Objektno orijentirano programiranje

9: Kolekcije

Creative Commons

You are free to

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material

under the following terms

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **NonCommercial** — You may not use the material for commercial purposes.
- **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

- https://creativecommons.org/licenses/by-nc-sa/4.0/

Nizovi

- U prethodnim primjerima podaci su bili pohranjeni u nizovima
  - *Item[], Perishable[], …*
  - Moguće stvoriti i polje parametriziranog tipa (uz određena ograničena i zaobilazna rješenja za kreiranje takvih nizova)
- Nedostatci u radu s nizovima
  - Veličina niza se ne može mijenjati i mora se navesti prilikom stvaranja niza (ne smije biti premala, ne smije biti prevelika, …)
- Što napraviti kad je niz pun?
  - Stvoriti novi, veći niz, kopirati postojeće elemente u novi niz i promijeniti referencu u kojoj čuvamo niz
    - GC će u nekom trenutku obrisati stari niz
    - Kopirati možemo element po element ili koristiti ugrađenu metodu *Arrays.copy* (radi brže jer kopira blokove memorije)

ArrayList

- U Javi već postoji „niz koji se može proširiti” – *ArrayList*
  - Možemo postaviti inicijalnu veličinu, ali ako broj elemenata naraste preko inicijalne veličine stvara se novi veći niz
    - Navedeno ponašanje je enkapsulirano i korisnik se ne brine o tome

09_Collections/hr/fer/oop/collections/ArrayListMain.java

```
package hr.fer.oop.collections;
import java.util.ArrayList;
public class ArrayListMain {
    public static void main(String[] args) {
        ArrayList<Integer> arr = new ArrayList<>(10); //init.capacity
        System.out.println("Size: " + arr.size()); // 0
        for(int i=0; i<1000; i++)
                arr.add(2*i);
        System.out.println("Size: " + arr.size());  //1000
        System.out.println("ELement at pos. 750: " + arr.get(750));
    }
}
```

Neke od metoda klase *ArrayList*

- Tip podatka je *ArrayList<E>,* a ne *E[]*, stoga u Javi nije moguće koristiti uglate zagrade za dohvat elementa na nekoj poziciji
  - Dohvat elementa: *E get(int index)*
  - E je tip po kojim je *ArrayList* parametriziran (u prethodnom primjeru to je bio *Integer*)
- Još neke često korištene metode
  - *add(E element)*
    - Dodaje element na kraj
  - *add(int index, E element)*
    - ubacuje element na poziciju *index* (dolazi do pomaka)
  - *E set(int index, E element)*
    - mijenja element na poziciji *index*; metoda vraća stari element
  - E remove(int index)
    - uklanja (i vraća) element na poziciji *index* position (and returns it)

Vezana lista – klasa *LinkedList*

- Nizovi i *ArrayList* su praktični za dohvat pojedinog elementa, ali
  - Što se događa ako treba ubaciti ili izbaciti neki element?
    - Svi elementi nakon mjesta ubacivanja ili izbacivanja moraju se pomaknuti
  - Što kad se treba „povećati” kapacitet?
- Vezana lista sastoji se od elemenata (čvorova) gdje svaki element (čvor) sadrži referencu na podatak te referencu na sljedeći element
  - U slučaju dvostruko povezane liste sadrži i referencu na prethodni element
- Vezana lista se tada sastoji od reference na prvi (i zadnji) element koji čine listu
  - Ubacivanje i brisanje je brzo/jednostavno

```
class LinkedList<E> {
   Node<E> first;
   Node<E> last;
   ...
}
class Node<E> {
   E item;
   Node<E> next;
   Node<E> prev;
   ...
}
```

Ne izmišljajte toplu vodu – class *LinkedList*

- Java već sadrži klasu *LinkedList* s optimiranim kodom

```
public E get(int index) {
        checkElementIndex(index);
        return node(index).item;
}
Node<E> node(int index) {
    if (index < (size >> 1)) {
        Node<E> x = first;
        for (int i = 0; i < index; i++)
            x = x.next;
        return x;
    } else {
        Node<E> x = last;
        for (int i = size - 1; i > index; i--)
            x = x.prev;
        return x;
    }
}
```

