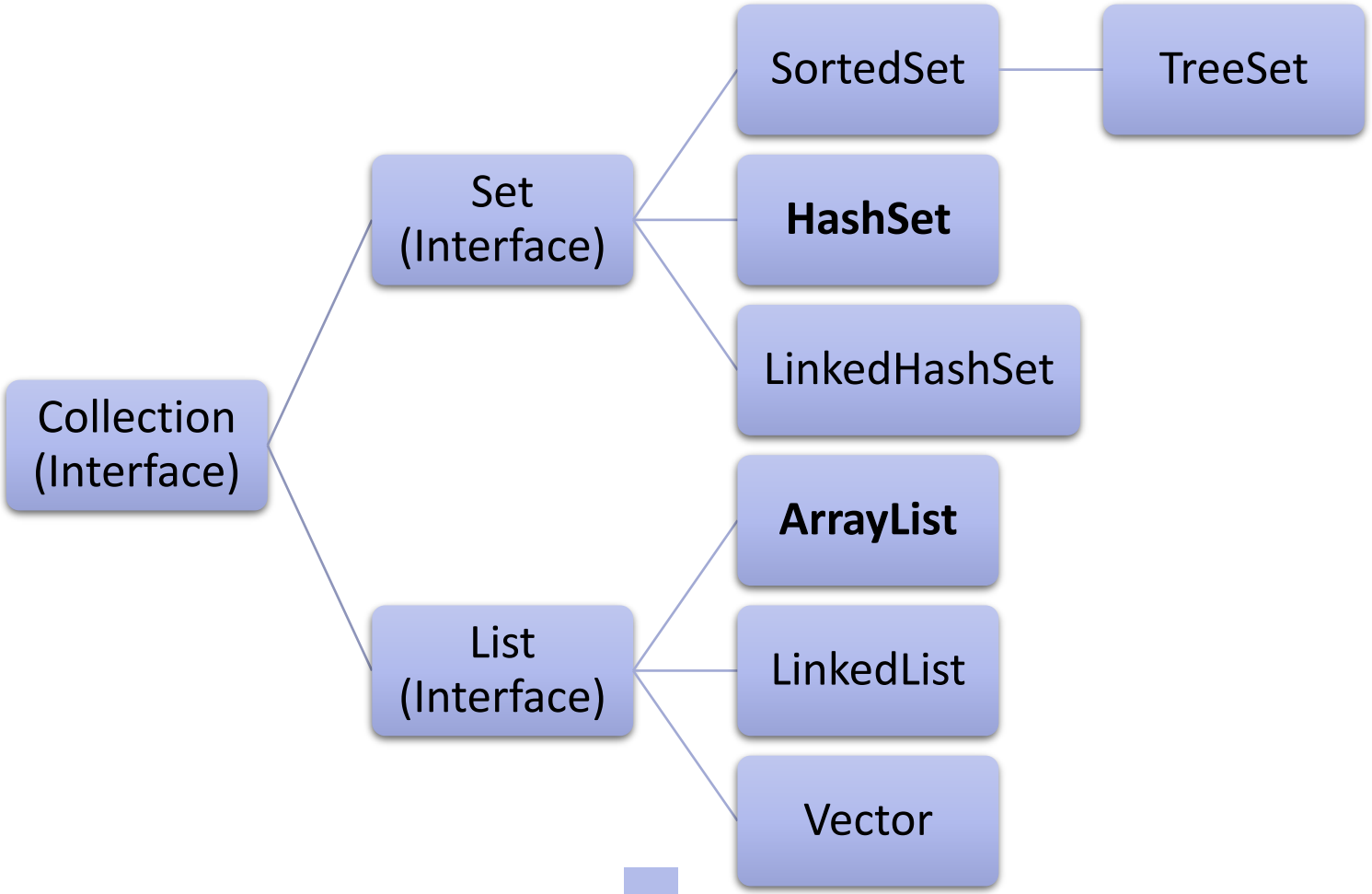
Programazioa

Datu-egitura dinamikoak

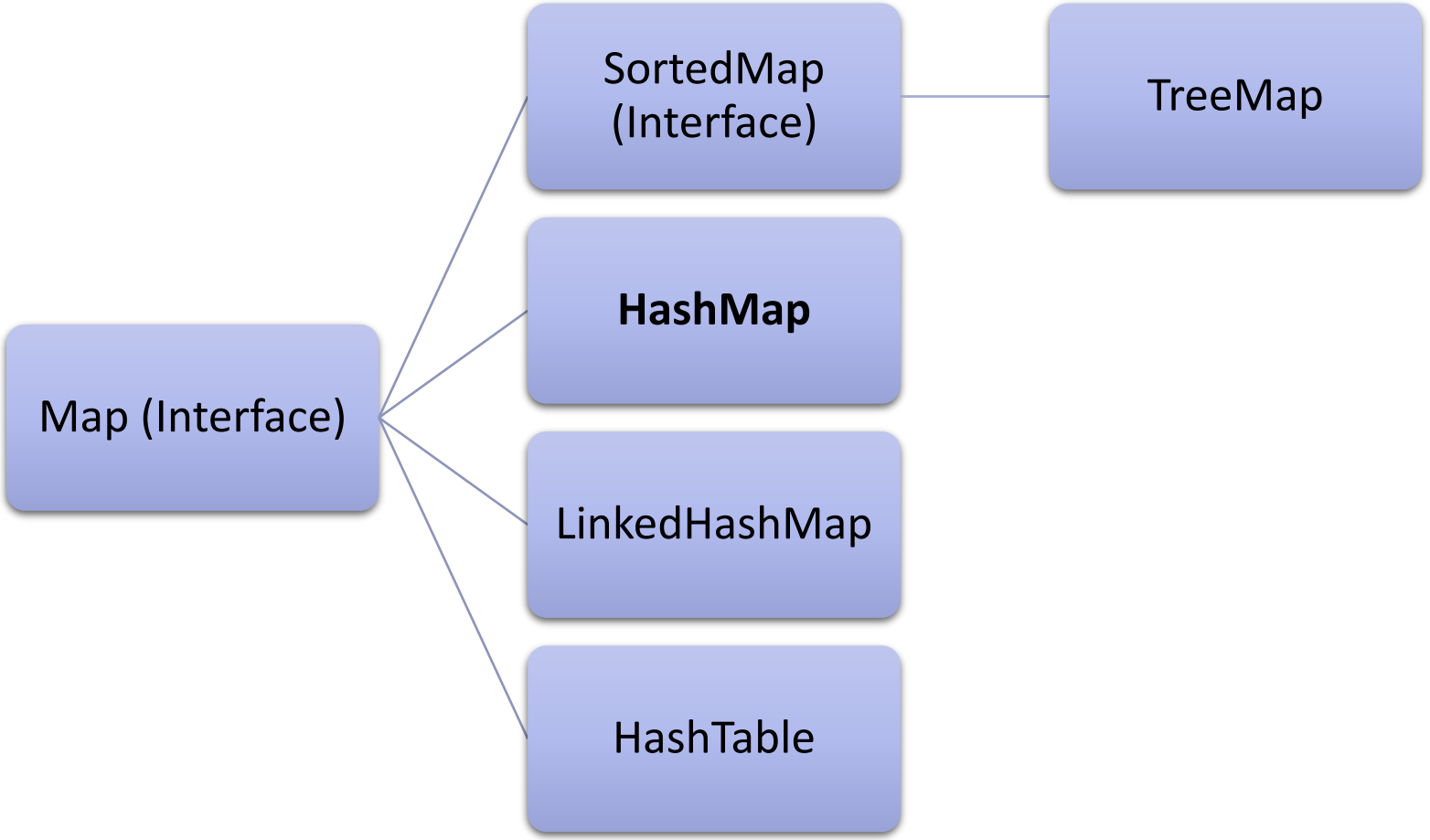
Bildumak

- Bildumen azpiegiturak bere barnean dinamikoki objektuak antolatzen dituzten zenbait interfaze eta klase hartzen ditu
  - Egitura batzuk ordenatuta daude eta beste batzuk ez, batzuk elementu bikoiztuak onartzen dituzte eta beste batzuk ez…
- Interfaze eta klase hauek beraien gainean lan egiteko metodo ugari eskaintzen dituzte
  - Bilatzeko, ordenatzeko, gehitu eta ezabatzeko…
- Bilduma gehienak ***Collection*** izeneko interfazetik eratortzen dira
- Mapak beste interfaze batetik eratortzen dira, baina praktikan bildumen parte direla onartzen da

Bildumak

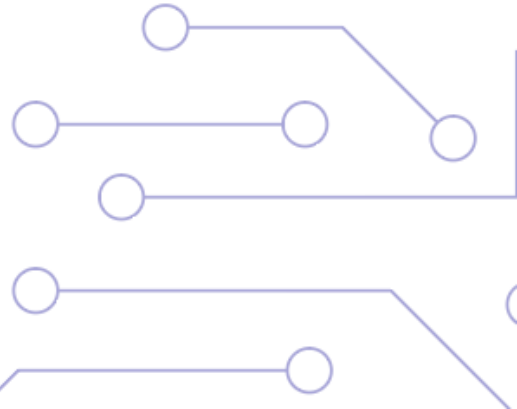

Collection (Interface)
Set (Interface)
List (Interface)
SortedSet
TreeSet
HashSet
LinkedHashSet
ArrayList
LinkedList
Vector

