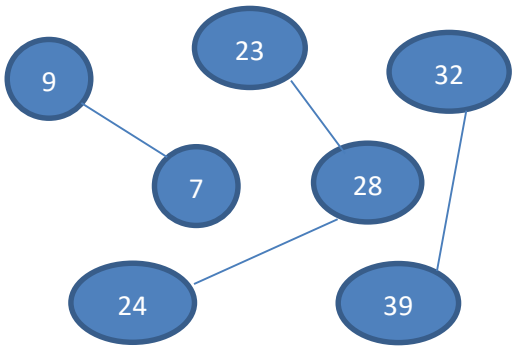
Stabla

Autori:
Goran Savić
Milan Segedinac

1. Strukture podataka bazirane na grafovima

Ovde ćemo predstaviti strukture podataka koje se baziraju na grafovima. Graf je pojam iz matematike i predstavlja skup objekata između kojih su definisane veze. Primer grafa je dat na slici.

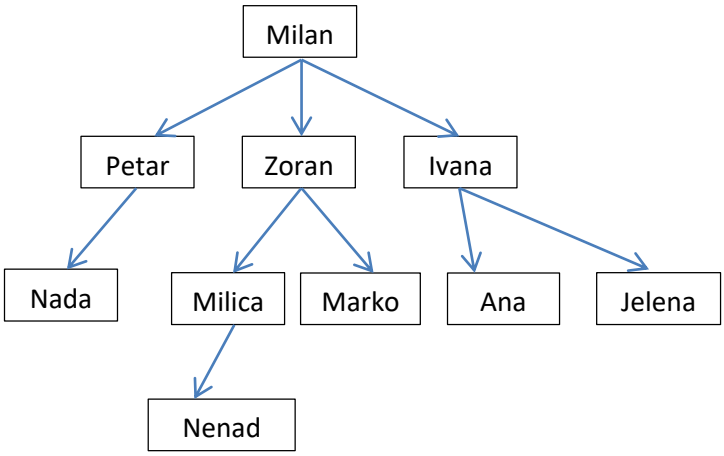


Graf se sastoji iz čvorova (objekti) i grana (veze između objekata). U grafu u prikazanom primeru, veze su uspostavljene između brojeva koji daju istu vrednost pri celobrojnom deljenju sa 10.

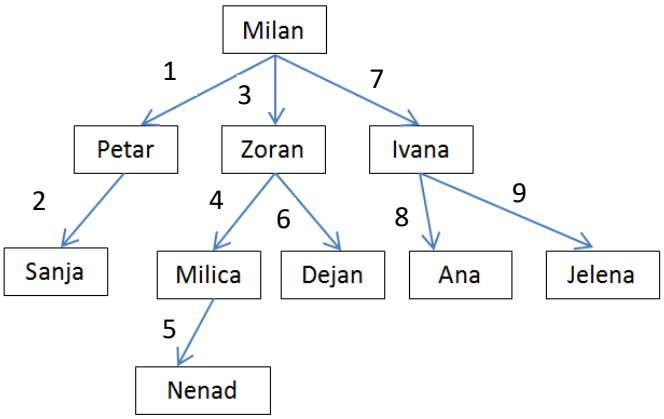
Od različitih struktura podataka baziranih na grafovima, ovde ćemo predstaviti stabla kao često korišćenu strukturu ovog tipa.

Stablo

Stablo je struktura podataka koja elemente skladišti hijerarhijski. Svaki čvor (osim korenskog čvora) sadrži roditeljski čvor i određen broj potomaka (može ih biti i nula). Time svaki čvor formira posebno podstablo. Čvorovi koji nemaju potomke nazivaju se listovi (eng. *leaf nodes*). Struktura tipa stablo je prikazana na sledećoj slici na primeru porodičnog stabla.



