



Funded by the  
Erasmus+ Programme  
of the European Union



---

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

---



# KI201 - JAVA 4: STRUKTURE PODATAKA I ALGORITMI – DEO A

## Setovi i mape

Lekcija 04

PRIRUČNIK ZA STUDENTE

# KI201 - JAVA 4: STRUKTURE PODATAKA I ALGORITMI – DEO A

## Lekcija 04

### *SETOVI I MAPE*

- ✓ Setovi i mape
- ✓ Poglavlje 1: Setovi
- ✓ Poglavlje 2: Poređenje preformansi setova i listi
- ✓ Poglavlje 3: Studija slučaja: Brojanje ključnih reči
- ✓ Poglavlje 4: Mape
- ✓ Poglavlje 5: Studija slučaja: Broj reči u tekstu
- ✓ Poglavlje 6: Singularne i nepromenljive kolekcije i mape
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

# ▼ Uvod

## UVOD

### *Teme koje se obrađuju u ovoj lekciji*

Ova lekcija ima za cilj da student nauči sledeće:

1. Kako se skladište neuređeni elemenati bez dupliranja upotrebom seta (skupa)
2. Kako i kada upotrebiti **HashSet**, **LinkedList** ili **TreeSet** radi skladišćenja elemenata.
3. Upoređenje performansi setova i listi
4. Kako da upotrebi setove u razvoju programa koji broji ključne reči u Java izvornoj datoteci.
5. Koje su razlike između **Collection** i **Map** i kako da opiše kada i kako da se upotrebi **HashMap**, **LinkedHashMap** ili **TreeMap** radi skladišćenja vrednosti koje su povezane sa ključevima.
6. Kako upotrebiti mape radi razvoja programa koji broji javljanje reči u nekom tekstu.
7. Kako da se dobiju singularni setovi, liste i mape i nepromenljivi setovi, liste i mape, upotrebom statičkih metoda u klasi **Collections**.

**Referenca:** Y. Daniel Liang, INTRODUCTION TO JAVA PROGRAMMING (COMPREHENSIVE VERSION), Tenth Edition, Pearson, ISBN 10: 0-13-376131-2, ISBN 13: 978-0-13-376131-3

Ovo je osnovni udžbenik za ovaj predmet i preporučuje se studentima da ga koriste

## ▼ Poglavlje 1

# Setovi

## ŠTA JE HEŠIRANJE?

*Heširanje (hashing) je tehnika koja nalazi vrednost u nekom nizu upotrebom indeksa, tj. heš koda, koji se dobija iz ključa, bez sprovođenja pretraživanja.*

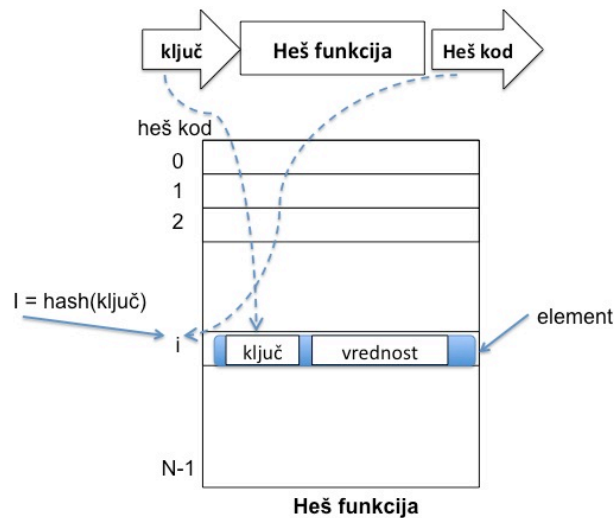
U ovoj sekciji će se često pominjati termin heš (engl. **hash**). Koncept heša se izučava u posebnoj lekciji predmeta CS103 Algoritmi i strukture podataka, a ovde se objašnjava u elementarnoj formi kako bi se razuelo njegova primena kod setova.

Ako znate indeks nekog elementa u nizu (engl. **array**), možete dobiti element upotrebom indeksa u vremenskom trenutku  $O(1)$ . Da li to znači da vi možete da uskladištite vrednost u neki niz i onda da koristite ključ kao indeks da bi našli vrednost? Odgovor je - da, ako možete da preslikate (engl. **map**) neki ključ u neki indeks. *Niz koji skladišti vrednosti se naziva heš tabelom. Funkcija koja preslikava ( **maps** ) neki ključ u indeks u heš tabeli se naziva heš funkcijom..* Kao što se vidi na slici 1, heš funkcija dobija indeks iz ključa i onda ga upotrebljava da nađe vrednost u nizu kojoj odgovara taj ključ. **Heširanje (hashing)** je tehnika koja nalazi vrednost u nekom nizu upotrebom indeksa koji se dobija iz ključa, bez sprovođenja pretraživanja.

Kako projektujete heš funkciju koja proizvodi indeks polazeći od nekog ključa? U idealnom slučaju, voleli bi da definišemo heš funkciju koja preslikava svak ključ pretraživanja u različiti indeks heš tabele. Takva perfektna heš funkcija se naziva perfektnom heš funkcijom.

Na žalost, vrlo je teško definisati perfektnu heš funkciju. Kada se dva ili više ključa preslikaju u istu heš vrednost, dolazi do sudara (collision), što treba izbegavati.

Zbog toga, treba da nađete brzu heš funkciju koja se lako računa, a koja minimizira sudare. Tipična heš funkcija prvo konvertuje neki ključ pretraživanja u ceo broj koji se naziva **heš kod** a onda sabija heš kod u indeks heš tabele.



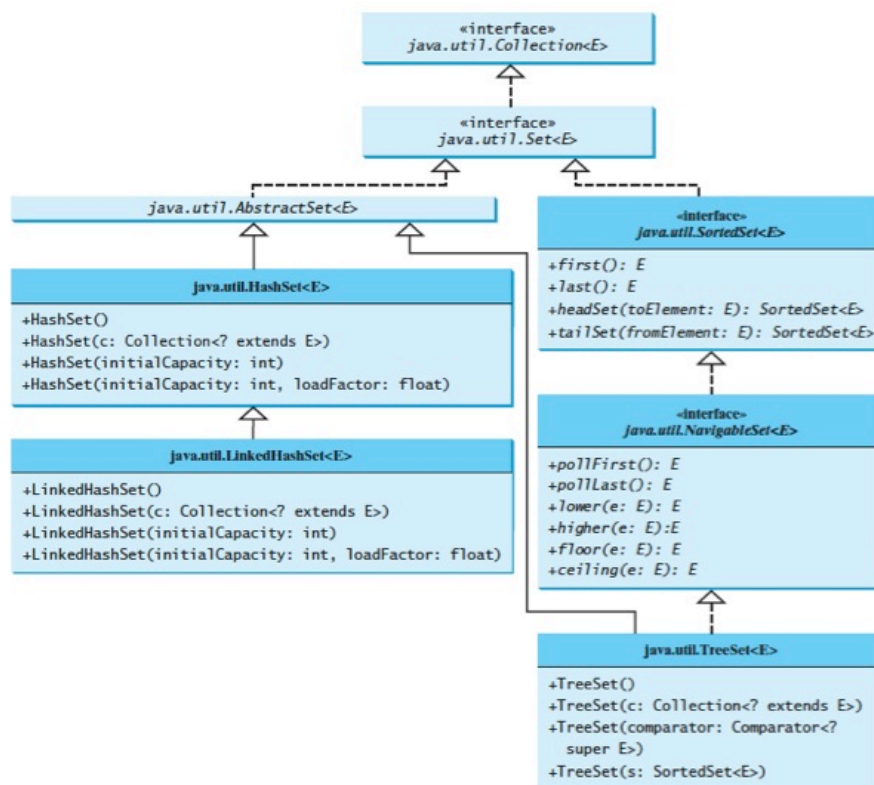
Slika 1.1 Ako je poznat ključ nekog elementa, heš funkcija mu određuje indeks (mesto) u nizu u kome se on skladišti

## ŠTA JE SET?

*Set (skup) je efikasna struktura podataka za skladištenje elemenata bez njihovog dupliranja (ponavljanja).*

Interfejs **Set** proširuje interfejs **Collection**, kao što je pokazano na slici 2. On ne obezbeđuje nove metode ili konstante, već obezbeđuje da **Set** primerak ne sadrži duplirane elemente, tj. da se elementi ne ponavljaju. *Konkretno klase koje primenjuju interfejs **Set** mora da obezbede da se ne mogu dodati duplikati elemenata koji su već u setu.* Ovo znači da dva elementa  $e1$  i  $e2$  u jednom setu ne mogu da daju: true (istinito) u logičkom izrazu:  **$e1.equals(e2)$** .

Klasa **AbstractSet** proširuje klasu **AbstractCollection** i delimično primenjuje interfejs **Set**. Ona obezbeđuje konkretne primene za metod **equals()** i metod **hashCode()**. *Heš kod seta je zbir heš kodova svih elemenata u setu.* Kako metodi **size()** i **iterator()** nisu primenjeni u klasi **AbstractSet**, **AbstractSet** je apstraktna klasa. Na slici 1 su prikazane tri konkretne klase sa **Set** interfejsom: **HashSet**, **LinkedHashSet**, i **TreeSet**.



Slika 1.2 Java Collections Framework obezbeđuje tri konkretne klase za setove

## KLASA HASHSET

*Klasa **HashSet** se koristi za skladištenje elemenata bez dupliranja, tj. koji se ne ponavljaju*

Klasa **HashSet** je konkretna klasa koja primenjuje interfejs **Set**. Možete da kreirate prazan heš set upotrebom konstruktora bez argumenata ili da kreirate heš set koristeći neku postojeću kolekciju. *Početno podešeni kapacitet heš seta je 16, a faktor opterećenja je 0,75.* Ako znate veličinu vašeg seta, onda možete da specificirate početni kapacitet i faktor opterećenja u konstruktoru. U suprotnom, koristite početno podešene vrednosti.