Bildumak


Map (Interface)
SortedMap (Interface)
TreeMap
HashMap
LinkedHashMap
HashTable

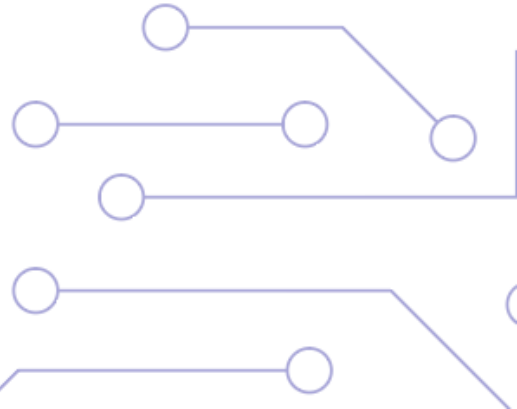
Bildumak

- Bilduma mota bakoitzak bere ezaugarriak ditu:
  - Zerrendak (*List*): Ordenak garrantzia duenerako
    - Objektuen indizea kontrolatzen dute
    - Badakite objektuak egitura barruan non dauden
    - Elementu errepikatuak onartzen ditu
  - Multzoak (*Set*): Bakarra izateak garrantzia duenerako
    - Ez dute elementu bikoizturik onartzen
    - Badakite multzoan objektu jakin bat dagoen ala ez



Bildumak

- Bilduma mota bakoitzak bere ezaugarriak ditu:
  - Mapak (*Map*): Objektu bat gako baten bidez bilatzea behar denerako
    - Gako-balio pareak gordetzen ditu
    - Gako bat emanda honi dagokion balioa zein den badakite
    - Ezin dira gako bikoiztuak izan, baina balioak bai bikoiztuak egon daitezke eta gako desberdinekin lotuta egon
    - Normalean gakoak karaktere-kate motakoak dira



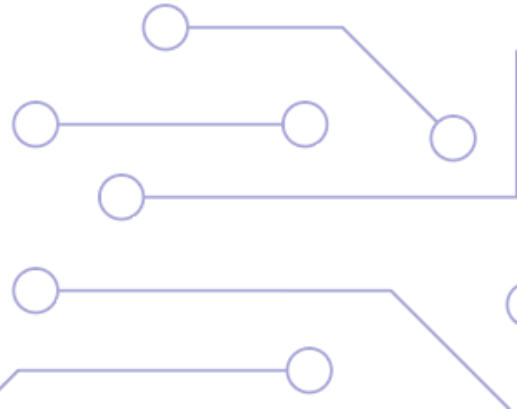
Edukiontzi klaseak (*wrapper*)

- Datu-mota primitiboak objektuak izango balira bezala erabiltzea ahalbidetzen dute
- Datu-mota primitibo bakoitzeko edukiontzi klase bat dago

Primitiboa	Edukiontzia
boolean	Boolean
char	Character
byte	Byte
int	Integer
long	Long
float	Float
double	Double

Edukiontzi klaseak (*wrapper*)

- Bi abantaila nagusi ematen dituzte:
  - Bildumetan erabili daitezke.
  - Balio hutsa (*null*) gorde dezakete. Hau oso erabilgarria izan daiteke konparaketak eta bilaketak egiterako orduan.
- Honetaz aparte, objektu izateagatik zenbait metodo dituzte erabilgarri
  - toString, equals, compareTo…



Edukiontzi klaseak (*wrapper*)

- Datu-mota primitibo bat dagokion edukiontzi objektuan gordetzeko balio primitiboa valueOf() metodoa erabili daiteke

```
int osoa = 100;
Integer obj = Integer.valueOf(osoa);
System.out.println(osoa + " - " + obj);
```

- Oraingo Javan ez da beharrezkoa, zuzenean primitiboa objektuari esleitu daiteke

```
Integer obj = osoa;
```

- Zenbakiak karaktere-kate bihurtzeko ere metodo bera erabiltzen da

Edukiontzi klaseak (*wrapper*)

- Edukiontzi baten balio primitiboa lortzeko intValue, floatValue… metodoak erabiltzen dira

```
Integer obj = 100;
int osoa = obj.intValue();
System.out.println(osoa + " - " + obj);
```

