Objektno orijentirano programiranje

Ljetni ispitni rok

7.7.2020.

Ispit nosi ukupno 50 bodova i piše se 150 minuta.
U zadacima nije potrebno pisati dio u kojem se uključuju klase ili paketi klasa (import)
Rješenja je potrebno pisati na stranice ispita. *Ako na nekoj stranici ne budete imali dovoljno mjesta, ostatak možete napisati na zadnju (praznu) stranicu ispita.*

IZJAVA

Tijekom ove provjere znanja neću od drugoga primiti niti drugome pružiti pomoć te se neću koristiti nedopuštenim sredstvima.
Ove su radnje povreda Kodeksa ponašanja te mogu uzrokovati trajno isključenje s Fakulteta.
Zdravstveno stanje dozvoljava mi pisanje ovog ispita.

Vlastoručni potpis studenta: _____

1. zadatak (10 bodova)

Potrebno je modelirati sustav male društvene mreže ljubitelja životinja koji je opisan nastavku teksta. **Vaš zadatak je nadopuniti predložak UML dijagrama klasa ovog sustava koji se nalazi na sljedećoj stranici**. Pritom **možete dodati nove klase** (ili sučelja) ako smatrate da je to potrebno**. Nije potrebno pisati programski kod sustava i to neće donositi nikakve bodove**. Na ovom predlošku pažljivo navedite:

- oznake za sučelje (**I**), enumeraciju (**E**), apstraktnu (**A**) ili običnu klasu (**C**)
- oznake za apstraktnu metodu (**A**) te oznake za statičku (**S**) i finalnu (**F**) metodu ili atribut
- modifikatore vidljivosti metoda i atributa (**+** public, **#** protected, **-** private, **~** package-private)
- tipove povratnih vrijednosti i argumenata za metode te tipove atributa
- odgovarajuće strelice da naznačite nasljeđivanje klasa (puna linija ⟶▷) ili implementaciju sučelja (iscrtkana linija ⇢▷)

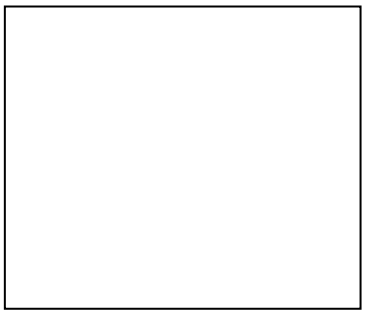
Na društvenoj mreži imamo korisnike tipa User. Korisnik ima svoje ime `name` tipa `String`, datum registriranja `signupDate` tipa `Date`, popis praćenih korisnika `following` tipa `List<User>` i popis svojih objava `posts` tipa `List<Post>`. Status korisnika `status` može biti u jednom od 3 stanja tipa `UserStatus`: neaktiviran (`NOT_ACTIVATED`), aktiviran (`ACTIVATED`) i deaktiviran (`DEACTIVATED`). Klasa `User`, uz *gettere* za ime (`getName`), datum registriranja (`getDate`), popis praćenih korisnika (`getFollowing`) i popis svojih objava (`getPosts`), također ima i metodu za dodavanje nove osobe na popis praćenih (`addFollowing`) te metodu za dodavanje nove objave (`addPost`).

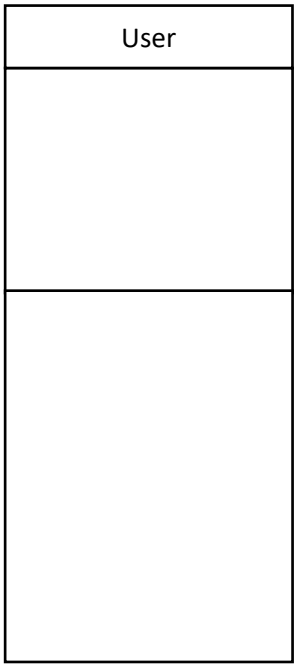
Korisnici su pretraživi. Objekt je pretraživ ako implementira metodu `boolean searchHash(int hashCode)`. Ova metoda uspoređuje predani `hashCode` s vrijednošću koju vraća metoda `hashCode()` tog objekta te kao rezultat usporedbe vraća `true` ako su iste, a inače `false`.