**Faktor opterećenja** je veličina između 0,0 i 1,0. *Faktor opterećenja meri stepena do koga je dozvoljeno punjenje heš seta, pre nego što se njegov kapacitet poveća.*

*Kada broj elemenata bude veći od proizvoda kapaciteta seta i njegovog faktora opterećenja, kapacitet seta se automatski duplira.* Na primer, ako je kapacitet 16 i ako je faktor opterećenja 0,75, kapacitet seta će se duplirati (32) kada veličina seta bude 12 (tj.  $16 \times 0,75 = 12$ ). Važi faktor opterećenja smanjuje cenu troškova prostora skladištenja ali povećava vreme pretraživanja seta. Smatra se da je faktor opterećenja 0,75 dobar kompromis. Heširanje ćete učiti u predmetu KI201 Algoritmi i strukture podataka.

Klasa **HashSet** se koristi za skladištenje elemenata bez dupliranja, tj. koji se ne ponavljaju. Zbog efikasnosti rada, objekti

koji se dodaju heš setu moraju da primenjuju metod **hashCode()** na način koji pravilno rastura heš kod. Metod **hashCode()** je definisan u klasi **Object**.

*Heš kod dva objekta mora da bude isti ako su dva objekta jednaka. Dva nejednaka objekta mogu da imaju isti heš kod, ali bi trebalo da primenite metod **hashCode()** tako da izbegnete puno takvih situacija.* Najveći broj klasa u Java API primenjuje metod **hashCode()**. Na primer, metod **hashCode()** u klasi **Integer** vraća vrednost **int**. Metod **hashCode()** u klasi **Character** vraća **Unicode** oznake (**character**). U slučaju klase **String**, metod **hashCode()** vraća  $s_0 * 31^{(n-1)} + s_1 * 31^{(n-2)} + \dots + s_{n-1}$ , gde  $s_i$  predstavlja **s.charAt(i)**.

Primena metoda **hashCode()** za nalaženje nekog elementa u setu je efikasna jer se element direktno nalazi na osnovu dobijenog keš koda, koji ustvari predstavlja indeks niza i koji označava poziciju gde je uskladišten traženi element. Umesto nekog pretraživanja niza, kada se u svakom koraku upoređuju neke vrednosti da bi se pronašao traženi element, kod primene keširanja, ako se zna ključ koji povezan sa vrednošću koja predstavlja neki element u nišu, taj element se preko keširanja direktno nalazi, bez pretraživanja. Zato je nalaženje nekog elementa u setu znatno brže primenom heširanja.

## PRIMER KORIŠĆENJE HEŠ SETA

*Klasa **TestHashSet** kreira heš set za skladišćenje stringova. Ona ga koristi u svakoj petlji u kojoj se prelazi po elementima seta.*

Ovde se daje listing klase **TestHashSet** koja kreira heš set za skladišćenje stringova i koja ga koristi u svakoj petlji u kojoj se prelazi po elementima seta.

U set se ubacuju stingovi (linije 9-14). **String** "New York" se dodaje više od jednog puta, ali se skladišti samo jedanput, jer set ne dozvoljava dupliranje elemenata (za razliku od liste). Kao što se vidi na prikazanim izlaznim rezultatima (slika 3), stringovi nisu uskladišćeni po redosledu po kom su dodavani u set. Ne postoji neki poseban redosled po kom se ređaju elementi u heš setu. Ako želite da uvedete neki red u ređanju elemenata, treba da koristite klasu **LinkedHashSet**, koja je opisana usledećoj sekciji

Kao što znate, interfejs **Collection** proširuje interfejs **Iterable**, te su elementi onda uskladišćeni u setu koji je iterabilan, tj. nad njim se mogu primeniti iteracije. Petja **foreach** se koristi da bi se prešli svi elementi seta (linije 19-21).

```
[San Francisco, New York, Paris, Beijing, London]
SAN FRANCISCO NEW YORK PARIS BEIJING LONDON
```

Slika 1.3 Dobijen rezultat izvršenjem programa **TestHashSet**

```
1 import java.util.*;
2
3 public class TestHashSet {
4     public static void main(String[] args) {
5         // Kreiranje heš seta
6         Set<String> set = new HashSet<>();
7     }
8 }
```



```
8 // Dodavanje stringova u set
9 set.add("London");
10 set.add("Paris");
11 set.add("New York");
12 set.add("San Francisco");
13 set.add("Beijing");
14 set.add("New York");
15
16 System.out.println(set);
17
18 // Prikaz elemenata u heš setu
19 for (String s: set) {
20     System.out.print(s.toUpperCase() + " ");
21 }
22 }
23 }
```

## PRIMENA METODA INTERFEJSA COLLECTION

*Listing klase `TestMethodsInCollection` daje primer korišćenja metoda interfejsa `Collection` u setovima.*

Kako set indirektno primenjuje interfejs **Collection**, mogu se u setovima koristiti svi metodi koje obezbeđuje interfejs **Collection**. Listing klase **TestMethodsInCollection** daje primer korišćenja metoda interfejsa **Collection** u setovima. Program kreira dva seta (linije 4 i 22). Metod **size()** vraća broj elemenata u setu (linija 14). Linija 17 uklanja London iz **seta1**. Metod **contains()** (linija 32) proverava da li je naveden element u setu. Linija 34 dodaje **set2** setu **set1**, te **set 2** postaje: [San Francisco, New York, Shanghai, Paris, Beijing, London] . Linija 38 uklanja **set2** iz seta **set1**, te **set1** postaje: [San Francisco, New York, Beijing] . Linija 42 zadržava zajedničke elemente u setu **set1**. Kako setovi **set1** i **set2** nemaju zajedničke elemente, **set1** postaje prazan.

```
set1 is [San Francisco, New York, Paris, Beijing, London]
5 elements in set1

set1 is [San Francisco, New York, Paris, Beijing]
4 elements in set1

set2 is [Shanghai, Paris, London]
3 elements in set2

Is Taipei in set2? false

After adding set2 to set1, set1 is
[San Francisco, New York, Shanghai, Paris, Beijing, London]

After removing set2 from set1, set1 is
[San Francisco, New York, Beijing]

After removing common elements in set2 from set1, set1 is []
```

Slika 1.4 Rezultat izvršenja programa

Slika 4 prikazuje rezultate izvršenja programa.

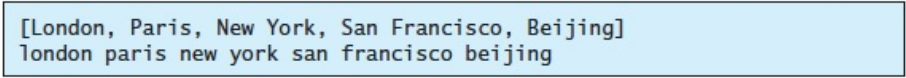
```
1 public class TestMethodsInCollection {
2     public static void main(String[] args) {
3         // Kreiranje seta set1
4         java.util.Set<String> set1 = new java.util.HashSet<>();
5
6         // Dodavanje stringova u set1
7         set1.add("London");
8         set1.add("Paris");
9         set1.add("New York");
10        set1.add("San Francisco");
11        set1.add("Beijing");
12
13        System.out.println("set1 is " + set1);
14        System.out.println(set1.size() + " elements in set1");
15
16        // Brisanje stringova iz set1
17        set1.remove("London");
18        System.out.println("\nset1 is " + set1);
19        System.out.println(set1.size() + " elements in set1");
20
21        // Kreiranje seta set2
22        java.util.Set<String> set2 = new java.util.HashSet<>();
23
24        // Dodavanje stringova u set2
25        set2.add("London");
26        set2.add("Shanghai");
27        set2.add("Paris");
28        System.out.println("\nset2 is " + set2);
29        System.out.println(set2.size() + " elements in set2");
30
31        System.out.println("\nIs Taipei in set2? "
32            + set2.contains("Taipei"));
33
34        set1.addAll(set2);
35        System.out.println("\nAfter adding set2 to set1, set1 is "
36            + set1);
37
38        set1.removeAll(set2);
39        System.out.println("After removing set2 from set1, set1 is "
40            + set1);
41
42        set1.retainAll(set2);
43        System.out.println("After removing common elements in set2 "
44            + "from set1, set1 is " + set1);
45    }
46 }
```

## KLASA LINKEDHASHSET

*Klasa `LinkedHashSet` proširuje klasu `HashSet` sa primenom povezane liste koja podržava ređanje elemenata u setu.*

Klasa `LinkedHashSet` proširuje klasu `HashSet` sa primenom povezane liste koja podržava ređanje elemenata u setu. Elementi u setu koji formira `HashSet` klasa nisu poređani, ali elementi u nizu koji kreira `LinkedHashSet` jesu, tj. dobijaju se u istom redosledu u kom su i bili skladišteni u setu.

`LinkedHashSet` objekat se kreira upotrebom jednog od četiri definisana konstruktora (vidi sliku 1 ranije). Ovi konstruktori su slični konstruktorima u klasi `HashSet`. Linsting daje test program za klasu `TestLinkedHashSet`. Program jednostavno zamenjuje `HashSet` sa `LinkedHashSet` u listing klase `TestHashSet`. Slika 5 prikazuje dobijane izlazne rezultate izvršenja programa `TestLinked HashSet`.



```
[London, Paris, New York, San Francisco, Beijing]
London Paris New York San Francisco Beijing
```

Slika 1.5 Rezultat izvršenja programa

`LinkedHashSet` set kreiran je u liniji 6. Kao što se vidi u prikazu rezultata, stringovi su poređani u redosledu po kom su i skladišćeni. Kako je `LinkedHashSet` set, on ne sadrži duple elemente. On održava redosled po kom su elementi skladišteni. Da bi se primenio drugojačiji redosled, možete koristiti klasu `TreeSet` koja se opisuje u sledećoj sekciji.

```
1 import java.util.*;
2
3 public class TestLinkedHashSet {
4     public static void main(String[] args) {
5         // Kreiranje heš seta
6         Set<String> set = new LinkedHashSet<>();
7
8         // Dodavanje stringova u set
9         set.add("London");
10        set.add("Paris");
11        set.add("New York");
12        set.add("San Francisco");
13        set.add("Beijing");
14        set.add("New York");
15
16        System.out.println(set);
17
18        // Prikaz elemenata u keš setu
19        for (Object element: set)
20            System.out.print(element.toLowerCase() + " ");
21    }
22 }
```

# KLASA TREESSET

*Listing klase `TestTreeSet` daje primer ređanja elemenata upotrebom interfejsa `Comparable`*

**SortedSet** je podinterfejs interfejsa **Set** koji obezbeđuje sortirani redosled elemenata u setu. Pored toga, obezbeđuje metode **first()** i **last()** koje vraćaju prvi i poslednji element u setu, i metode **headSet(toElement)** i **tailSet(fromElement)**. Prvi vraća deo seta čiji su elementi manji od elementa **toElement**, a drugi vraća elemente kojisu veći ili jednaki elementu **fromElement**.