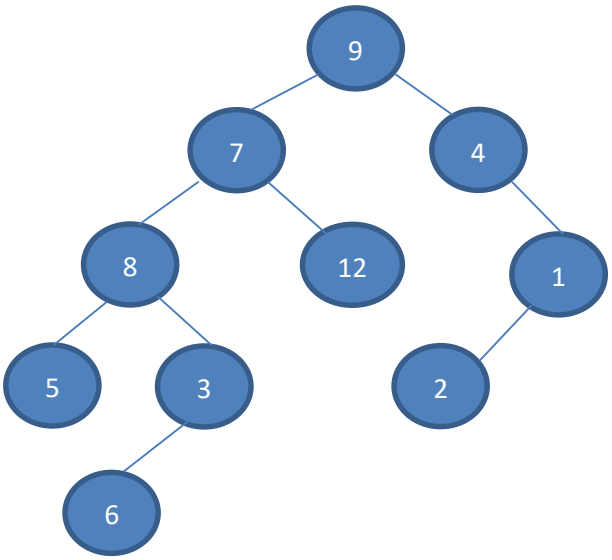
Pri dodavanju novog elementa u stablo, mora se definisati koji će biti roditeljski čvor novom elementu. Pronalaženje elementa u stablu podrazumeva prolazak kroz čvorove stabla. Algoritam koji se često koristi se naziva „prvi u dubinu“. Redosled prolaska kroz čvorove stabla prema ovom algoritmu je ilustrovan na slici.



Dakle, kada se pristupi jednom čvoru, pristupa se njegovom prvom potomku, za koji se dalje pristupa njegovom prvom potomku i tako redom dok se ne naiđe na čvor koji više nema potomak. Tada se algoritam vraća na roditeljski čvor tog poslednjeg čvora i po istom principu prolazi kroz sve njegove potomke. Postupak se ponavlja dok se ne prođu svi čvorovi stabla.

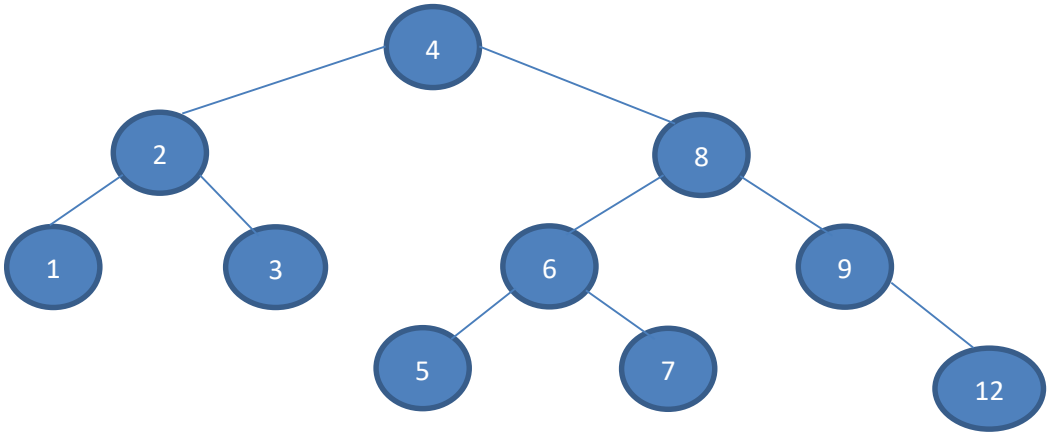
Binarno stablo

U praksi se često koristi podtip strukture stablo pod nazivom binarno stablo. Binarno stablo je stablo u kojem svaki čvor ima najviše dva potomka. Ovi potomci se najčešće nazivaju levi i desni potomak. Binarno stablo je ilustrovano na sledećoj slici.