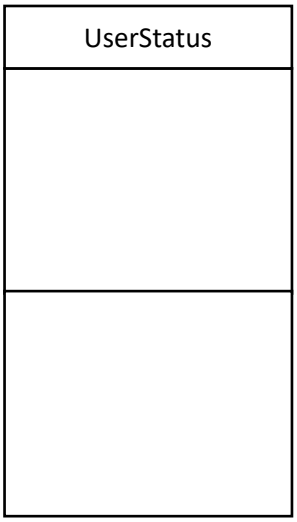
Objava tipa `Post` sadrži vrijeme zadnje modifikacije `lastModified` tipa `Date`, korisnika `user` koji je napravio objavu tipa `User` te status objave `status` koje može biti u jednom od 3 stanja tipa `PostStatus`: skica (`DRAFT`), skriveno (`HIDDEN`) i objavljeno (`PUBLISHED`). Osim toga, objava treba imati *gettere* za autora (`getAuthor`), status (`getStatus`) i vrijeme zadnje modifikacije (`getLastModified`). Zatim, *settera* za status (`setStatus`) te `void` metodu `displayPost` bez argumenata koja prikazuje objavu. **Objave su također pretražive.**

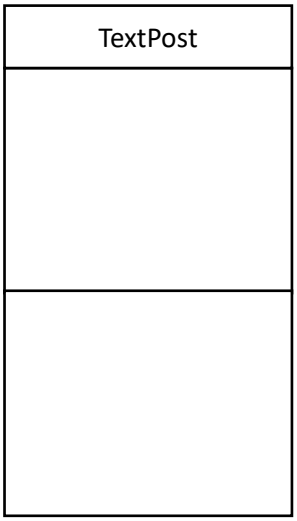
Postoje tri vrste objava: objava teksta (`TextPost`), objava slike (`ImagePost`) i objava zvuka (`AudioPost`). **Svaka od ovih objava se prikazuje na drugačiji sebi svojstven način**. Objava teksta, uz sve što svaka objava mora imati, dodatno sadrži i tekst objave `text` tipa `String`. Analogno, objava slike dodatno sadrži sliku `image` tipa byte[], a zvučna objava sadrži zvuk `audio` tipa byte[] te opis zvučne datoteke `description` tipa String.

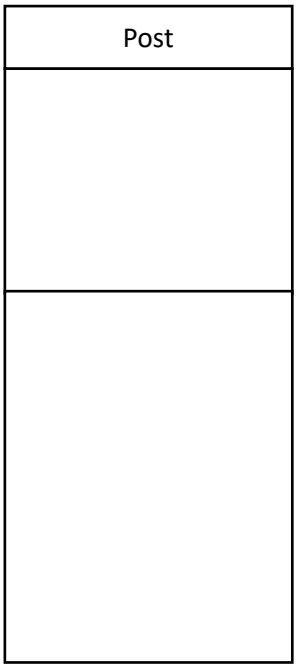
U sustavu postoji mogućnost **objave** grupe objava `PostsGroup` **iste vrste** u obliku albuma slika, popisa tekstova ili popisa zvukova. **U grupi objava se mogu nalaziti samo objave jednog korisnika**. Modelirajte ovu klasu tako da neće biti potrebe za njenim mijenjanjem ako se u budućnosti bude dodao neki novi tip objava. **Grupe objava su također pretražive.**