**NavigableSet** proširuje **SortedSet** da bi obezbedio navigacione metode **lower(e)**, **floor(e)**, **ceiling(e)**, i **higher(e)** koji vraćaju elemente koji su manji, odn. manji ili jednaki, veći ili jednaki, i veći od datog elementa i vraćaju **null** ako nema takvog elementa.

Metodi **pollFirst()** i **pollLast()** uklanjaju, odn. vraćaju prvi, odn. poslednji element u **setu stabla** (**tree set**). Klasa **TreeSet** primenjuje interfejs **SortedSet**, a primenjuje konstruktor prikazan na slici 1. Možete dodati elemente u set stabla sve dok su oni međusobno uporedljivi (koriste interfejsa **Comparable** ili **Comparator**).

Listing klase **TestTreeSet** daje primer ređanja elemenata upotrebom interfejsa **Comparable**. U prethodnom primeru, klase **TestLinkedHashSet**, elementi su poređani po redosledu skladišćenja. Ovaj primer menja prethodni primer prikazom stringova po alfabetskom redosledu upotrebom **TreeSet** klase.

Listing klase **TreeSet**:

```
1 import java.util.*;
2
3 public class TestTreeSet {
4     public static void main(String[] args) {
5         // Kreiranje heš seta
6         Set<String> set = new HashSet<>();
7
8         // Dodavanje stringova u set
9         set.add("London");
10        set.add("Paris");
11        set.add("New York");
12        set.add("San Francisco");
13        set.add("Beijing");
14        set.add("New York");
15
16        TreeSet<String> treeSet = new TreeSet<>(set);
17        System.out.println("Sorted tree set: " + treeSet);
18
19        // Upotreba metoda interfejsa SortedSet
20        System.out.println("first(): " + treeSet.first());
21        System.out.println("last(): " + treeSet.last());
22        System.out.println("headSet(\"New York\"): " +
23        treeSet.headSet("New York"));
```

```
24 System.out.println("tailSet(\"New York\"): " +
25     treeSet.tailSet("New York"));
26
27 // Upotreba metoda interfejsa NavigableSet
28 System.out.println("lower(\"P\"): " + treeSet.lower("P"));
29 System.out.println("higher(\"P\"): " + treeSet.higher("P"));
30 System.out.println("floor(\"P\"): " + treeSet.floor("P"));
31 System.out.println("ceiling(\"P\"): " + treeSet.ceiling("P"));
32 System.out.println("pollFirst(): " + treeSet.pollFirst());
33 System.out.println("pollLast(): " + treeSet.pollLast());
34 System.out.println("New tree set: " + treeSet);
35 }
36 }
```

## PRIKAZ REZULTATA IZVRŠENJA PROGRAMA TESTTREESET

*Primer kreira heš set popunjen stringovima, a onda kreira set stabla (tree set) za iste stringove. Stringovi se sortiraju u setu stabla upotrebom metoda `compareTo()` interfejsa `Comparable`*

Izvršenjem programa **TestTreeSet** se dobija na monitoru računara sledeći prikaz (slika 6):

```
Sorted tree set: [Beijing, London, New York, Paris, San Francisco]
first(): Beijing
last(): San Francisco
headSet("New York"): [Beijing, London]
tailSet("New York"): [New York, Paris, San Francisco]
lower("P"): New York
higher("P"): Paris
floor("P"): New York
ceiling("P"): Paris
pollFirst(): Beijing
pollLast(): San Francisco
New tree set: [London, New York, Paris]
```

Slika 1.6 Rezultat izvršenja programa TestTreeSet

Primer kreira heš set popunjen stringovima, a onda kreira set stabla (**tree set**) za iste stringove. Stringovi se sortiraju u setu stabla upotrebom metoda **`compareTo()`** interfejsa **`Comparable`**. Elementi u setu se jedanput sortiraju prilikom kreiranja objekta **`TreeSet`** iz objekta **`HashSet`**, upotrebom instrukcije u liniji 16: **`new TreeSet<String>(set)`**.

Možete promeniti program kreiranja primerka **`TreeSort`** upotrebom njegovog konstruktora bez argumenata i dodavanje stringova u objekat **`TreeSet`**.

Metod **`treeSet.first()`** vraća prvi element u **`treeSet`** (linija 20), a metod **`treeSet.last()`** vraća poslednji element u **`treeSet`**. (linija 21).

Instrukcija **`treeSet.headSet("New York")`** vraća elemente u **`treeSet`** pre stringa New York (linije 22-23), a instrukcija **`treeSet.tailSet("New York")`** vraća elemente u **`treeSet`** posle stringa New York, uključujući i string New York (linije 24-25).

Metod `treeSet.lower("P")` vraća najveći element koji je manji od P u treeSet (linija 28). Metod `treeSet.higher("P")` vraća najmanji element koji je veći od P u treeSet (linija 29).

Metod `treeSet.floor("P")` vraća najveći element a koji je manji ili jednak P u treeSet (linija 30). Metod `treeSet.ceilling("P")` vraća najmanji element koji je veći ili jednak sa P u treeSet (linija 31).

Metod `treeSet.pollFirst()` uklanja prvi element u treeSet i vraća uklonjen element (linija 32). Metod `treeSet.pollLast()` uklanja poslednji element u treeSet i vraća uklonjen element (linija 33).

Ako kreirate objekat TreeSet upotrebom njegovog konstruktora bez argumenata, upotrićete metod `compareTo()` za upoređenje elemenata u setu, pretpostavljajući da klasa elemenata primenjuje interfejs `Comparable`. Ako želite da koristite interfejs `Comparator`, korišćete konstruktor `TreeSet(Comparator comparator)` za kreiranje sortiranog seta koji koristi `compare()` metod za uređenje.

## SORTIRANJE ELEMENATA SETA STABLA

*Klasa `TestTreeSetWithComparator` predstavlja program koji pokazuje kako se sortiraju elementi seta stabla upotrebom interfejsa `Comparator`.*

Listing klase **`TestTreeSetWithComparator`** predstavlja program koji pokazuje kako se sortiraju elementi seta stabla upotrebom interfejsa `Comparator`.

```
1 import java.util.*;
2
3 public class TestTreeSetWithComparator {
4     public static void main(String[] args) {
5         // Kreiranje seta stabla za geometrijske objekte upotrebom komparatora
6         Set<GeometricObject> set =
7             new TreeSet<>(new GeometricObjectComparator());
8         set.add(new Rectangle(4, 5));
9         set.add(new Circle(40));
10        set.add(new Circle(40));
11        set.add(new Rectangle(4, 1));
12
13        // Display geometric objects in the tree set
14        System.out.println("A sorted set of geometric objects");
15        for (GeometricObject element: set)
16            System.out.println("area = " + element.getArea());
17    }
18 }
```

Na slici 7 su prikazan izlazni rezultati:

```
A sorted set of geometric objects
area = 4.0
area = 20.0
area = 5021.548245743669
```

Slika 1.7 Rezultat izvršenja programa TestTreeSetWithComparator

## VIDEO: SETOVI

*Sets: Java Collections Framework Tutorial Part 5 (15,54 minuta)*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## VIDEO: KORISTITE VAŠE OBJEKTE U SETOVIMA I MAPAMA

*Using Your Own Objects in Sets and Maps: Java Collections Framework Tutorial Part 6 (11,2 min.)*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ZADATAK 1

*Cilj ovog zadatka je pravljenje primera korišćenja HashSetova.*

*Napraviti klasu Student (indeks, ime, prezime), Implementirati equals i hashCode metode tako da koriste parametar indeks za poređenje. U Main klasi kreirati HashSet studenata i prikazati da HashSet neće dodavati studente sa istim indeksom u svoju kolekciju.*

*Objašnjenje i uputstva za rešavanje zadatka:*

1. kreiramo klasu Student sa navedenim atributima
2. Override-ujemo metodu hashCode
3. Override-ujemo metodu equals na "naš" način - dva studenta su "jednaka ako imaju isti broj indeksa", u suprotnom nisu.

```
@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
}
```

```

    }
    final Student other = (Student) obj;
    if (this.indeks != other.indeks) {
        return false;
    }
    return true;
}

```

4. Kreiramo Main klasu

5. Pravimo objekte klase Student sa proizvoljnim atributima

6. Pravimo Set naredbom:

```
Set<Student> setovi = new HashSet<Student>();
```

7. metodom add() dodajemo Studente u Set

8. iteriramo kroz Set i ispisujemo studente

9. Primetiti da Set iz pokaznog primera sadrži samo 2 studenta, iako su 3 dodata. Razlog tome je što s i s3 imaju isti broj indeksa - te su po metodu equals oni jednaki.