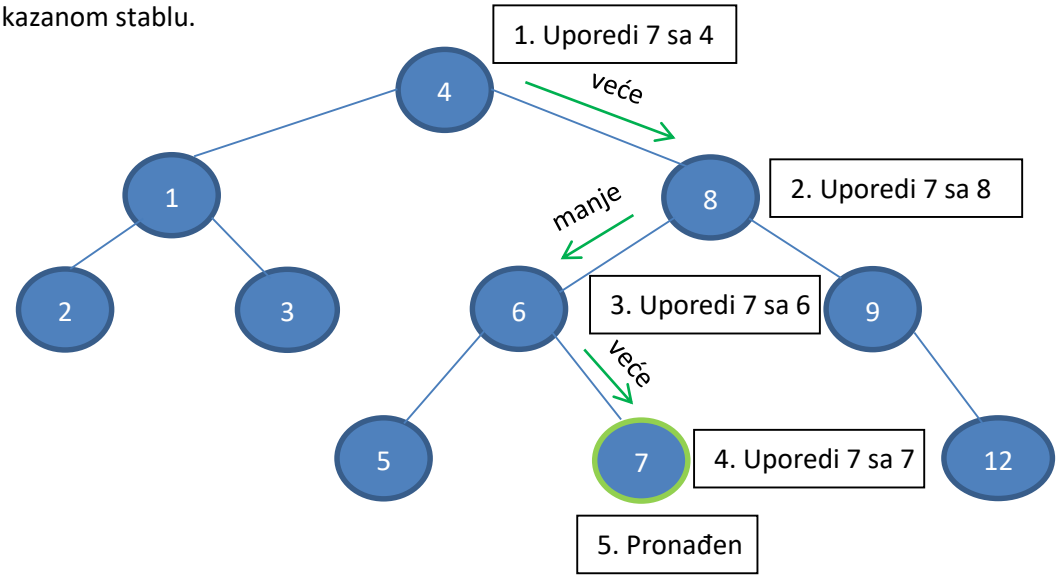


Binarno stablo pretrage

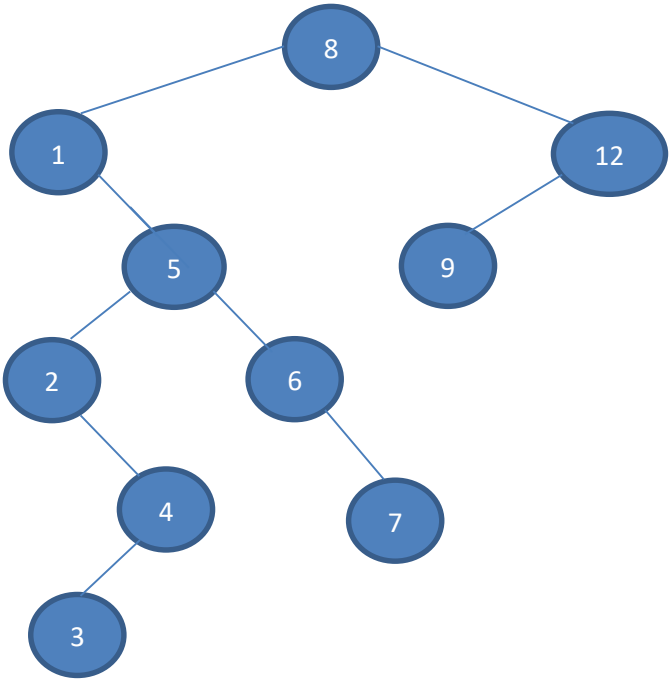
Binarna stabla se najčešće koriste u obliku svog specijalnog podtipa pod nazivom binarno stablo pretrage (eng. *binary search tree – BST).* Binarno stablo pretrage je binarno stablo u kojem su elementi sortirani. Preciznije, svaki element mora biti veći od svih elemenata u svom levom podstablu i manji u odnosu na sve elemente u svom desnom podstablu. Elementi iz gore prikazanog binarnog stabla prikazani su organizovani u binarno stablo pretrage na narednoj slici.



Binarno stablo pretrage omogućuje brzo pronalaženje elementa. Pronalaženje se zasniva na principima binarne pretrage. Na osnovu poređenja vrednosti određenog čvora i tražene vrednosti odlučuje se da li se prelazi u levo ili desno podstablo (u levom podstablu su manje vrednosti, a u desnom podstablu veće). Ovaj postupak se ponavlja za svaki čvor dok se ne pronađe traženi element. Za razliku od sekvencijalne strukture u kojoj je složenost pronalaženja elementa linearna, tj. $\Theta(n)$, kod binarnog stabla pretrage složenost pronalaženja elementa je logaritamsko, tj. $\Theta(logn)$. Naredna slika ilustruje postupak za pronalaženje elementa kroz traženje elementa broja 7 u ranije prikazanom stablu.