LinkedList

- *LinkedList* ima velik broj metoda istih kao klasa *ArrayList*
  - Implementiraju isto sučelje: *List* (detaljnije malo kasnije)

```
package hr.fer.oop.collections;
import java.util.LinkedList;

public class LinkedListMain {
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<>();
        System.out.println("Size: " + list.size());
        for(int i=0; i<1000; i++)
                list.add(2*i);
        System.out.println("Size: " + list.size());
        System.out.println("ELement at pos. 750: " + list.get(750));
    }
}
```

09_Collections/hr/fer/oop/collections/LinkedListMain.java

Iteriranje kroz *LinkedList* i/ili *ArrayList*

- U oba slučaja može se koristiti for petlja i metoda *get*, ali dohvat i-tog elementa vezane liste je sporiji, jer je potrebno slijedno proći kroz vezanu listu
- Kao i kod nizova, može se koristiti *foreach* varijanta *for* petlje
  - Zašto je ovo moguće i kad, bit će diskutirano kasnije

09_Collections/hr/fer/oop/collections/ListIterate.java

```
package hr.fer.oop.collections;
import java.util.LinkedList;
import java.util.List;
public class ListIterate {
	public static void main(String[] args) {
		List<Integer> list = new LinkedList<>();
		for(int i=0; i<10; i++)
			list.add(2*i);
		for(Integer i : list)
			System.out.println(i);
...
```

Nepromjenjive liste i liste nepromjenjive veličine

- Java omogućava jednostavno kreiranje lista nepromjenjive veličine korištenjem *Arrays.asList* i nepromjenjivih lista metodama *List.of* i *List.copyOf*

09_Collections/ hr/fer/oop /collections/UnmodifiableList.java

```
import java.util.Arrays;
import java.util.List;
public class UnmodifiableList {
	public static void main(String[] args) {
		List<Integer> list = List.of(1,  2,  3);
		//list.add(4); //throws an Exception
		//list.set(0, 5); //throws and Exception
		System.out.println(list);

		list = Arrays.asList(1, 2, 3);
		//list.add(4); //throws an Exception
		list.set(0, 5);
		System.out.println(list);
...
```

Java Collections Framework (1)

- *List*, *ArrayList*, i *LinkedList* su dijelovi Javinog okvira kolekcija pod nazivom *Java Collection Framework*

- Kolekcija (spremnik, kontejner)
  - objekt koji grupira više drugih elemenata (drugih objekata)
  - kolekcija uobičajeno omogućava:
    - pohranu podataka
    - dohvat podataka
    - manipulaciju podacima (traženje elementa, brisanje, …)

- Kolekcijski okvir
  - standardizirana (unificirana) biblioteka klasa i sučelja za rad s kolekcijama

Java Collections Framework (2)

- Javin okvir kolekcija (engl. *Java Collection Framework*) sastavljen je od:
  - **sučelja** – omogućavaju rad s kolekcijama neovisno o samoj implementaciji
  - **implementacija** – klase koje predstavljaju konkretne implementacije sučelja uz različite složenosti podržanih operacija
  - **algoritama**
    - općenite metode za sortiranje, pretraživanje, i sl. nad objektima klasa iz kolekcijskih sučelja
    - isti algoritam (odnosno ista metoda) za različite implementacije (polimorfizam)