10. Izlaz iz programa je:

```
Student{ime=Pera, prezime=Vasic, indeks=1640}Student{ime=Vuk, prezime=Vasic, indeks=1630}
```

## ZADATAK 1 - REŠENJE

### Rešenje zadatka 1

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Z01;

/**
 *
 * @author Aleksandra
 */
public class Student {

    // definišemo attribute klase Student
    private String ime;
    private String prezime;
    private int indeks;

    //podrazumevani konstruktor
    public Student() {

```



```
}

//konstruktor polja
public Student(String ime, String prezime, int indeks) {
    this.ime = ime;
    this.prezime = prezime;
    this.indeks = indeks;
}

//getter za atribut ime
public String getIme() {
    return ime;
}

//setter za atribut ime

public void setIme(String ime) {
    this.ime = ime;
}

//getter za atribut prezime
public String getPrezime() {
    return prezime;
}

//setter za atribut prezime
public void setPrezime(String prezime) {
    this.prezime = prezime;
}

//getter za atribut indeks

public int getIndeks() {
    return indeks;
}

//setter za atribut indeks

public void setIndeks(int indeks) {
    this.indeks = indeks;
}

@Override
public String toString() {
    return "Student{" + "ime=" + ime + ", prezime=" + prezime + ", indeks=" +
indeks + '}';
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 23 * hash + this.indeks;
    return hash;
}
```

```
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Student other = (Student) obj;
    if (this.indeks != other.indeks) {
        return false;
    }
    return true;
}
}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Z01;

import java.util.HashSet;
import java.util.Set;

/**
 *
 * @author Aleksandra
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        new Main();
    }

    public Main() {
        Set<Student> setovi = new HashSet<Student>();
        Student s = new Student("Vuk", "Vasic", 1630);
        Student s2 = new Student("Pera", "Vasic", 1640);
        Student s3 = new Student("Peric", "Vasic", 1630);
        setovi.add(s);
        setovi.add(s2);
        setovi.add(s3);
        for (Student st : setovi) {
```

```

        System.out.println(st);
    }

}

}

```

## ZADACI ZA SAMOPROVERU

### *Uraditi slede'e zadatke za samoproveru*

**Zadatak 1.** Neka set1 sadrži sledeće stringove red, yellow, and green, a set2 neka sadrži red, yellow, and blue. Odgovoriti na sledeća pitanja:

- Šta je u skupu set1, a šta u skupu set2 posle set1.addAll(set2)?
- Šta je u skupu set1, a šta u skupu set2 posle set1.add(set2)?
- Šta je u skupu set1, a šta u skupu set2 posle set1.removeAll(set2)?
- Šta je u skupu set1, a šta u skupu set2 posle set1.remove(set2)?
- Šta je u skupu set1, a šta u skupu set2 posle set1.retainAll(set2)?
- Šta je u skupu set1 posle set1.clear()?

**Zadatak 2.** Koji je izlaz ovog programa:

```

import java.util.*;
public class Test {
    public static void main(String[] args) {
        HashSet<String> set1 = new HashSet<String>();
        set1.add("New York");
        HashSet<String> set2 = set1;
        HashSet<String> set3 =
            (HashSet<String>)(set1.clone());
        set1.add("Atlanta");
        System.out.println("set1 is " + set1);
        System.out.println("set2 is " + set2);
        System.out.println("set3 is " + set3);
    }
}

```

**Zadatak 3.** Koji je izlaz ovog programa:

```

import java.util.*;
import java.io.*;
public class Test {
    public static void main(String[] args) throws Exception {
        ObjectOutputStream output = new ObjectOutputStream(
            new FileOutputStream("c:\\test.dat"));
        HashSet<String> set1 = new HashSet<String>();
        set1.add("New York");
        HashSet<String> set2 =
            (HashSet<String>)set1.clone();
    }
}

```

```
        set1.add("Atlanta");  
        output.writeObject(set1);  
        output.writeObject(set2);  
        output.close();  
        ObjectInputStream input = new ObjectInputStream(  
            new FileInputStream("c:\\test.dat"));  
        set1 = (LinkedHashSet<String>)input.readObject();  
        set2 = (LinkedHashSet<String>)input.readObject();  
        System.out.println(set1);  
        System.out.println(set2);  
        output.close();  
    }  
}
```

## ZADACI ZA SAMOSTALNI RAD

*Na osnovu obradjenog uraditi samostalno sledeće zadatke*

**Zadatak 1.** (Izvršiti operacija skupa na heš setu) Kreirati 2 heš seta {"George", "Jim", "John", "Blake", "Kevin", "Michael"} i {"George", "Katie", "Kevin", "Michelle", "Ryan"} i pronaći uniju, razliku i presek. (Moguće je klonirati skup da bi sačuvali originalni skup od upotrebe ovih metoda). Ponoviti sve isto sa prioritetnim redovima.

**Zadatak 2.** (Prikaz reči bez duplikata u rastućem poretku) Napisati program koji čita reči iz tekstualnog fajla i prikazuje sve reči koje nemaju duplikate u rastućem poretku. Naziv fajla se prosleđuje kao argument komandne linije.

## ▼ Poglavlje 2

# Poređenje preformansi setova i listi

## PRIMER TESTIRANJA PERFORMANSI SETOVA I LISTI

*Setovi su efikasniji od listi za skladištenje elemenata bez njihovog dupliranja. Liste su korisne za pristup elementima preko indeksa.*

Elementima u listi se pristupa preko indeksa, a setovi ne podržavaju indekse, jer elementi nisu smešteni po nekom redosledu. Prelaženje preko svih elemenata usetu se vrši petljom `foreach`. Ovde se daje listing programa kojim ćemo testirati performanse setova i listi. Meri se izvršenje kada (1) su elementi u heš setu, u povezanom heš setu, setu stabla, niza i povezane liste, i (2) se elementi uklanjaju iz vreme heš seta, povezanog seta, seta stabla, niza i povezane liste.

Program kreira listu sa brojevima od 0 do N-1 (za N=50000) (linije 8-10) i meša listuu (linija 11). Iz ove liste program kreira heš set (linija 14), povezan heš set (linija 21) i set stabla (linija 42). Program dobija vreme izvršenja testa za svaku od navedenih vrsti kolekcija. (linije 16, 23, 30, 37, i 44) i za uklanjanje elemenata (linije 18, 25, 32, 39 i 46).

Metod **`getTestTime()`** poziva metod **`contains()`** da proveri da li je bro u kontejneru (linija 54) i metod **`getRemoveTime()`**, koji poziva metod **`remove()`** koji uklanja element iz kontejnera (linija 63).

Kao što se vidi, setovi su mnogo efikasniji od listi kod testa da li je neki broj u setu ili listi. Razlog je u implementacijama listi i setova. Na slici 1 prikazani su dobijeni rezultati u toku izvršenja programa.

```
Member test time for hash set is 20 milliseconds
Remove element time for hash set is 27 milliseconds
Member test time for linked hash set is 27 milliseconds
Remove element time for linked hash set is 26 milliseconds
Member test time for tree set is 47 milliseconds
Remove element time for tree set is 34 milliseconds
Member test time for array list is 39802 milliseconds
Remove element time for array list is 16196 milliseconds
Member test time for linked list is 52197 milliseconds
Remove element time for linked list is 14870 milliseconds
```

Slika 2.1 Dobijeni rezultati testiranja pri izvršenju programa

```
1 import java.util.*;
2
3 public class SetListPerformanceTest {
4     static final int N = 50000;
5
6     public static void main(String[] args) {
7         // Dodavanje brojeve 0, 1, 2, ..., N-1 u niz
```

```

8 List<Integer> list = new ArrayList<>();
9 for (int i = 0; i < N; i++)
10     list.add(i);
11 Collections.shuffle(list); // Mešanje niza
12
13 // Kreiranje heš seta i testiranje njegovih performansi
14 Collection<Integer> set1 = new HashSet<>(list);
15 System.out.println("Member test time for hash set is " +
16     getTestTime(set1) + " milliseconds");
17 System.out.println("Remove element time for hash set is " +
18     getRemoveTime(set1) + " milliseconds");
19
20 // Kreiranje povezanog heš seta, i testiranje njegovih performansi
21 Collection<Integer> set2 = new LinkedHashSet<>(list);
22 System.out.println("Member test time for linked hash set is " +
23     getTestTime(set2) + " milliseconds");
24 System.out.println("Remove element time for linked hash set is "
25     + getRemoveTime(set2) + " milliseconds");
26
27 // Kreiranje seta stabla i testiranje njegovig performansi
28 Collection<Integer> set3 = new TreeSet<>(list);
29 System.out.println("Member test time for tree set is " +
30     getTestTime(set3) + " milliseconds");
31 System.out.println("Remove element time for tree set is " +
32     getRemoveTime(set3) + " milliseconds");
33
34 // Kreiranje niza, i testiranje njegovih performansi
35 Collection<Integer> list1 = new ArrayList<>(list);
36 System.out.println("Member test time for array list is " +
37     getTestTime(list1) + " milliseconds");
38 System.out.println("Remove element time for array list is " +
39     getRemoveTime(list1) + " milliseconds");
40
41 // Kreiranje povezane liste, i testiranje njegovih performansi
42 Collection<Integer> list2 = new LinkedList<>(list);
43 System.out.println("Member test time for linked list is " +
44     getTestTime(list2) + " milliseconds");
45 System.out.println("Remove element time for linked list is " +
46     getRemoveTime(list2) + " milliseconds");
47 }
48
49 public static long getTestTime(Collection<Integer> c) {
50     long startTime = System.currentTimeMillis();
51
52     // Test if a number is in the collection
53     for (int i = 0; i < N; i++)
54         c.contains((int)(Math.random() * 2 * N));
55
56     return System.currentTimeMillis() - startTime;
57 }
58
59 public static long getRemoveTime(Collection<Integer> c) {
60     long startTime = System.currentTimeMillis();

```

```
61
62  for (int i = 0; i < N; i++)
63      c.remove(i);
64
65  return System.currentTimeMillis() - startTime;
66  }
67 }
```

## ZADATAK: PRIMENA SETOVA I NIZOVA

*Cilj sekcije je utvrdjivanje rada sa Setovima*

*Napraviti Set čiji su objekti celi brojevi, smestiti prvih 100 pozitivnih celih brojeva u Set. Potom, iz seta izbaciti sve parne brojeve. Prikazati sadržaj seta pre kraja programa.*

*Objašnjenje i uputstva za rešavanje zadatka:*

1. kreiramo Main klasu
2. kreirati set naredbom

```
Set<Integer> set = new HashSet<Integer>();
```

3. Dodati 100 elemenata u set, najbolje za to je koristiti petlju  

```
for (int i = 1; i < 100; i++) {  
    set.add(i);  
}
```

4. Prebaciti set u array naredbama  

```
Integer[] niz = new Integer[set.size()];  
set.toArray(niz);
```

5. Izbaciti sve parne elemente  

```
for (Integer element : niz) {  
    if (element % 2 == 0) {  
        set.remove(element);  
    }  
}
```

6. Na kraju štampamo sadržaj seta  

```
for (int p : set) {  
    System.out.println(p);  
}
```

## REŠENJE ZADATKA

*Rešenje zadatka*

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Z06;

import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

/**
 *
 * @author Aleksandra
 */
public class Main {

    public static void main(String[] args) {
        new Main();
    }

    public Main() {
        Set<Integer> set = new HashSet<Integer>();
        for (int i = 1; i < 100; i++) {
            set.add(i);
        }
        Integer[] niz = new Integer[set.size()];
        set.toArray(niz);
        for (Integer element : niz) {
            if (element % 2 == 0) {
                set.remove(element);
            }
        }
        for (int p : set) {
            System.out.println(p);
        }
    }
}
```



## ▼ Poglavlje 3

# Studija slučaja: Brojanje ključnih reči

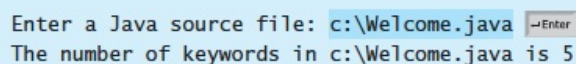
## PRIMER PROGRAMA ZA BROJANJE KLJUČNIH REČI

*U ovoj sekciji se predstavlja aplikacija koja broji broj ključnih reči u izvornoj Java datoteci.*

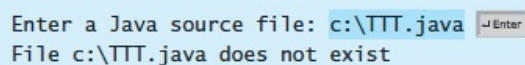
Za svaku reč i izvornoj Java datoteci, treba odrediti da li je reč ključna reč. Radi efikasnosti, sve ključne reči se skladište u heš set (**HashSet**) i koristi se metod **contains()** radi testiranja da li je reč u setu ključnih reči. Listing klase **CountKeywords** daje program za testiranje.