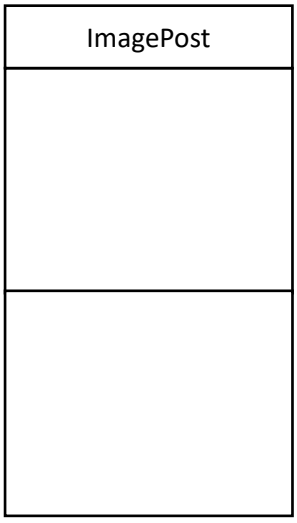


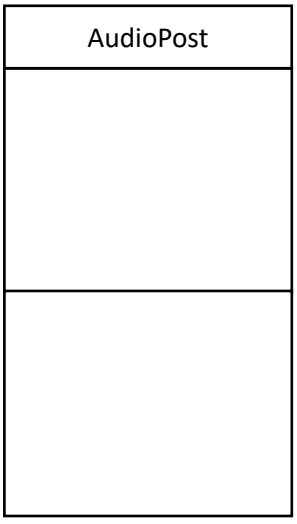

User

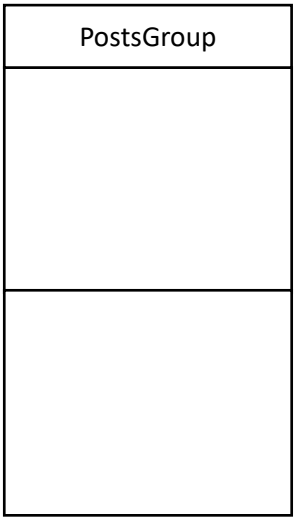

UserStatus

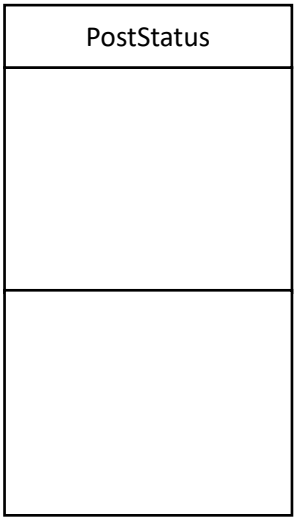

TextPost


Post


ImagePost


AudioPost

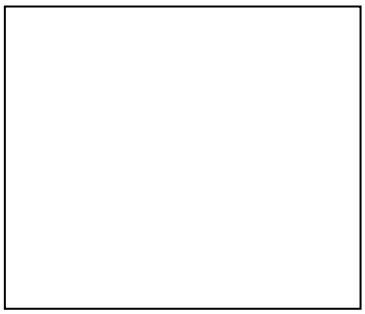

PostsGroup


PostStatus

2. zadatak (10 bodova)

Vaš zadatak je nadopuniti klasu `ShelterCollection` koja se nalazi ispod. Ovom klasom modeliramo sklonište za životinje. Svaka vrsta životinje (npr. `Cat`, `Bird`) je zasebna klasa koja nasljeđuje apstraktnu klasu `Animal`. Pojedina vrsta životinje ima svoj tip koji je modeliran atributom `type` tipa `String` u klasi `Animal` (npr. za mačku "`Korat`", a za pticu "Cockatoo"). Dodatno, svaka životinja ima jedinstveni identifikator `id` tipa `int`. Prilikom implementacije klase `ShelterCollection` omogućite izvođenje programskog koda u metodi `main` klase `Main` prikazane ispod koji kad se izvrši na konzolu ispisuje: **„`1 4 6 2 5 7 3 8 ---- 21 23 22 24`“**. Pritom zamijetite da se iteriranje po skloništu za životinje radi tako da obilazi kružno po tipovima životinja te se odjednom vrati samo po jedna svakog tipa ako takva postoji. **Poredak tipova prilikom obilaska je nebitan, za razliku od poretka životinja određenog tipa koje je potrebno vratiti po redoslijedu dodavanja u sklonište.** U ispisu je poredak tipova pri obilasku bio *Korat-Ragdoll-Toyger* za prvo sklonište, a *Minskin-Peterbald* za drugo.

```
public class Main {

	public static void main(String[] args) {
		ShelterCollection<Cat> catSC1 = new ShelterCollection<>();
		catSC1.add(new Cat(1, "Korat"), new Cat(2, "Korat"), new Cat(3, "Korat"));
		catSC1.add(new Cat(4, "Ragdoll"), new Cat(5, "Ragdoll"));
		catSC1.add(new Cat(6, "Toyger"), new Cat(7, "Toyger"), new Cat(8,
"Toyger"));
		for (Cat cat : catSC1) {
			System.out.print(cat.getId() + " ");
		}
		System.out.print("  ----  ");
		ShelterCollection<Cat> catSC2 = new ShelterCollection<>();
		catSC2.add(new Cat(21, "Minskin"));
		catSC2.add(new Cat(22, "Minskin"));
		catSC2.add(new Cat(23, "Peterbald"), new Cat(24, "Peterbald"));
		for (Cat cat : catSC2) {
			System.out.print(cat.getId() + " ");
		}
	}
}

public class ShelterCollection ____________________________________________________{
	Map<String, List<T>> shelterCollection = new TreeMap<>();

	public void add(T t) {









	}
	public void add(_________ elements) {









	}
```



```
        @Override
        public Iterator<T> iterator() {
                return new MyIterator();
        }

        private class MyIterator _____________________________{
                private List<Iterator<T>> lists;
```

```
        }
        private MyIterator() {
```

```
        }
        @Override
        public boolean hasNext() {
```

```
        }
        @Override
        public T next() {
```

```
        }
    }
}
```