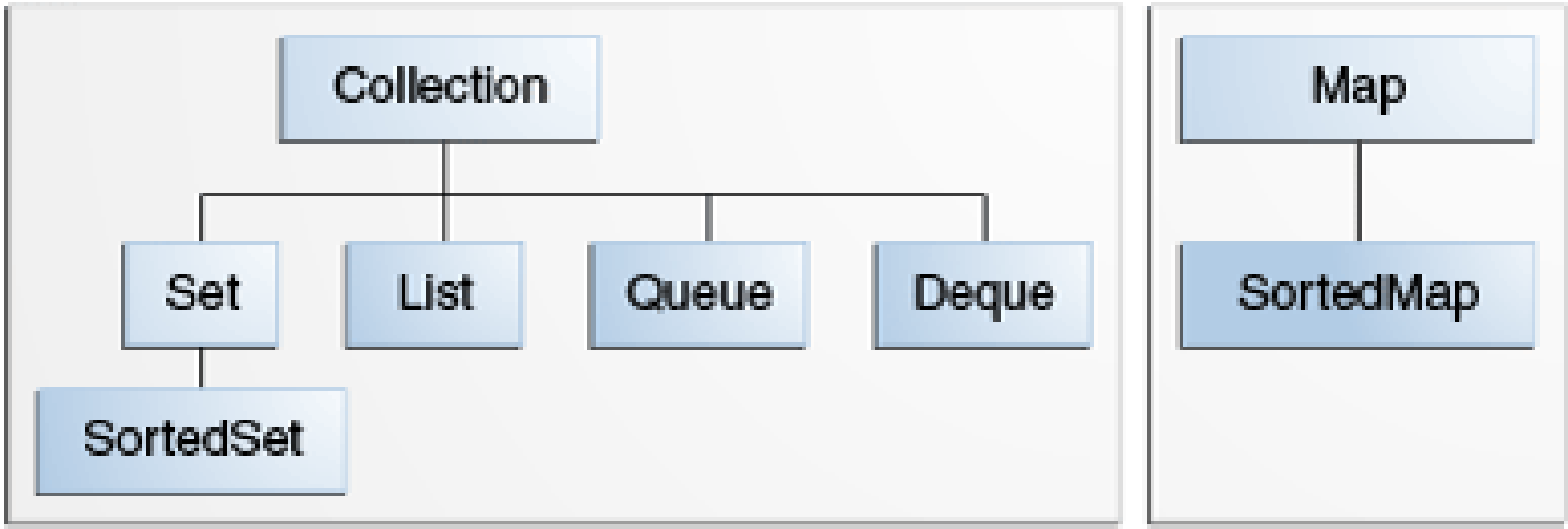
Prednosti uporabe Javinog okvira kolekcija

- Promovira višestruku iskoristivost koda
- Smanjenje potrebnog truda i količine koda uz povećanje kvalitete i brzine koda
  - Optimirane implementacije (npr. metoda *get* u *LinkedList*)
  - Algoritmi ovise o sučeljima – promjena implementacije ne utječe na ostatak koda
- Ubrzano učenje i olakšan razvoj novih API-ja

Osnovna sučelja Javinog okvira kolekcija

- Dvije hijerarhije kolekcija
  - jedna izvedena iz sučelja *Collection*
  - druga izvedena iz sučelja *Map*
  - Napomena: Slika ne prikazuje cijelu hijerarhiju, već samo najznačajnije dijelove. Detaljnije na:

    https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html



Sučelje *Collection*

- Modelira općenitu kolekciju elemenata
  - ne propisuje ništa vezano za poredak, duplikate i null elemente
    - Specifične vrste kolekcija modelirane su drugim sučeljima izvedenim iz sučelja *Collection* (npr. *List* za poredane kolekcije)
- Sučelja ne mogu propisati postojanje konstruktora, ali je uobičajena praksa da implementacije iz Javinog okvira kolekcija imaju barem dva konstruktora
  - konstruktor bez argumenata za stvaranje praznih kolekcija
  - konstruktor koji prima referencu na drugu kolekciju za stvaranje nove kolekcija na temelju postojeće
    - Ovaj konstruktor se naziva još i *conversion constructor* jer omogućava stvaranje kolekcije drugačijeg tupa od one u argumentu konstruktora