Program pita korisnika da unese ime izvorne Java datoteke (linija 7) i čita ime datoteke (linija 8). Ako datoteka postoji, metod **countKeywords()** se poziva da broji ključne reči u datoteci (linija 13). Metod **countKeywords()** kreira niz sa stringovima sa ključnim rečima (linije 22-32) i kreira heš set iz niza (linija 41). Kada se to utvrdi, broj se povećava za 1 (linija 42).

Možete izmeniti program da koristi **LinkedHashSet**, **TreeSet**, **ArrayList**, ili **LinkedList** za skladištenje ključnih reči. Međutim, upotrebe heš seta (HashSet) je najefikasnija kolekcija za ova program. Slika 1 prikazuje dobijene rezultate tokom izvršenja programa



Enter a Java source file: c:\Welcome.java ↵ Enter  
The number of keywords in c:\Welcome.java is 5



Enter a Java source file: c:\TTT.java ↵ Enter  
File c:\TTT.java does not exist

Slika 3.1 Prikaz dobijenih rezultata tokom izvršenja programa

```
1 import java.util.*;
2 import java.io.*;
3
4 public class CountKeywords {
5     public static void main(String[] args) throws Exception {
6         Scanner input = new Scanner(System.in);
7         System.out.print("Enter a Java source file: ");
8         String filename = input.nextLine();
9
10        File file = new File(filename);
11        if (file.exists()) {
12            System.out.println("The number of keywords in " + filename
```

```
13     + " is " + countKeywords(file));
14 }
15 else {
16     System.out.println("File " + filename + " does not exist");
17 }
18 }
19
20 public static int countKeywords(File file) throws Exception {
21     // Array of all Java keywords + true, false and null
22     String[] keywordString = {"abstract", "assert", "boolean",
23         "break", "byte", "case", "catch", "char", "class", "const",
24         "continue", "default", "do", "double", "else", "enum",
25         "extends", "for", "final", "finally", "float", "goto",
26         "if", "implements", "import", "instanceof", "int",
27         "interface", "long", "native", "new", "package", "private",
28         "protected", "public", "return", "short", "static",
29         "strictfp", "super", "switch", "synchronized", "this",
30         "throw", "throws", "transient", "try", "void", "volatile",
31         "while", "true", "false", "null"};
32
33     Set<String> keywordSet =
34         new HashSet<>(Arrays.asList(keywordString));
35     int count = 0;
36
37     Scanner input = new Scanner(file);
38
39     while (input.hasNext()) {
40         String word = input.next();
41         if (keywordSet.contains(word))
42             count++;
43     }
44
45     return count;
46 }
47 }
```

## ZADACI ZA SAMOSTALAN RAD

*Proverite vaše razumevanje, vaše novostečeno znanje.*

**Zadatak 1.** Da li program CountKeywords može da radi ako se linije 33-34 zamene sa:

```
Set<String> keywordSet =
    new LinkedHashSet<>(Arrays.asList(keywordString));
```

**Zadatak 2.** Da li program CountKeywords može da radi ako se linije 33-34 zamene sa:

```
List<String> keywordSet =  
    new ArrayList<>(Arrays.asList(keywordString));
```

**Zadatak 3.** Izmeniti kod priloženog zadatka na sledeći način: Ukoliko je ključna reč u komentaru ili stringu onda je ignorisati. Proslediti ime fajla iz komandne linije. Pretpostaviti da je Java izvorni kod korektan i da se linije sa komentarima i paragrafi ne preklapaju.

**Zadatak 4.** Napisati program koji od korisnika zahteva unos imena tekstualne datoteke i prikazuje broj suglasnika i samoglasnika. Koristiti set da bi se vršilo brojanje samoglasnika: A, E, I, O, i U.

## ▼ Poglavlje 4

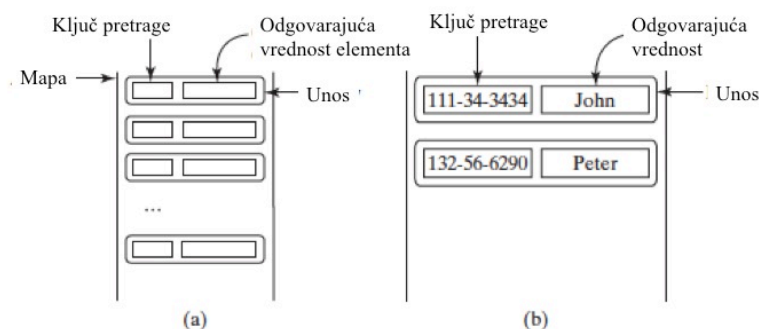
# Mape

## ŠTA JE MAPA?

*Mapa je kao rečnik koji obezbeđuje brzo nalaženje neke vrednosti upotrebom ključa pretrage. Postoji tri tipa mapa definisanih klasama: **HashMap**, **LinkedHashMap** i **TreeMap**.*

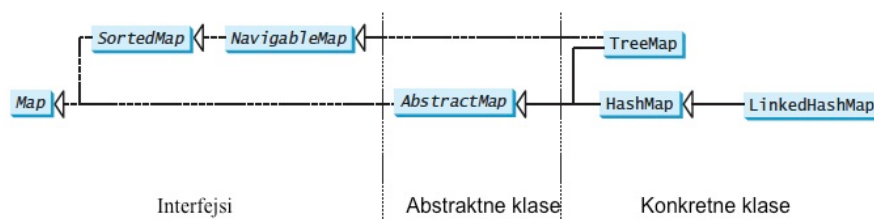
Mapa je kontejner objekat koji skladišti kolekciju parova ključ/vrednost. Ona omogućava brzu pretragu, brisanje i menjanje parova primenom ključa pretrage. Mapa skladišti vrednosti zajedno sa ključevima.

Ključevi su kao indeksi. U listi (**List**), indeksi su celi brojevi. U mapi (**Map**) ključevi mogu biti bilo kakvi objekti. Mapa ne može da sadrži duple ključeve, tj. dva ista ključa. Svaki ključ odgovara jednoj vrednosti. Ključ i njegova odgovarajuća vrednost čine ulaz uskladišten u mapi, kao što je prikazano na slici 1a. Slika 2b pokazuje mapu u kojoj svaki unos sadrži broj socijalnog osiguranja (**Social Security number**) kao ključ i ime kao vrednost.



Slika 4.1 Ulaz u vidu parova ključ/vrednost se skladište u mapi

Postoji tri tipa mapa: heš mapa (**HashMap**), povezana keš mapa (**LinkedHashMap**) i mapa stabla (**TreeMap**). Zajednička svojstva ovih mapa su definisana u interfejsu **Map**. Njihove veze su prikazane na slici 2.



Slika 4.2 Postoji tri konkretne klase za tri tipa mapa

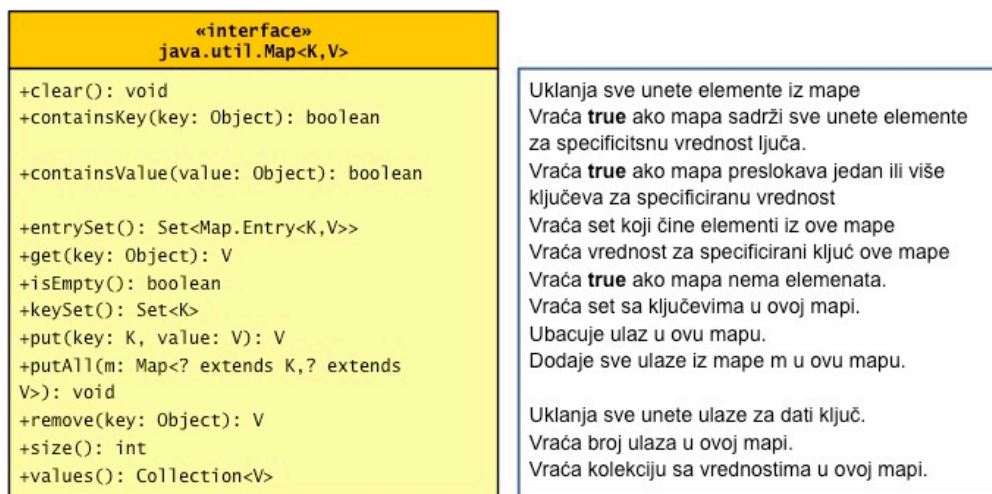
## INTERFEJS MAP

*Intefejs Map obezbeđuje metode za pretraživanje, menjanje i dobijanje kolekcija vrednosti i seta ključeva.*

Intefejs **Map** obezbeđuje metode za pretraživanje, menjanje i dobijanje kolekcija vrednosti i seta ključeva (slika 3). Metodi za promene mape su: **put()**, **putAll()** i **remove()**. Metod **clear()** uklanja sve ulaze iz mape. Metod **put(K key, V value)** dodaje ulaz sa specificiranim ključem i sa odgovarajućom vrednosti u mapu. Ako mapa sadrži za isti ključ unet par ključ/vrednost, stara vrednost se menja sa novom. i vraća se novi par. Metod **putAll(Map m)** dodaje sve ulaze u **m** u ovu mapu. Metod **remove(Object key)** uklanja ulaz za specificiran ključ iz mape.

