2. labos, tema 1

Zadatak 1.

Napišite implementaciju razreda `SimpleHashtable`. Razred predstavlja tablicu raspršenog adresiranja koja omogućava pohranu uređenih parova (ključ, vrijednost). Postoje dva javna konstruktora: podrazumjevani (default) koji stvara tablicu veličine 16 slotova, te konstruktor koji prima jedan argument: broj koji predstavlja željeni početni kapacitet tablice i koji stvara tablicu veličine koja je potencija broja 2 koja je prva veća ili jednaka predanom broju (npr. ako zada 30, bira se 32).

Jedan slot tablice modelirajte ugniježđenim razredom `TableEntry`. Primjerci ovog razreda imaju člansku varijablu `key` u kojoj pamte predani ključ, člansku varijablu `value` u kojoj pamte pridruženu vrijednost te člansku varijablu `next` koja pokazuje na sljedeći primjerak razreda `TableEntry` koji se nalazi u istom slotu tablice (izgradnjom ovakve liste riješavat ćete problem preljeva – situacije kada u isti slot treba upisati više uređenih parova). I ključ i vrijednost mogu biti bilo kakvi objekti – ne moraju nužno biti stringovi.

Ideju uporabe ovakve kolekcije ilustrira sljedeći kod.

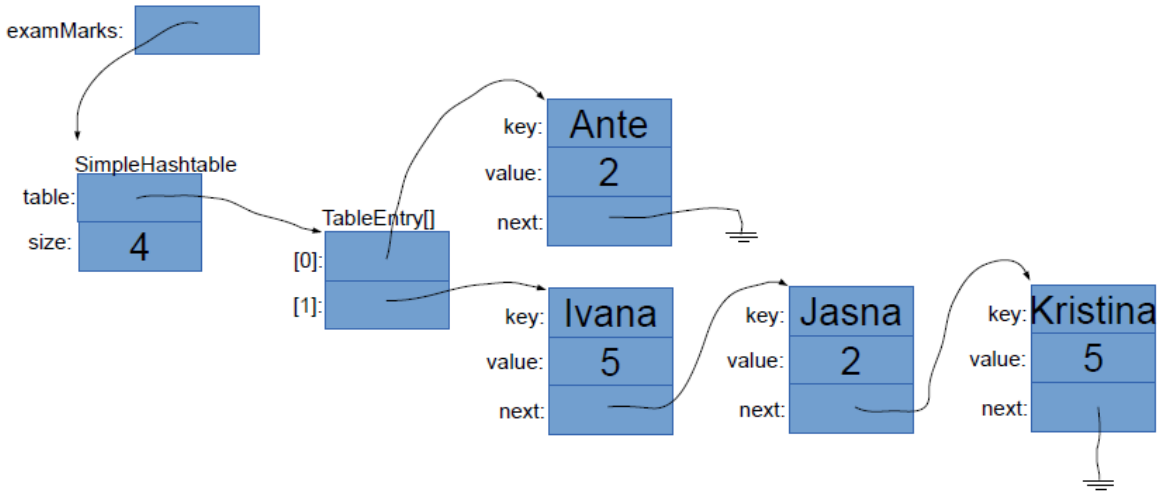
```
// create collection:
SimpleHashtable examMarks = new SimpleHashtable(2);

// fill data:
examMarks.put("Ivana", Integer.valueOf(2));
examMarks.put("Ante", Integer.valueOf(2));
examMarks.put("Jasna", Integer.valueOf(2));
examMarks.put("Kristina", Integer.valueOf(5));
examMarks.put("Ivana", Integer.valueOf(5)); // overwrites old grade for Ivana

// query collection:
Integer kristinaGrade =  (Integer)examMarks.get("Kristina");
System.out.println("Kristina's exam grade is: " + kristinaGrade); // writes: 5

// What is collection's size? Must be four!
System.out.println("Number of stored pairs: " + examMarks.size()); // writes: 4
```

Za potrebe izračuna slota u koji treba ubaciti uređeni par koristite metodu `hashCode()` ključa, pa modulo veličina tablice. Ključ uređenog para ne smije biti `null` dok vrijednost može biti `null`.

