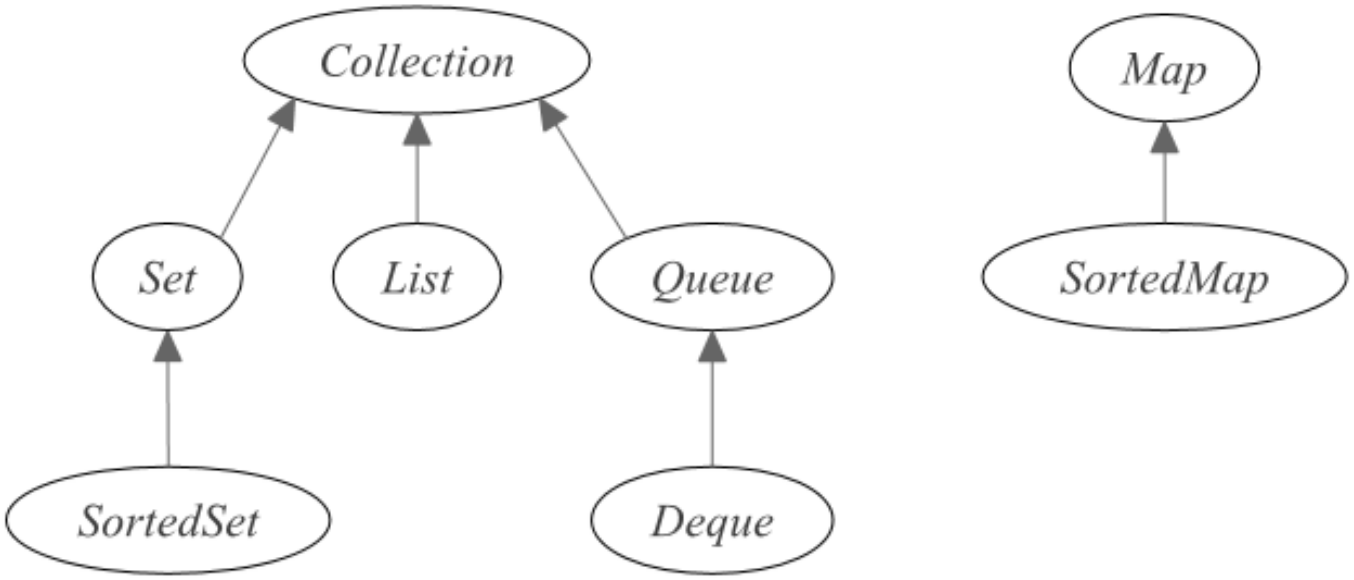
Kolekcije

- Objektno-orjentisano programiranje 1

Kolekcije

- **Kolekcija je objekat koji sadrži druge objekte**
  - Liste, skupovi, stekovi, redovi opsluživanja
- **Mapa je kolekcija parova (ključ, vrednost) u kojoj nema duplikata ključeva**
- Interfejsi i klase koje realizuju standardne kolekcije su u paketu **java.util**
- **Centralni interfejsi iz paketa java.util**



Interfejs Collection<E>

- **Generički interfejs koji definiše sledeće operacije**
  - `boolean add(E e)`
  - `void clear()`
  - `boolean remove(Object o)`
  - `boolean contains(Object o)`
  - `boolean isEmpty()`
  - `int size()`
  - `Iterator<E> iterator()`
- **Iterator<E> je generički interfejs koji definiše sledeće operacije**
  - `boolean hasNext()`
  - `E next()`
  - `void remove()`
    - Briše iz kolekcije poslednji element isporučen kroz iterator
    - **Ponašanje iteratora nije specificirano ako se iz kolekcije uklanjaju elementi dok se kroz kolekciju iterira osim koristeći remove() metod**

Interfejsi Set i SortedSet

- **Generički interfejsi koji opisuju operacije u radu sa skupovima**
  - Operacije iz interfejsa Collection
  - **add(E e)** uspeva ukoliko u skupu ne postoji **x** takvo da je **x.equals(e)** tačno
- **Generičke klase HashSet i LinkedHashSet implementiraju interfejs Set**
  - **HashSet** – skup realizovan heš tabelom (otvoreno hešovanje)
  - **LinkedHashSet** – skup realizovan istovremeno i heš tabelom i listom (možemo iterirati u redosledu umetanja)
- **SortedSet – skup kod koga su elementi sortirani po prirodnom uređenju ili po proizvoljnom komparatoru (iteriranje po poretku)**
  - metode koje vraćaju najmanji i najveći element skupa
  - metoda koja vraća podskup koji sadrži sve elemente veće/manje od nekog elementa
  - metoda koja vraća podskup koji sadrži sve elemente u nekom intervalu
- **Generička klasa TreeSet implementira interfejs SortedSet**

```java
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedHashSet;
import java.util.Set;
import java.util.TreeSet;

public class SkupoviDemo {

	private static void formSkup(Set<String> s, String[] el) {
		for (int i = 0; i < el.length; i++)
			s.add(el[i]);
	}

	private static void ispisi(Set<String> s) {
		Iterator<String> it = s.iterator();
		while (it.hasNext()) {
			String e = it.next();
			System.out.print(e + " ");
		}
		System.out.println();
	}

	...
```

```
public static void main(String[] args) {
        String[] niz = {"Mina", "Ceca", "Mara", "Beba"};

        Set<String> s1 = new HashSet<String>();
        Set<String> s2 = new LinkedHashSet<String>();
        Set<String> s3 = new TreeSet<String>();

        formSkup(s1, niz);
        formSkup(s2, niz);
        formSkup(s3, niz);

        ispisi(s1); ispisi(s2); ispisi(s3);
}
```

