



LocalDateTime & LocalDate

LocalDateTime/LocalDate

- **LocalDateTime** – tai datos ir laiko duomenų tipas
- **LocalDate** – tai tik datos duomenų tipas
- **LocalTime** – tai tik laiko duomenų tipas

Reikalingi *import*

```
import java.time.LocalDateTime;  
import java.time.LocalDate;
```

Pavyzdžiai

```
LocalDateTime laikas = LocalDateTime.now(); // 2018-07-09T22:49:49.280

System.out.println(laikas.getDayOfMonth()); // 9
System.out.println(laikas.getHour());        // 22
System.out.println(laikas.getMonth());       // JULY
LocalDate data = laikas.toLocalDate();
System.out.println(data);                    // 2018-07-09
LocalDateTime naujasLaikas = laikas.plusDays(6);
System.out.println(naujasLaikas);            // 2018-07-15T22:49:49.280
```

DateTimeFormatter

```
LocalDate siandien = LocalDate.now(); // 2018-07-09
String formatuotaSiandien = siandien.format(DateTimeFormatter.BASIC_ISO_DATE);

System.out.println(formatuotaSiandien); // 20180709

DateTimeFormatter formateris = DateTimeFormatter.ofPattern("yyyy!!MM!!dd");
formatuotaSiandien = siandien.format(formateris);

System.out.println(formatuotaSiandien); // 2018!!07!!09
```

Laiko įvedimas

```
DateTimeFormatter f = DateTimeFormatter.ofPattern("HH:mm");

System.out.println("Iveskite laika: ");
Scanner sc = new Scanner(System.in);

String textFromConsole = sc.next();

sc.close();

LocalTime time = LocalTime.parse(textFromConsole, f);

System.out.println("Ivestas laikas: " + time);

System.out.println("pridejus viena valanda bus: " + time.plusHours(1));
```

Datų palyginimas naudojant *isBefore* ir *isAfter*

```
LocalDate zalgirioMuisis = LocalDate.of(1410, 7, 15);
LocalDate saulesMuisis = LocalDate.of(1236, 9, 22);

if (zalgirioMuisis.isBefore(saulesMuisis)) {
    System.out.println("Zalgirio muisis ivyko anksciau");
}

if (zalgirioMuisis.isAfter(saulesMuisis)) {
    System.out.println("Zalgirio muisis ivyko veliau");
}
```

Datų palyginimas naudojant *compareTo*

```
LocalDate zalgirioMysis = LocalDate.of(1410, 7, 15);
LocalDate saulesMysis = LocalDate.of(1236, 9, 22);

int compareToRezultatas = zalgirioMysis.compareTo(saulesMysis);

if (compareToRezultatas > 0) {
    System.out.println("Zalgirio musis ivyko veliau");
} else if (compareToRezultatas == 0) {
    System.out.println("Abu musiai vyko tuo paciu metu");
} else {
    System.out.println("Zalgirio musis ivyko anksciau");
}
```


Nuorodos

- LocalDateTime:
<https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html>
- LocalDate: <https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html>
- LocalTime: <https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html>
- DateTimeFormatter:
<https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>



`==` vs *equals()*

==

- Naudojama palyginti:
 - Skaičiams (int, long, short, byte)
 - Simboliams (char)
 - Loginiams kintamiesiems (boolean)
 - Objektų nuorodoms

== su primitiviais tipais

```
class ComparePrimitives {  
    public static void main(String[] args) {  
        System.out.println("char 'a' == 'a'? " + ('a' == 'a'));  
        System.out.println("char 'a' == 'b'? " + ('a' == 'b'));  
        System.out.println("5 != 6? " + (5 != 6));  
        System.out.println("5.0 == 5L? " + (5.0 == 5L));  
        System.out.println("true == false? " + (true == false));  
    }  
}
```

```
char 'a' == 'a'? true  
char 'a' == 'b'? false  
5 != 6? true  
5.0 == 5L? true  
true == false? false
```

Lyginamos primitiviųjų
tipų reikšmės

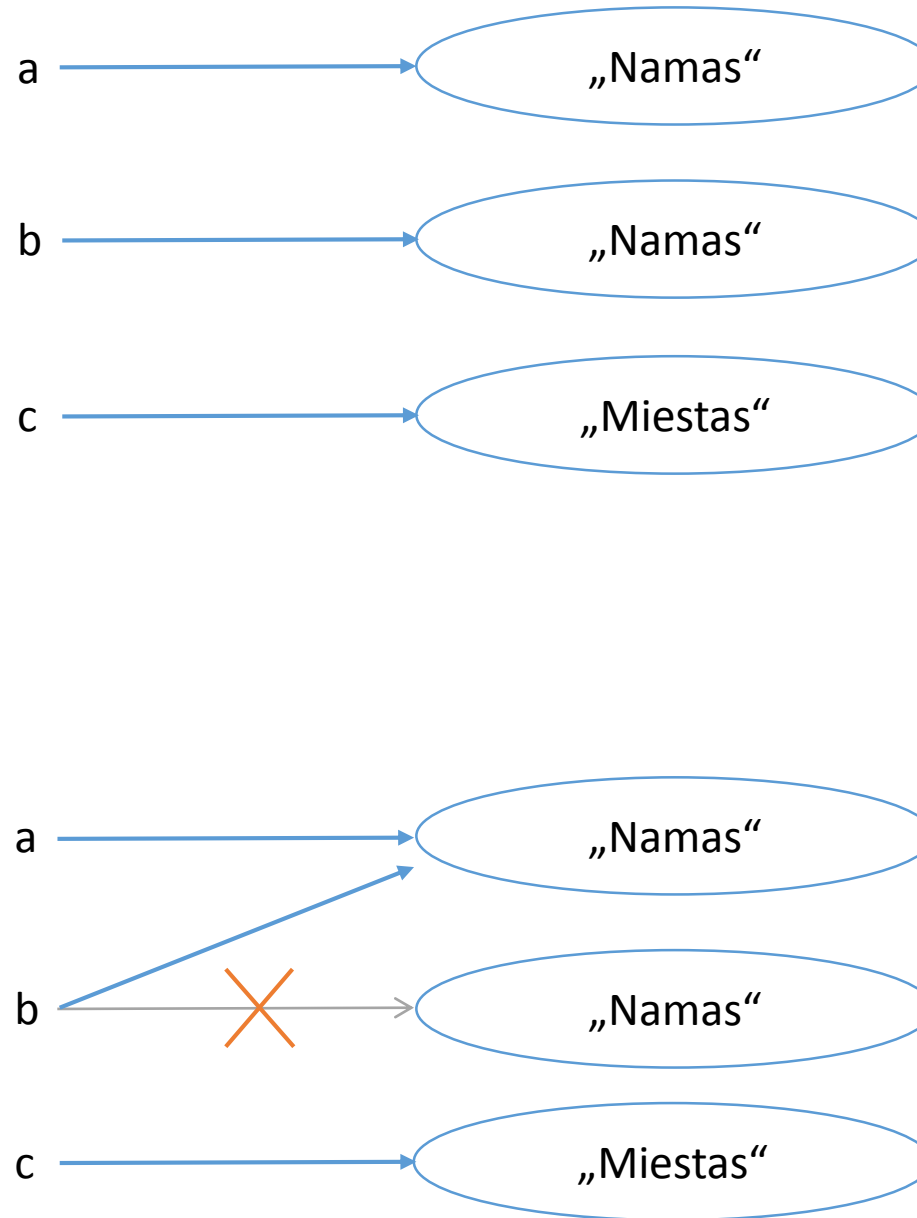
== su *String*

```
String a = new String("Namas");  
String b = new String("Namas");  
String c = new String("Miestas");
```

```
System.out.println(a == b); // false  
System.out.println(a == c); // false  
System.out.println(b == c); // false
```

```
b = a;
```

```
System.out.println(a == b); // true  
System.out.println(a == c); // false  
System.out.println(b == c); // false
```



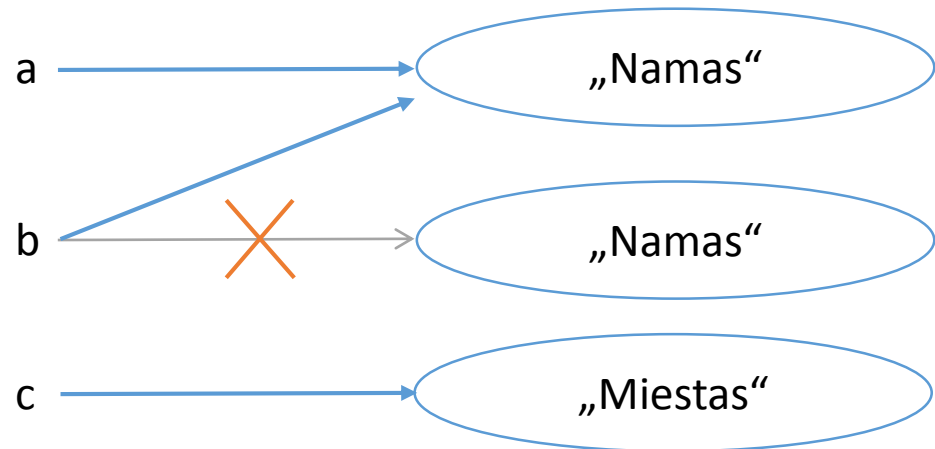
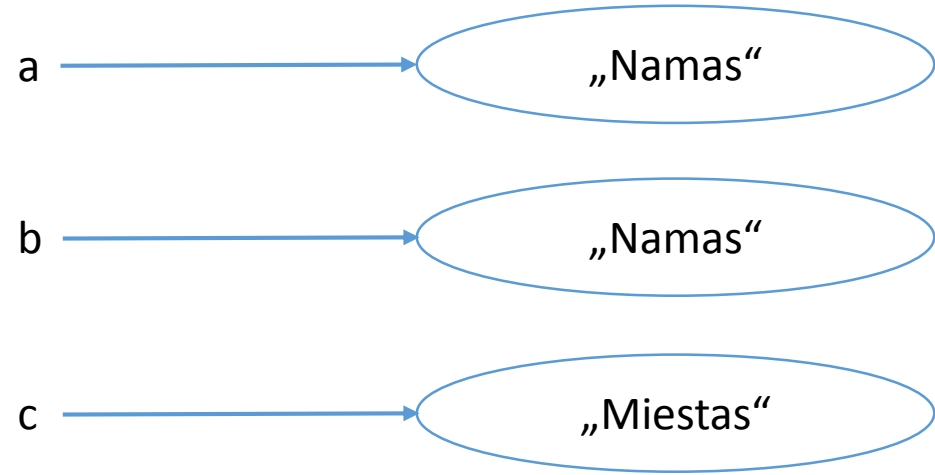
equals() su *String*

```
String a = new String("Namas");  
String b = new String("Namas");  
String c = new String("Miestas");
```

```
System.out.println(a.equals(b)); // true  
System.out.println(a.equals(c)); // false  
System.out.println(b.equals(c)); // false
```

```
b = a;
```

```
System.out.println(a.equals(b)); // true  
System.out.println(a.equals(c)); // false  
System.out.println(b.equals(c)); // false
```



