

**Гимназија „Јован Јовановић Змај“**

**Нови Сад, Златне Греде 4**

**Матурски рад из Објектно Оријентисаног Програмирања  
Концепти Објектно Оријентисаног Програмирања  
Објашњени Кроз Пројекат**

**Ментор:**

**Милан Станковић**

**Ученик:**

**Душица Трбовић, IV-7**

**Нови Сад, април 2021. год.**

# САДРЖАЈ

1. Увод у објектно оријентисано програмирање .....	3
2. Израда Пројекта.....	5
2.1. Израда апликације у програмском језику „java“ .....	5
2.2. Увод у тему апликације „Ресторан“ .....	5
3. Имплементација апликације.....	6
3.1. UML дијаграм .....	6
3.2. Главне класе „Menadzer.java“ и „Konobar.java“ .....	7
3.2.1. „Menadzer.java“ .....	7
3.2.2. „Konobar.java“ .....	7
3.3. Споредне класе .....	9
3.3.1. „GotoviProizvodi.java“ .....	9
3.3.2. „Cenovnik.java“ .....	10
3.3.3. „Racun.java“ .....	10
4. Кориснички интерфејс .....	12
4.1. Увод у „javaFX“ .....	12
4.2. Имплементација „javaFX“ у апликацији .....	12
5. Приказ коначног изгледа апликације .....	15
6. Закључак.....	20
Литература .....	21
Биографија.....	22

## 1. УВОД У ОБЈЕКТНО ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ

Да би концепти објектно оријентисаног програмирања били јасни, потребно је да објаснимо шта је у основи објектно оријентисано програмирање, за шта се оно користи, и које су његове предности и мане. Наравно да није у циљу да користимо објектно оријентисано програмирање у сваком моменту, и да није прилагодљиво за сваки тип пројекта и програмирања. Ово је разлог због ког морамо да научимо у којим ситуацијама се овакав начин програмирања користи, и чему он тачно служи.

Приликом објектно оријентисаног програмирања, подаци су много битнији од самих функција, док то није случај у, на пример, процедуралном програмирању.

Објектно оријентисано програмирање је заправо једна од многобројних програмских парадигми које одређују начин програмирања. Ако се бавимо објектно оријентисаним програмирањем, неопходно је да се упознамо са појмом објекта и класе. Класе дефинишу објекат, односно оне садрже атрибуте и методе које објекат треба да има. Да би се креирао објекат, потребно је претходно имати класу, а у тој класи конструкторе. Конструктори су методе класа које се позивају када се креира нова инстанца те класе. Сви конструктори имају исти назив као и класа у којој се налазе. Још једна од предности објектно оријентисаног програмирања јесте могућност наслеђивања. Наслеђивање нам омогућава да од једне класе ми изведемо њене подкласе. На тај начин подкласа садржи све атрибуте и функције главне класе, али можемо да додајемо или мењамо податке који нам не одговарају, ово омогућава уштеду меморије, и избегава се непотребно понављање одређеног дела пројекта, које је у процедуралном програмирању неизбежно. Објектно оријентисано програмирање такође садржи и могућност енкапсулације и апстракције. Они представљају различите нивое заштите података приликом приказивања атрибута неке класе. Примера ради, нема потребе да на сајту на ком обављамо куповину остављамо наш јмбг, али ми као особа га ипак поседујемо. Наш јмбг био би на нивоу заштите „private“ (подаци су доступни само унутар ове класе). Преостала два нивоа су: „public“ (подаци су доступни свим класама) и „protected“ (подаци су доступни самој класи и њеним изведеним класама).