Izlaz:

Mara Mina Beba Ceca
Mina Ceca Mara Beba
Beba Ceca Mara Mina

```
public static void main(String[] args) {
	String[] niz = {"Marina", "Ceca", "Anastasija", "Iva"};

	Set<String> s1 = new TreeSet<String>();
	formSkup(s1, niz);
	ispisi(s1);

	// Interfejs Comparator je funkcijski
	// Promenljivoj tipa Comparator mozemo
	// dodeliti anonimnu metodu (lambda izraz)
	Comparator<String> cmp =
		(str1, str2) -> str1.length() - str2.length();

	Set<String> s2 = new TreeSet<String>(cmp);
	formSkup(s2, niz);
	ispisi(s2);
}
```

Izlaz:

Anastasija Ceca Iva Marina

Iva Ceca Marina Anastasija

Interfejs List<E>

- **Lista je (uređena kolekcija) sekvenca elemenata:**
  - **elementima liste možemo pristupati preko indeksa**
- Metode definisane interfejsom
  - `boolean add(int index, Object o)`
  - `E get(int index)`
  - `int indexOf(Object o)`
  - `E remove(int index)`
  - `void set(int index, E element)`
  - `ListIterator<E> listIterator()`
    - **`hasNext()`, `next()`, `hasPrevious()`, `previous()`**
- **Klase ArrayList i LinkedList implementiraju List**
  - ArrayList – sekvenca realizovana dinamički proširivim nizom
  - LinkedList – sekvenca realizovana dvostruko-povezanom listom

Interfejs List<E>

- **Klase ArrayList i LinkedList implementiraju List**
  - Kod obe klase metod **add()** dodaje na kraj liste
- **ArrayList je efikasnija kada elementima pristupamo preko indeksa i kada elemente dodajemo/brišemo sa kraja**
- **U svim ostalim slučajevima LinkedList je efikasnija**
- Klasa LinkedList takođe implementira interfejs **Deque** (**double ended queue**)
  - Metode: **addFirst, removeFirst, addLast, removeLast**
  - Stoga, LinkedList možemo koristiti i kao
    - Stek – last in first out
    - Queue (red opsluživanja) – first in first out

Klasa Collections iz java.util

- **Collections.sort(List<E> l)**
  - Sortiranje po prirodnom uređenju
  - Elementi liste su objekti klase koja implementira Comparable
- **Collections.sort(List<E> l, Comparator<E> cmp)**
  - Sortiranje po proizvoljnom komparatoru
- **Collections.binarySearch(List<E> l, E key)**
  - Binarno pretraživanje
  - Elementi liste su objekti klase koja implementira Comparable
- **Collections.copy(List<E> dst, List<E> src)**
  - Kopiranje liste dst u listu src
- **Collections.shuffle(List<E> src)**
  - Izmeša elemente liste

Interfejsi Queue i Dequeue

- **Generički interfejs koji opisuje operacije u radu sa redovima opsluživanja**

	Throws exception	*Returns special value*
Insert	`add(e)`	`offer(e)`
Remove	`remove()`	`poll()`
Examine	`element()`	`peek()`

- **Klasa PriorityQueue implementira Queue**
- **Interfejs Deque nasleđuje Queue**

	First Element (Head)		**Last Element (Tail)**	
	Throws exception	*Special value*	*Throws exception*	*Special value*
Insert	`addFirst(e)`	`offerFirst(e)`	`addLast(e)`	`offerLast(e)`
Remove	`removeFirst()`	`pollFirst()`	`removeLast()`	`pollLast()`
Examine	`getFirst()`	`peekFirst()`	`getLast()`	`peekLast()`

Interfejs Map<K, V>

- **Interfejs koji opisuje operacije u radu sa mapama**
- **Metode**
  - `V put(K key, V value)`
  - `V get(Object key)`
  - `V remove(Object key)`
  - `Set<Map.Entry<K, V>> entrySet();`

```
interface Entry {
    K getKey();
    V getValue();
}
```