Razred `SimpleHashtable` treba imati sljedeće članske varijable:

- `TableEntry[] table`: polje slotova tablice,
- `int size`: broj parova koji su pohranjeni u tablici.

Pojednostavljeni prikaz stanja u memoriji nakon izvođenja koda iz prethodnog primjera ilustriran je na sljedećoj slici.

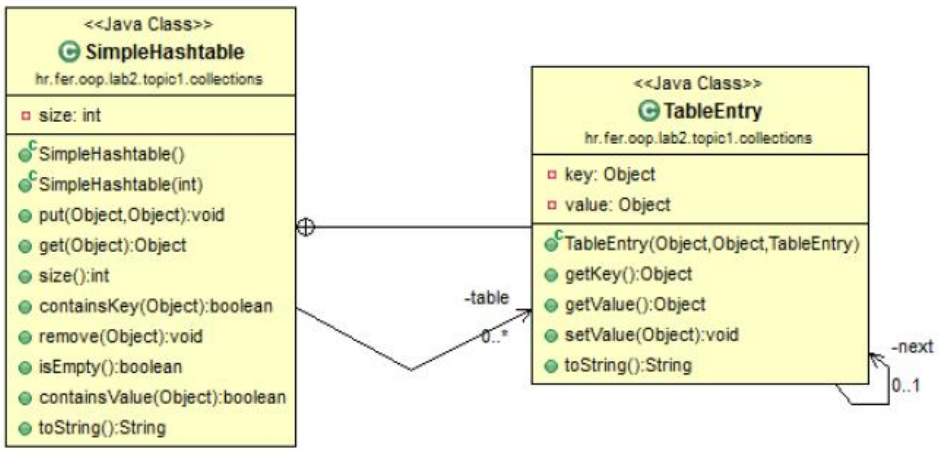


Pri tome na platformi Java 8 vrijedi:

Objekt	**hashCode()**	**|hashCode()|**	**slot=|hashCode()| % 2**
"Ivana"	71029095	71029095	1
"Ante"	2045822	2045822	0
"Jasna"	71344303	71344303	1
"Kristina"	-1221180583	1221180583	1

Stoga će ključevi Ivana, Jasna i Kristina biti u slotu 1 a ključ Ante u slotu 0. Razmislite odgovara li prikazana slika stvarnom stanju u memoriji ili bismo za stvarno stanje dio slike trebali drugačije nacrtati?

Dijagram razreda koji prikazuje razrede ovog zadatka prikazan je u nastavku.



Metode i konstruktori koje razred `SimpleHashtable` mora ponuditi navedeni su u nastavku bez posebne dokumentacije (iz imena bi moralo biti jasno što se od metode očekuje).

```
public SimpleHashtable();
public SimpleHashtable(int capacity);
public void put(Object key, Object value);
public Object get(Object key);
public int size();
public boolean containsKey(Object key);
public boolean containsValue(Object value);
public void remove(Object key);
```

```
public boolean isEmpty();
public String toString();
```

Metoda `put` pozvana s ključem koji u tablici već postoji ažurira postojeći par novom vrijednošću; metoda ne dodaje još jedan par s istim ključem ali drugom vrijednosti.

Metoda `get` pozvana s ključem koji ne tablici ne postoji vraća `null`.

Što možete zaključiti o složenosti metode `containsKey` a što o složenosti metode containsValue u ovako implementiranoj kolekciji (uz pretpostavku da je broj parova dodanih u tablicu dosta manji od broja slotova tablice te da funkcija sažetka radi dobro raspršenje)?