3. zadatak (10 bodova)

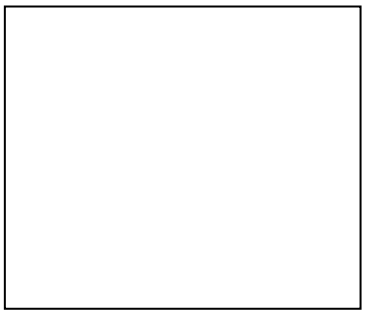
Vaš zadatak je nadopuniti programski kod klase Dog koji se nalazi ispod. Ovom klasom modeliramo pse koji imaju svoje ime `name` tipa `String`, datum okota `dateOfBirth` tipa `Date`, spol `gender` tipa `char`, pasminu `breed` tipa `Set<Breed>`, težinu u kilogramima `weight` tipa `int`, veličinu `size` tipa `Size` i vlasnika `owner` tipa `String`. Veličina psa `Size` je enumeracija koji može poprimiti vrijednosti `SMALL, MEDIUM` i `LARGE`, a određuje se prema njegovoj težini. Pritom vrijedi da su mali psi oni čija je težina između `10` i `20` (uključivo) kg, psi srednje veličine su oni koji teže `20`-`35` (uključivo) kg te konačno veliki psi teški su između `35` i `45` (uključivo).

Dodatno, pas može biti čistokrvan ili mješanac što je definirano njegovom pasminom. Pasmina `breed` psa je skup koji sadrži sve pasmine tipa `Breed` od kojih pas potječe. Konstruktor klase `Dog` prihvaća inicijalne vrijednosti za sve atribute osim za veličinu psa koju je potrebno postaviti temeljem težine.

Dva su psa jednaka ako imaju isto ime, datum okota, spol i vlasnika što morate uzeti u obzir kod implementacije metoda `hashCode` i `equals`. Prirodni poredak pasa je takav da se gleda starost psa (tj. atribut `dateOfBirth`), a potom se njegova težina. **Prilikom implementacije prirodnog poretka uzmite u obzir da će se psi stavljati u kolekciju tipa `TreeSet` sortiranu po njihovom prirodnom poretku (iako to nije prikazano u kodu).**

Klasa Dog ima tri javne konstante: atributni komparator `BY_BREED_PURITY` koji pse uspoređuje po čistokrvnosti, atributni komparator `BY_WEIGHT` koji pse uspoređuje po težini te kompozitni komparator `BY_BREED_PURITY_THEN_WEIGHT` s dvije razine usporedbe koji pse uspoređuje prvo po čistokrvnosti, a zatim po težini. Komparator `BY_BREED_PURITY` je potrebno implementirati kao primjerak ugniježđene klase `ByBreedPurityComparator`. Usporedbu po čistokrvnosti je potrebno napraviti na način da su psi jednaki ako su oba čistokrvna ili imaju jednak broj pasmina, a veći je onaj koji ima veći broj pasmina. Komparator `BY_WEIGHT` potrebno je definirati lambda izrazom (veći je onaj pas koji je teži). **Komparator `BY_BREED_PURITY_THEN_WEIGHT` potrebno je definirati isključivo pozivanjem metoda nad prethodno definiranim atributnim komparatorima** (te za njega nije dozvoljeno pisanje lambda izraza i/ili stvaranja objekata).

```
public class Dog _____________________________________________________________________{
	private String name;
	private char gender;
	private Date dateOfBirth;
	private Set<Breed> breed;
	private int weight;
	private String owner;
	private Size size;

	public static final Comparator<Dog> BY_BREED_PURITY = new
ByBreedPurityComparator();
	public static final Comparator<Dog> BY_WEIGHT =


	public static final Comparator<Dog> BY_BREED_THEN_WEIGHT =


	private static class ByBreedPurityComparator ____________________________________{




	}
```



```
        public Dog(String name, char gender, Date dateOfBirth,
Set<Breed> breed, int weight, String owner) {
                super();

                this.name=name;
                this.gender=gender;
                this.dateOfBirth=dateOfBirth;
                this.breed=breed;
                this.weight=weight;

                //TODO set dog size









        }
        public int hashCode() {









        }
        public boolean equals(Object obj) {









        }
        public int compareTo(Dog other) {









        }
}
```

4. zadatak (10 bodova)

Na raspolaganju nam stoji klasa `Loader` sa metodom `loadData()` koja nam vraća podatke o glasovima za stranke u pojedinim izbornim jedinicama. Metoda `loadData()` vraća mapu `Map<String, Map<String, Integer>>` kojoj je ključ oznaka izborne jedinice, a vrijednost mapa s nazivom stranke kao ključem i s brojem glasova (te stranke u toj izbornoj jedinici) kao vrijednošću.