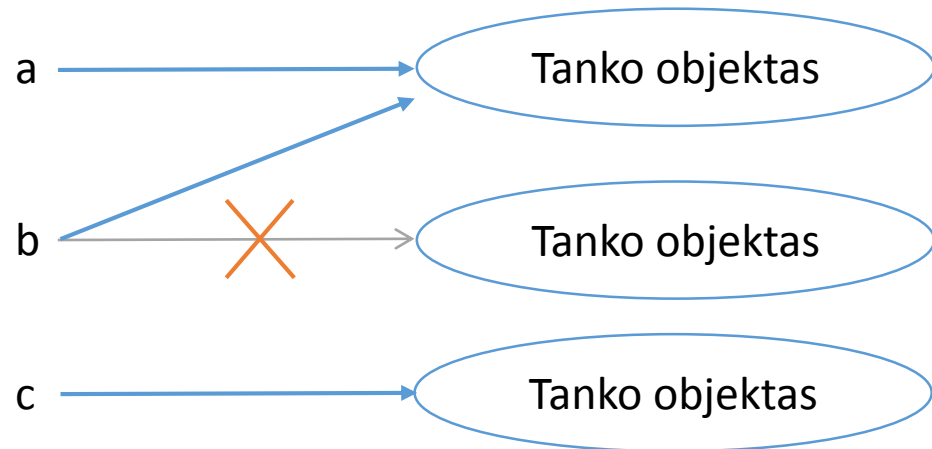
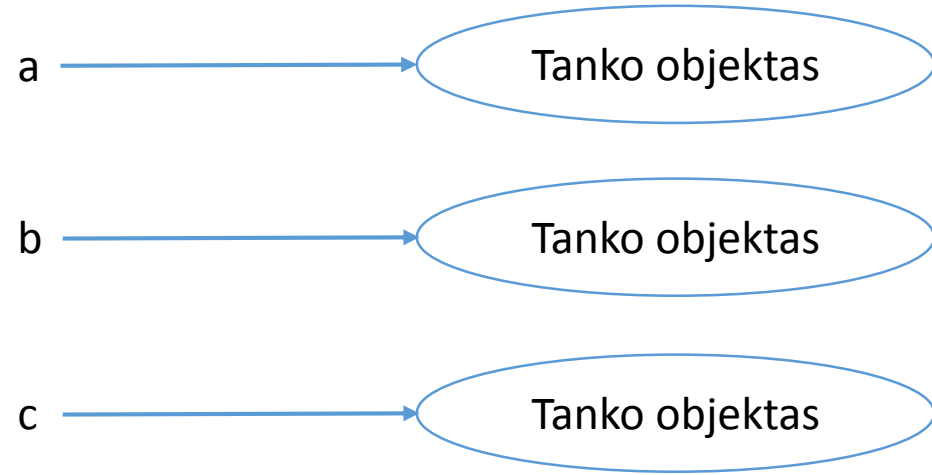
== su *Objektais*

```
Tankas a = new Tankas();  
Tankas b = new Tankas();  
Tankas c = new Tankas();
```

```
System.out.println(a == b); // false  
System.out.println(a == c); // false  
System.out.println(b == c); // false
```

```
b = a;
```

```
System.out.println(a == b); // true  
System.out.println(a == c); // false  
System.out.println(b == c); // false
```



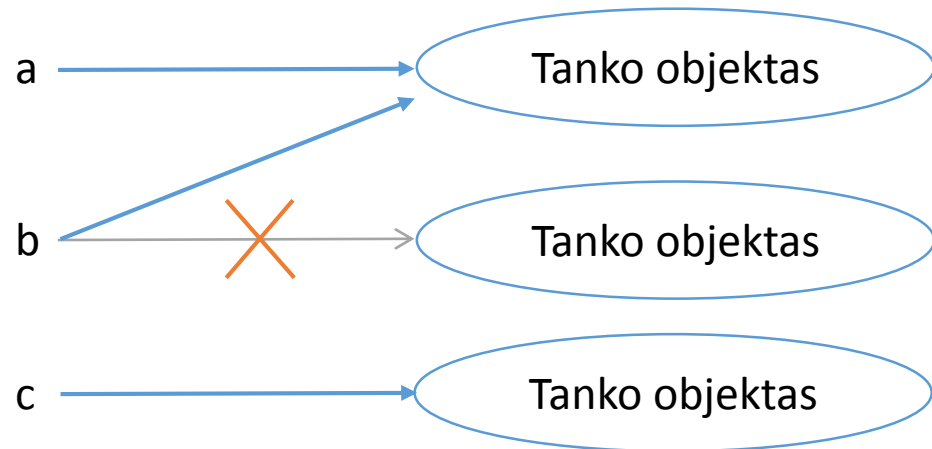
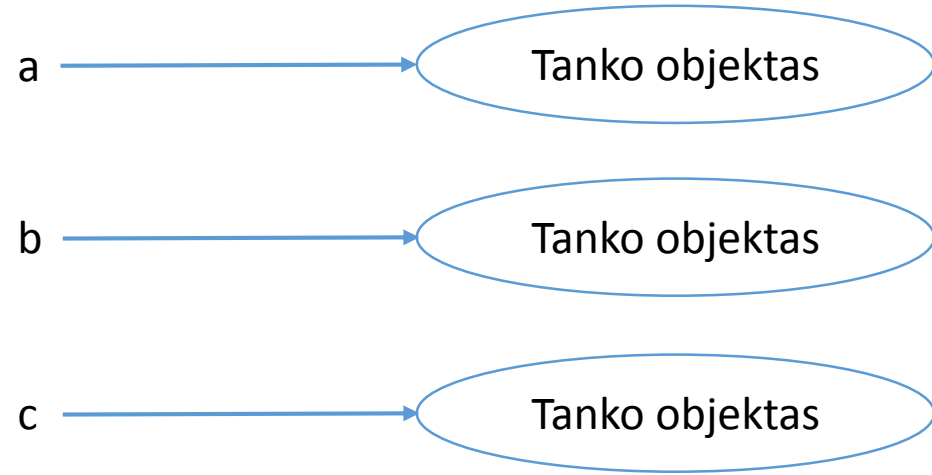
equals() su *Objektais*

```
Tankas a = new Tankas();  
Tankas b = new Tankas();  
Tankas c = new Tankas();
```

```
System.out.println(a.equals(b)); // false  
System.out.println(a.equals(c)); // false  
System.out.println(b.equals(c)); // false
```

```
b = a;
```

```
System.out.println(a.equals(b)); // true  
System.out.println(a.equals(c)); // false  
System.out.println(b.equals(c)); // false
```



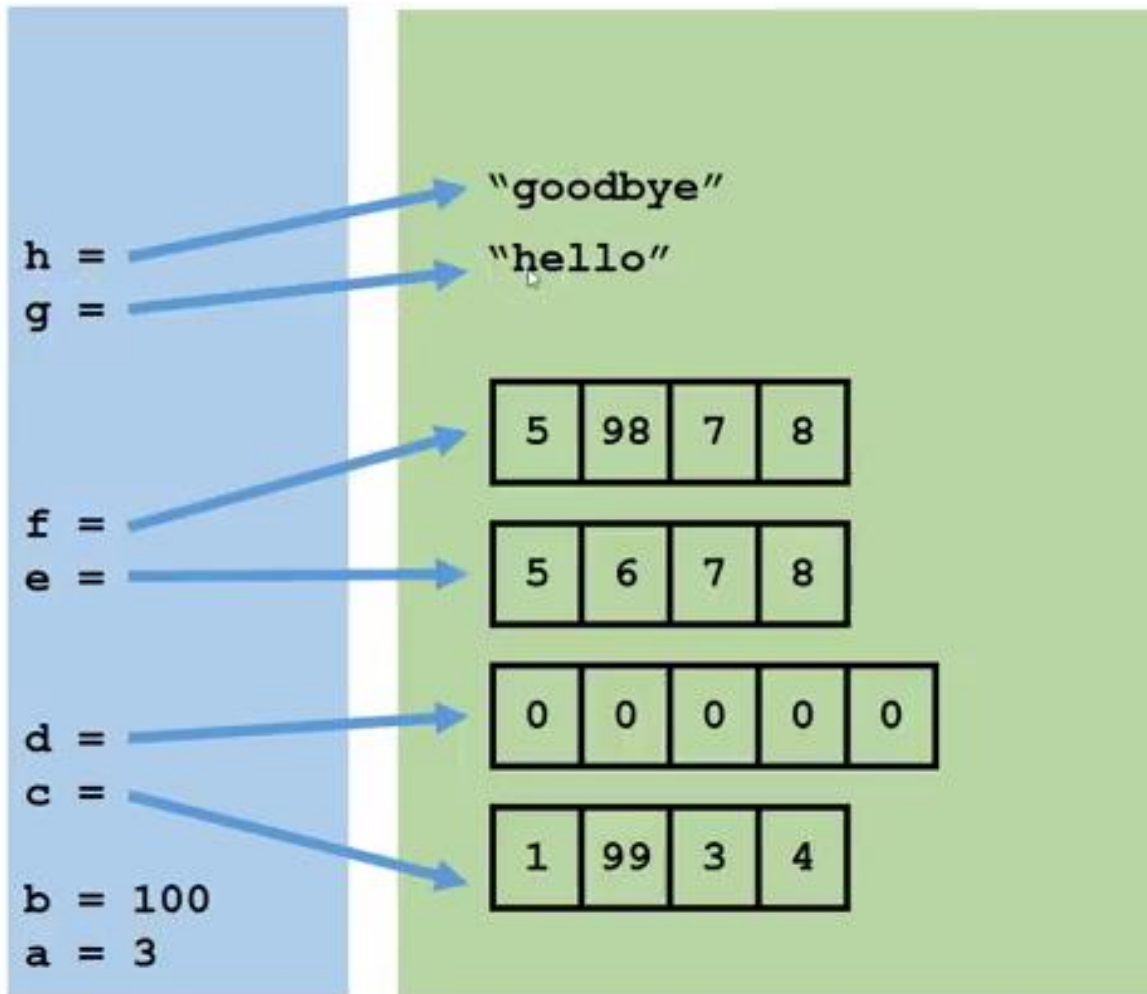
Kaip atrodo atmintis Javoje

```
int a = 3;  
int b = a;  
b = 100;
```

```
int[] c = {1, 2, 3, 4};  
int[] d = c;  
d[1] = 99;  
d = new int[5];
```

```
int[] e = {5, 6, 7, 8};  
int[] f = {5, 6, 7, 8};  
f[1] = 98;
```

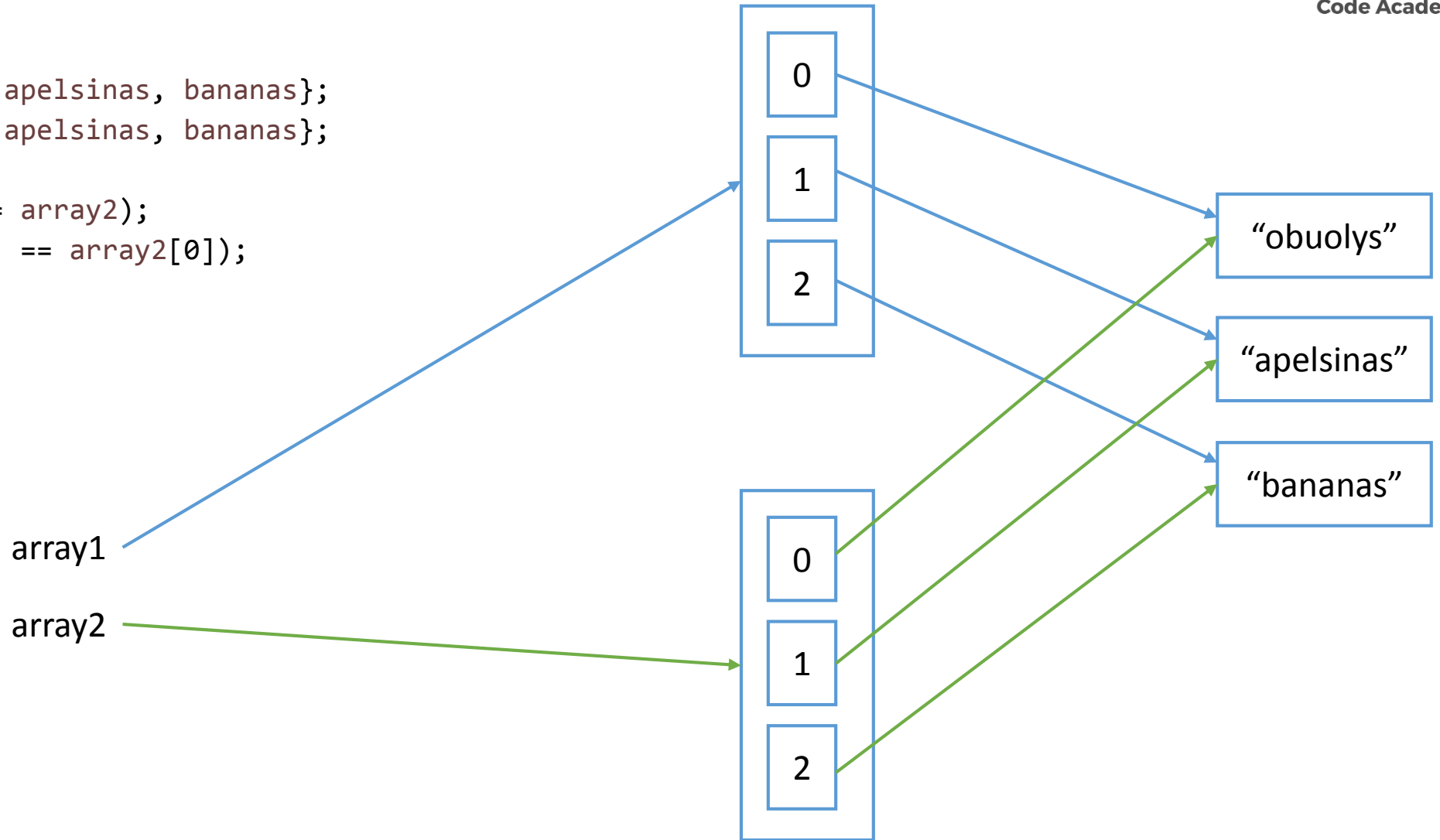
```
String g = "hello";  
String h = g;  
h = "goodbye";
```



```
String obuolys = "obuolys";  
String apelsinas = "apelsinas";  
String bananas = "bananas";
```

```
String[] array1 = {obuolys, apelsinas, bananas};  
String[] array2 = {obuolys, apelsinas, bananas};
```