- Oraingo Javan ez da beharrezkoa

```
int osoa = obj;
```

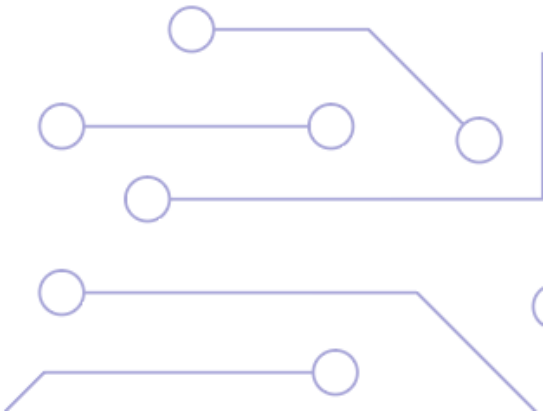
- Zenbaki moten arteko transformazioak egiteko ere erabili daitezke

```
Integer obj = 100;
Float hamartarObj = obj.floatValue();
```

Gogoratu: Bildumetan ezin dira balio primitiboak gorde

List interfazea

- Zerrenda bateko elementuak ordenatuta daude eta elementu bakoitzaren kokapena (indizea) ezaguna da
- Tamaina dinamikoa du, elementuak gehitu eta ezabatuz bere tamaina aldatzen da
- Erabiltzaileak indizea erabilita posizio zehatz bateko elementua jaso dezake
- Elementuak bilatu daitezke
- Elementuak bikoiztuta egon daitezke



List interfazea

- Interfazeko metodo batzuk:

Metodoa	Funtzionamendua
boolean add(Object obj)	Elementua zerrendaren bukaeran gehitzen du
void add(int pos, Object obj)	Elementua adierazitako posizioan gehitzen du. Atzerago dauden elementuak desplazatzen ditu.
Object set(int pos, Object obj)	Elementua adierazitako posizioan gordetzen du. Bertan zegoen objektua itzultzen du.
Object get(int pos)	Adierazitako posizioan dagoen objektua itzultzen du.
Object remove(int pos)	Adierazitako posizioan dagoen objektua ezabatu eta itzultzen du.
boolean remove(Object obj)	Adierazitako objektua ezabatzen du (baldin badago) eta ezabatu duen itzultzen du.

List interfazea

- Interfazeko metodo batzuk:

Metodoa	Funtzionamendua
int size()	Zerrendaren elementu kopurua itzultzen du.
boolean isEmpty()	Zerrenda hutsa dagoen itzultzen du.
int indexOf(Object obj)	Adierazitako objektuaren lehen agerpenaren indizea itzultzen du. -1 ez badago.
boolean contains(Object obj)	Adierazitako objektua zerrendan dagoen ala ez itzultzen du.

Metodo gehiago: https://docs.oracle.com/javase/8/docs/api/java/util/List.html

ArrayList klasea

- *List* interfazea inplementatzen duen klaseetako bat, tamaina dinamikoko array bat definitzen du
- Array-aren tamaina barnean dituen elementuen kopuruaren araberakoa da
  - Elementuak gehitzean tamaina handitzen da eta elementuak ezabatzean tamaina txikitzen da
- Ausazko atzipenak (indize jakin bat atzitzea) egiteko oso egitura aproposa da
- Elementuak gehitzea eta ezabatzea nahiko ekintza motelak dira

ArrayList klasea

- Klase honek eraikitzaile bakarra du, array-a hutsean hasieratzen duena
  - Hasierako tamaina 0 izango da hutsik dagoelako
  - Eraikitzailean gordeko diren objektuen mota adieraztea komenigarria da

```
ArrayList<Integer> nireArray = new ArrayList<>();
```

- Elementu bat bukaeran gehitzeko add() erabili daiteke

```
nireArray.add(10);
```

- Ezabatzea remove() metodoarekin egin daiteke, indizea edo elementua emanez

```
nireArray.remove(Integer.valueOf(10));
nireArray.remove(0);
```

ArrayList klasea

```
ArrayList<Integer> zenbakiak = new ArrayList<>();
zenbakiak.add(10);
zenbakiak.add(20);
zenbakiak.add(30);
zenbakiak.add(40);
int batura = 0;
for (int i=0;i< zenbakiak.size();i++)
{
    batura = batura + zenbakiak.get(i);
}
System.out.println(batura);
```

Iterator interfazea

- *for* egitura bat erabiliz zerrenda bat aztertu badaiteke ere, iteratzaile (*iterator*) objektu bat ere erabili daiteke
  - Zenbait ekintzetarako egokiagoa da, adibidez, elementuak ezabatzeko
- Metodo erabilienak:

Metodoa	Funtzionamendua
Object next()	Iteratzailearen hurrengo elementua bueltatzen du.
boolean hasNext()	Hurrengo elementu bat dagoen itzultzen du. Bukaera kontrolatzeko.
void remove()	Itzulitako azken elementua ezabatzen du.

