9-2

Objektno orijentisano programiranje vežbe

Staša Vujičić Stanković

Povezana lista

- Konstruktori:
  - bez argumenata
  - sa argumentom *Collection*<> koji kreira *LinkedList*<> objekat koji sadrži objekte kolekcije koja se prosleđuje kao argument.
- Metode:
  - **add(), addAll()** – kao kod klase Vector<>
  - **addFirst()** – dodaje objekat na početak liste
  - **addLast()** – dodaje objekat na kraj liste
  - **get(int)** – kao kod klase Vector<>

Povezana lista

- **getFirst(), getLast()**
- **remove(), removeFirst(), removeLast()**
- **set()**
- **size()** – vraća broj elemenata liste

- Metodom *iterator()* dobija se *Iterator*<> objekat nad listom
- Metodom *listIterator()* objekat *ListIterator*<>, kao i kod klase *Vector*<>.

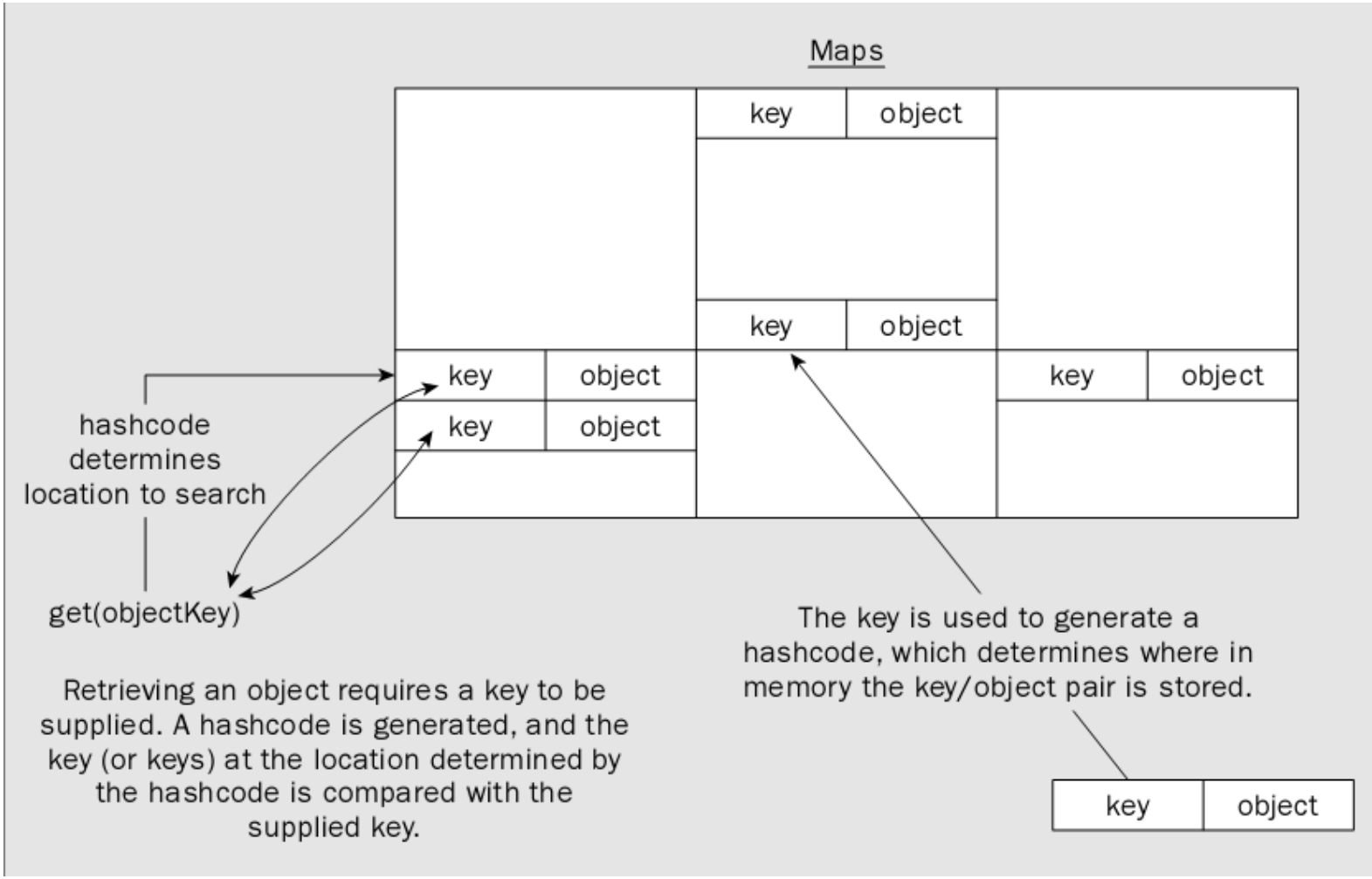
Zadatak - linkedList

- Point.java
- PolyLine.java
- TryPolyLine.java

Katalozi

- **Katalog (HashMap<K, V>)** takođe nazivamo i rečnikom. Zajedno se čuvaju i ključ i objekat.
- Svaki objekat je jedinstveno određen svojim ključem.
- Svi ključevi, iz tog razloga, moraju da budu različiti.
- Izdvajanje objekata iz kataloga vrši se pomoću odgovarajućeg ključa,
  jer ključ određuje gde se u katalogu nalazi objekat.

KATALOZI


Maps
key
object
key
object
key
object
key
object
key
object
hashcode
determines
location to search
get(objectKey)
Retrieving an object requires a key to be supplied. A hashcode is generated, and the key (or keys) at the location determined by the hashcode is compared with the supplied key.
The key is used to generate a hashcode, which determines where in memory the key/object pair is stored.
key
object