Stablo kakvo je prikazano na slici je izbalansirano binarno stablo pretrage. Kod ovakvog stabla, visina stabla je minimalno moguća. Visina stabla predstavlja broj grana na najdužoj putanji od korenskog čvora do nekog lista. Na primer, stablo sa prethodne slike ima visinu 3. Samo izbalansirano binarno stablo pretrage omogućuje dobre performanse. Stablo koje nije izbalansirano se u najgorem slučaju pretvara u sekvencu. Dat je primer neizbalansiranog binarnog stabla pretrage sa istim elementima kao u prethodnom primeru.



Kao što vidimo, ovo neizbalansirano binarno stablo pretrage ima visinu 5. Za pronalaženje elementa 7 bi bilo potrebno izvršiti 5 provera za razliku od 4 kod balansiranog stabla. Za pronalaženje elementa 3 kod ranije prikazanog balansiranog stabla treba izvršiti tri provere, a kod ovakvog neizbalansiranog stabla čak 6 provera.

Vidimo da izbalansiranost stabla presudno utiče na efikasnost pretrage elemenata u stablu. Obzirom na to koliko je važno da bude izbalansirano, binarno stablo pretrage se može implementirati i kao samobalansirajuće binarno stablo pretrage. Kod ovakve strukture, pri svakom dodavanju novog elementa se automatski vrši reorganizacija čvorova stabla kako bi stablo bilo izbalansirano.

Ovde ćemo još pomenuti i strukturu pod nazivom B-stablo, koje predstavlja generalizaciju binarnog stabla pretrage i omogućuje da čvor ima više od dva potomka. I kod B-stabla za svaki čvor važi pravilo da se u levom podstablu nalaze manji, a u desnom podstablu veći čvorovi od tog čvora.

Ranije smo pomenuli da se struktura tipa stablo može iskoristiti za realizaciju asocijativnog niza. Sada kada smo se upoznali sa stablima možemo objasniti na koji način se ovo vrši. Za realizaciju asocijativnog niza se koristi samobalansirajuće binarno stablo pretrage čiji su čvorovi parovi *{ključ, vrednost}*. Mesto u stablu određeno je vrednošću ključa. Dakle, par se pri dodavanju smešta u levo ili desno podstablo određenog čvora u zavisnosti od toga da li je njegov ključ manji ili veći od ključa para koji se nalazi u čvoru stabla. Kada su parovi organizovani u ovakvu strukturu, pronalaženje vrednosti na osnovu ključa svodi se na prolazak kroz binarno stablo pretrage putem poređenja traženog ključa sa ključevima u stablu. Čvor koji sadrži traženi ključ sadrži i vrednost koja odgovara tom ključu.

U realizaciji asocijativnog niza moguće je i kombinovati heš tabelu i binarno stablo pretrage. Za organizaciju parova u slotove se koristi heš tabela, a parovi koji se nalaze u istom slotu se uvezuju u binarno stablo pretrage. Podsetimo da su u ranije objašnjenoj implementaciji asocijativnog niza isključivo putem heš tabele, parovi bili uvezani u spregnutu listu. Binarno stablo pretrage obezbeđuje

bolje performanse pronalaženja elementa, ali u odnosu na spregnutu listu predstavlja komplikovaniju strukturu koja uvodi dodatne troškove održavanja. Zato se ovo rešenje primenjuje samo kada je broj elemenata u slotu veliki, pa je racionalno organizovati ih u komplikovaniju strukturu koja će biti brže pretraživa.

2. Implementacija struktura podataka u Javi

Java biblioteka klasa sadrži klase koje predstavljaju implementacije nekih od struktura podataka koje smo do sada pominjali. Ovaj skup klasa se u Java biblioteci naziva *collections framework* i nalazi se u paketu `java.util`. Pod kolekcijom se podrazumeva objekat koji reprezentuje grupu objekata. Ovakav tip klasa često nazivamo i kontejnerske klase obzirom da njihovi objekti skladište grupe drugih objekata. U nastavku su predstavljene neke od klasa iz Java *collections framework* grupe. Osim opisa njihovog API-ja, dati su i neki detalji implementacije. Podsećamo da je Java samo specifikacija, pa ćemo u opisu implementacije ovih klasa predstaviti kako su one implementirane u OpenJDK implementaciji Java specifikacije.