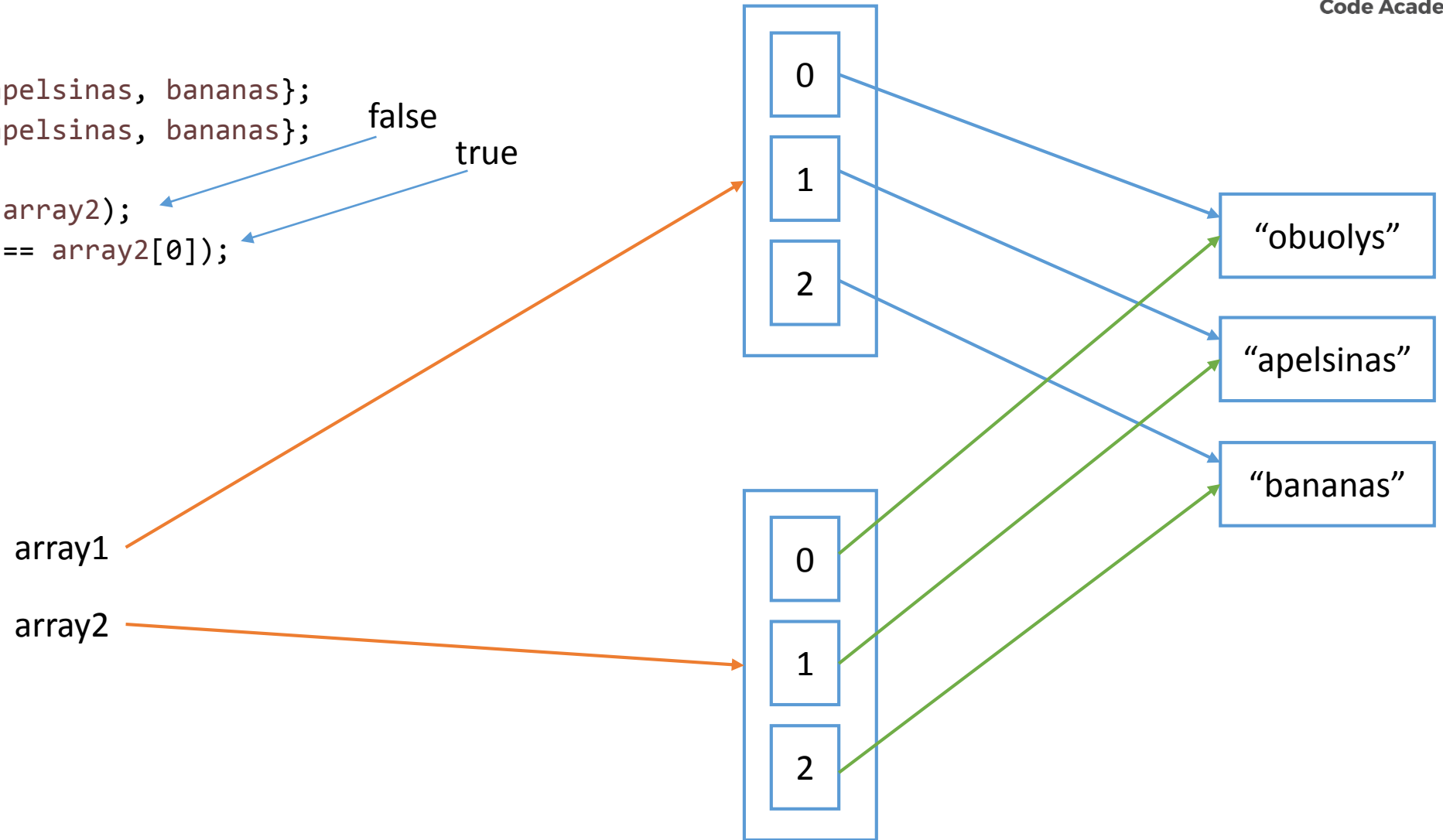
```
System.out.println(array1 == array2);  
System.out.println(array1[0] == array2[0]);
```



```
String obuolys = "obuolys";  
String apelsinas = "apelsinas";  
String bananas = "bananas";
```

```
String[] array1 = {obuolys, apelsinas, bananas};  
String[] array2 = {obuolys, apelsinas, bananas};
```

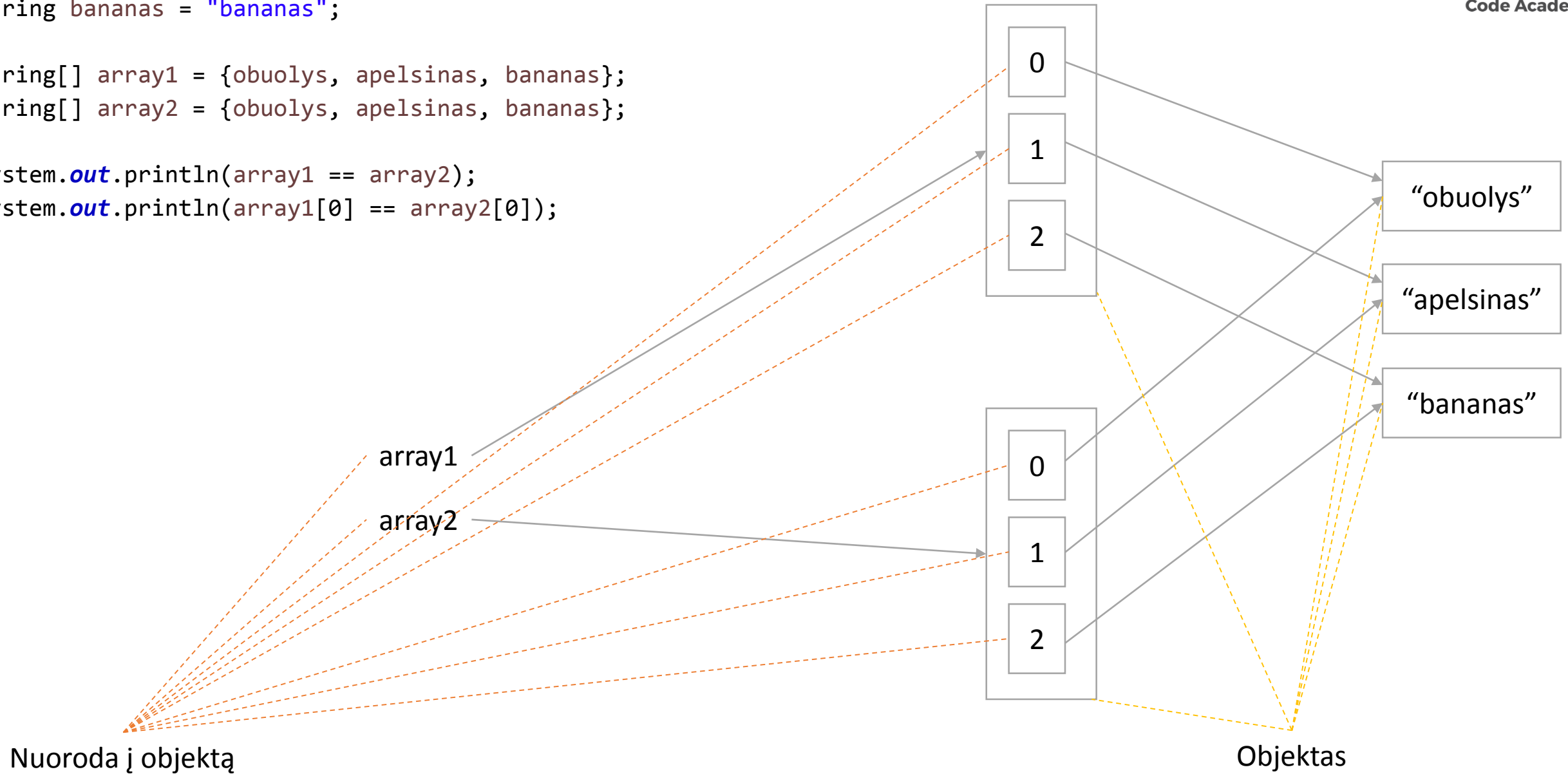
```
System.out.println(array1 == array2);  
System.out.println(array1[0] == array2[0]);
```



```
String obuolys = "obuolys";  
String apelsinas = "apelsinas";  
String bananas = "bananas";
```

```
String[] array1 = {obuolys, apelsinas, bananas};  
String[] array2 = {obuolys, apelsinas, bananas};
```