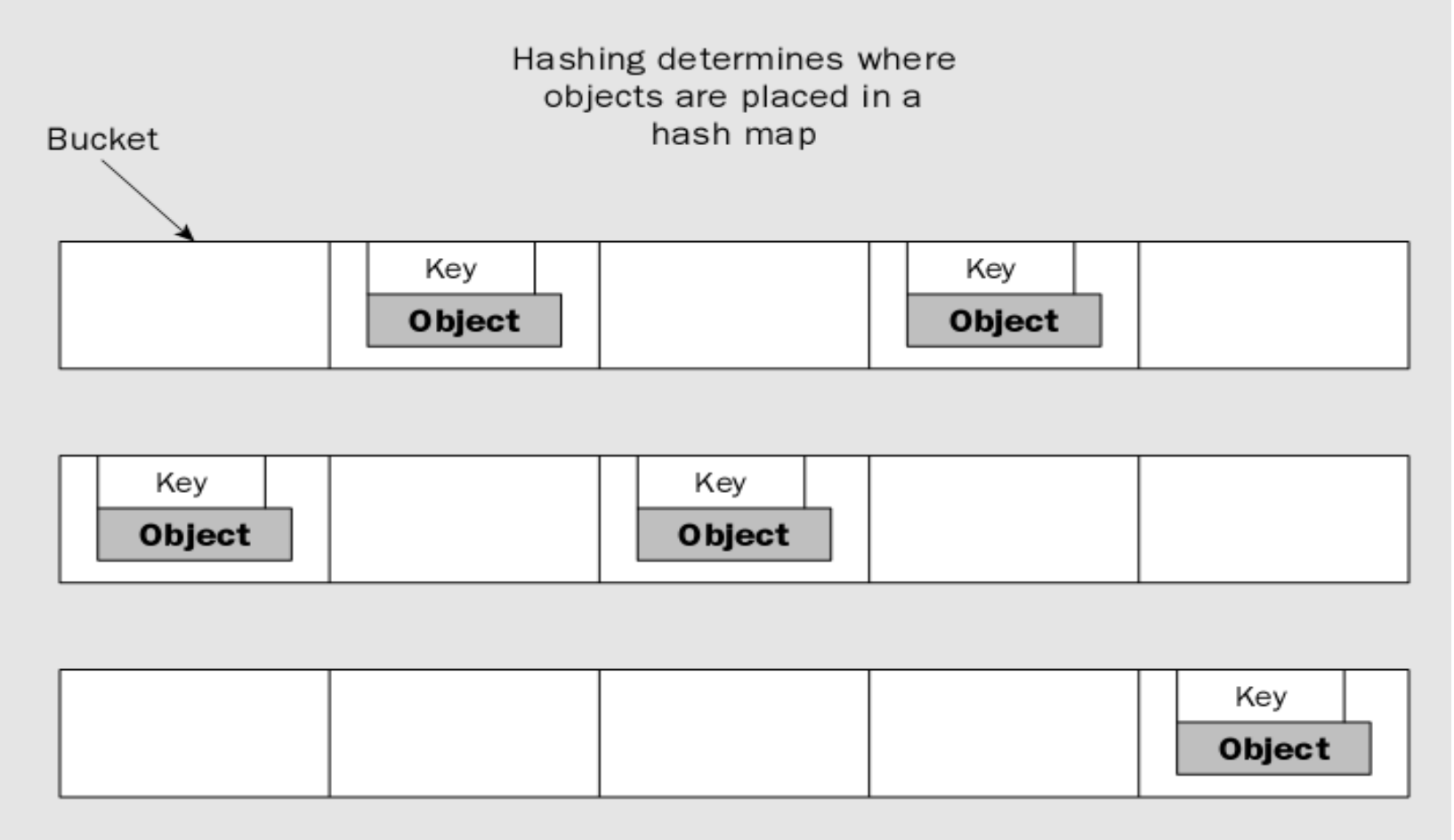
Katalozi

- Sve klase za rad sa katalozima implementiraju generički interfejs **Map<K,V>.**
- Razmatraćemo generičku klasu **HashMap<K,V>.**
- Implementacija kataloga pomoću generičke klase HashMap<> podrazumeva da su parovi ključ/objekat smešteni u niz.

Katalozi

- Indeks u nizu dobija se na osnovu ključa za šta se koristi metod **hashCode().**
- Ovaj metod nasleđuje se iz klase Object i on proizvodi podrazumevani heškod,
  osim ako nije predefinisan u nekoj od izvedenih klasa.
- Stoga, bilo koji objekat može da bude korišćen kao ključ. Na osnovu njega se metodom hashCode() generiše heškod kojim se određuje indeks para sa datim ključem u nizu.

KATALOZI


Hashing determines where objects are placed in a hash map
Bucket
Key
Object
Key
Object
Key
Object
Key
Object
Key
Object

Heširanje

- Prilikom heširanja može da se javi problem **kolizije** – dva različita ključa proizvode istu heš-vrednost.
- HashMap<> rešava problem kolizije smeštajući parove sa istom heš-vrednošću u povezanu listu. Pristup objektu se onda vrši na sledeći način:
  - računa se heškod na osnovu ključa i potom pronalazi lokacija para
  - pretražuje se povezana lista kako bi se pronašao par sa datim ključem.