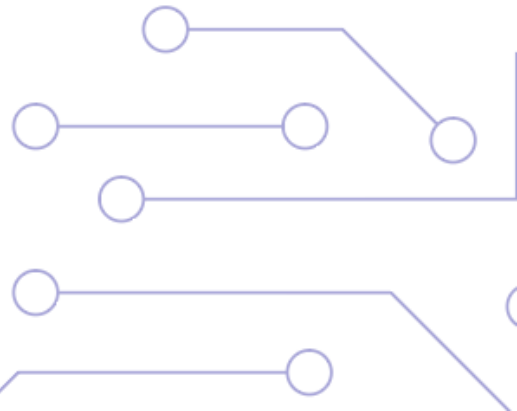
Iterator interfazea

- Iteratzailearen sorrera bilduma batean oinarritzen da

```
Iterator<ObjectType> it = list.iterator();
```

- Iteratzailea aztertzeko *while* egitura erabiltzen da, ez *for*

```
while (it.hasNext())
{
  ObjectType obj = it.next();
}
```



Iterator klasea

```
ArrayList<Integer> zenbakiak = new ArrayList<>();
zenbakiak.add(10);
zenbakiak.add(20);
zenbakiak.add(30);
zenbakiak.add(40);
int batura = 0;
Iterator<Integer> it = zenbakiak.iterator();
while(it.hasNext())
{
    batura = batura + it.next();
}
System.out.println(batura);
```

Kontuz: zerrenda bateko elementuak ezabatzeko *for* egitura erabiltzeak gaizki funtziona dezake tamaina dinamikoki aldatzen delako

Iterator klasea

```
ArrayList<Integer> zenbakiak = new ArrayList<>();
zenbakiak.add(10);
zenbakiak.add(10);
zenbakiak.add(20);
zenbakiak.add(30);
System.out.println(zenbakiak);
for (int i = 0; i < zenbakiak.size(); i++)
{
    if (zenbakiak.get(i) == 10)
    {
        zenbakiak.remove(i);
    }
}
System.out.println(zenbakiak);
```

Ez du prozesua ondo egiten array-aren tamaina dinamikoki aldatzen delako

Iterator klasea

```
ArrayList<Integer> zenbakiak = new ArrayList<>();
zenbakiak.add(10);
zenbakiak.add(10);
zenbakiak.add(20);
zenbakiak.add(30);
System.out.println(zenbakiak);
Iterator<Integer> it = zenbakiak.iterator();
while(it.hasNext())
{
    if (it.next() == 10)
    {
        it.remove();
    }
}
System.out.println(zenbakiak);
```

ListIterator interfazea

- *Iterator* interfazearen hedapen bat da, metodo berriak gehitzen ditu:
  - Atzetik aurrera aztertzeko aukera
  - Uneko indizea jakitea
- Metodo erabilienak:

Metodoa	Funtzionamendua
Object previous()	Aurreko elementua itzultzen du.
boolean hasPrevious()	Aurreko elementu bat dagoen itzultzen du. Bukaera kontrolatzeko.
int previousIndex()	Aurreko elementuaren indizea itzultzen du.
int nextIndex()	Hurrengo elementuaren indizea itzultzen du.

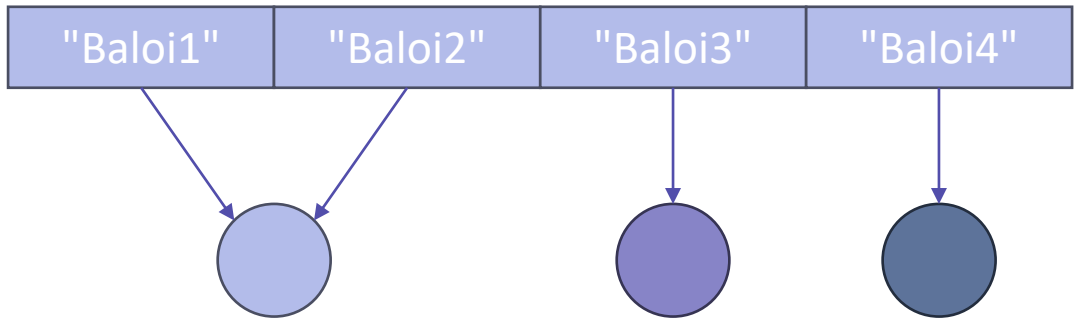
ListIterator interfazea

- Metodo erabilienak:

Metodoa	Funtzionamendua
void add(Object obj)	Pasatutako objektua gehitzen du aurreko elementuaren atzetik (baldin badago) eta hurrengo elementuaren aurretik (baldin badago). Hau da, *previous* eta *next* metodoek itzuliko zituzten elementuen artean.
void set(Object obj)	Itzuli duen azken elementua pasatutako elementuarengatik aldatzen du.