ArrayList

Prva složena struktura podataka koju smo opisivali bio je niz. Klasa ArrayList omogućuje standardne operacije nad nizom. Objekti ove klase sadrže kolekciju elemenata organizovanu u klasičan Java niz. Klasa je realizovana kao generička klasa tako da skladišti bilo koji tip elementa. Tip elemenata koji se skladište se definiše pri instanciranju klase. Dat je primer kreiranja ArrayList objekta koji skladišti objekte klase Student.

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

Klasa pruža određen broj standardnih operacija od kojih su najvažnije navedene u tabeli. Generički tip podatka koji ArrayList skladišti je u spisku označen sa E.

Naziv metode	Opis
`add(E e)`	Dodaje novi element na kraj liste
`add(int index, E e)`	Dodaje novi element na specificiranu poziciju u listi
`contains(Object o)`	Utvrđuje da li lista sadrži prosleđeni objekat
`get(int index)`	Vraća element na specificiranoj poziciji
`indexOf(Object o)`	Vraća poziciju na kojoj se prosleđeni element nalazi u listi
`remove(int index)`	Uklanja element na specificiranoj poziciji u listi
`remove(Object o)`	Uklanja specificirani objekat iz liste
`set(int index, E e)`	Zamenjuje element na specificiranoj poziciji u listi specificiranim elementom
`size()`	Vraća broj elemenata u listi
`toArray()`	Vraća Java niz koji sadrži elemente iz liste

Što se tiče implementacije ArrayListe, klasa elemente skladišti u nizu objekata klase Object. Pri kreiranju klase, zauzima se inicijalni niz određene veličine. Ako se inicijalna veličina ne specificira, zauzima se niz od deset elemenata. Pri dodavanju elementa može se desiti da niz nije dovoljno veliki da se upiše novi element. Na primer, želimo da upišemo jedanaesti element u inicijalno kreirani niz od deset elemenata. Tada je potrebno proširiti niz koji skladišti elemente. Vrši se kreiranje novog niza koji je veći od trenutnog i u koji se kopiraju elementi trenutnog niza. Novi niz će sadržati za

polovinu veći broj elemenata od trenutnog niza. Npr. ako je niz imao 10 elemenata, pri proširenju niza se kreira niz od 15 elemenata. Dakle, niz se uvek uvećava za faktor 1.5. Iako bi dodavanje novog elementa bilo moguće i kada bi se niz uvećao samo za jedan element, uvećavanje za faktor 1.5 obezbeđuje da se uvećavanje niza neće morati vršiti pri dodavanju svakog novog elementa. Ovo je veoma važno jer kreiranje novog niza i kopiranje trenutnih elemenata značajno utiče na performanse. Cena očuvanja performansi je trošenje nešto više memorijskog prostora nego što je neophodno. Za prošli primer sa proširenim nizom od 15 elemenata, poslednja 4 elementa će biti neiskorišćena (bar do dodavanja novih elemenata).

Videli smo da metoda add(int index, E e) radi umetanje elementa u niz. Ova operacija kopira sve elemente niza za jednu poziciju unapred, počevši od specificiranog indeksa. Treba imati u vidu da ovo može da bude zahtevno sa stanovišta performansi. Slično, metode remove() uklanjaju element iz niza tako što sve elemente koji se nalaze nakon tog elementa moraju da pomere za jednu poziciju unazad.

Kada je reč o Java strukturama koje enkapsuliraju niz, treba spomenuti i klasu Vector. Ova klasa radi vrlo slično klasi ArrayList. Od nekoliko karakteristika po kojima se klasa Vector razlikuje od klase ArrayList ovde ćemo pomenuti samo to da je kod klase Vector moguće definisati za koliko elemenata će niz biti proširen kada trenutni kapacitet niza više nije dovoljan za smeštanje svih elemenata. Ako se ovaj podatak ne specificira, klasa Vector će duplirati trenutni broj elemenata niza.