Heširanje

- Ako želimo da kao ključeve koristimo objekte koje sami definišemo, treba da predefinišemo metod **equals()** iz klase Object.
- Da bi se obezbedila potrebna funkcionalnost, nova verzija treba da vrati true kada dva različita objekta sadrže iste vrednosti.
- Takođe, moguće je predefinisati metod **hashCode()** tako da raspodela bude prilično uniformna na skupu mogućih vrednosti za ključeve.
- Jedan način je da se za svaki podatak-članicu klase generiše ceo broj npr. postojećim metodom hashCode() koji se zatim množi prostim brojem (svaki član različitim) i na kraju se dobijeni rezultati sumiraju.

Heširanje

- Generisanje celog broja za podatak-članicu klase se obično vrši pozivanjem metoda **hashCode()**.
- Primer: neka se objekat klase **Osoba** koristi kao ključ. Članice klase su: *ime, prezime* tipa *String* i *godine* tipa *int*.

```
public int hashCode() {
   return 13*ime.hashCode() +
   17*prezime.hashCode() + 19*godine;
}
```

- Proste brojeve treba birati tako da ne budu preveliki, kako rezultat ne bi bio van opsega za *int*.
- Kad god je podatak-članica klase objekat neke druge klase, a ne primitivnog tipa, neophodno je implementirati hashCode() metod za datu klasu.