Map interfazea

- Mapak hiztegi baten moduan egiten du lan
  - Ez dago indizerik
- Elementu bakoitza bi objektuengatik dago osatuta: gakoa eta balioa
  - Biak objektuak dira (normalean gakoa String motakoa da)
- Gakoak ezin dira bikoiztuta egon, baina balioak bai
  - Hau da, gerta daiteke bi gako ezberdinek objektu berdinari erreferentzia egitea



Map interfazea

- Metodo erabilienak:

Metodoa	Funtzionamendua
Object put(Object gako, Object balio)	Gako-balio bikotea mapan gehitzen du. Dagoeneko gakoa erabiltzen bada, balioa berridazten da.
Object remove(Object gako)	Adierazitako gakoa duen gako-balio bikotea ezabatzen du (existitzen bada) eta balioa itzultzen du.
Object get(Object gako)	Adierazitako gakoari dagokion balioa itzultzen du, *null* ez bada existitzen.
int size()	Maparen gako-balio bikoteen kopurua itzultzen du.
boolean isEmpty()	Mapa hutsik dagoen ala ez itzultzen du.

Map interfazea

- Metodo erabilienak:

Metodoa	Funtzionamendua
boolean containsKey(Object gako)	Adierazitako gakoa mapan dagoen itzultzen du.
boolean containsValue(Object balio)	Adierazitako balioa mapan behin edo gehiagotan dagoen itzultzen du.
Set keySet()	Mapan gordeta dauden gakoen multzoa itzultzen du
Collection values()	Mapan gordeta dauden balioen bilduma itzultzen du.

Metodo gehiago: https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

HashMap klasea

- *Map* interfazearen inplementazio erabiliena da
- Gako-balio bikoteak gordetzen ditu
  - Normalean gakoak String motakoak dira
- Gako bidezko bilaketak oso azkarrak dira
- Lehenetsitako eraikitzailean gako eta balioaren motak adieraztea komenigarria da

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

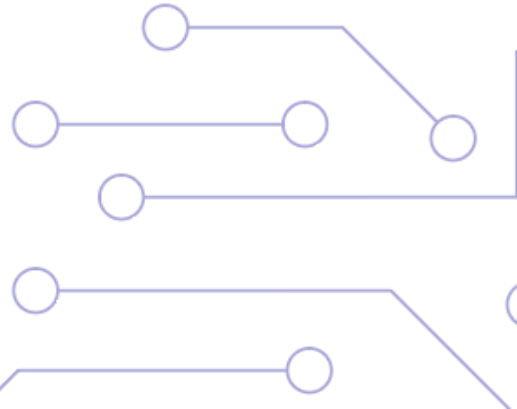
HashMap klasea

```
HashMap<String, Integer> puntuazioak = new HashMap<>();
puntuazioak.put("Ander", 42);
puntuazioak.put("Miren", 343);
puntuazioak.put("Amaia", 125);
System.out.println(puntuazioak);
System.out.println(puntuazioak.get("Miren"));
```

HashMap klasea

- Mapa bat guztiz aztertu nahi bada lehenik gakoen multzoa jaso behar da *getKeys()* metodoa erabilita eta ondoren gako hauek aztertu behar dira iteratzaile bat erabiliz, adibidez

```
HashMap<String,Produktu> nireMapa = new HashMap<>();
...
Iterator<String> it = nireMapa.keySet().iterator();
while (it.hasNext()) {
  ...
}
```



HashMap klasea

```
HashMap<String,Produktu> nireMapa = new HashMap<>();
nireMapa.put("k1", new Produktu(1, "M1"));
nireMapa.put("k2", new Produktu(2, "M2"));
nireMapa.put("k3", new Produktu(3, "M3"));
Iterator<String> it = nireMapa.keySet().iterator();
while (it.hasNext()) {
  String gakoa = it.next();
  Produktu p = nireMapa.get(gakoa);
  System.out.println("Gakoa: " + gakoa + ", Produktua: " + p.toString());
}
```

Set interfazea

- Multzoa errepikatzen ez diren elementuen bilduma bat definitzeko erabiltzen da
  - *equals* metodoak garrantzia handia hartzen du, hau erabiliko baita elementu bat multzoan dagoen ala ez identifikatzeko
  - Multzoan dagoen elementu bat gehitzen saiatzen bada ez du gehituko
- Elementuak ez daude ordenatuta eta ez dute indizerik
  - Edukia inprimatzen bada aterako den ordena desberdina izan daiteke exekuzio bakoitzean
- Multzoaren tamaina dinamikoki aldatzen da elementuak gehitu eta ezabatzen diren heinean

Set interfazea

- Metodo erabilienak:

Metodoa	Funtzionamendua
boolean add(Object obj)	Elementua multzoan gehitzen du baldin eta ez badago
boolean remove(Object obj)	Adierazitako objektua ezabatzen du (baldin badago) eta ezabatu duen itzultzen du.
boolean contains(Object obj)	Adierazitako objektua multzoan dagoen ala ez itzultzen du.
int size()	Multzoaren elementu kopurua itzultzen du.
boolean isEmpty()	Multzoa hutsik dagoen ala ez itzultzen du.
void clear()	Multzoko elementu guztiak ezabatzen ditu

Set interfazea

- Aurreko metodoez gain, interfaze honek multzo matematikoen inguruko eragiketak ere definitzen ditu
  - Suposatu *m1* eta *m2* datu mota berdineko elementuak gordetzen dituzten bi multzo direla

Metodoa	Funtzionamendua
m1.addAll(m2)	m2 multzoko elementu guztiak m1 multzoan gehitzen ditu. Bildura eragiketa.
m1.retainAll(m2)	m1 eta m2 multzoetan (bietan) dauden elementuak mantentzen dira. Ebakidura eragiketa.
m1.removeAll(m2)	m2 multzoan dauden elementu guztiak m1 multzotik ezabatzen ditu. Osaketa eragiketa.

Metodo gehiago: https://docs.oracle.com/javase/8/docs/api/java/util/Set.html

HashSet klasea

- *Set* interfazearen inplementazio erabiliena da
- Ez du ordenik gordetzen, bakarrik multzoan elementu errepikatuak ez egotea kontrolatzen du
  - *SortedSet* interfazeak ordenatutako multzoa inplementatzen du
- Lehenetsitako eraikitzailean gordeko diren elementuen mota adieraztea komenigarria da

```
HashSet<Integer> nireMultzoa = new HashSet<>();
```

- Elementu guztiak aztertzeko *for-each* egitura edo iteratzaile bat erabili daiteke

HashSet klasea

```
HashSet<String> cars = new HashSet<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("BMW");
cars.add("Mazda");
System.out.println(cars);
```

HashSet klasea

```
HashSet<Herrialde> herrialdeak = new HashSet<>();
herrialdeak.add(new Herrialde("India", 1428));
herrialdeak.add(new Herrialde("Australia", 26));
herrialdeak.add(new Herrialde("Frantzia", 68));
herrialdeak.add(new Herrialde("Peru", 34));
herrialdeak.add(new Herrialde("Hego Afrika", 60));
// Elementuak for-each egitura erabiliz
for(Herrialde herri: herrialdeak)
{
    System.out.println(herri);
}
// Beste aukera bat iteratzaile bat erabiliz
Iterator<Herrialde> i = herrialdeak.iterator();
while (i.hasNext())
{
    System.out.println(i.next());
}
```

HashSet klasea

```
HashSet<Integer> m1 = new HashSet<>(Arrays.asList(1, 3, 5, 7, 9));
HashSet<Integer> m2 = new HashSet<>(Arrays.asList(2, 4, 6, 8));
System.out.println("m1 bildura aurretik: " + m1);
m1.addAll(m2);
System.out.println("m1 bildura ondoren: " + m1);
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-ja
m1 bildura aurretik: [1, 3, 5, 7, 9]
m1 bildura ondoren: [1, 2, 3, 4, 5, 6, 7, 8, 9]

Process finished with exit code 0
```

HashSet klasea

```
HashSet<Integer> m1 = new HashSet<>(Arrays.asList(1, 2, 3, 4, 5));
HashSet<Integer> m2 = new HashSet<>(Arrays.asList(2, 4, 6, 8));
System.out.println("m1 ebakidura aurretik: " + m1);
m1.retainAll(m2);
System.out.println("m1 ebakidura ondoren: " + m1);
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe
m1 ebakidura aurretik: [1, 2, 3, 4, 5]
m1 ebakidura ondoren: [2, 4]

Process finished with exit code 0
```

HashSet klasea

```
HashSet<Integer> m1 = new HashSet<>(Arrays.asList(1, 2, 3, 4, 5));
HashSet<Integer> m2 = new HashSet<>(Arrays.asList(2, 4, 6, 8));
System.out.println("m1 osaketa aurretik: " + m1);
m1.removeAll(m2);
System.out.println("m1 osaketa ondoren: " + m1);
```

```
"C:\Program Files\Java\jdk-17\bin\java
m1 osaketa aurretik: [1, 2, 3, 4, 5]
m1 osaketa ondoren: [1, 3, 5]

Process finished with exit code 0
```