Opcionalne i *default* metode

- Java Collection Framework sadrži puno korisnih *default* metoda
  - Naknadno dodane u Javina sučelja kako bi se sučelje proširilo s (u većini slučajeva) odgovarajućim metoda za manipulaciju kolekcijom, a bez narušavanja kompatibilnosti sa starim kodom.
- Neke implementacije stvaraju nepromjenjive kolekcije ili kolekcije nepromjenjive veličine, pa stoga ne podržavaju sve metode opisane sučeljem
  - u popisu metoda takve metode su navedene kao opcionalne
    - U dokumentaciji implementacijske klase potrebno navesti podržane opcionalne operacije
  - ako operacija nije podržana tada metoda u konkretnoj implementaciji treba baciti iznimku *UnsupportedOperationException*
- Napomena: optional ≠ default !

Metode sučelja *Collection*

- Popis nekih od korisnih metoda sučelja Collection

```
public interface Collection<E> extends Iterable<E> {
        int size();
        boolean isEmpty();
        boolean contains(Object element);
        boolean add(E element); //optional
        boolean remove(Object element); //optional
        Iterator<E> iterator();
        boolean containsAll(Collection<?> c);
        boolean addAll(Collection<? extends E> c); //optional
        boolean removeAll(Collection<?> c); //optional
        boolean retainAll(Collection<?> c); //optional
        void clear(); //optional
        Object[] toArray();
        <T> T[] toArray(T[] a);
        default boolean removeIf(Predicate<? super E> filter) {...}
        default Stream<E> stream() {...}
```

Napomena veza za neke metode iz *Collection*

- Zašto neke metode imaju naoko neuobičajen potpis, npr. :

  boolean remove(Object element) umjesto
  boolean remove(E element)

  - Implementacije koriste statičku metodu *Objects.equals* (koja koristi metodu *equals* iz klase *Object*) za provjeru jesu li objekti isti

  boolean addAll(Collection<? extends E> c) umjesto
  boolean addAll(Collection<E> c);

  - Zamislimo da imamo klasu *Food* koja nasljeđuje *Item*. Ovakav potpis metode omogućava nam da dodamo elemente iz Collection<Food> u Collection<Item>

- Neki slični primjeri će biti objašnjeni kasnije
  - npr. super i sučelje *Predicate* u metodi *removeIf*

Sučelje *Iterable* i iteratori

- Sučelje *Collection* nasljeđuje sučelje *Iterable* koje omogućava obilazak svih elemenata kolekcije *for-each* varijantom for petlje

```
for(Type item : collection)
        radi nešto s item (ali ne mijenjaj kolekciju)
```

- Ovo je pojednostavljena verzija koncepta *iteratora* - objekta koji omogućava obilazak (i u nekim slučajevima uklanjanje elemenata iz kolekcije)
  - Sučelje *Iterable<T>* definira metodu *Iterator<T> iterator()*
  - Sučelje *Iterator* definira tri metode: *hasNext, next*, i (opcionalno) *remove*

```
Iterator<SomeType> it = collection.iterator();
while(it.hasNext()) {
        Type item = it.next();
        radi nešto s item
}
```

Sučelje *List*

- *List* "je" *Collection*
  - Sučelje *List* nasljeđuje (proširuje) sučelje *Collection* metodama za poredak elemenata u kolekciji

```
public interface List<E> extends Collection<E> {
   E get(int index);
   E set(int index, E element);    //optional
   boolean add(E element);         //optional
   void add(int index, E element); //optional
   E remove(int index);            //optional
   boolean addAll(int index, Collection<? extends E> c);  //optional
   int indexOf(Object o);
   int lastIndexOf(Object o);
   ListIterator<E> listIterator();
   ListIterator<E> listIterator(int index);
   List<E> subList(int from, int to);
```

Pristup na temelju pozicije

Pretraživanje

Omogućava iteriranje u oba smjera

Primjer 1.