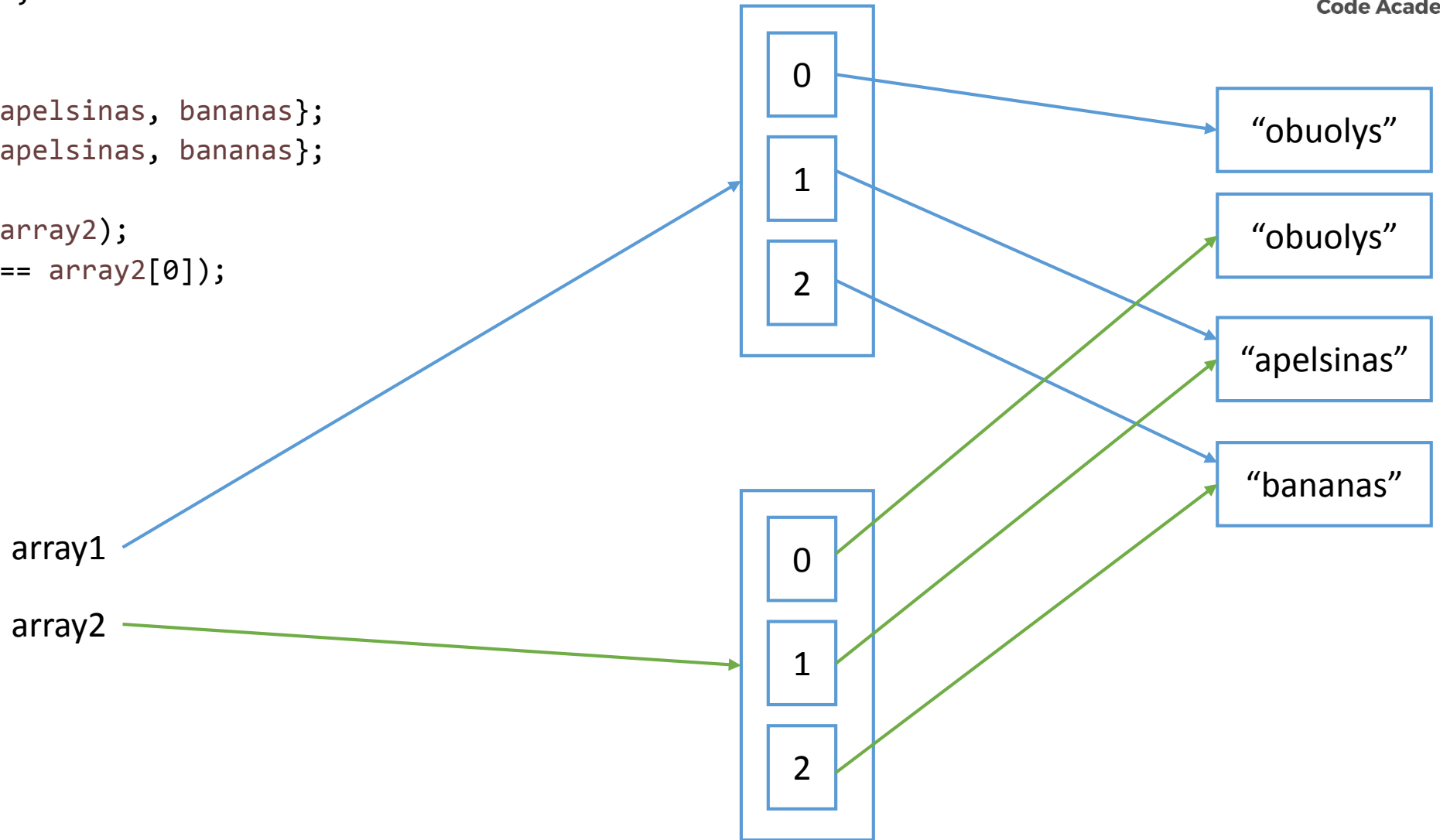
```
System.out.println(array1 == array2);  
System.out.println(array1[0] == array2[0]);
```



```
String obuolys1 = "obuolys";  
String obuolys2 = new String("obuolys");  
String apelsinas = "apelsinas";  
String bananas = "bananas";
```

```
String[] array1 = {obuolys1, apelsinas, bananas};  
String[] array2 = {obuolys2, apelsinas, bananas};
```

```
System.out.println(array1 == array2);  
System.out.println(array1[0] == array2[0]);
```

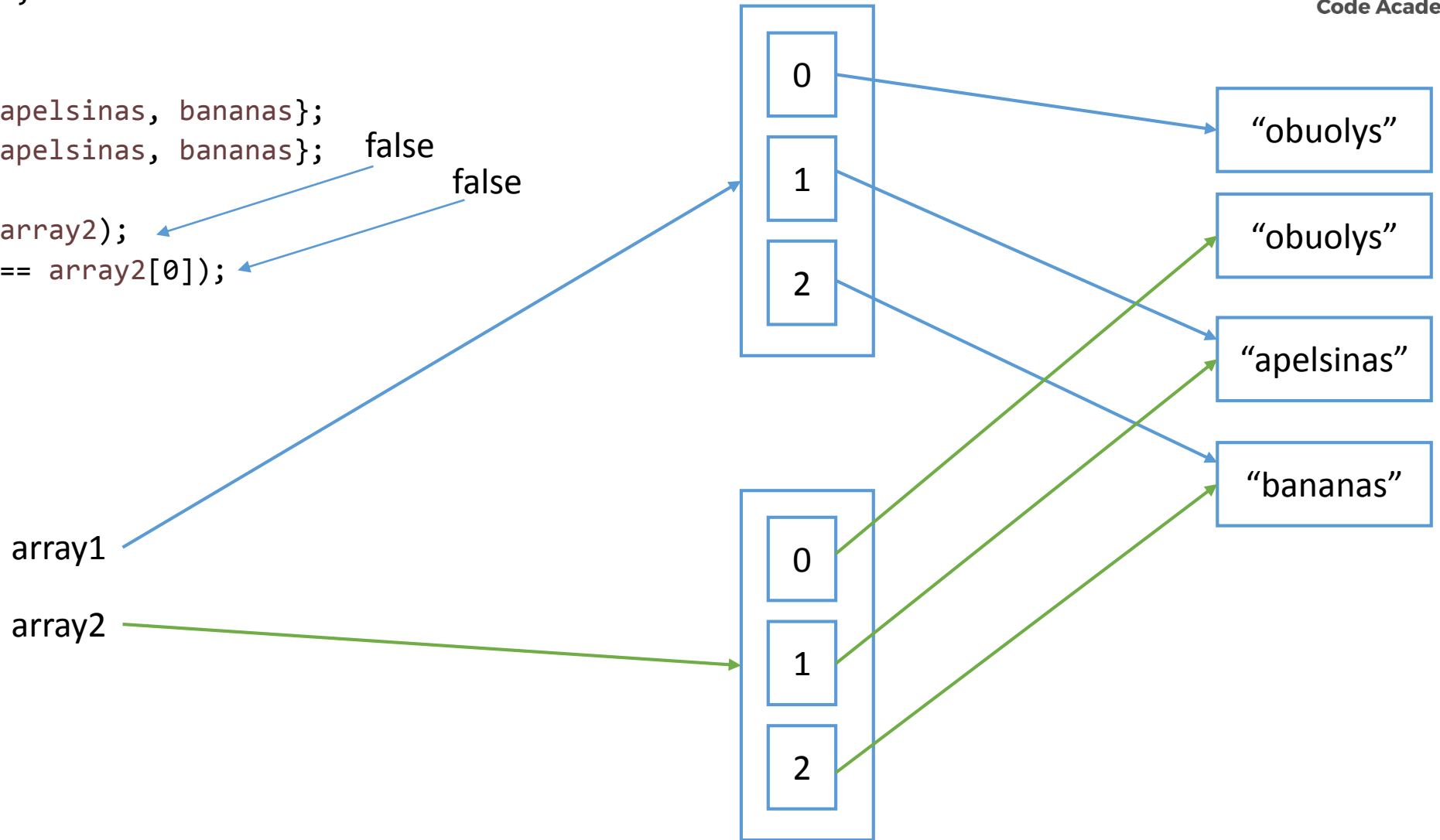


```
String obuolys1 = "obuolys";  
String obuolys2 = new String("obuolys");  
String apelsinas = "apelsinas";  
String bananas = "bananas";
```

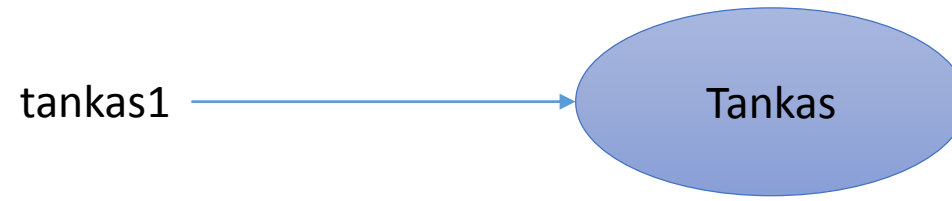
```
String[] array1 = {obuolys1, apelsinas, bananas};  
String[] array2 = {obuolys2, apelsinas, bananas};
```

```
System.out.println(array1 == array2);
```

```
System.out.println(array1[0] == array2[0]);
```



Kas yra kas?



```
Tankas tankas1 = new Tankas();
```

Kokio tipo yra objektas,
į kurį rodo *tankas1*
kintamais

Tai kintamasis, kuris yra
nuoroda į sukurtą Tanko
objektą

new – reiškia, kad kuriame objektą

Kviečiame klasės *Tankas* konstruktorių.
Šiuo atveju kviečiame *default* konstruktorių
be parametrų. Galime kviesti konstruktorių
su parametrais, jei toks yra aprašytas klasėje
Tankas.



Dinaminiai sąrašai *ArrayList*

Ką galime nuveikti su kolekcjomis (*Collections*)



- Pridėti objektus į kolekcijas
- Išimti elementus iš kolekcijos
- Ieškoti objektų kolekciijoje
- Gauti objektus iš kolekcijos
- Iteruoti visus kolekcijos elementus vienas po kito

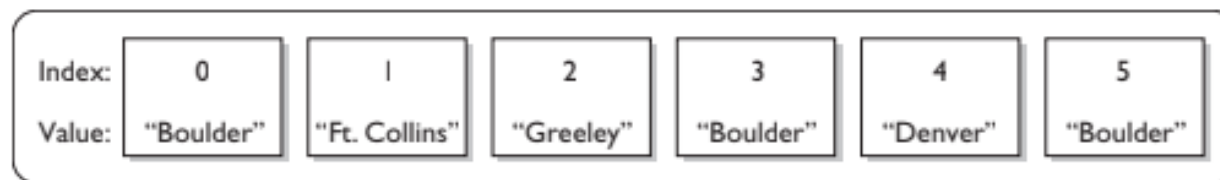
Java kolekcijos

Maps	Sets	Lists	Queues	Utilities
HashMap	HashSet	ArrayList	PriorityQueue	Collections
Hashtable	LinkedHashSet	Vector		Arrays
TreeMap	TreeSet	LinkedList		
LinkedHashMap				

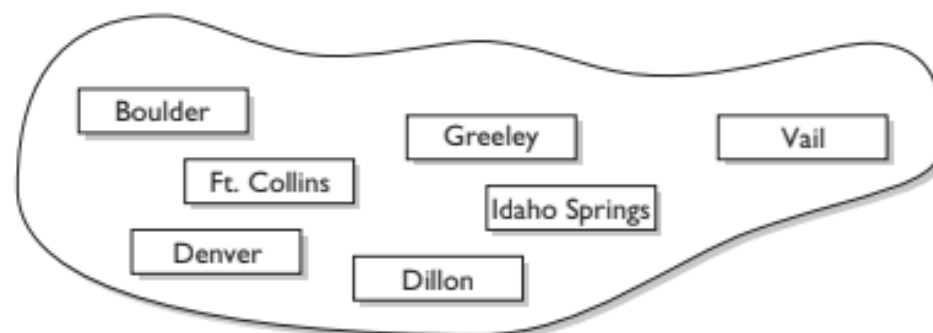
Java kolekcijos

- *List* - objektų sąrašai
- *Set* - unikalių objektų sąrašas
- *Map* - key -> value sąrašas

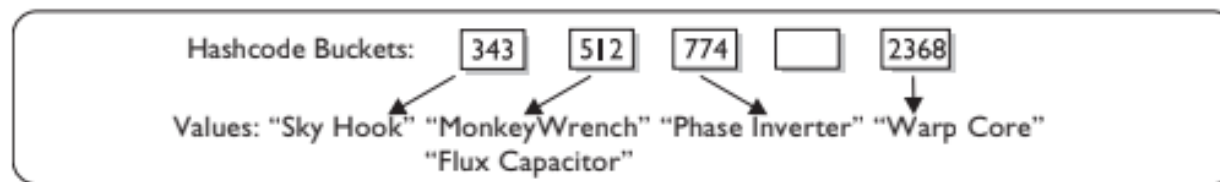
Java kolekcijos



List: The salesman's itinerary (Duplicates allowed)



Set: The salesman's territory (No duplicates allowed)