Metodi pretraživanja uključuju: **containsKey()**, **containsValue()**, **isEmpty()**, i **size()**. Metod **containsKey(Object key)** proverava da li mapa sadrži ulaz sa datom vrednošću. Metod **isEmpty()** proverava da li mapa sadrži neki ulaz, a metod **size()** vraća broj ulaza u mapi.

Možete da dobijete set ključeva u mapi upotrebom **keySet()** metoda i kolekciju vrednosti u mapi upotrebom metoda **values()**. Metod **entrySet()** vraća set ulaza



Slika 4.3 UML dijagram interfejsa Map sa metodama za rad sa mapama

## INTERFEJS MAP.ENTRY

*Interfejs Map.Entry služi za unos parova ključ/vrednost u mapu.*

Ulazi su primerci od interfejsa **Map.Entry<K, V>** gde je Entry unutrašnji interfejs interfejsa **Map**, kao što je pokazano na slici 4. Svaki ulaz u setu je par ključ/vrednost u mapi.

Slika 4.4 Interfejs `Map.Entry` služi za unos parova ključ/vrednost u mapu

## KONKRETNE KLASSE ZA MAPE

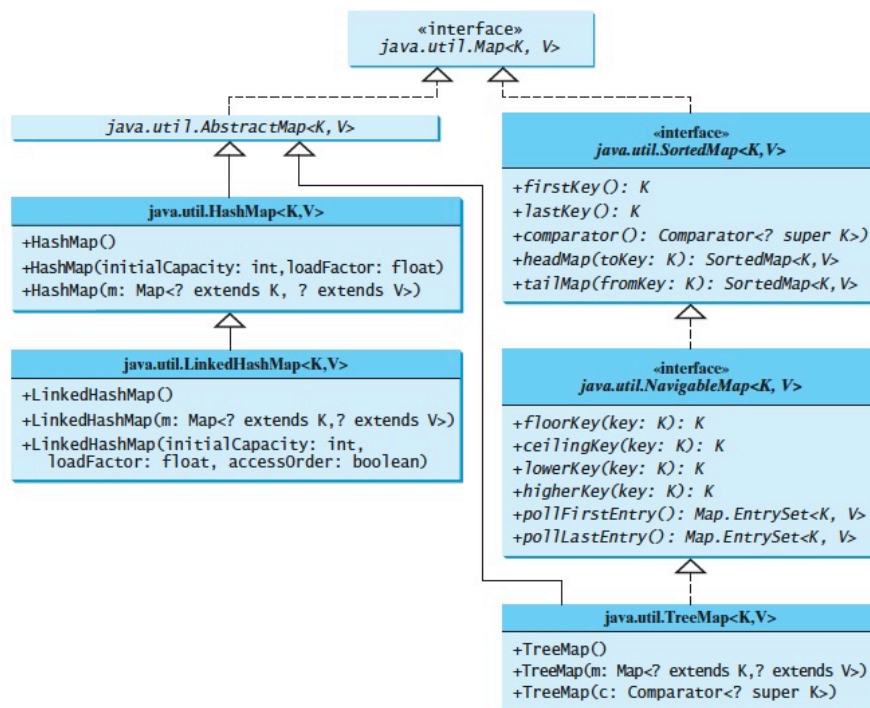
*Klase `HashMap`, `LinkedHashMap` i `TreeMap` su tri konkretne klase koje primenjuju interfejs `Map`*

Klasa **`AbstractMap`** i pogodna konkretna klasa koja primenjuje sve metode u interfejsa **`Map`**, sem metoda **`entrySet()`**.

Klase **`HashMap`**, **`LinkedHashMap`**, i **`TreeMap`** su tri konkretne klase koje primenjuju interfejs **`Map`**, kao što je prikazano na slici 5. Klasa **`HashMap`** je efikasna za lociranje neke vrednosti, ubacivanje ulaza i za brisanje ulaza. Klasa **`LinkedHashMap`** proširuje **`HashMap`** sa implementacijom povezane liste koja podržava redosled ulaza u mapi. Ulazi u **`HashMap`** nisu uređeni po redosledu, ali ulazi u **`LinkedHashMap`** mogu se dobijati ili po redosledu u kom su uneti u mapu, ili po redosledu po kom su poslednji put bili imali pristup. Konstruktor bez argumenata klase **`LinkedHashMap`** sa redosledom pristupa upotrebljava potpis: **`LinkedHashMap(initialCapacity, loadFactor, true)`**.

Klasa **`TreeMap`** je efikasna u pretraživanju sortiranih ključeva. Ključevi se mogu sortirati upotrebom interfejsa **`Comparable`** ili **`Comparator`**. Ako kreirate objekat **`TreeMap`** upotrebom konstruktora bez argumenata, metod **`compareTo()`** u interfejsu **`Comparable`** upoređuje ključeve u mapi, pretpostavljajući da klasa za ključeve primenjuje interfejs **`Comparable`**. Upotreba **`Comparator`** interfejsa se vrši upotrebom konstruktora: **`TreeMap(Comparator comparator)`** radi dobijanja sortirane mape koja koristi **`compareTo()`** metod interfejsa **`comparator`** da bi se sortirali ulazi u mapu na osnovu ključeva.

**`SortedMap`** je podinterfejs interfejsa **`Map`** koji sortira ulaze u mapu. Takođe, obezbeđuje metode **`firstKey()`** i **`lastKey()`** za dobijanje prvog i poslednjeg ključa u mapi, metodi **`headMap(toKey)`** i **`tailMap(fromKey)`** vraćaju mape čiji su ključevi manji od **`toKey`**, odn. veći ili jednaki od **`fromKey`**. Interfejs **`NavigableMap`** obezbeđuje metode za navigaciju po ključevima.



Slika 4.5 Java Collections Framework obezbeđuje tri konkretne klase za rad sa mapama

## PRIMER: KLASA TESTMAP

*Listing klase **TestMap** daje primer koji kreira heš mapu, povezanu heš mapu i mapu stabla ređanje studenata po starosti*

Listing klase **TestMap** daje primer koji kreira heš mapu, povezanu heš mapu i mapu stabla ređanje studenata po starosti. Program prvo kreira heš mapu sa imenom studenta kao svojim ključem, a sa godinama starosti kao svojom vrednošću. Zatim, program kreira mapu stabla iz heš mape i prikazuje ulaze u rastućem redosledu ključeva. Na kraju, program kreira povezanu heš mapu, dodaje iste ulaze u mapu i prikazuje ulaze.

Ulazi u heš mapu su u slučajnom redosledu. Ulazi u mapu stabla su poređani po rastućem redosledu ključeva, a i špbezanoj heš tabeli po redusledu poslednjeg pristupa (slika 6).

```

Display entries in HashMap
{Cook=29, Smith=30, Lewis=29, Anderson=31}

Display entries in ascending order of key
{Anderson=31, Cook=29, Lewis=29, Smith=30}

The age for Lewis is 29
Display entries in LinkedHashMap
{Smith=30, Anderson=31, Cook=29, Lewis=29}
  
```

Slika 4.6 Prikaz unosa i rezultata u ovom primeru

Linije 16-17 konstruišu mapu stabli iz heš mape. Na linijama 22-21 kreirana povezana heš mapa po redosledu sa prethodnog pristupa. U linjama 34 ulaz sa ključem Lewis u poslednjem prikazu u liniji 31.

Listing klase TestMap:

```

1 import java.util.*;
2
3 public class TestMap {
4     public static void main(String[] args) {
5         // Kreiranje keš mape HashMap
6         Map<String, Integer> hashMap = new HashMap<>();
7         hashMap.put("Smith", 30);
8         hashMap.put("Anderson", 31);
9         hashMap.put("Lewis", 29);
10        hashMap.put("Cook", 29);
11
12        System.out.println("Display entries in HashMap");
13        System.out.println(hashMap + "\n");
14
15        // Kreiranje TreeMap iz prethodne HashMap
16        Map<String, Integer> treeMap =
17            new TreeMap<>(hashMap);
18        System.out.println("Display entries in ascending order of key");
19        System.out.println(treeMap);
20
21        // Kreiranje LinkedHashMap
22        Map<String, Integer> linkedHashMap =
23            new LinkedHashMap<>(16, 0.75f, true);
24        linkedHashMap.put("Smith", 30);
25        linkedHashMap.put("Anderson", 31);
26        linkedHashMap.put("Lewis", 29);
27        linkedHashMap.put("Cook", 29);
28
29        // Prikaz starosti zaD Lewis-a
30        System.out.println("\nThe age for " + "Lewis is " +
31            linkedHashMap.get("Lewis"));
32
33        System.out.println("Display entries in LinkedHashMap");
34        System.out.println(linkedHashMap);
35    }
36 }

```

## ZADATAK 1

*Cilj ovog zadatka je pravljenje primera korišćenja TreeMape*

Koristeći TreeMapu napravite HighScore listu kao na slici. Key TreeMape treba da bude pozicija igrača dok value treba da bude ima igrača. Za prikaz koristite JFrame, JPanel i JTable.