09_Collections/hr/fer/oop/collections/example1/*.java

- Sa standardnog ulaza učitavati cijele brojeve i dodavati ih u listu dok se ne pojavi negativni broj. Nakon toga, iz liste ukloniti brojeve ispod prosjeka i sortirati listu.
- Rješenje je podijeljeno u nekoliko dijelova
  - vlastita klasa *Loader* koja učitava ne-negativne brojeve koristeći *Scanner* (bit će inicijaliziran na *System.in* u glavnom programu)
  - izračun prosječne vrijednosti u listi
  - vlastita klasa *BelowThreshold* (implementira sučelje *Predicate*)
    - **predikat** (općenito): metoda koja za neki objekt provjerava zadovoljava li neki uvjet (vraća istinu ili laž)
  - uklanjanje elemenata metodom *removeIf* i napisanim predikatom
  - Sortiranje liste korištenjem statičke metode iz klase *Collections*
    - *class Collections ≠ interface Collection*
    - Collections sadrži samo statičke metode poput sort, reverse, shuffle, …

Primjer 1. – Napomena uz *Predicate* i *super*

09_Collections/hr/fer/oop/collections/example1/*.java

- Zadana je lista cijelih brojeva: List<Integer>
- Default metoda *removeIf* očekuje predikat s kojim će testirati treba li ukloniti neki element ili ne
  - *removeIf* ima sljedeći potpis

    default boolean removeIf(Predicate<? super E> filter)
  - Ovo znači da valjani argument može biti Predicate<Integer>, ali i Predicate<Number> , t.j. *Predicate<? super Integer>*
  - *removeIf* šalje *Integer* metodi *test* sučelja *Predicate*. Stoga možemo napisati bilo koji predikat kojem se može predati *Integer*
    - *Integer extends Number*

```
public class BelowThreshold implements Predicate<Number> {
        public boolean test(Number d) {
                ...
        }
}
```

Sučelje *Set*

- ***Skup** (engl. Set)* **je kolekcija koja ne može sadržavati duplikate**.
  - Sučelje Set sadrži samo metode naslijeđene iz *Collection* uz gore naveden uvjet vezan za duplikate.
  - Ovo ograničenje je semantičko
    - sučelje ne može sintaksno izvesti takvo ograničenje
- Javin okvir kolekcija dolazi s tri implementacije sučelja *Set*:
  - *HashSet, TreeSet, LinkedHashSet*
  - Koju implementaciju sučelja *Set* odabrati?
    - Je li važan redosljed elemenata pri iteriranju?
    - Koliko ima zapisivanja (izmjena skupa), a koliko dohvata podataka?
    - Odabrati implementaciju koja nudi zadovoljavajuću funkcionalnost i performanse

Implementacije sučelja *Set*

- *HashSet*
  - sprema elemente u pretince
  - uz pretpostavku da su elementi ravnomjerno raspršeni po pretincima osigurava najbolje performance
    - većina osnovnih operacija ne ovisi o broju elemenata u skupu
  - ne garantira poredak kod obilaska elemenata
- *TreeSet*
  - Sprema elemente u crveno-crna stabla (specijalni oblik binarnog stabla traženje)
    - iterator elemente skupa obilazi sortiranim poretkom
- *LinkedHashSet*
  - Slično HashSetu uz dodatnu vezanu listu za očuvanje redoslijeda dodavanja u skup (koristi se za iteriranje po skupu)

Primjer 2. Ispis argumenata programa bez duplikata (1/2)

- Vlastita metoda *addToSet* puni skup i vraća referencu na njega
- Vlastita metoda *print* iterira kroz bilo što što je *Iterable*

```
public static void main(String[] args) {
	System.out.println("Using HashSet:");
	print(addToSet(new HashSet<String>(), args));

	System.out.println("Using TreeSet:");
	print (addToSet(new TreeSet<String>(), args));

	System.out.println("Using LinkedHashSet:");
	print (addToSet(new LinkedHashSet<String>(), args));
}
private static Set<String> addToSet(Set<String> set, String[] arr) {
	for (String element : arr)
		set.add(element);
	return set;
}
```