Zadatak 2. (nova verzija – ako prvi puta radite vježbu, radite ovu verziju)

Napišite klase koje modeliraju jednostavan cjelobrojni kalkulator. Kalkulator može unositi dekadske znamenke, ali ne i decimalnu točku. Ima dvije operacije: **zbrajanje** i **oduzimanje** te omogućava: **brisanje** i **dohvat rezultat**. Kalkulator ima tri registra: **display** (broj na zaslonu), **memory** (memorija prethodnog broja) i **operator** (operacija). Dva kalkulatora su ista ako imaju iste sadržaje registara.

Ispis objekta se svodi na ispis registara, u obliku "`(D=*, M=*, O=*)`", gdje se umjesto zvjezdica nalaze brojevi, odnosno simbol operacije (npr. "`(D=12, M=15, O=+)`"). Ako je neki registar prazan, ne mora se ispisivati.

Prije početka rada na programu kalkulatora, potrebno je **nacrtati UML dijagram** predloženog rješenja.On mora imati ove klase/sučelja/enum-ove:

- `ICalculator` (sučelje koje trebaju implementirati svi kalkulatori)
- `SimpleCalc` (konkretna klasa kalkulatora koja je nastala implementacijom sučelja ICalculator i koja sadrži registre)
- `Register` (konkretna klasa kojom se modelira registar)
- Klasu/Klase kojima se modeliraju gumbi ili različite vrste gumba
- Enumove koji službe za pobrojanje ponavljajućih elemenata u kodu koje je moguće kategorizirati

Sučelje `ICalculator` definira metode:

- `String getDisplay()`: vrati sadržaj zaslona kalkulatora.
- `void press ( /*parametri kojima se identificira gumb */ )`: pritisnut je gumb

Konkretna klasa `SimpleCalc` implementira sučelje `Calculator` i koristi klasu `Register` za memorije. Kalkulator poznaje samo operacije "+" i "-" te komande "=" i "C". Ako se pritisne = prije nego što je zadana operacija, metoda `press`() uzrokuje završetak rada programa.

Unutar klase `SimpleCalc` potrebno je minimalno implementirati sljedeće metode:

- `public String getDisplay()` – vraća sadržaj zaslona kalkulatora
- `public void pressDigit( int )` – dojavljuje kalkulatoru da je pritisnuta jedna od brojevnih tipki (dekadske znamenke od "**0**" do "**9**")
- `public void pressEquals()` – pritisnut je "=" (rezultat). Treba obaviti operaciju spremljenu u O nad brojevima u D i M. Rezultat treba staviti u D, a M poništiti.
- `public void pressPlus()` – pritisnut je "+" (zbrajanje). D -> M, "+" -> O, poništi D.
- `public void pressMinus()` – pritisnut je "-" (oduzimanje)
- `public void pressClear()` – pritisnut je "**C**" (brisanje cijele memorije)

Naredba	**vrijednosti nekih metoda nakon naredbe**	
`SimpleCalc c = new SimpleCalc();`	`c.getDisplay()` `c.toString()`	`"0"` `"()"`
`c.pressDigit( 1 );`	`c.getDisplay()`	`"1"`

	`c.toString()`	`"(P=1)"`
`c.pressDigit( 0 );`	`c.getDisplay()` `c.toString()`	`"10"` `"(P=10)"`
`c.pressMinus();`	`c.getDisplay()` `c.toString()`	`"0"` `"(S=10, O=-)"`
`c.pressDigit( 2 );`	`c.getDisplay()` `c.toString()`	`"2"` `"(P=2, S=10, O=-)"`
`c.pressDigit( 9 );`	`c.getDisplay()` `c.toString()`	`"29"` `"(P=29, S=10, O=-)"`
`c.pressPlus;`	`c.getDisplay()` `c.toString()`	`"0"` `"(S=-19, O=+)"`
`c.pressDigit( 3 );`	`c.getDisplay()` `c.toString()`	`"3"` `"(P=3, S=-19, O=+)"`
`c.pressEquals();`	`c.getDisplay()` `c.toString()`	`"-16"` `"(P=-16)"`
`c.pressClear();`	`c.getDisplay()` `c.toString()`	`"0"` `"()"`

