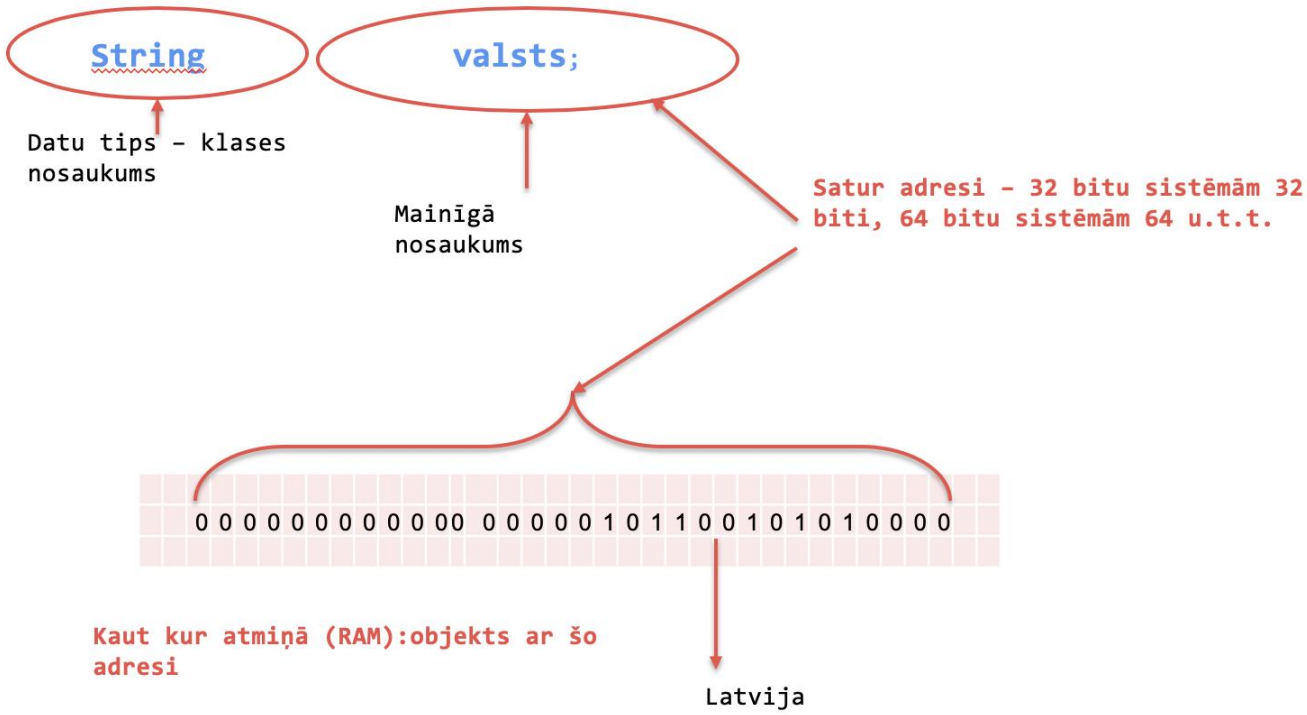
KLAŠU PAPLAŠINĀŠANA II



JAVA DATU DEKLARĀCIJAS: PRIMITĪVIE MAINĪGIE



JAVA DATU DEKLARĀCIJAS: VISI PĀRĒJIE MAINĪGIE



OBJEKTU INSTANCIĒŠANA AR OPERATORU «NEW»

```
import lv.startit.Telefons;
```
← klase

```
Telefons mansTelefons;
Telefons tavsTelefons;
Telefons mammasTelefons;
```
← Definīcijas mainīgajiem ar tipu «Telefons». Tiem satur adresi objektam ar tipu Telefons. Šobrīd tie satur **null**.

```
mansTelefons = new Telefons();
```
← Tiek radīts jauns objekts. Mainīgais `mansTelefons` tagad šī objekta adresi.

Objekta instanciēšana

```
mansTelefons.zvanit(3333333);
```
← Tagad varam sākt izmantot objekta metodes.

STATIC METODES

Izveidojot jaunu klasi, bieži redzam kaut ko šādu:

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

Redzam triku, kā iedarbināt programmu, neizveidojot pašiem tās objektu.

Static pie metodes nozīmē, ka objekts šai klasei izveidosies automātiski, kad java programma sāks izpildīties, un to nevajag instanciēt ar new operatoru.

```
int x =Integer.parseInt("9");  //nevis x.parseInt("9")
```

Statisku metodi izsauc klasei, nevis objektam.

Statiskās metodes bieži lieto aprēķiniem un palīgdarbībām, kas neizmanto objekta instances datus.

STATIC MAINĪGIE

Mēs varam nodefinēt klases mainīgo kā static:

```
static Motors mansMotors;
```

Kas no tā mainīsies?