Kreiranje kataloga

- HashMap<> objekat ima 4 konstruktora:
  - **HashMap()** – podrazumevani, kreira katalog podrazumevanog kapaciteta 16, a podrazumevani load faktor je 0.75.
  - **HashMap(int capacity)** – kreira katalog datog kapaciteta, sa podrazumevanim load faktorom.
  - **HashMap(int capacity, float loadFactor)** – kreira katalog sa datim kapacitetom i load faktorom.
  - četvrti konstruktor kreira katalog na osnovu postojećeg kataloga.

Kreiranje kataloga

- Kapacitet kataloga je broj parova ključ/objekat koji mogu da se čuvaju. Automatski se povećava, ako je neophodno.
- Poželjno je da se za kapacitet kataloga zadaje prost broj, kako bi se izbegla kolizija.
- Faktor punjenja (load faktor) se koristi da odlučimo kada da se poveća veličina heš tabele. Kada veličina tabele dostigne vrednost proizvoda faktora punjenja i kapaciteta, kapacitet se automatski povećava dva puta uz dodavanje 1 da bi se osiguralo da je novi kapacitet barem neparan, ako već nije prost.

Smeštanje, dobijanje i uklanjanje objekata kataloga

- **V put(K key, V value)** – smešta objekat *value* u katalog koristeći ključ *key*.
- **V remove(Object key)** – uklanja par povezan sa ključem ako postoji i vraća referencu na objekat. Ukoliko ne postoji odgovarajući par sa datim ključem, ili je objekat pridružen ključu *null*, vraća se *null*.

Smeštanje, dobijanje i uklanjanje objekata kataloga

- **V get(Object key)** – vraća objekat sa istim ključem kao *key*. Objekat ostaje u katalogu. Ako nema nijednog objekta sa datim ključem, ili je *null* smešteno umesto objekta, vraća se *null*.
- Ako **get()** vrati *null*, ne znamo da li objekat povezan sa ključem ne postoji ili je objekat *null*. Za ovo služi metod **containsKey()** koji kao argument ima dati ključ. On vraća *true* ako je ključ smešten u katalogu.

Obrada svih elemenata u katalogu

- Map<> interfejs obezbeđuje 3 načina za dobijanje kolekcionog pregleda sadržaja kataloga:
  - **keySet()** – vraća Set<K> objekat koji referiše na ključeve kataloga
  - **entrySet()** – vraća Set<Map.Entry<K,V>> objekat koji referiše na parove ključ/objekat – svaki par je objekat tipa Map.Entry<K,V>. Entry<K,V> je generički interfejsni tip definisan unutar interfejsa Map<>.
  - **values()** – vraća Collection<V> objekat koji referiše na objekte iz kataloga.

Primeri

- Neka je **map** objekat klase **HashMap<String,Integer>.**
  Set<String> keys = map.keySet();
  Sada se može dobiti iterator za dati skup ključeva:
  Iterator<String> keyIter = keys.iterator();

Ili:

```
Iterator<String> keyIter = map.keySet().iterator();
while(keyIter.hasNext())
        System.out.println(keyIter.next());
```

Primeri

```
Collection<Integer> collection = map.values();
for(Integer i : collection)
        System.out.println(i);
```

- Pošto interfejs Set<> nasleđuje interfejs Iterable<>, može se koristiti kolekcijska forma for petlje direktno sa objektima koje vraća metod keySet(), bez upotrebe iteratora.

```
Set<String> keys = map.keySet();
for(String key : keys)
        System.out.println(key);
```

Zadatak – Primer Telefonski imenik

- BrojTelefona.java
- Osoba.java
- ElementTelefonskogImenika.java
- TelefonskiImenik.java
- Test.java