Klasa `Register` predstavlja registar koji pamti jednu vrijednost. Može sadržavati neku vrijednost ili biti prazan. Vrijednost je jedan općeniti `Object`. Metode (minimalno):

- `public /* odabrati tip povratne vrijednosti */ getValue()`: vrati pohranjenu vrijednost ili `null` ako je prazan.
- `public void setValue( /* potebni parametri */ )`: posprema vrijednost.
- `public void clear()`: isprazni se

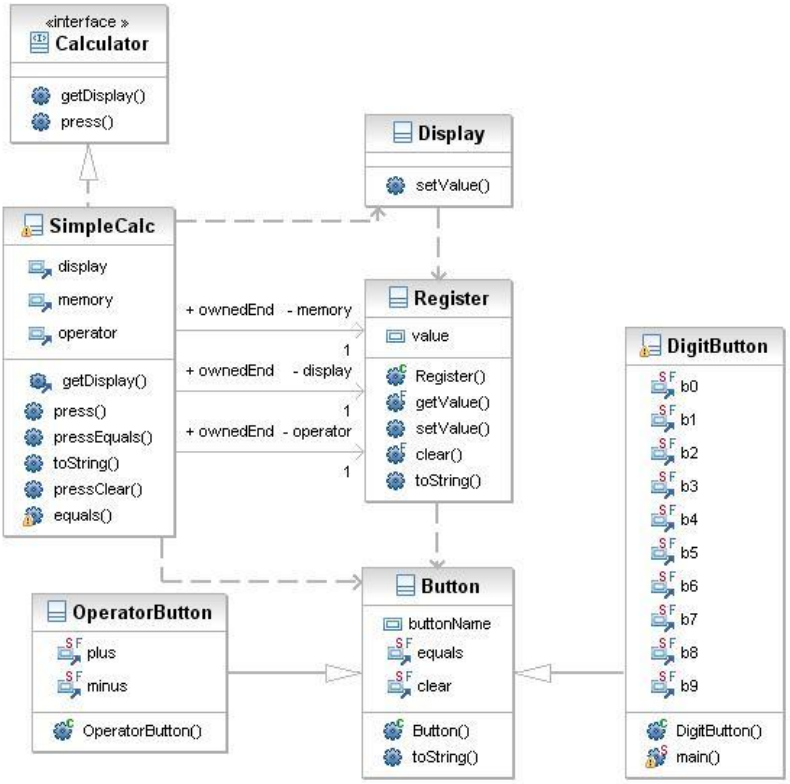
Zadatak 2. (stara verzija – ako ste već radili ovaj zadatak, slobodno nastavite po ovoj verziji)

Prije početka rješavanja potrebno je samostalno proučiti temu enuma-a u Javi. Potrebno je koristiti samostalno identificirane web-resurse ili samostalno odabranu literaturu.

Napišite klase koje modeliraju jednostavan cjelobrojni kalkulator. Kalkulator može unositi dekadske znamenke, ali ne i decimalnu točku. Ima dvije operacije: **zbrajanje** i **oduzimanje** te dvije posebne tipke: **brisanje** i **rezultat**. Kalkulator ima tri registra: **display** (broj na zaslonu), **memory** (memorija prethodnog broja) i **operator** (operacija). Dva kalkulatora su ista ako imaju iste sadržaje registara.

Ispis objekta se svodi na ispis registara, u obliku "`(D=*, M=*, O=*)`", gdje se umjesto zvjezdica nalaze brojevi, odnosno simbol operacije (npr. "`(D=12, M=15, O=+)`"). Ako je neki registar prazan, ne mora se ispisivati.

Dijagram klasa kalkulatora dan je sljedećom slikom.



Sučelje `Calculator` definira metode:

- `String getDisplay()`: vrati sadržaj zaslona kalkulatora.
- `void press ( Button button )`: pritisnuta je tipka zadana objektom klase `Button`.