LinkedList

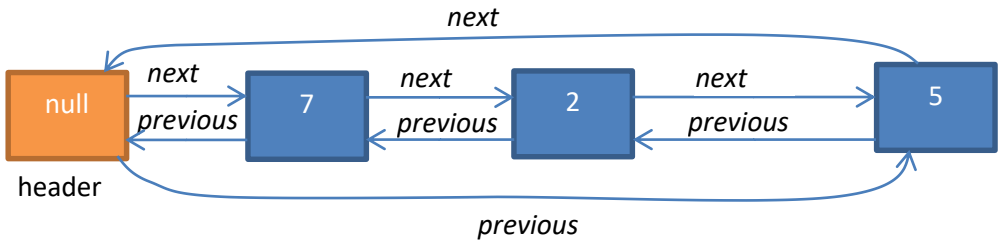
Java biblioteka klasa pruža i implementaciju spregnute liste. Konkretno, reč je o dvostruko spregnutoj listi predstavljenoj klasom LinkedList. I ova klasa je takođe generička i omogućuje skladištenje elemenata bilo kog tipa. U tabeli su prikazane glavne metode klase.

Naziv metode	Opis
`add(E e)`	Dodaje novi element na kraj liste
`add(int index, E e)`	Dodaje novi element na specificiranu poziciju u listi
`contains(Object o)`	Utvrđuje da li lista sadrži prosleđeni objekat
`get(int index)`	Vraća element na specificiranoj poziciji
`getFirst()`	Vraća prvi element liste
`getLast()`	Vraća poslednji element liste
`indexOf(Object o)`	Vraća poziciju na kojoj se prosleđeni element nalazi u listi
`remove(int index)`	Uklanja element na specificiranoj poziciji u listi
`remove(Object o)`	Uklanja specificirani objekat iz liste
`set(int index, E e)`	Zamenjuje element na specificiranoj poziciji u listi specificiranim elementom
`size()`	Vraća broj elemenata u listi
`toArray()`	Vraća Java niz koji sadrži elemente iz liste

Kao što vidimo, većina metoda su iste kao i kod ArrayListe. Ovo je i razumljivo, obzirom da je u oba slučaja reč o sekvencijalnoj strukturi. Razlike su u implementaciji skladištenja elemenata.

Obzirom da je reč o dvostruko spregnutoj listi, klasa LinkedList skladišti svaki objekat u elementima liste koji osim vrednosti samog objekta sadrže i referencu na prethodni, kao i na naredni element u listi. Element liste sa pomenutim podacima predstavljen je unutrašnjom klasom *Entry*. Pored

elemenata koji sadrže vrednosti u listi, klasa sadrži i jedan dodatni element pod nazivom *header*. Uloga ovog čvora je da čuva reference na prvi i na poslednji element u listi (ne računajući *header* element). Razlog za postojanje ovog elementa je jednostavnija implementacija, obzirom da lista tada nikad nije potpuno prazna. To pojednostavljuje operacije u listi obzirom da je prazna lista specijalni slučaj koji bi se morao posebno u kodu obrađivati. Postojanje *header* elementa je stvar interne implementacije liste i korisnik klase nije svestan njegovog postojanja. U svim operacijama koje API klase nudi figurišu samo elementi liste koji skladište objekte, bez *header* elementa. Na slici je prikazana struktura liste predstavljene LinkedList klasom.



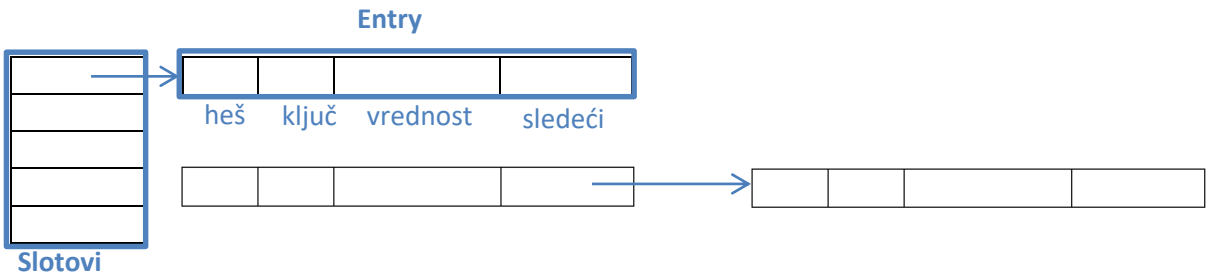
Prikazana lista skladišti celobrojne vrednosti 7, 2 i 5. *Header* element označen je drugom bojom.