09_Collections/hr/fer/oop/collections/UniqueArguments.java

Primjer 2. Ispis argumenata programa bez duplikata (1/2)

Program arguments:

```
23 76 55 23 12 99 76 11 10
```

```
private static void print(Iterable<String> col) {
        for (String element : col)
                System.out.println(element);
        System.out.println();

//if using iterator instead of for-each
//      Iterator<String> iterator = col.iterator();
//      while(iterator.hasNext())
//              System.out.println(iterator.next());
//      System.out.println();
}
```

```
Using HashSet:
55
99
11
23
12
76
10

Using TreeSet:
10
11
12
23
55
76
99

Using LinkedHashSet:
23
76
55
12
99
11
10
```

09_Collections/hr/fer/oop/collections/UniqueArguments.java

- Napomena: Argumenti su stringovi
  - Kakav bi ispis bio da je jedan od argumenata bio 150?

Složenosti uobičajenih operacija sučelja *Set* i *List* u različitim implementacijama

- (Vremenska) složenost
  - Umjesto u vremenskim jedinicama, izražava se kao funkcija veličine ulaznih podataka
  - **red veličine** potrebnih koraka/naredbi da se obavi pojedini postupak
    - npr. zanima nas očekujemo li da se, ako se broj elemenata u nekoj kolekciji podupla, vrijeme izvođenja (broj izvršenih naredbi) poveća za 2 ili 2 puta, da li možda 4 puta, 8 puta, … ili će možda ostati približno isto

- Koja je složenost metoda
  - *contains(Object e)?*
  - *remove(Object e)?*
  - *add(E e)?*
    za *HashSet i TreeSet* i sličnih operacije u *ArrayList* i *LinkedList*?

Primjer 3.

- Napisati funkciju koja će primiti polje imena (polje stringova) i ispisati svako ime samo jednom, ali u obrnutom poretku od pojavljivanja u polju
  a) koristeći samo primljeno polje (dakle, bez dodatnih podatkovnih struktura)
  b) koristeći listu i skup
     - *uputa*: elemente dodati u listu ako već nisu u listi, upotrijebiti skup radi brže provjere da li je element već u listi
  c) Koristeći prikladnu implementaciju skupa
     - *uputa*: imamo implementaciju skupa koja čuva poredak!

09_Collections/hr/fer/oop/collections/example3/*.java

- Diskutirati složenost pojedine varijante
  - Koliko naredbi će se izvršiti ako polje sadrži 10 različitih imena, a koliko ako sadrži 50 različitih imena

Sučelje *Map*

- **Mapa** ili preslikavanje (engl. *Map*) je kolekcija uređenih parova (ključ, vrijednost)
  - mapa ne može pohranjivati više **istih** ključeva
  - svaki ključ ima pridruženu **jednu** vrijednost
  - više ključeva može imati istu vrijednost
- Par opisan sučeljem *Map.Entry* (definirano unutar sučelja *Map*)
  - ključevi i vrijednosti mogu biti bilo kojeg tipa (osim primitivnih tipova)
  - jednom dodani ključ je nepromjenjiv
    - moguće ga je ukloniti iz mape
- Primjeri preslikavanja
  - Osoba → Telefonski broj (može i obrnuto)
  - Predmet → Skup upisanih studenata
  - Ime → broj pojavljivanja
- Rječnik (*Dictionary, C#*), *asocijativno polje* (JavaScript, PHP)

```
interface Entry<K,V> {
  K getKey();
  V getValue();
  V setValue(V value);
}
```

Sučelje *Map*

- Primijetiti da *Map* nije *Iterable* (posebno stablo nasljeđivanja)

```
public interface Map<K,V> {
        int size();
        boolean isEmpty();
        boolean containsKey(Object key);
        boolean containsValue(Object value);
        V get(Object key);
        V put(K key, V value); //optional
        V remove(Object key); //optional
        void putAll(Map<? extends K, ? extends V> m); //opt.
        void clear(); //optional
        Set<K> keySet();
        Collection<V> values();
        Set<Map.Entry<K, V>> entrySet();

        boolean equals(Object o);
        int hashCode();
```