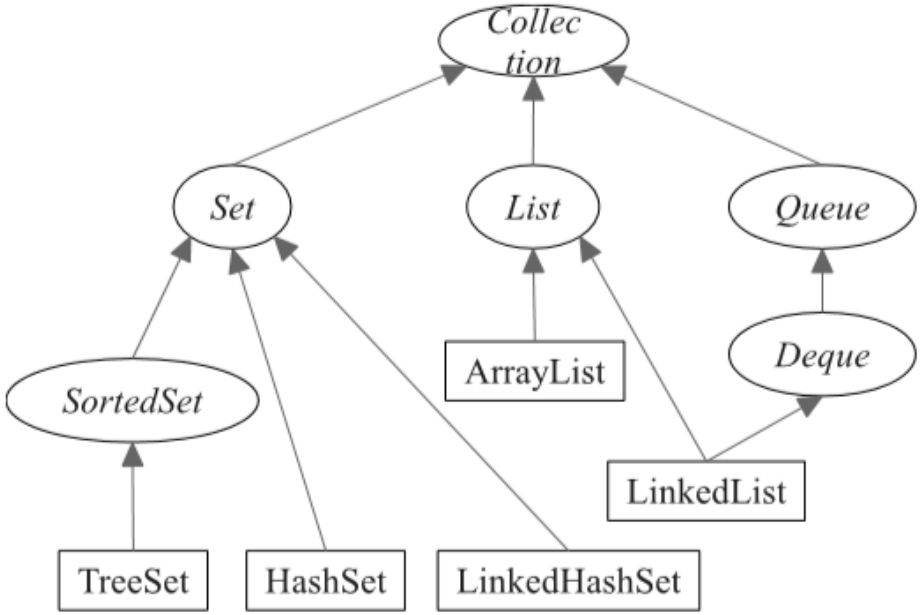
    - **Skup pogled na mapu, operacije nad skupom utiču na mapu i obratno**

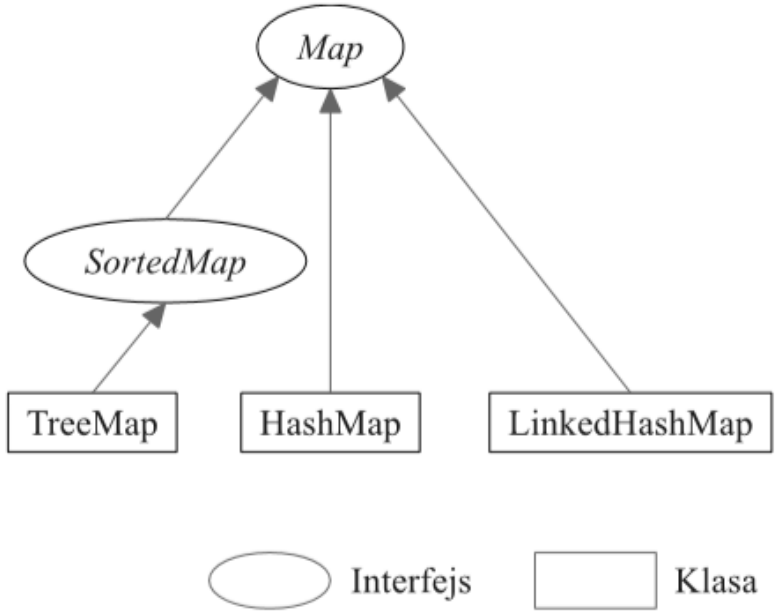
Interfejs Map

- **Klase HashMap i LinkedHashMap implementiraju interfejs Map**
  - **HashMap** – mapa realizovana heš tabelom (otvoreno hešovanje)
  - **LinkedHashMap** – mapa realizovana istovremeno i heš tabelom i listom
    - Iteriranje kroz mapu u redosledu umetanja
- Interfejs **SortedMap** nasleđuje **Map** i dodaje sledeće operacije
  - Vraćanje najvećeg/najmanjeg ključa
  - Kreiranje podmape koja sadrži ključeve veće/manje od nekog ključa
  - Kreiranje podmate koja sadrži ključeve iz nekog intervala
- Klasa **TreeMap** implementira interfejs **SortedMap**

```java
import java.util.*;
import java.util.Map.Entry;

public class MapExample {
	public static void main(String[] args) {
		Map<String, Integer> uceniciOcene = new HashMap<>();
		uceniciOcene.put("Mika", 3);
		uceniciOcene.put("Pera", 4);
		uceniciOcene.put("Zika", 2);
		uceniciOcene.put("Mara", 5);

		// pretrazivanje mape
		Integer ocena = uceniciOcene.get("Stavra");
		if (ocena == null)
			System.out.println("Stavra nije ocenenjen");
		else
			System.out.println("Ocena je " + ocena);

		// iteriranje kroz mapu
		Set<Entry<String, Integer>> es = uceniciOcene.entrySet();
		Iterator<Entry<String, Integer>> it = es.iterator();
		while (it.hasNext()) {
			Entry<String, Integer> e = it.next();
			System.out.println(e.getKey() + ", " + e.getValue());
		}
	}
}
```

Kolekcije


Collec
tion
Set
List
Queue
SortedSet
ArrayList
Deque
LinkedList
TreeSet
HashSet
LinkedHashSet


Map
SortedMap
TreeMap
HashMap
LinkedHashMap
Interfejs
Klasa