HashMap

Klasa HashMap predstavlja Java implementaciju heš mape. Heš mapa je realizovana tako da skladišti parove {ključ, vrednost} predstavljene unutrašnjom klasom Entry. Ovo je generička klasa koja omogućuje skladištenje ključeva i vrednosti proizvoljnog tipa. Pri instanciranju objekta klase HashMap navodi se kojeg su tipa ključevi i vrednosti u toj konketnoj instanci HashMap klase. Dat je primer instanciranja HashMap klase u kojoj su ključevi tipa String, a vrednosti tipa Student.

```
HashMap<String, Student> studenti = new HashMap<>();
```

Parovi se skladište u slotovima (u implementaciji klase se slot naziva *bucket*) koji formiraju niz slotova. U kojem slotu će se par nalaziti zavisi od vrednosti ključa. Preciznije, na osnovu vrednosti ključa izračunava se njegova heš vrednost korišćenjem heš funkcije. Kao heš funkcija koristi se metoda hashCode() koja vraća celi broj koji predstavlja heš vrednost. Svaka klasa sadrži ovu metodu obzirom da je nasleđuje iz klase Object. Na osnovu izračunate heš vrednosti se određuje indeks slota u nizu slotova u koji će par biti smešten. Ako heš funkcija različite ključeve pretvara u istu heš vrednost, tada će se ti objekti nalaziti u istom slotu. Klasa HashMap objekte koji se nalaze u istom slotu organizuje u jednostruko spregnutu listu. Skladištenje podataka korišćenjem klase HashMap je prikazano na slici.



Treba primetiti da objekat klase Entry pored ključa, vrednosti i reference na sledeći element u spregnutoj listi, skladišti i heš vrednost izračunatu na osnovu ključa. Razlog je optimizacija pri

pronalaženju elementa. Naime, pošto u spregnutoj listi može biti više elemenata (jer se različiti ključevi mapiraju na isti slot), potrebno je linearnim prolaskom kroz listu pronaći element sa traženim ključem. Pri pronalaženju, efikasnije je uporediti heš vrednosti traženog i uskladištenog ključa (koji su celobrojne vrednosti) nego vršiti poređenje ključeva. Ako se heš vrednosti razlikuju, onda to sigurno nije traženi element i pretraga se nastavlja.

Ranije smo pominjali da samo dobro izabrana heš funkcija može da obezbedi da se što manje vrednosti mapira na istu heš vrednost. Obzirom da heš vrednost ukazuje na slot u koji će objekat biti smešten, važno je da distribucija heš vrednosti bude takva da nemamo previše parova u istom slotu, tj. da spregnute liste nisu prevelike. Idealno je da svaki slot sadrži samo jedan par, jer tada pri pronalaženju elementa ne treba linearno prolaziti kroz spregnutu listu. Iz tog razloga, kao zaštita od loše implementirane hashCode() funkcije koja ne pravi dobru distribuciju heš vrednosti, klasa HashMap implementira i svoju funkciju pod nazivom hash() za izračunavanje heš vrednosti. Heš vrednost dobijena pozivom funkcije hashCode() nad ključem se dalje prosleđuje funkciji hash() koja daje konačnu heš vrednost i na osnovu koje se pronalazi odgovarajući slot u kojem je par smešten.

Jasno je da heš mapa može da obezbedi dobre performanse dodavanja i preuzimanja elemenata samo ako je dobra distribucija heš vrednosti i ako spregnute liste u slotovima imaju malo elemenata. Da bi ovo obezbedila, klasa HashMap po potrebi vrši reorganizaciju strukture. Reorganizacija podrazumeva povećanje veličine niza slotova kako bi što više parova bilo smešteno u različite slotove i time se smanjile dužine spregnutih lista. Ovo se vrši po potrebi pri dodavanju novog elementa u mapu. Veličina niza slotova je uvek stepen broja 2. Kada se ovaj niz povećava, vrši se dupliranje veličine.

U tabeli je prikazan API klase HashMap.