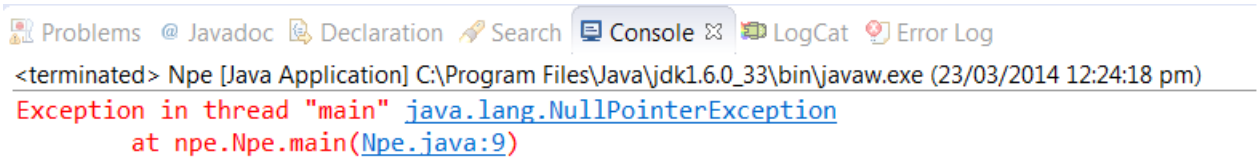
Ja Jūs izveidosiet vairākus šīs klases objektus, tad mainīgais **mansMotors** visiem objektiem rādīs uz vienu un to pašu klases **Motors** instanci. Citiem vārdiem, visiem automobiļiem būs viens un tas pats motors.

To var darīt, bet rūpīgi jāapdomā, vai tas ir tas, ko Jūs tiešām gribat.

NULL POINTER EXCEPTION I

```
public class Npe {
   static Random myRandom;
   public static void main(String[] args) {
     System.out.println("gadijuma skaitlis:" + myRandom.nextInt());
   }
}
```

Kas notiks?



NullPointerException, iepējams, ir visbiežākā kļūda, ar ko java programmētāji sastopas.

NULL POINTER EXCEPTION II

```
public class Npe {
   static Random myRandom;
   public static void main(String[] args) {
     System.out.println("gadijuma skaitlis:" + myRandom.nextInt());
   }
}
```

Kā situāciju labot?

Jāinstancē objekts, pirms to lieto!

```
public class Npe {
   static Random myRandom = new Random();
   public static void main(String[] args) {
     System.out.println("gadijuma skaitlis:" + myRandom.nextInt());
   }
}
```

MAINĪGO PĀRSLĒGŠANA UZ CITU OBJEKTU

Objektu var instanciēt mainīgā deklarācijas laikā:

```
Random myRandom = new Random();
```

Vai vēlāk:

```
Random myRandom;
```

```
myRandom = new Random();
```

Ja vajag, vēlāk var «pārslēgt» mainīgo uz citu objektu:

```
Random myRandom;
myRandom = new Random();
```

Vecais objekts tiek aizmirsts. To vēlāk satīra «Garbage Collector».

```
. . .
myRandom = new Random(33);
```

Tiek radīts jauns objekts. Mainīgais **myRandom** tagad satur šī jaunā objekta adresi.

KONSTRUKTORI

Viens no iespējamiem veidiem, kā uzstādīt objekta mainīgajiem sākuma vērtības, ir ar konstruktora palīdzību. Sk. piemēru:

```
public class Skolnieks {
   String vards;
   String klase;
   String ePasts;
      Konstruktors saucas tāpat, kā klase
   public Skolnieks(String v) {
      vards = v;
   }
}
```

Instanciējot klasi, varam rakstīt tā:

```
Skolnieks sk1 = new Skolnieks("Anna");
Skolnieks sk2 = new Skolnieks("Peteris");
```

JAVA ĪPATNĪBAS – DATU STRUKTŪRAS

C++ ir šāda iespēja definēt sarežģītas struktūras mainīgos. Līdzīgas iespējas ir lielākajā daļā programmēšanas valodu:

```
struct Employee
{
    int nID;
    int nAge;
    float fWage;
};
```

Java valodā vienīgā iespēja definēt sarežģītākas datu struktūras ir veidot jaunu klasi.

OBJEKTU PIEŠĶIRŠANA

```
Ritenis prieksejais = new Ritenis();
Ritenis aizmugurejais = prieksejais;
```

Vai Jūs domājat, ka esam nodefinējuši divus vienādus riteņus?

Nē, mūsu velosipēdam ir tikai viens ritenis un divas šī riteņa «iesaukas».

```
Ritenis prieksejais = new Ritenis();
Ritenis aizmugurejais = new Ritenis();
```

Tagad gan mūsu velosipēdam ir 2 riteņi.

Objektus nevar piešķirt vienu otram. Var piešķirt tikai to adreses.

OBJEKTU SALĪDZINĀŠANA

```
Ritenis prieksejais = new Ritenis();
Ritenis aizmugurejais = new Ritenis();
if (prieksejais == aizmugurejais) ....
```

Vai salīdzināšanas rezultāts būs true vai false? Riteņi taču ir vienādi!

Atbilde ir **false**

Objektus nevar salīdzināt, ja tas īpaši nav uzprogrammēts. Var salīdzināt tikai to adreses.

Bet vienmēr ir atļauts pašiem uzprogrammēt savu metodi equals un to izsaukt:

```
Ritenis prieksejais = new Ritenis();

Ritenis aizmugurejais = new Ritenis();

if (prieksejais.equals(aizmugurejais) ....
```