Klasa Main

```

public class Main {

```



```

public static void main(String[] args) {
    final Map<String, String> st = new TreeMap<String, String>();
    st.put("1", "Djordje Cvarkov");
    st.put("2", "Mita Mrkic");
    st.put("3", "Olivera Mikicevic");
    JTable t = new JTable(toTableModel(st));
    JPanel p = new JPanel();
    p.setLayout(new GridLayout(1, 1));
    p.add(t);
    JFrame f = new JFrame();
    f.setContentPane(p);
    f.setSize(500, 200);
    f.setVisible(true);
}

public static TableModel toTableModel(Map<?, ?> map) {
    DefaultTableModel model = new DefaultTableModel(
        new Object[]{"Key", "Value"}, 0
    );
    for (Map.Entry<?, ?> entry : map.entrySet()) {
        model.addRow(new Object[]{entry.getKey(), entry.getValue()});
    }
    return model;
}
}

```

## ZADATAK 2

*Cilj ovog zadatka je pravljenje primera korišćenja ArrayListe u kombinaciji sa HashMap-om*

*Napraviti klasu Student kao uzadtku sa setovima a potom kreirati ArrayList-u studenta i ubaciti dva studenta u nju. Nakon ovoga kreirati Mapu koja će kao ključ imati Integer, a kao vrednost String. U String treba ubaciti sve položene ispite ovih studenata dok u key smetiti njihov indeks. Petljom prođite sve studente u ArrayListi i ispišite sve njihove položene ispite.*

Objašnjenje i uputstva za rešavanje zadatka:

1. kreiramo klasu Student sa navedenim atributima
2. kreiramo Main klasu
3. U Main klasi parvimo ArrayList naredbom

```
ArrayList<Student> studenti = new ArrayList<Student>();
```

4. metodom add() dodajemo studente u listu studenti

5. Keriramo Mapu naredbom

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

Ovde smo definisali kog je tipa KEY, a kog VALUE u Mapi

6. Metodom PUT() dodajemo nove podatke u Mapu

7. Ispisujemo sadržaj liste i uparujemo ga sa vrednostima iz mape

8. Izlaz iz programa:

*Student: Student{ime=Vuk, prezime=Vasic, indeks=1630}Polozeni ispiti:CS111, CS112\_\_\_\_\_Student: Student{ime=Ana, prezime=Peric, indeks=2005}Polozeni ispiti:\_\_\_\_\_*

## ZADATAK 2 - REŠENJE

### Rešenje zadatka 2

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Z02;

/**
 *
 * @author Aleksandra
 */
public class Student {

    // definišemo atribut klase Student
    private String ime;
    private String prezime;
    private int indeks;

    //podrazumevani konstruktor
    public Student() {
    }

    //konstruktor polja
    public Student(String ime, String prezime, int indeks) {
        this.ime = ime;
        this.prezime = prezime;
        this.indeks = indeks;
    }

    //getter za atribut ime
    public String getIme() {
        return ime;
    }

    //setter za atribut ime
```

```
public void setIme(String ime) {
    this.ime = ime;
}

//getter za atribut prezime
public String getPrezime() {
    return prezime;
}

//setter za atribut prezime
public void setPrezime(String prezime) {
    this.prezime = prezime;
}

//getter za atribut indeks
public int getIndeks() {
    return indeks;
}

//setter za atribut indeks
public void setIndeks(int indeks) {
    this.indeks = indeks;
}

@Override
public String toString() {
    return "Student{" + "ime=" + ime + ", prezime=" + prezime + ", indeks=" +
indeks + '}';
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 23 * hash + this.indeks;
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Student other = (Student) obj;
    if (this.indeks != other.indeks) {
        return false;
    }
    return true;
}
}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Z02;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

/**
 *
 * @author Aleksandra
 */
public class Main {

    public static void main(String[] args) {
        new Main();
    }

    public Main() {
        ArrayList<Student> studenti = new ArrayList<Student>();
        studenti.add(new Student("Vuk", "Vasic", 1630));
        studenti.add(new Student("Ana", "Peric", 2005));
        Map<Integer, String> polozeniIspiti = new HashMap<Integer, String>();
        polozeniIspiti.put(1630, "CS111, CS112");
        polozeniIspiti.put(1565, "IT250");

        for (Student s : studenti) {
            System.out.println("Student: " + s);
            System.out.println("Polozeni ispit:");
            for (Entry<Integer, String> ispit : polozeniIspiti.entrySet()) {
                if (ispit.getKey().equals(s.getIndeks())) {
                    System.out.println(ispit.getValue());
                }
            }
            System.out.println("_____");
        }
    }
}
```

## ZADATAK 3

*Cilj ovog zadatka je pravljenje primera korišćenja LinkedHashMapa.*

*Napraviti LinkedHashMapu koja u sebi ima 5 korisnika i njihov trenutni novac na racunu (String, Double). Ispisati informaciju o stanju svakog korisnika a potom promeniti jednom korisniku stanje na trenutno + 1000. Prikazati novo stanje klijenta.*

*Objašnjenje i uputstva za rešavanje zadatka:*

1. kreiramo Main klasu

2. kreiramo LinkedHashMap

```
LinkedHashMap lhm = new LinkedHashMap();
```

3. metodom PUT() dodajemo objekte

4. Uzimamo elemente mape naredbom

```
Set set = lhm.entrySet();
```

5. pravimo iterator po setu

6. Prikazujemo elemente

```
while (i.hasNext()) {  
    Map.Entry me = (Map.Entry) i.next();  
    System.out.print(me.getKey() + ": ");  
    System.out.println(me.getValue());  
}
```

7. Dodajemo zari 1000 dinara

```
double balance = ((Double) lhm.get("Zara")).doubleValue();  
lhm.put("Zara", new Double(balance + 1000));
```

## ZADATAK 3 - REŠENJE

### *Rešenje zadatka 3*

```
/*  
 * To change this license header, choose License Headers in Project Properties.  
 * To change this template file, choose Tools | Templates  
 * and open the template in the editor.  
 */  
package Z04;  
  
import java.util.Iterator;  
import java.util.LinkedHashMap;  
import java.util.Map;  
import java.util.Set;  
  
/**  
 *  
 * @author Aleksandra  
 */  
public class Main {  
  
    public static void main(String[] args) {
```

```

        new Main();

    }

    public Main() {
        LinkedHashMap lhm = new LinkedHashMap();
        lhm.put("Zara", new Double(3434.34));
        lhm.put("Mahnaz", new Double(123.22));
        lhm.put("Ayan", new Double(1378.00));
        lhm.put("Daisy", new Double(99.22));
        lhm.put("Qadir", new Double(-19.08));

        // Uzimamo elemente mape
        Set set = lhm.entrySet();
        // Uzimamo iterator
        Iterator i = set.iterator();
        //Prikazujemo elemente
        while (i.hasNext()) {
            Map.Entry me = (Map.Entry) i.next();
            System.out.print(me.getKey() + ": ");
            System.out.println(me.getValue());
        }
        System.out.println();
        // Dodajemo zari 1000 dinara
        double balance = ((Double) lhm.get("Zara")).doubleValue();
        lhm.put("Zara", new Double(balance + 1000));
        System.out.println("Zarino novo stanje je: " + lhm.get("Zara"));
    }
}

```

## ZADATAK 4

*Cilj ovog zadatka je pravljenje primera korišćenja TreeMape.*

*Napraviti TreeMap-u koja ima od podataka Broj indeksa kao ključ i ime studenta kao vrednost.*

*Prikazati da će TreeMapa automatski sortirati ljude u mapu po broju indeksa bez obzira na unet redosled.*

*Objašnjenje i uputstva za rešavanje zadatka:*

1. kreiramo Main klasu
2. kreiramo TreeMap koja kao KEY ima Integer tip, a VALUE String tip  

```
TreeMap<Integer, String> studenti = new TreeMap<Integer, String>();
```
3. metodom put() dodajemo u mapu proizvoljan broj studenata
4. primetiti da student nisu uredjeni po broju indeksa
5. pravimo Set od TreeMap

```
Set set = studenti.entrySet();
```

6. kreiramo iterator kroz set

```
Iterator i = set.iterator();
```

7. ispisujemo sadržaj seta

```
while (i.hasNext()) {  
    Map.Entry me = (Map.Entry) i.next();  
    System.out.print(me.getKey() + ": ");  
    System.out.println(me.getValue());  
}
```

8. Izlaz iz programa je:

1000: Milena

1254: Nikola

1565: Ana

1630: Vuk

Primetiti da je sadržaj sortiran po broju indeksa rastuće.

## ZADATAK 4 - REŠENJE

### *Rešenje zadatka 4*

```
/*  
 * To change this license header, choose License Headers in Project Properties.  
 * To change this template file, choose Tools | Templates  
 * and open the template in the editor.  
 */  
package z05;  
  
import java.util.Iterator;  
import java.util.Map;  
import java.util.Set;  
import java.util.TreeMap;  
  
/**  
 *  
 * @author Aleksandra  
 */  
public class Main {  
  
    public static void main(String[] args) {  
        new Main();  
    }  
  
    public Main() {  
        TreeMap<Integer, String> studenti = new TreeMap<Integer, String>();  
        studenti.put(1630, "Vuk");  
    }  
}
```

```

        studenti.put(1565, "Ana");
        studenti.put(1000, "Milena");
        studenti.put(1254, "Nikola");

        Set set = studenti.entrySet();
        Iterator i = set.iterator();
        while (i.hasNext()) {
            Map.Entry me = (Map.Entry) i.next();
            System.out.print(me.getKey() + ": ");
            System.out.println(me.getValue());
        }
    }
}

```

## ZADACI ZA SAMOSTALAN RAD

*Proverite svoje novostečeno znanje.*

**Zadatak 1.** Pokažite izlazni rezultat sledećeg koda:

```

public class Test {
    public static void main(String[] args) {
        Map<String, String> map = new LinkedHashMap<>();
        map.put("123", "John Smith");
        map.put("111", "George Smith");
        map.put("123", "Steve Yao");
        map.put("222", "Steve Yao");
        System.out.println("(1) " + map);
        System.out.println("(2) " + new TreeMap<String, String>(map));
    }
}

```

**Zadatak 2.** Napisati program koji učitava neodređen broj celih brojeva i pronalazi onaj koji se javlja najviše puta. Unos se prekida ako se unese broj 0. Na primer, ako se unese sekvenca 2 3 40 3 5 4 -3 3 3 2 0, broj 3 će biti onaj koji se najviše puta pojavljuje. Ukoliko postoji više brojeva koji se istovremeno javljaju najviše puta, svi brojevi treba biti odštampani. Na primer, pošto se brojevi 9 i 3 pojavljuju dva puta u listi 9 30 3 9 3 2 4, oba broja treba biti odštampana.

**Zadatak 3.** (Pogađanje glavnih gradova korišćenjem mapa) Napisati program koji će omogućiti da se u mapi čuvaju parovi država/glavni grad. Kreirati mapu sa ulaznim podacima. Vaš program treba da od korisnika zahteva da unese državu, a da onda za nju prikaže glavni grad.

## VIDEO: MAPE

*HashMap: Java Collections Framework Tutorial Part 3 (9,57 minuta)*



**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ Poglavlje 5

# Studija slučaja: Broj reči u tekstu

## STUDIJA SLUČAJA: BROJANJE REČI U TEKSTU

*Ova studija slučaja razvija program koji broji javljanje reči u nekom tekstu i prikazuje reči i njihovo javljanje u alfabetskom redosledu reči.*

Program upotrebljava TreeMap za skladištenje ulaza koji čine reč i broj njenog javljanja u tekstu. Za svaku reč, proverite da li postoji ključ u mapi. Ako ne postoji, dodajte ulaz u mapu koji čine reč i broj njenog javljanja. Za svaku reč, proveriti da li je ta reč već u mapi kao ključ. Ako nije, dodaj ulaz u mapu, tj. reč kao ključ i vrednost 1. U suprotnom, povećajte vrednost za tu reč (ključ) za 1 u mapi. Predpostavimo da su reči neosetljive za veličinu slova (npr. Good i good je isto). Listing klase CountOccurrenceOfWords predstavlja rešenje ovog problema.

Program kreira TreeMap (linija 10) za skladištenje parova reči (kao ključa) i broja njenih javljanja u tekstu. Sve vrednosti u mapi moraju da se uskladište kao objekti. Zato se umesto *int* koristi klasa *Integer* za broj javljanja reči. Program izvlači jednu reč iz teksta upotrebom metoda *split()* (linija 12) u klasi String. Za svaku izvučenu reč, program proverava da li je već uskladišćena u mapi kao ključ u mapi (linija 17). Ako nije, uskladišćuje se u mapu novi par: reči njen početni broj pojavljivanja u tekstu 1 (linija 18). U suprotnom, broj javljanja reči se povećava za 1. (linije 21-23). Program dobija ulaze u mapu u vidu seta (linija 29) i prelazi preko preko seta radi prikaza broja javljanja i same reči, za svaki ulaz (linije 32-33). Kako se koristi mapa stabla, ulazi se prikazuju po rastućem redosledu broja javljanja (vrednost).

Slika 1 prikazuje dobijeni rezultat.

a	2
class	1
fun	1
good	3
have	3
morning	1
visit	1

Slika 5.1 Rezultat prikazan u toku izvršenja programa

```
1 import java.util.*;
2
3 public class CountOccurrenceOfWords {
4     public static void main(String[] args) {
5         // Stavljanje teksta u String
6         String text = "Good morning. Have a good class. " +
7             "Have a good visit. Have fun!";
8
9         // Kreiranje TreeMap ya smeštaj reči, kao ključeva, i broja javljanja, kao
vrednost
```

```

10 Map<String, Integer> map = new TreeMap<>();
11
12 String[] words = text.split("[ \\n\\t\\r.,;:!?(){}"]);
13 for (int i = 0; i < words.length; i++) {
14     String key = words[i].toLowerCase();
15
16     if (key.length() > 0) {
17         if (!map.containsKey(key)) {
18             map.put(key, 1);
19         }
20         else {
21             int value = map.get(key);
22             value++;
23             map.put(key, value);
24         }
25     }
26 }
27
28 // Uzimanje svih ulaza u u set
29 Set<Map.Entry<String, Integer>> entrySet = map.entrySet();
30
31 // Dobijanje ključa i vrednosti za svaki ulaz
32 for (Map.Entry<String, Integer> entry: entrySet)
33     System.out.println(entry.getValue() + "\\t" + entry.getKey());
34 }
35 }

```

## ZADACI ZA SAMOSTALNI RAD

*Proverite vaše novostečeno znanje.*

**Zadatak 1.** Da li program CountOccurrenceOfWords može da radi ako se promeni linija 10 i sada sadrži:

3. Da li će program CountOccurrenceOfWords program raditi ako se linije 32-33 promene sa:

```
for (String key: map)
```

```
System.out.println(key + "\\t" + map.getValue(key));
```

```
Map<String, int> map = new TreeMap<>();
```

**Zadatak 2.** Da li program CountOccurrenceOfWords može da radi ako se linija 17 promeni sa:

```
if (map.get(key) == null) {
```

**Zadatak 3.** Da li će program CountOccurrenceOfWords raditi ako linije 32-33 promenimo u

```
for (String key: map)
    System.out.println(key + "\t" + map.getValue(key));
```

**Zadatak 4.** (Broj pojavljivanja reči u tekstualnom fajlu) Izmeniti prethodni zadatak tako da čita tekst iz tekstualnog fajla. Ime tekstualnog fajla se prosleđuje kao argument komandne linije. Reči su razdvojene prazninama, znacima interpunkcije (,;.:?), navodnicima (""), i otvorenim/zatvorenim zagradama. Izbrojati reči pod uslovom da je nebitno da li se unose mala ili velika slova (tj. smatrati da su Good i good iste reči). Reči moraju početi slovom. Prikazati izlaz u alfabetskom redosledu reči, pri čemu svakoj reči prethodi broj njenog pojavljivanja.

## ▼ Poglavlje 6

# Singularne i nepromenljive kolekcije i mape

## KLASA COLLECTIONS

*Možete kreirati singularne setove, liste i mape, kao i nepromenljive setove, upotrebom statičkih metoda klase Collections.*

Klasa **Collection** sadrži statičke metode za liste i kolekcije, kao i metode za kreiranje nepromenljivih singularnih setova, listi i mapa, kao i za kreiranje nepromenljivih (read-only) setova, listi i mapa (slika 1).

Klasa Collection definiše tri konstante: **EMPTY\_SET**, **EMPTY\_LIST**, **EMPTY\_MAP**—za oznaku praznih setova, listi i mapa. Ove kolekcije su nepromenljive. Klasa takođe obezbeđuje metod **singleton(Object o)** za kreiranje nepromenljivog seta koji sadrži samo singularnu stavku, metod **singletonList(Object o)** za kreiranje nepromenljive liste koja sadrži samo jednu singularnu stavku, i metod **singletonMap(Object key, Object value)** za kreiranje nepromenljive metode sa samo jednim ulazom.

Klasa Collection takođe obezbeđuje šest statičkih metoda koje vraćaju nepromenljive poglede na kolekcije: **unmodifiableCollection(Collection c)**, **unmodifiableList(List list)**, **unmodifiableMap(Map m)**, **unmodifiableSet(Set set)**, **unmodifiableSortedMap(SortedMap m)**, i **unmodifiableSortedSet(SortedSet s)**. Ovaj tip pogleda je kao referenca stvarnoj kolekciji. Ali, ne možete je promeniti preko nepromenljivog pogleda, jer to izaziva javljanje izuetka **UnsupportedOperationException**.

java.util.Collections	
<code>+singleton(o: Object): Set</code>	Vraća nepromenljiv set koji sadrži specificiran objekat
<code>+singletonList(o: Object): List</code>	Vraća nepromenljivu listu koji sadrži specificiran objekta
<code>+singletonMap(key: Object, value: Object): Map</code>	Vraća nepromenljivu mapu koja sadrži specificiran obj.
<code>+unmodifiableCollection(c: Collection): Collection</code>	Vraća nepromenljiv pogled na kolekciju.
<code>+unmodifiableList(list: List): List</code>	Vraća nepromenljiv pogled na listu.
<code>+unmodifiableMap(m: Map): Map</code>	Vraća nepromenljiv pogled na mapu.
<code>+unmodifiableSet(s: Set): Set</code>	Vraća nepromenljiv pogled na set.
<code>+unmodifiableSortedMap(s: SortedMap): SortedMap</code>	Vraća nepromenljiv pogled na sortiranu mapu.
<code>+unmodifiableSortedSet(s: SortedSet): SortedSet</code>	Vraća nepromenljiv pogled na sortirani set.

Slika 6.1 Klasa Collections sadrži statičke metode koji kreiraju singularne i nepromenljive liste, setove i mape.

## ZADACI ZA SAMOSTALAN RAD

*Proverite vaše novostečeno znanje.*

2. Šta se dešava kada izvršavate ovaj program?

**Zadatak 1.** Šta je pogrešno u sledećem kodu?

```
Set<String> set = Collections.singleton("Chicago");  
set.add("Dallas");
```

**Zadatak 2.** Šta se dešava kada pokrenete sledeći kod?

```
List list = Collections.unmodifiableList(Arrays.asList("Chicago", "Boston"));  
list.remove("Dallas");
```

## VIDEO: KOJU KOLEKCIJU KORISTITI?

*Deciding Which Java Collection to Use: Java Collections Framework  
(14,24 minuta)*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ Zaključak

### REZIME

#### *Pouke ove lekcije*

1. Set skladišti elemente bez dupliranja. Kolekcija koja dozvoljava duplirane elemente je lista..
2. Mapa skladišti parove ključ/vrednost. On brzo nalazi vrednost korišćenjem ključa.
3. Podržavaju se tri tipa setova: HashSet, LinkedHashSet, i TreeSet. HashSet skladišti elemente na nepredvidljiv način. LinkedHashSet skladišti elemente po redosledu sa kojim su elementi ubačivani u set. TreeSet skladišti sortirane elemente. Svi metodi u klasama HashSet, LinkedHashSet, i TreeSet se nassleđuju sa interfejsa Collection.
4. Interfejs Map povezuje ključeve sa elementima. Ključevi su kao indeksi. U kolekciji List indeksi su celi brojevi. U interfejsu Map ključevi mogu biti bilo kakvi objekti. Mapa ne može da sadrži duple ključeve. Svaki ključ se povezuje najviše za jednu vrednost. Interfejs Map obezbeđuje metode za pretraživanje, promenu i dobijanje kolekcije vrednosti i set ključeva.
5. Podržani su tri tipa mapa: HashMap, LinkedHashMap, i TreeMap. HashMap je efikasan u lociranju vrednosti, ubacivanju ulaza i u brisanju ulaza. LinkedHashMap podržava ređanje ulaza u mapi. Ulazi u Hash-Map nisu poređani, ali ulazi u LinkedHashMap može se pretraživati ili po redosledu po kom su poslednji put bili ubačeni u mapu, ili po redosledu po kom su poslednji put bile korišćene, tj. počev od elementa koji je najredje korišćen, pa do elementa koji je poslednji put korišćen. TreeMap je efikasan za prelaženje preko ključeva po sortiranom redosledu. Ključevi se mogu sortirati upotrebom interfejsa Comparable ili Comparator.