Objektuen konparaketak

- Datu-egitura dinamikoen zenbait metodo elementuen arteko konparaketetan oinarritzen dira
  - *indexOf, contains, containsKey* eta *containsValue*
- Era berean, multzoetan elementuak eta mapetan gakoak errepikatuta dauden identifikatzeko ere horietan oinarritzen dira
- Ikusi dugun bezala "==" ikurrak erabiliz objektuak beraien memoriarako erreferentziagatik konparatzen dira
- *Object* klaseak eskaintzen duen *equals* metodoa erabiltzen bada, modu lehenetsian konparaketa berdina egiten du, erreferentzia bidez
- Hau ez da zenbait testuinguruetan egokia, hala nola, konparaketa objektuen atributuen arabera egin nahi bada

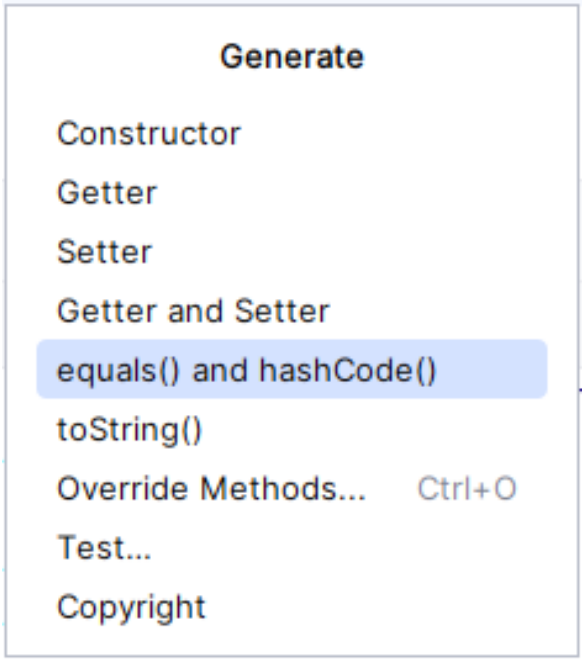
Objektuen konparaketak

```
public class Puntu
{
  private float x;
  private float y;

  public Puntu(float x, float y)
  {
    this.x = x;
    this.y = y;
  }

  @Override
  public String toString()
  {
    return x + " - " + y;
  }
}
```

```
public static void main(String[] args) {
  HashSet<Puntu> arr=new HashSet<>();
  arr.add(new Puntu(4,5));
  arr.add(new Puntu(6,7));
  arr.add(new Puntu(3,4));
  arr.add(new Puntu(5,7));
  arr.add(new Puntu(1,2));
  System.out.println(arr);
  Puntu p = new Puntu(3,4);
  System.out.println(arr.contains(p));
  arr.add(p);
  System.out.println(arr);
}
```

Objektuen konparaketak

- Soluzioa: ***equals*** eta ***hashCode*** metodoak berridaztea
- Garapen inguruneak kodea automatikoki sortzen du erabiltzaileak aukeratutako atributuetan oinarrituta



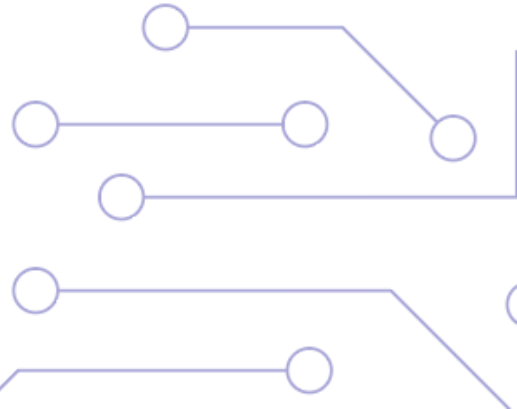
Objektuen konparaketak

```
public class Puntu
{
  ...
  @Override
  public boolean equals(Object o)
  {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Puntu puntu = (Puntu) o;
    return Float.compare(puntu.x, x) == 0 && Float.compare(puntu.y, y) == 0;
  }
  @Override
  public int hashCode()
  {
    return Objects.hash(x, y);
  }
}
```

ADI: Gogoratu bakarrik behar diren atributuak aukeratzeaz

Ordenatze algoritmoak

- Zerrenda edo bektore bateko elementuak emandako sekuentzia batean jartzeko erabiltzen dira
- Ikerketa asko egin dira
  - Problema sinplea bada ere, konputazionalki garesti atera daiteke
- Ordena duten bildumek dagoeneko badute ordenatzeko metodo bat, *sort()*
  - Konparaketa modua aukeratu daiteke *Comparator* klasetik



Ordenatze algoritmoak

- https://youtu.be/Hd5jp935ays?si=tLoXXNolxNdekqA1

Ordenatze algoritmoak

- https://youtu.be/kPRA0W1kECg?si=PzfdMeuqiDRSWmmX