Konkretna klasa `SimpleCalc` implementira sučelje `Calculator` i koristiklasu `Register` za memorije. Kalkulator poznaje samo operacije "+" i "-" te komande "=" i "C". Ako se pritisne = prije nego što je zadana operacija, metoda `press`() uzrokuje završetak rada programa.

Unutar klase `SimpleCalc` potrebno je implementirati sljedeće metode:

- `public String getDisplay()` – vraća sadržaj zaslona kalkulatora

- `public void pressDigit( int )` – dojavljuje kalkulatoru da je pritisnuta jedna od brojevnih tipki (dekadske znamenke od "**0**" do "**9**")
- `public void pressEquals()` – pritisnut je "**=**" (rezultat). Treba obaviti operaciju spremljenu u O nad brojevima u D i M. Rezultat treba staviti u D, a M poništiti.
- `public void pressPlus()` – pritisnut je "**+**" (zbrajanje). D -> M, "+" -> O, poništi D.
- `public void pressMinus()` – pritisnut je "**-**" (oduzimanje)
- `public void pressClear()` – pritisnut je "**C**" (brisanje cijele memorije)

Naredba	**vrijednosti nekih metoda nakon naredbe**	
`SimpleCalc c = new SimpleCalc();`	`c.getDisplay()` `c.toString()`	`"0"` `"()"`
`c.pressDigit( 1 );`	`c.getDisplay()` `c.toString()`	`"1"` `"(P=1)"`
`c.pressDigit( 0 );`	`c.getDisplay()` `c.toString()`	`"10"` `"(P=10)"`
`c.pressMinus();`	`c.getDisplay()` `c.toString()`	`"0"` `"(S=10, O=-)"`
`c.pressDigit( 2 );`	`c.getDisplay()` `c.toString()`	`"2"` `"(P=2, S=10, O=-)"`
`c.pressDigit( 9 );`	`c.getDisplay()` `c.toString()`	`"29"` `"(P=29, S=10, O=-)"`
`c.pressPlus;`	`c.getDisplay()` `c.toString()`	`"0"` `"(S=-19, O=+)"`
`c.pressDigit( 3 );`	`c.getDisplay()` `c.toString()`	`"3"` `"(P=3, S=-19, O=+)"`
`c.pressEquals();`	`c.getDisplay()` `c.toString()`	`"-16"` `"(P=-16)"`
`c.pressClear();`	`c.getDisplay()` `c.toString()`	`"0"` `"()"`

Klasa `Register` predstavlja registar koji pamti jednu vrijednost. Može sadržavati neku vrijednost ili biti prazan. Vrijednost je jedan općeniti `Object`. Metode:

- `public final Object getValue()`: vrati pohranjenu vrijednost ili `null` ako si prazan.
- `public void setValue( Object o )`: pospremi vrijednost. Potklase ju nadjačavaju radi dodatne provjere da je objekt prave vrste (*instanceof* nešto).
- `public final void clear()`: isprazni se

Klasa `Button` je *typesafe enum*. Sadrži statička svojstva:

- `public static final Button equals = new Operator( "=" );`
- `public static final Button clear = new Operator( "C" );`

Implementira metodu `toString`() tako da vraća string koji je dan u konstruktoru. Taj string se, naravno, mora pamtiti u svojstvu objekta, recimo `private String buttonName`.

Klasa `DigitButton` je *typesafe enum*. Izvedena iz Button. Sadrži statička svojstva:

- `public static final DigitButton b0 = new DigitButton( 0 );`
- itd. za b1 .. b9.

Njen konstruktor poziva konstruktor natklase sa stringom dobivenim pomoću standardne Javine metode Integer.toString(int).

Klasa `OperatorButton` je *typesafe enum*. Izvedena iz `Button`. Sadrži statička svojstva:

- `public static final OperatorButton plus = new OperatorButton( "+" );`
- `public static final OperatorButton minus = new OperatorButton( "-" );`