HashMap: The salesman's products (Keys generated from product IDs)

```
/* Sukuriamas dinaminio saraso objektas */
ArrayList<String> sarasas = new ArrayList<>();

/* Dedami objektai i sarasa */
sarasas.add("Pirmas");
sarasas.add("Antras");
sarasas.add("Trecias");
sarasas.add("Ketvirtas");
sarasas.add("Penktas");

System.out.println("Saraso elementai: " + sarasas);

/* Idedama dar elementu i saraso pradzia */
sarasas.add(0, "Sestas");
sarasas.add(1, "Septintas");

/* Isimami/istrinami elementai is saraso pagal reiksme */
sarasas.remove("Trecias");
sarasas.remove("Penktas");

System.out.println("Saraso elementai: " + sarasas);

/* Isimamams/istrinamas 1-as elementas */
sarasas.remove(1);

System.out.println("Saraso elementai: " + sarasas);
```

```
/* Sukuriamas dinaminio saraso objektas */
ArrayList<String> sarasas = new ArrayList<>();

/* Dedami objektai i sarasa */
sarasas.add("Pirmas");
sarasas.add("Antras");
sarasas.add("Trecias");
sarasas.add("Ketvirtas");
sarasas.add("Penktas");
System.out.println("Saraso elementai: " + sarasas);

// pakeiciamas 2-as elementas
sarasas.set(2, "*****");
System.out.println("Saraso elementai: " + sarasas);

// gaunama elemento pozicija sarase
int pos = sarasas.indexOf("*****");
System.out.println(pos);

// gaunamas elementas pagal indeksa
String elementas = sarasas.get(2);
System.out.println(elementas);

// gaunamas saraso dydis
int sarasoDydis = sarasas.size();
System.out.println(sarasas.size());

boolean arYraAntras = sarasas.contains("Antras");
System.out.println(arYraAntras);
```

```
// isvalo sarasa
sarasas.clear();
System.out.println(sarasas);
System.out.println(sarasas.size());

// patikrina ar sarasas tuscias
boolean arTuscias = sarasas.isEmpty();
System.out.println(arTuscias);
```

```
/* Sukuriamas dinaminio saraso objektas */
ArrayList<String> sarasas = new ArrayList<>();

/* Dedami objektai i sarasa */
sarasas.add("Pirmas");
sarasas.add("Antras");
sarasas.add("Trecias");
sarasas.add("Ketvirtas");
sarasas.add("Penktas");
System.out.println("Saraso elementai: " + sarasas);

for (String elementas : sarasas) {
    int i = sarasas.indexOf(elementas);
    System.out.println("Elementas: " + elementas + ", Indeksas: " + i);
}

System.out.println("Saraso elementai: " + sarasas);
```



Dinaminiai sąrašai :
ArrayList, LinkedList

Skirtingi iteravimo būdai

```
ArrayList<String> arrlist = new ArrayList<>();
arrlist.add("Vilnius");
arrlist.add("Kaunas");
arrlist.add("Klaipeda");
arrlist.add("Siauliai");

/* Iprastas FOR ciklas */
for (int counter = 0; counter < arrlist.size(); counter++) {
    System.out.println(arrlist.get(counter));
}

/* FOR ciklas masyvams ir sarasams */
for (String str : arrlist) {
    System.out.println(str);
}

/* WHILE ciklas */
int count = 0;
while (arrlist.size() > count) {
    System.out.println(arrlist.get(count));
    count++;
}

/* WHILE + ITERATOR */
Iterator<String> iter = arrlist.iterator();
while (iter.hasNext()) {
    System.out.println(iter.next());
}
```

Sąrašo *String* tipo elementų rūšiavimas

```
ArrayList<String> arrlist = new ArrayList<>();  
arrlist.add("Vilnius");  
arrlist.add("Kaunas");  
arrlist.add("Klaipeda");  
arrlist.add("Siauliai");  
  
System.out.println(arrlist);  
  
Collections.sort(arrlist);  
  
System.out.println(arrlist);  
  
Collections.sort(arrlist, Collections.reverseOrder());  
  
System.out.println(arrlist);
```

Vieno sąrašo papildymas kitu sąrašu

```
ArrayList<String> pirmasSarasas = new ArrayList<>();  
pirmasSarasas.add("Vilnius");  
pirmasSarasas.add("Kaunas");  
pirmasSarasas.add("Klaipeda");  
pirmasSarasas.add("Siauliai");
```

```
System.out.println("Pirmas sarasas: " + pirmasSarasas);
```

```
ArrayList<String> antrasSarasas = new ArrayList<>();  
antrasSarasas.add("Panevezys");  
antrasSarasas.add("Utena");
```

```
System.out.println("Antras sarasas: " + antrasSarasas);
```

```
pirmasSarasas.addAll(antrasSarasas);
```

```
System.out.println("Papildytas pirmas sarasas: " + pirmasSarasas);
```

Dviejų sąrašo elementų sukeitimas

```
ArrayList<String> pirmasSarasas = new ArrayList<>();  
pirmasSarasas.add("Vilnius");  
pirmasSarasas.add("Kaunas");  
pirmasSarasas.add("Klaipeda");  
pirmasSarasas.add("Siauliai");  
  
System.out.println("Pirmas sarasas: " + pirmasSarasas);  
  
Collections.swap(pirmasSarasas, 0, 2);  
  
System.out.println("Pirmas sarasas po apkeitimo: " +  
pirmasSarasas);
```

Primityviųjų tipų elementai sąrašė

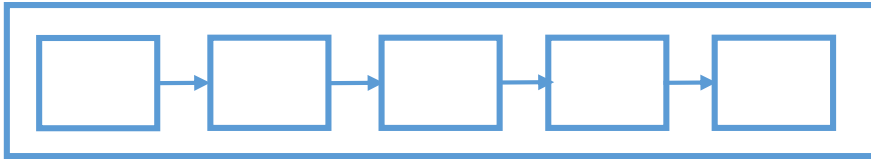
- Primityviųjų tipų reikšmių negalima tiesiogiai įdėti į sąrašą
- Į sąrašą dedami tik objektai
- Kad galėtume įdėti į sąrašą primityvaus tipo reikšmes, jas turime paversti į objektus
 - `Integer intObjektas = new Integer(5);`
 - `Float floatObjektas = new Float(6.4f);`
 - `Double doubleObjektas = new Double(5.5d);`
 - `Boolean booleanObjektas = new Boolean(true);`

Primityviųjų tipų elementai sąrašė

```
ArrayList<Integer> pirmasSarasas = new ArrayList<>();  
pirmasSarasas.add(2);  
pirmasSarasas.add(4);  
pirmasSarasas.add(new Integer(5));  
pirmasSarasas.add(6);  
  
System.out.println(pirmasSarasas);  
  
Integer intObjektas = pirmasSarasas.get(1);  
  
int primityvusIntKinatamasis = intObjektas.intValue();  
  
System.out.println(primityvusIntKinatamasis);
```

LinkedList

- Elementai sujungti ir eina vienas po kito
- Visada galime gauti pirmąjį ir paskutinįjį elementus



```
// sukuriamas tuscias susietas sarasas
LinkedList<String> sujungtasSarasas = new LinkedList<String>();

// i sarasa dedami elementai
sujungtasSarasas.add("vasaris");
sujungtasSarasas.add("kovas");
sujungtasSarasas.add("balandis");
sujungtasSarasas.add("geguze");

System.out.println(sujungtasSarasas);

// idedame elementus i sarso i pradzia ir i pabaiga
sujungtasSarasas.addFirst("SAUSIS");
sujungtasSarasas.addLast("BIRZELIS");

System.out.println(sujungtasSarasas);
// istriname pirma ir paskutini elementus
sujungtasSarasas.removeFirst();
sujungtasSarasas.removeLast();

System.out.println(sujungtasSarasas);

// idedame ir istriname pagal indeksa
sujungtasSarasas.add(0, "Dar vienas menuo");
sujungtasSarasas.remove(2);

// dar yra removeFirst() ir removeLast() metodai,
// kurie istrina pirma ir paskutini elementus

System.out.println(sujungtasSarasas);

// grazina saraso pirmaji elementa ir ji istrina is saraso
String menuo = sujungtasSarasas.poll();
System.out.println(menuo);
System.out.println(sujungtasSarasas);

// grazina saraso pirmaji elementa ir ji istrina is saraso
menuo = sujungtasSarasas.pollFirst();
System.out.println(menuo);
System.out.println(sujungtasSarasas);

// grazina saraso pirmaji elementa ir ji istrina is saraso
menuo = sujungtasSarasas.pollLast();
System.out.println(menuo);
System.out.println(sujungtasSarasas);
```

Push & Pop

- Push – deda elementą į sąrašo pradžią
- Pop – išima pirmąjį sąrašo elementą

```
LinkedList<String> sujungtasSarasas = new LinkedList<String>();
```

```
sujungtasSarasas.add("vasaris");  
sujungtasSarasas.add("kovas");  
sujungtasSarasas.add("balandis");  
sujungtasSarasas.add("geguze");
```

```
System.out.println(sujungtasSarasas);
```

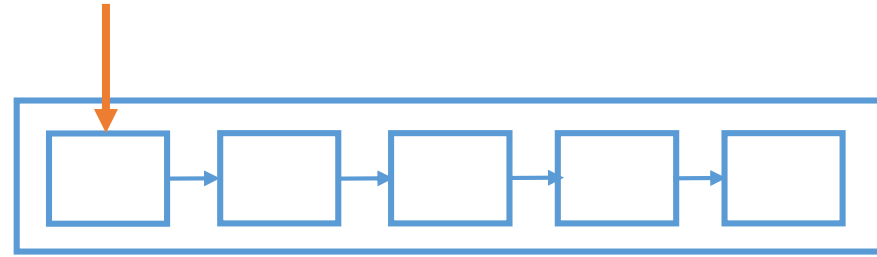
```
sujungtasSarasas.push("sausis");
```

```
System.out.println(sujungtasSarasas);
```

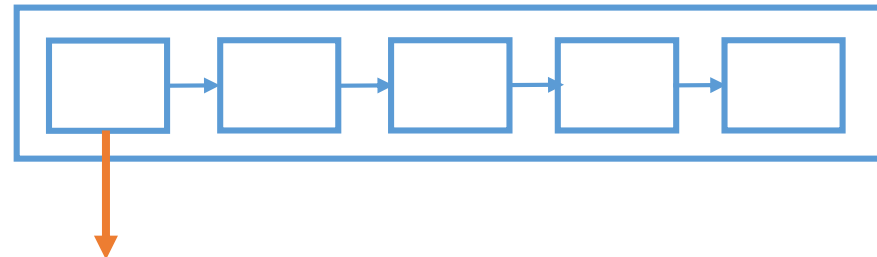
```
String ispopintasElementas = sujungtasSarasas.pop();
```

```
System.out.println(sujungtasSarasas);  
System.out.println(ispopintasElementas);
```

Push



Pop



Nuorodos

- ArrayList dokumentacija:
<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>
- LinkedList dokumentacija:
<https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html>



HashSet

Aibė

- Aibė negarantuoja, kad elementai jos viduje visada bus išdėstyti ta pačia tvarka
- Aibė neleidžia pasikartojančių reikšmių. Jas galime dėti, bet nuo to aibė nepasikeis (tiksliau bus ta pati reikšmė perrašyta nauja), o ir klaidos nebus
- Galime įdėti null reikšmes

Aibė

```
// Aibes sukurimas
HashSet<String> aibe = new HashSet<String>();

// Elementu idejimas i aibe
aibe.add("Apple");
aibe.add("Mango");
aibe.add("Grapes");
aibe.add("Orange");
aibe.add("Fig");

System.out.println(aibe);

// Dedame pasikartojancius el., bet niekas nepasikeis
aibe.add("Apple");
aibe.add("Mango");

System.out.println(aibe);
```

Iteravimas

```
HashSet<String> hset = new HashSet<String>();
```

```
// add elements to HashSet
```

```
hset.add("Chaitanya");
```

```
hset.add("Rahul");
```

```
hset.add("Tim");
```

```
hset.add("Rick");
```

```
hset.add("Harry");
```

```
Iterator<String> it = hset.iterator();
```

```
while (it.hasNext()) {
```

```
    System.out.println(it.next());
```

```
}
```