Osnovne operacije

Masovne operacije

Različiti pogledi na mapu

Sučelje *Map* – *default* methode

- Mnoštvo korisnih metoda
  - neke od njih će biti korištene u sljedećim predavanjima

```
default V getOrDefault(Object key, V defaultValue)
default void forEach(BiConsumer<? super K, ? super V> action)
default void replaceAll(BiFunction<? super K, ? super V, ? extends V>
                                                                function)
default V putIfAbsent(K key, V value)
default boolean remove(Object key, Object value)
default boolean replace(K key, V oldValue, V newValue)
default V replace(K key, V value)
default V computeIfAbsent(K key,
            Function<? super K, ? extends V> mappingFunction)
default V computeIfPresent(K key,
   BiFunction<? super K, ? super V, ? extends V> remappingFunction)
default V compute(K key,
     BiFunction<? super K, ? super V, ? extends V> remappingFunction)
default V merge(K key, V value,
   BiFunction<? super V, ? super V, ? extends V> remappingFunction)
```

Ugrađene implementacije sučelja *Map*

- *HashMap, TreeMap, LinkedHashMap*

- Implementacija, ponašanje i performance slične implementacijama sučelja *Set*

Primjer sa sučeljem *Map* (1/2)

- Prebrojati pojavljivanja pojedinog imena (unos završiti s *quit*)
  - Pokrenite program s različitim implementacija mape i uočite razliku

09_Collections/hr/fer/oop/collections/MapExample.java

```
public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Map<String, Integer> names = new HashMap<>();
        // Map<String, Integer> names = new TreeMap<>();
        // Map<String, Integer> names = new LinkedHashMap<>();
        System.out.println("Enter names (quit for end):");
        String name;
        while (!(name = scanner.next()).equals("quit")) {
                Integer val = names.get(name);
                names.put(name, val == null ? 1 : val + 1);
        }
        for (Map.Entry<String, Integer> entry : names.entrySet())
                System.out.format("%s occured %d time(s)%n",
                        entry.getKey(), entry.getValue());
}
```

Primjer sa sučeljem *Map* (2/2)

- Mapa nije izvedena iz *Collection* ili *Iterable* i nema *iterator*
- Korisnicima nudi poglede kroz tri dodatne kolekcije
  - *keySet* : skup ključeva
  - *values*: kolekcija vrijednosti koje su pridružene ključevima u mapi
    - razmislite zašto ovo nije skup?
  - *entries*: skup uređenih parova (ključ, vrijednost)

```
public interface Map<K,V> {
   Set<K> keySet();
   Collection<V> values();
   Set<Map.Entry<K, V>> entrySet();
   ...
```

```
public static void main(String[] args) {
        Map<String, Integer> names = new ...
        ...
        for (Map.Entry<String, Integer> entry : names.entrySet())
                System.out.format("%s occured %d time(s)%n",
                        entry.getKey(), entry.getValue());
}
```

09_Collections/hr/fer/oop/collections/MapExample.java

- Prilikom iteriranja moguće je promijeniti vrijednost pridruženu nekom ključu, ali ne ključ i mapu

Ostala sučelja iz *Java Collection Frameworka*

- Java Collection Framework sadrži i ostale korisne strukture, npr.
  - *Queue* — red na principu FIFO (first-in, first-out)
  - *PriorityQueue* – prioritetni red (elementi složeni po veličini)
  - *Deque* — (double ended queue) red s mogućnošću dodavanja i uklanjanja elemenata s oba kraja

- Napomena: Java još uvijek sadrži i neke starije, baštinjene (engl. *legacy*) klase poput *Stack* i *Vector*
  - Umjesto *Stack* preporuča se korištenje *Deque* na način da se elementi stavljaju i skidaju iz iste strane reda
  - *Vector* → *ArrayList* (osim u nekim iznimnim slučajevima*)