Ova mapa se koristi kao jedini argument statičkih javnih metoda `getVotesPerElectionUnit` i `getAverageVotesPerParty` u klasi `Main` ispod. **Vaš zadatak je nadopuniti programski kod ove dvije metode pri čemu morate koristiti kolekcijske tokove i ne smijete koristiti pomoćne kolekcije podataka (osim onih koje su izričito navedene u zadatku da se trebaju koristiti)**.

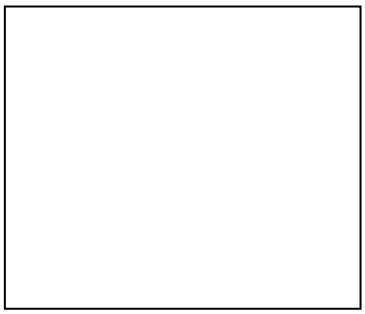
Metoda `getVotesPerElectionUnit` računa ukupni broj glasova (za sve stranke) u svakoj izbornoj jedinici. Kao rezultat vraća mapu `Map<String, Long>` kojoj je ključ oznaka izborne jedinice, a vrijednost ukupan broj glasova (za sve stranke) u toj izbornoj jedinici.

Metoda `getAverageVotesPerParty` računa prosječan broj glasova određene stranke u izbornoj jedinici. Kao rezultat vraća mapu Map<String, Double> kojoj je ključ naziv stranke, a vrijednost prosječan broj glasova te stranke u izbornoj jedinici. Ovu metodu implementirajte u 2 koraka. U prvom koraku **korištenjem metode merge** napunite pomoćnu mapu `tempMap` tipa `Map<String, List<Integer>>` u kojoj je ključ oznaka stranke, a vrijednost lista glasova te stranke po izbornim jedinicama. Npr. za stranku "Blue" se u toj listi trebaju nalaziti vrijednosti 100 i 400. U drugom koraku korištenjem pomoćne mape `tempMap` izračunajte povratnu mapu (tj. rezultat) metode. Npr. Pod ključem "Blue" bi u povratnoj mapi trebala biti vrijednost 250 (tj. prosjek od 100 i 400). **U ovoj metodi ne smijete koristiti metodu `Collectors.groupingBy`.** JavaDoc metode `merge` iz sučelja `java.util.Map` je sljedeći:

```
default V merge(K key, V value, BiFunction<? super V,? super V,? extends
V> remappingFunction)
```

If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value. Otherwise, replaces the associated value with the results of the given remapping function, or removes if the result is `null`.