Iteravimas II

```
HashSet<String> hset = new HashSet<String>();
```

```
// add elements to HashSet
```

```
hset.add("Chaitanya");
```

```
hset.add("Rahul");
```

```
hset.add("Tim");
```

```
hset.add("Rick");
```

```
hset.add("Harry");
```

```
for (String temp : hset) {  
    System.out.println(temp);  
}
```

HashSet metodai

- **boolean add(Element e)**: įdeda elementą e į aibę.
- **void clear()**: ištrina visus elementus iš aibės.
- **boolean contains(Object o)**: patikrina, ar objektas (elementas) o yra aibėje.
- **boolean isEmpty()**: grąžina true, jei aibė yra tuščia.
- **int size()**: grąžina aibės elementų skaičių, t.y. aibės dydį.
- **boolean remove(Object o)**: ištrina elementą o iš aibės

Nuorodos

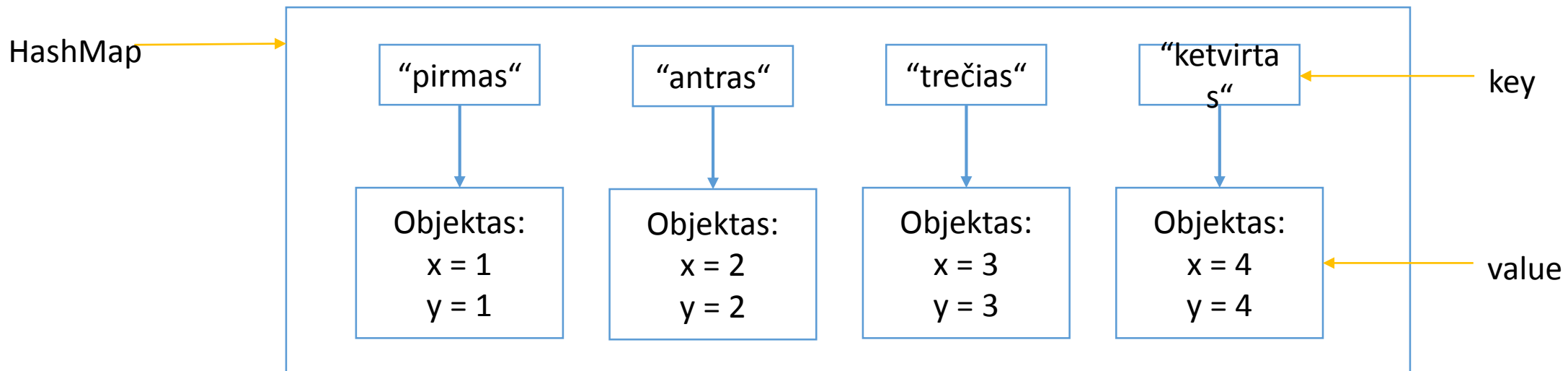
- HashSet dokumentacija:
<https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html>



HashMap

HashMap

- Saugo sąrašą poros key-value reikšmių
- Negarantuoja, kad porų eilės tvarka bus visada ta pati
- Norint naudoti, reikia importuoti `java.util.HashMap`
- Naudojamas norint išsaugoti objektus (values), kuriuos vėliau galėsime greit rasti pagal raktus (keys)



HashMap sukūrimas ir porų įdėjimas

```
/* Sukuriamas HashMap objektas */
```

```
HashMap<Integer, String> hmap = new HashMap<Integer, String>();
```

```
/* idedamos poros (key->value) i HashMap */
```

```
hmap.put(12, "Chaitanya");
```

```
hmap.put(2, "Rahul");
```

```
hmap.put(7, "Singh");
```

```
hmap.put(49, "Ajeet");
```

```
hmap.put(3, "Anuj");
```

```
System.out.println(hmap);
```

—————→ {49=Ajeet, 2=Rahul, 3=Anuj, 7=Singh, 12=Chaitanya}


Reikšmės gavimas pagal rakta

```
String gautaReiksme = hmap.get(12);  
System.out.println(gautaReiksme);
```

→ Chaitanya

Visų HashMap raktų gavimas

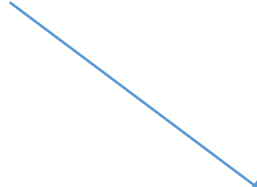
```
Set<Integer> visiRaktai = hmap.keySet();  
System.out.println(visiRaktai);
```



[49, 2, 3, 7, 12]

HashMap iteravimas naudojant While


```
Set set = hmap.entrySet();
Iterator iterator = set.iterator();
while(iterator.hasNext()) {
    Map.Entry mentry = (Map.Entry)iterator.next();
    System.out.print("key is: "+ mentry.getKey() + " & Value is: ");
    System.out.println(mentry.getValue());
}
```



key is: 49 & Value is: Ajeet
key is: 2 & Value is: Rahul
key is: 3 & Value is: Anuj
key is: 7 & Value is: Singh
key is: 12 & Value is: Chaitanya

HashMap iteravimas naudojant For

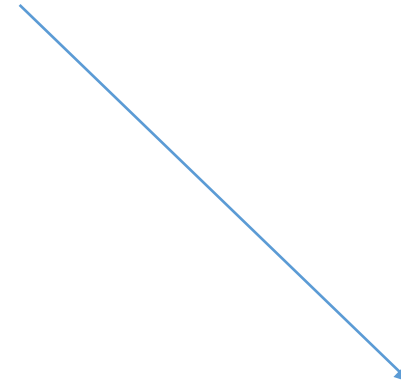
```
for (Map.Entry me : hmap.entrySet()) {  
    System.out.println("Key: "+me.getKey() + " & Value: " + me.getValue());  
}
```



key is: 49 & Value is: Ajeet
key is: 2 & Value is: Rahul
key is: 3 & Value is: Anuj
key is: 7 & Value is: Singh
key is: 12 & Value is: Chaitanya

HashMap dydis

```
System.out.println("HashMap dydis: " + hmap.size());
```



HashMap dydis: 5

Poros ištrynimas iš HashMap pagal rakta

```
Object removed = hmap.remove(12);  
System.out.println("Istrinta reiksme: " + removed);
```



Istrinta reiksme: Chaitanya

Kiti HashMap metodai

- `hmap.isEmpty()` – grąžina `true`, jei `HashMap` yra tuščias
- `hmap.containsKey(12)` – grąžina `true`, jei `HashMap`’e egzistuoja pora su raktu 12
- `hmap.containsValue("Singh")` – grąžina `true`, jei `HashMap`’e egzistuoja pora su reikšme "Singh,,
- `hmap.clear()` – išvalo visą `HashMap`.

DRY principas

Visada programuoti taip, kad programoje nekartotume jau parašyto kodo. Naudingiau yra parašyti metodą, kuris atlieka veiksmus (logiką) ir kurį galime iškviesti daug kartų, nei tuos pačius veiksmus (logiką) daryti daugelyje pasikartojančių kodo vietų.

DRY: Don't Repeat Yourself



DRY ✓

Don't Repeat Yourself

WET ✗

Write Everything Twice

We Enjoy Typing

Waste Everyone's Time

KISS ✓

Keep It Simple, Stupid

Keep It Short and Simple

Nuorodos

- HashMap dokumentacija:
<https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>
- DRY:
https://en.wikipedia.org/wiki/Don%27t_repeat_yourself
- KISS:
https://en.wikipedia.org/wiki/KISS_principle



Paveldējimas

Kas yra paveldėjimas?

- Procesas, kai viena klasė paveldi kitos klasės kintamuosius bei metodus, yra vadinama **paveldėjimu** (*inheritance*).
- Paveldėjimo tikslas – sumažinti to paties Java kodo rašymą kelis kartus ir perpanaudoti jau turimą funkcionalumą.
- Klasė, kuri paveldi kitos klasės funkcionalumą yra vadinama **paveldinčia** klase, **išvestine** klase, **vaikine** klase arba **sub** klase (*child class, sub clas, derived class*)
- Klasė, kurios funkcionalumą paveldi kita klasė, yra vadinama **tėvine** klase, **bazine** klase arba **super** klase (*parent class, super class, base class*)
- Vaikinė klasė praplečia tėvinę klasę pridėdama naujo funkcionalumo
- Visos klasės tiesiogiai ar netiesiogiai paveldi klasės **java.lang.Object** savybes

Paveldėjimas

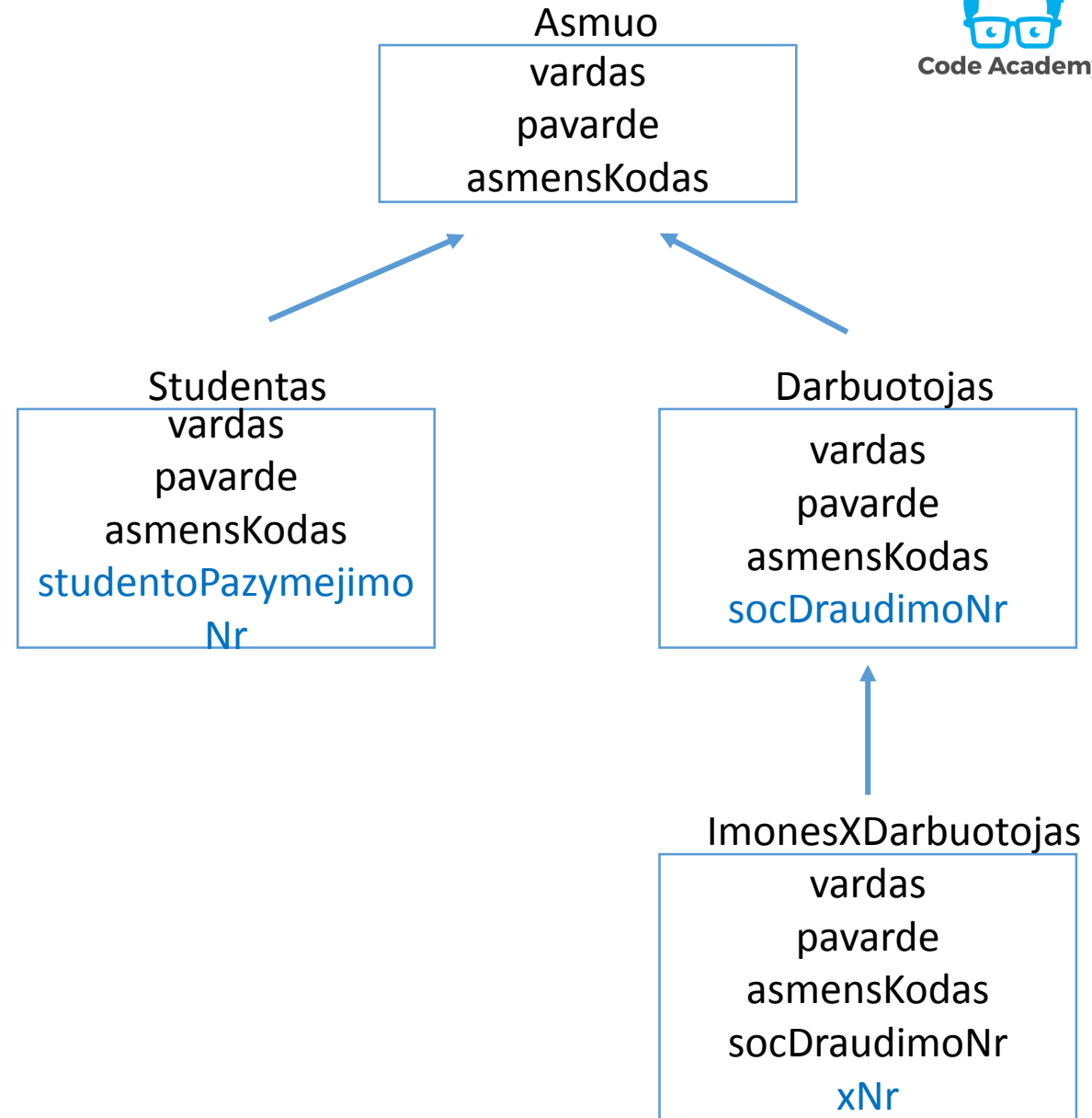
- Norėdami parodyti, kad klasė paveldi kitos klasės funkcionalumą, vaikinėje klasėje naudojame žodį „extends“.

```
class A {  
    ...  
}
```

```
class B extends A {  
    ...  
}
```



```
public class Asmuo {  
    protected String vardas;  
    protected String pavarde;  
    protected String asmensKodas;  
    //...konstruktorius/getteriai/setteriai...  
}  
  
class Studentas extends Asmuo {  
    protected String studentoPazymejimoNr;  
    //...konstruktorius/getteriai/setteriai...  
}  
  
class Darbuotojas extends Asmuo {  
    protected String socDraudimoNr;  
    //...konstruktorius/getteriai/setteriai...  
}  
  
class ImonesXDarbuotojas extends Darbuotojas {  
    protected String xNr;  
    //...konstruktorius/getteriai/setteriai...  
}
```