Naziv metode	Opis
`put(K key, V value)`	Ubacuje specificirani par {ključ, vrednost} u mapu
`get(K key)`	Preuzima vrednost koja odgovara specificiranom ključu
`remove(K key)`	Izbacuje par koji odgovara specificiranom ključu
`keySet()`	Vraća sve ključeve uskladištene u mapi
`values()`	Vraća sve vrednosti uskladištene u mapi
`entrySet()`	Vraća sve parove uskladištene u mapi u formi objekata klase Entry

Stack

Ranije smo spominjali da stek pripada strukturama podataka sa ograničenim pristupom. Praktično, stek je specijalan oblik neke od ranije prikazanih struktura i može se realizovati korišćenjem neke od ovih struktura. Java biblioteka klasa sadrži klasu Stack koja predstavlja stek strukturu podataka. Stack klasa se pri implementaciji oslanja na klasu Vector. Preciznije, klasa Stack je specijalizacija implementacije klasu Vector. Pri implementaciji operacija steka, samo se pozivaju odgovarajuće metode klase Vector. Dakle, stek je u Javi predstavljen kao niz elemenata, jer klasa Vector enkapsulira niz, isto kao i klasa ArrayList. Nad ovim nizom moguće je izvršiti standardne stek operacije koje su interno realizovane pozivom metoda klase Vector. U tabeli je predstavljen API klase Stack.

Naziv metode	Opis
`push(E item)`	Dodaje novi element na vrh steka. Poziva metodu addElement(e) klase Vector koja dodaje element na kraj niza.

`pop()`	Uklanja element sa vrha steka. Poziva metodu removeElementAt() klase Vector. Ova metoda kao parametar prima indeks elementa koji se uklanja. Prosleđuje se indeks poslednjeg elementa u nizu.
`peek()`	Preuzima element sa vrha steka bez uklanjanja. Koristi metodu elementAt() klase Vector. Ova metoda kao parametar prima indeks elementa koji se preuzima. Prosleđuje se indeks poslednjeg elementa u nizu.
`search(Object o)`	Vraća informaciju koliko je traženi objekat udaljen od vrha steka ili vrednost -1 ako se objekat ni ne nalazi na steku. Za element na vrhu vraća udaljenost 1. Poziva metodu lastIndexOf() klase Vector koja vraća poziciju na kojoj se neki element nalazi u nizu.

ArrayDequeue

Java biblioteka klasa sadrži i klase koje predstavljaju strukturu tipa red. Jednu takvu klasu smo već videli. Klasa LinkedList omogućuje pristup i dodavanje na početak i kraj liste, pa se može posmatrati i kao red sa dva kraja (eng. *double ended queue*). Osim putem klase LinkedList, red sa dva kraja implementiran je i putem klase ArrayDeque. Ova klasa koristi klasičan Java niz za skladištenje elemenata. U tabeli je prikazan API ove klase.

Naziv metode	Opis
`addFirst(E e)`	Dodaje novi element na početak reda.
`addLast(E e)`	Dodaje novi element na kraj reda. Isto radi i metoda add(E e)
`getFirst()`	Vraća element koji se nalazi na početku reda.
`getLast()`	Vraća element koji se nalazi na kraju reda.
`removeFirst()`	Uklanja element sa početka reda.
`removeLast()`	Uklanja element sa kraja reda.
`poll()`	Vraća i uklanja element sa početka reda

Treba primetiti da za razliku od klasa ArrayList i LinkedList, klasa ArrayDeque nema podršku za pristup proizvoljnom elementu u sekvenci obzirom da predstavlja red sa dva kraja u kojem se pristupa elementu sa početka ili kraja reda.

HashSet

Kada je reč o strukturi tipa skup, Java biblioteka klasa sadrži različite klase koje predstavljaju ovu strukturu. Ovde ćemo predstaviti klasu pod nazivom HashSet. Ova klasa predstavlja skup koristeći ranije prikazanu klasu HashMap. Klasa HashSet sadrži atribut tipa HashMap koji koristi za skladištenje elemenata. Obzirom da je glavna osobina skupa da elementi moraju biti jedinstveni (nema duplikata), korišćenjem heš mape se efikasno utvrđuje da li se element već nalazi u skupu. U tabeli je prikazan API ove klase.

Naziv metode	Opis
`add(E e)`	Dodaje novi element u skup, ako se već ne nalazi u skupu.
`contains(Object o)`	Vraća informaciju da li se specificirani objekat nalazi u skupu.
`remove(Object o)`	Uklanja specificirani objekat iz skupa ukoliko se nalazi u skupu.