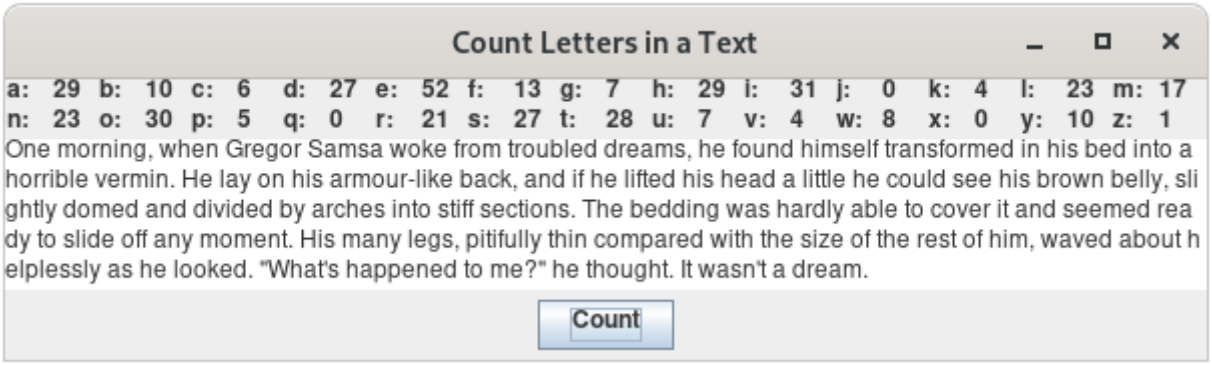
```
public class Loader {
    public static Map<String, Map<String, Integer>> loadData() {
        Map<String, Integer> electionUnit1 = new HashMap<>();
        electionUnit1.put("Blue", 100);
        electionUnit1.put("Green", 200);
        electionUnit1.put("Yellow", 300);
        electionUnit1.put("Red", 400);
        Map<String, Integer> electionUnit2 = new HashMap<>();
        electionUnit2.put("Yellow", 100);
        electionUnit2.put("Green", 200);
        electionUnit2.put("Red", 300);
        electionUnit2.put("Blue", 400);
        Map<String, Map<String, Integer>> votes = new HashMap<>();
        votes.put("eu1", electionUnit1);
        votes.put("eu2", electionUnit2);
        return votes;
    }
}
```



```
public class Main {
    public static void main(String[] args) {

        Map<String, Map<String, Integer>> votes-
Map = Loader.loadData();
        System.out.println("Votes per election unit: " + get-
VotesPerElectionUnit(votesMap)); //Votes per elec-
tion unit: {eu1=1000, eu2=1000}
        System.out.println("Average votes per party: " + get-
AverageVotesPerParty(votesMap)); //Average votes per party: {Red=350.0, Yel-
low=200.0, Blue=250.0, Green=200.0}
    }
    private static Map<String, Long> getVotesPerElectionUnit(Map<String, Map<String, In-
teger>> votesMap) {
        return votesMap.









    }
    private static Map<String, Double> getAverageVotesPerParty(Map<String,
    Map<String, Integer>> votesMap) {
        Map<String, List<Integer>> tempMap = new HashMap<>();

        //fill tempMap using merge
        votesMap.



            tempMap.merge(









        //calculate the result using tempMap
        return tempMap.









    }
}
```

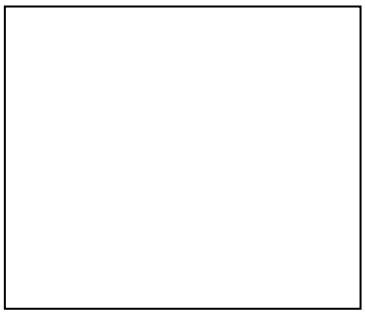
5. zadatak (10 bodova)

Vaš zadatak je nadopuniti programski kod aplikacije s grafičkim korisničkim sučeljem koja je prikazana ispod. Klikom na gumb *Count*, aplikacija treba izračunati broj pojavljivanja pojedinog slova engleske abecede te zatim te izračunate vrijednosti prikazati u gornjem dijelu aplikacije. **Pretpostavite da ovaj posao nije zahtjevan i da se smije obavljati na grafičkoj dretvi.** Prilikom prvog pokretanja aplikacije za sva slova treba prikazati vrijednost *0*.

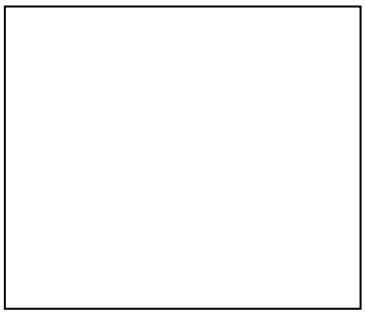



```
public final class LetterCounterFrame extends JFrame {

public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new LetterCounterFrame();
            //TODO show the frame, set its title, etc.
```

```
        });
    }

    private JTextArea textArea;
    private JLabel[] labels = new JLabel[26]; //JLabels for counting characters

    public LetterCounterFrame() {
        textArea = new JTextArea(5, 60); //sets JTextArea size
        textArea.setLineWrap(true);
        add(textArea, BorderLayout.CENTER);
```



```
            //TODO add JLabels for letters and their counts



        //TODO add button



        button.addActionListener((ActionEvent e) -> {
           //TODO implement the button listener
           String text = textArea.getText().toLowerCase();

        });
    }
}
```



Na ovoj stranici možete napisati kod koji vam nije stao na prethodne stranice.