Klasė Asmuo

```
class Asmuo {  
    protected String vardas;  
    protected String pavarde;  
    protected String asmensKodas;  
    public Asmuo(String vardas, String pavarde, String asmensKodas) {  
        this.vardas = vardas;  
        this.pavarde = pavarde;  
        this.asmensKodas = asmensKodas;  
    }  
    public String getVardas() {  
        return vardas;  
    }  
    public void setVardas(String vardas) {  
        this.vardas = vardas;  
    }  
    public String getPavarde() {  
        return pavarde;  
    }  
    public void setPavarde(String pavarde) {  
        this.pavarde = pavarde;  
    }  
    public String getAsmensKodas() {  
        return asmensKodas;  
    }  
    public void setAsmensKodas(String asmensKodas) {  
        this.asmensKodas = asmensKodas;  
    }  
}
```

Ši klasė nieko nepaveldi
ir turi vardą, pavardę ir
asmens kodą

Klasė Studentas

```
class Studentas extends Asmuo {  
    protected String studentoPazymejimoNr;  
    public Studentas(String vardas, String pavarde,  
                      String asmensKodas,  
                      String studentoPazymejimoNr) {  
        super(vardas, pavarde, asmensKodas);  
        this.studentoPazymejimoNr = studentoPazymejimoNr;  
    }  
    public String getStudentoPazymejimoNr() {  
        return studentoPazymejimoNr;  
    }  
    public void setStudentoPazymejimoNr(String studentoPazymejimoNr) {  
        this.studentoPazymejimoNr = studentoPazymejimoNr;  
    }  
}
```

Ši klasė paveldi visas klasės Asmuo savybes. Klasė turi vardą, pavardę, asmens kodą ir studento pažymėjimo numerį.

Norėdami iškviesti tėvinės klasės konstruktorių rašome **super()**

Klasė Darbuotojas

```
class Darbuotojas extends Asmuo {  
    protected String socDraudimoNr;  
    public Darbuotojas(String vardas, String pavarde,  
                        String asmensKodas,  
                        String socDraudimoNr) {  
        super(vardas, pavarde, asmensKodas);  
        this.socDraudimoNr = socDraudimoNr;  
    }  
    public String getSocDraudimoNr() {  
        return socDraudimoNr;  
    }  
    public void setSocDraudimoNr(String socDraudimoNr) {  
        this.socDraudimoNr = socDraudimoNr;  
    }  
}
```

Ši klasė paveldi visas klasės Asmuo savybes. Klasė turi vardą, pavardę, asmens kodą ir socialinio draudimo numerį.

Klasė ImonesXDarbuotojas

```
class ImonesXDarbuotojas extends Darbuotojas {  
    protected String xNr;  
    public ImonesXDarbuotojas(String vardas, String pavarde,  
                                String asmensKodas,  
                                String socDraudimoNr,  
                                String xNr) {  
        super(vardas, pavarde, asmensKodas, socDraudimoNr);  
        this.xNr = xNr;  
    }  
    public String getxNr() {  
        return xNr;  
    }  
    public void setxNr(String xNr) {  
        this.xNr = xNr;  
    }  
}
```

Ši klasė paveldi visas klasės Darbuotojas savybes. Klasė turi var pavardę, asmens kodą, socialinio draudimo numerį ir x numerį.

```
Asmuo asmuo = new Asmuo("Petras", "Petraitis", "39901010000");
System.out.println(String.format("Asmuo: %s %s %s", asmuo.getVardas(),
                                asmuo.getPavarde(), asmuo.asmensKodas));

Studentas studentas = new Studentas("Antanas", "Antanaitis", "39901011111", "123456");
System.out.println(String.format("Studentas: %s %s %s %s", studentas.getVardas(),
                                studentas.getPavarde(), studentas.asmensKodas,
                                studentas.getStudentoPazymejimoNr()));

Darbuotojas darbuotojas = new Darbuotojas("Mantas", "Mantauskas", "39901012222", "333");
System.out.println(String.format("Darbuotojas: %s %s %s %s", darbuotojas.getVardas(),
                                darbuotojas.getPavarde(), darbuotojas.asmensKodas,
                                darbuotojas.getSocDraudimoNr()));

ImonesXDarbuotojas xDarbuotojas = new ImonesXDarbuotojas("Onute", "Mantauskiene",
                                                           "49901012222", "444", "X000");
System.out.println(String.format("ImonesXDarbuotojas: %s %s %s %s %s",
                                xDarbuotojas.getVardas(),
                                xDarbuotojas.getPavarde(),
                                xDarbuotojas.getAsmensKodas(),
                                xDarbuotojas.getSocDraudimoNr(),
                                xDarbuotojas.getxNr()));
```

Asmuo: Petras Petraitis 39901010000

Studentas: Antanas Antanaitis 39901011111 123456

Darbuotojas: Mantas Mantauskas 39901012222 333

ImonesXDarbuotojas: Onute Mantauskiene 49901012222

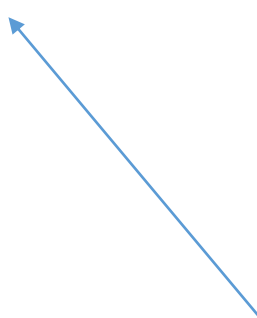
444 X000

Metodų užklotis

- Turime tėvinę klasę, kuri turi metodą X. Turime vaikinę klasę, kuri paveldi visą funkcionalumą iš tėvinės klasės. Paveldi ir X metodą. Bet mes norime, kad vaikinės klasės X metodas elgtųsi kitaip. Tada vaikinėje klasėje tą metodą taip pat apsirašome su tuo pačiu metodo pavadinimu, ir tais pačiais metodo parametrais. Pridedame anotaciją **@Override**. Tai reiškia, kad metodą užklojome (*overriding*).

Klasė Asmuo

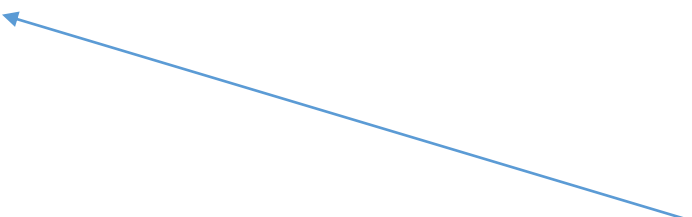
```
class Asmuo {  
    protected String vardas;  
    protected String pavarde;  
    protected String asmensKodas;  
    public Asmuo(String vardas, String pavarde, String asmensKodas) {  
        this.vardas = vardas;  
        this.pavarde = pavarde;  
        this.asmensKodas = asmensKodas;  
    }  
    public String getInfo() {  
        return String.format("%s-%s-%s", vardas, pavarde, asmensKodas);  
    }  
}
```



Metodas getInfo() grąžina suformuotą String eilutę iš klasės Asmuo kintamųjų reikšmių

Klasė Studentas

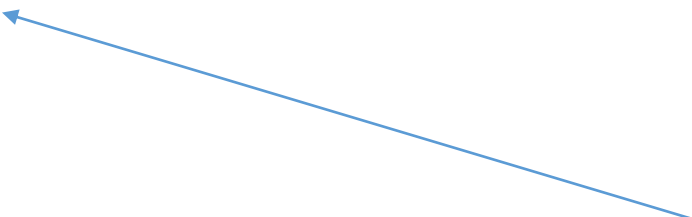
```
class Studentas extends Asmuo {  
    protected String studentoPazymejimoNr;  
    public Studentas(String vardas, String pavarde,  
                      String asmensKodas,  
                      String studentoPazymejimoNr) {  
        super(vardas, pavarde, asmensKodas);  
        this.studentoPazymejimoNr = studentoPazymejimoNr;  
    }  
    @Override  
    public String getInfo() {  
        return String.format("%s", studentoPazymejimoNr);  
    }  
}
```



Metodas getInfo() užkloja iš klasės Asmuo paveldėtą metodą su tuo pačiu pavadinimu

Klasė Darbuotojas

```
class Darbuotojas extends Asmuo {  
    protected String socDraudimoNr;  
    public Darbuotojas(String vardas, String pavarde,  
                        String asmensKodas, String socDraudimoNr) {  
        super(vardas, pavarde, asmensKodas);  
        this.socDraudimoNr = socDraudimoNr;  
    }  
    @Override  
    public String getInfo() {  
        return String.format("%s", socDraudimoNr);  
    }  
}
```



Metodas getInfo() užkloja iš klasės
Asmuo paveldėtą metodą su tuo pačiu
pavadinimu

Klasė ImonesXDarbuotojas

```
class ImonesXDarbuotojas extends Darbuotojas {  
    protected String xNr;  
    public ImonesXDarbuotojas(String vardas, String pavarde,  
                               String asmensKodas,  
                               String socDraudimoNr, String xNr) {  
        super(vardas, pavarde, asmensKodas, socDraudimoNr);  
        this.xNr = xNr;  
    }  
    @Override  
    public String getInfo() {  
        return String.format("%s", xNr);  
    }  
}
```

Metodas getInfo() užkloja iš
klasės
Darbuotojas paveldėtą metodą
su tuo pačiu pavadinimu

```
Asmuo asmuo = new Asmuo("Petras", "Petraitis", "39901010000");  
System.out.println(asmuo.getInfo());
```

```
Studentas studentas = new Studentas("Antanas", "Antanaitis", "39901011111",  
                                     "123456");  
System.out.println(studentas.getInfo());
```

```
Darbuotojas darbuotojas = new Darbuotojas("Mantas", "Mantauskas",  
                                           "39901012222", "333");  
System.out.println(darbuotojas.getInfo());
```

```
ImonesXDarbuotojas xDarbuotojas = new ImonesXDarbuotojas("Onute",  
                                                           "Mantauskiene", "49901012222",  
                                                           "444", "X000");  
System.out.println(xDarbuotojas.getInfo());
```

```
Petras-Petraitis-39901010000  
123456  
333  
X000
```

Vaikinės klasės metodas kviečia tėvinės klasės metodą

- Galime vaikinėje klasėje išreikštinau iškviešti tėvinės klasės metodą
- Tam naudojame ***super***
- Į kiekvienos išvestinės klasės getInfo metodą pridėkime
 - `super.getInfo()`
 - Pvz. `return String.format("%s-%s", super.getInfo(), studentoPazymejimoNr);`
- *Tada programa spausdins:*
 - Petras-Petraitis-39901010000
 - Antanas-Antanaitis-39901011111-123456
 - Mantas-Mantauskas-39901012222-333
 - Onute-Mantauskiene-49901012222-444-X000

```
Asmuo asmuo = new Asmuo(...)  
Studentas studentas = new Studentas(...)  
Darbuotojas darbuotojas = new Darbuotojas(...)  
ImonesXDarbuotojas xDarbuotojas = new ImonesXDarbuotojas(...)
```

galime pakeisti į

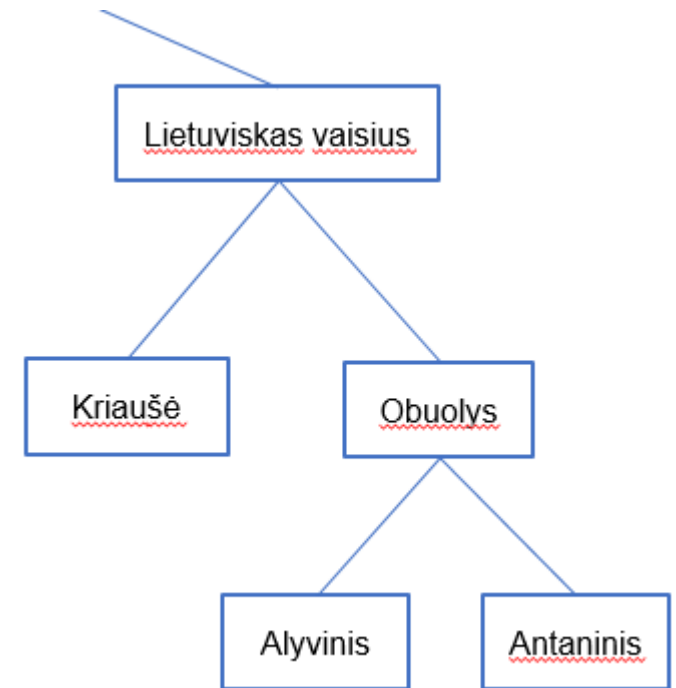
```
Asmuo asmuo = new Asmuo(...)  
Asmuo studentas = new Studentas(...)  
Asmuo darbuotojas = new Darbuotojas(...)  
Asmuo xDarbuotojas = new ImonesXDarbuotojas(...)
```

ir programos veikimas nepasikeis, nes studentas yra tuo pačiu asmuo, xDarbuotojas irgi yra asmuo. Kai kviesime getInfo() metodą Java supras, kad reikia kviesti vaikinės klasės metodą, jei jis yra užklotas.

Objekto tipo nustatymas *instanceof*

```
Alyvinis alyvObuol = new Alyvinis();  
alyvObuol instanceof Alyvinis // true  
alyvObuol instanceof Obuolys // true  
alyvObuol instanceof String // false
```

```
Obuolys obuol = new Obuolys();  
obuol instanceof Obuolys // true  
Obuol instanceof Alyvinis // false
```



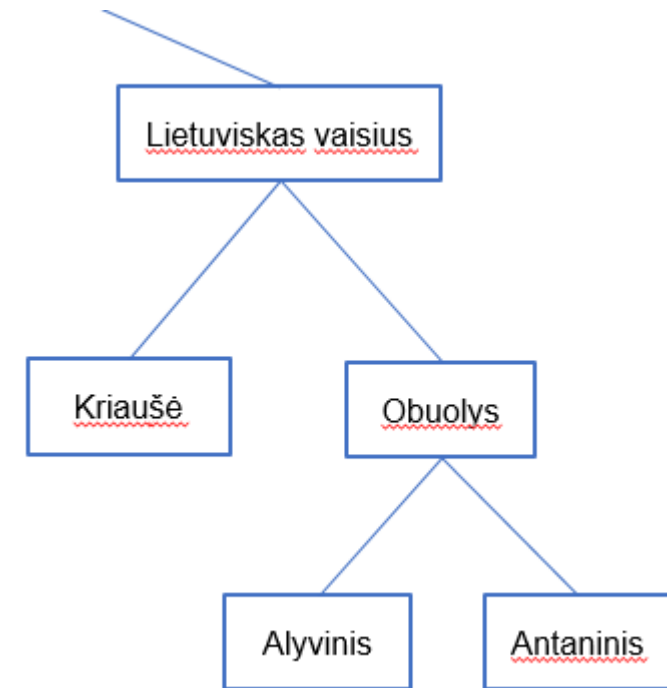
Tipo keitimas *cast*

Obuolys obuol = new Obuolys;

LietuviskasVaisius lietVai = (LietuviskasVaisius) obuol;

~~Alyvinis alyv = (Alyvinis) obuol;~~

~~String eilute = (String) obuol;~~



Paveldėjimas iš *java.lang.Object*

- Klasė ***Object*** taip pat turi savo metodų ir juos galime užkloti (*override*)
- ***toString*** - grąžina tekstinę objekto informaciją
- ***clone*** – sukuria naują objektą - to paties objekto kopiją
- ***equals*** – lygina objektų turinį

```
Object
  Object()
  registerNatives() : void
  getClass() : Class<?>
  hashCode() : int
  equals(Object) : boolean
  clone() : Object
  toString() : String
  notify() : void
  notifyAll() : void
  wait(long) : void
  wait(long, int) : void
  wait() : void
  finalize() : void
```

Object.toString()

```
public class Object {  
  
    ...  
  
    public String toString() {  
        return getClass().getName() + "@" + Integer.toHexString(hashCode());  
    }  
  
    ...  
}
```

Object.toString() užklojimas

```
public class Koordinate {  
  
    private int x;  
    private int y;  
  
    public Koordinate(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int getX() {  
        return x;  
    }  
    public void setX(int x) {  
        this.x = x;  
    }  
    public int getY() {  
        return y;  
    }  
    public void setY(int y) {  
        this.y = y;  
    }  
  
    @Override  
    public String toString() {  
        return "Koordinate [x=" + x + ", y=" + y + "];"  
    }  
}
```

```
public class Test {  
  
    public static void main(String[] args) {  
  
        ArrayList<Koordinate> koordinaciuSarasas = new ArrayList<>();  
  
        koordinaciuSarasas.add(new Koordinate(1, 2));  
        koordinaciuSarasas.add(new Koordinate(0, 0));  
        koordinaciuSarasas.add(new Koordinate(3, 5));  
  
        System.out.println(koordinaciuSarasas);  
  
    }  
}  
  
// vietoje tokio teksto  
// [Koordinate@7852e922, Koordinate@4e25154f, Koordinate@70dea4e]  
// spausdins toki tekstą  
// [Koordinate [x=1, y=2], Koordinate [x=0, y=0], Koordinate [x=3, y=5]]
```

Object.equals() užklojimas

```
public class Koordinate {  
  
    private int x;  
    private int y;  
  
    public Koordinate(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int getX() {  
        return x;  
    }  
  
    public void setX(int x) {  
        this.x = x;  
    }  
  
    public int getY() {  
        return y;  
    }  
  
    public void setY(int y) {  
        this.y = y;  
    }  
  
    @Override  
    public boolean equals(Object o) {  
        if (o instanceof Koordinate) {  
            Koordinate ko = (Koordinate) o;  
            if (x == ko.getX() && y == ko.getY()) {  
                return true;  
            }  
        }  
        return false;  
    }  
}
```

```
public class Test {  
  
    public static void main(String[] args) {  
  
        Koordinate koordinate1 = new Koordinate(2, 2);  
        Koordinate koordinate2 = new Koordinate(2, 2);  
  
        System.out.println(koordinate1.equals(koordinate2));  
  
    }  
}  
  
// vietoje false spausdins true
```

Object.clone() užklojimas

```
public class Koordinate {  
  
    private int x;  
    private int y;  
  
    public Koordinate(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int getX() {  
        return x;  
    }  
    public void setX(int x) {  
        this.x = x;  
    }  
    public int getY() {  
        return y;  
    }  
    public void setY(int y) {  
        this.y = y;  
    }  
  
    @Override  
    public Koordinate clone() {  
        return new Koordinate(x, y);  
    }  
}
```

```
public class Test {  
  
    public static void main(String[] args) {  
  
        Koordinate koordinateOriginali = new Koordinate(2, 2);  
        Koordinate koordinateKlonuota = koordinateOriginali.clone();  
  
        System.out.println("Originali koordinate: "  
                            + koordinateOriginali.getX()  
                            + ";" + koordinateOriginali.getY());  
  
        System.out.println("Originali klonuota: "  
                            + koordinateKlonuota.getX()  
                            + ";" + koordinateKlonuota.getY());  
  
    }  
  
    // Originali koordinate: 2;2  
    // Originali klonuota: 2;2  
}
```



Abstrakti klasė

Kas yra abstrakti klasė?

- Abstrakti klasė yra tokia klasė, kuri yra pažymėta žodžiu **abstract**.
- **abstract class Animal {}**
- Tokia klasė gali (bet nebūtinai) turėti abstrakčių metodų, kurie neturi kūno.
- Įprasta (ne abstrakti) klasė negali turėti abstrakčių metodų. Visi jos metodai privalo būti su kūnu
- Jei **Animal** yra abstrakti klasė, tai jos pagrindu negalime kurti objekto, t.y. negalime daryti **new Animal()**;

Abstrakti klasė

- Sakykime turime abstrakčią klasę Gyvunas su abstrakčiu metodu `skleidžiamasGarsas()`;
- Ir turime klases Suo, Katinas ir Bite, kurių tėvinė klasė yra Gyvūnas.
- Tada šios klasės privalo užkloti tėvinės klasės abstraktų metodą `skleidžiamasGarsas()`;


```
abstract class Gyvunas {  
    public abstract String skleidziamasGarsas();  
}  
  
class Suo extends Gyvunas {  
    @Override  
    public String skleidziamasGarsas() {  
        return "Au";  
    }  
}  
  
class Katinas extends Gyvunas {  
    @Override  
    public String skleidziamasGarsas() {  
        return "Miau";  
    }  
}  
  
class Bite extends Gyvunas {  
    @Override  
    public String skleidziamasGarsas() {  
        return "Bzzz";  
    }  
}  
  
public class GyvunasAbstract {  
    public static void main(String[] args) {  
        Gyvunas suo = new Suo();  
        Gyvunas katinas = new Katinas();  
        Gyvunas bite = new Bite();  
  
        System.out.println("Suns garsas: " + suo.skleidziamasGarsas());  
        System.out.println("Katino garsas: " + katinas.skleidziamasGarsas());  
        System.out.println("Bites garsas: " + bite.skleidziamasGarsas());  
    }  
}
```

Suns garsas: Au
Katino garsas: Miau
Bites garsas: Bzzz

Abstrakčios klasės taisyklės

- Kai klasės metodo realizacija nėra žinoma ar negali būti įgyvendinama toje klasėje, tada reikia klasę deklaruoti kaip **abstract** ir aprašyti abstraktų metodą.
- Paveldinčios klasės privalo realizuoti tėvinėje klasėje esantį abstraktų metodą.
- Jei vaikinė klasė metodo nerealizuoja, tada ji taip pat privalo būti deklaruota kaip abstrakti.
- Abstraktus metodas neturi kūno
- Abstraktaus metodo aprašas visada baigiasi kabliataškiu ;



Interfeisas

Kas yra Interfeisas?

- Interfeisas yra pilna abstrakcija
- Interfeisas atrodo kaip klasė, bet nėra klasė.
- Interfeisas gali turėti tik metodų aprašus, t.y. be kūno (lygiai taip, kaip abstraktūs metodai abstrakčiose klasėse).
- Interveiso metodai visada būna public, todėl žodžio public galime nerašyti.
- Visi interfeiso kintamieji gali būti tik public static final, t.y. interfeisas gali turėti tik konstantas.
- Interfeisas žymimas žodžiu interface
 - `interface Animal {...}`
- Kaip ir abstrakčiai klasei, taip ir interfeiso pagrindu negalime kurti objekto, ty negalime sakyti `new Animal()`; kai Animal yra interfeisas.

Interfeisas

- Interfeise gali būti tik tokie nariai:
 - Laukai
 - Abstraktūs metodai
- Kiekvienas interfeiso laukas (by default) yra **public static final**, todėl nėra būtina rašyti šių modifikatorių
- Laukų reikšmių negalima keisti, todėl tai yra konstantos
- Kiekvienas interfeiso metodas (by default) yra **public abstract**, todėl nėra būtina rašyti šių modifikatorių
- Klasė gali paveldėti tik vieną klasę, bet tokia pati klasė gali implementuoti kelis interfeisus
- Interfeisas gali paveldėti kitus interfeisus