Познато нам је да је у програмирању битан тимски рад, уколико наш пројекат садржи много неповезаних линија кода, које су нама јасне, јер смо ми радили на том пројекту, то никако не иде у корист људима са којима радимо, или који у неком моменту треба да се прикључе раду на том пројекту. Објектно оријентисано програмирање решава овај проблем. Све је организовано кроз класе и објекте, па се тако чланови тима постепено упознају са програмом, и могу да буду упућени само у део истог, ако је то њихов резон, што такође омогућава знатну уштеду времена. На пример, ако бисмо имали класу типа Особа, можемо да направимо инстанцу те класе под називом Марко\_Марковић, затим додељујемо вредности њеним атрибутима, који могу бити име, презиме, јмбг, место рођења.. На овај начин избегавамо понављање делова кода који би били исти за сваку особу, тј свака особа има име, презиме, јмбг.. Када у програму постоји класа „Особа“, довољно је да направимо инстанцу те класе, и да је уредимо онако како нама одговара, тј доделимо јој јединствене атрибуте. Овим начином програмирања пројекат се разлаже на мање логичке целине. Ове целине имају своје специфичне карактеристике, тј атрибуте и функције. Између ових логичких целина постоји комуникација, без обзира на њихову раздвојеност. На овај начин акценат при програмирању стављамо на проблем који требамо решити, и фокусирамо се на мање делове који имају одређене задатке. Нека од главних предности код овог начина програмирања јесте што програмери не морају да мењају кодове када уносе нове објекте или податке.

Једноставно креирају објекат са наслеђеним особинама старог и мењају особине које не одговарају, што програмерима омогућава лако мењање објектно оријентисаних програма.

Кроз историју, објектно оријентисано програмирање се мењало, концепти су усавршавани, и настајали су многи језици који су користили овај начин програмирања. Један од првих језика био је „Симула“, још 1967. године. Он је креиран са намером да представља програм где су објекти најбитнији део презентовања информација, што у основи и јесте идеја оваквог начина програмирања. Данас, неки од популарнијих језика јесу *Java*, *C++*, *JavaScript*. Сваки има своје предности и мане и у односу на наше потребе бирамо језик који нама највише одговара за реализацију програма. Језик који је коришћен у изради пројекта за овај матурски рад јесте *Java*. *Java* је језик који је представљен 1995. године, и базиран је на класама. Популарност овог језика је расла због његове универзалности. Користи се највише приликом израда апликација које се баве односом клијент-сервер, каква јесте направљена апликација „Ресторан“.

Битно је да напоменемо и разлике у процедуралном програмирању и објектно оријентисаном програмирању. Кроз процедурално, наш програм подељен је на више мањих целина којима су основни делови функције програма и сви подаци важе само унутар одређене функције, док код објектно оријентисаног све је организовано по класама, свака класа организује своје податке, и функције које одређују њихово „понашање“.

## 2. ИЗРАДА ПРОЈЕКТА

Пре имплементације апликације „Ресторан“ потребно је да се фокусирамо на садржај који она треба да поседује. Потребно је да урадимо малу скицу апликације, која ће нам олакшавати даљу израду саме апликације. Битно је да велики пројекат поделимо на више мањих делова (како је и конципирано објектно оријентисано програмирање). Ресторан делимо на делове који ће касније представљати класе, а након тога ближе описујемо те класе и тако одређујемо њихове атрибуте и функције које требају да садрже.

### 2.1. Израда апликације у програмском језику „java“

Зашто је апликација „Ресторан“ рађена баш у језику *Java*? *Java* је једноставан језик који не користи неки посебан начин размишљања или потребу за предзнањем посебних алгоритама. Он је објектно оријентисан језик а самим тим фокус је над подацима, тј самим објектима и повезаности тих објеката. *Java* нам онемогућује приступ меморији, читање или уписивање датотека без дозволе. Ово нам обезбеђује додатну сигурност, какву треба да има свака апликација. Још једна од предности овог језика јесте што није битан оперативни систем на ком се покреће. Компајлирани код може се извршити на многим процесорима, услов јесте присуство извршног система *Java*. Језик је такође пројектован тако да се прилагођава окружењу, које се стално унапређује, тако уз само додавање нових библиотека, можемо да унапредимо код, а да то нема никаквог утицаја на клијенте.

### 2.2. Увод у тему апликације „Ресторан“

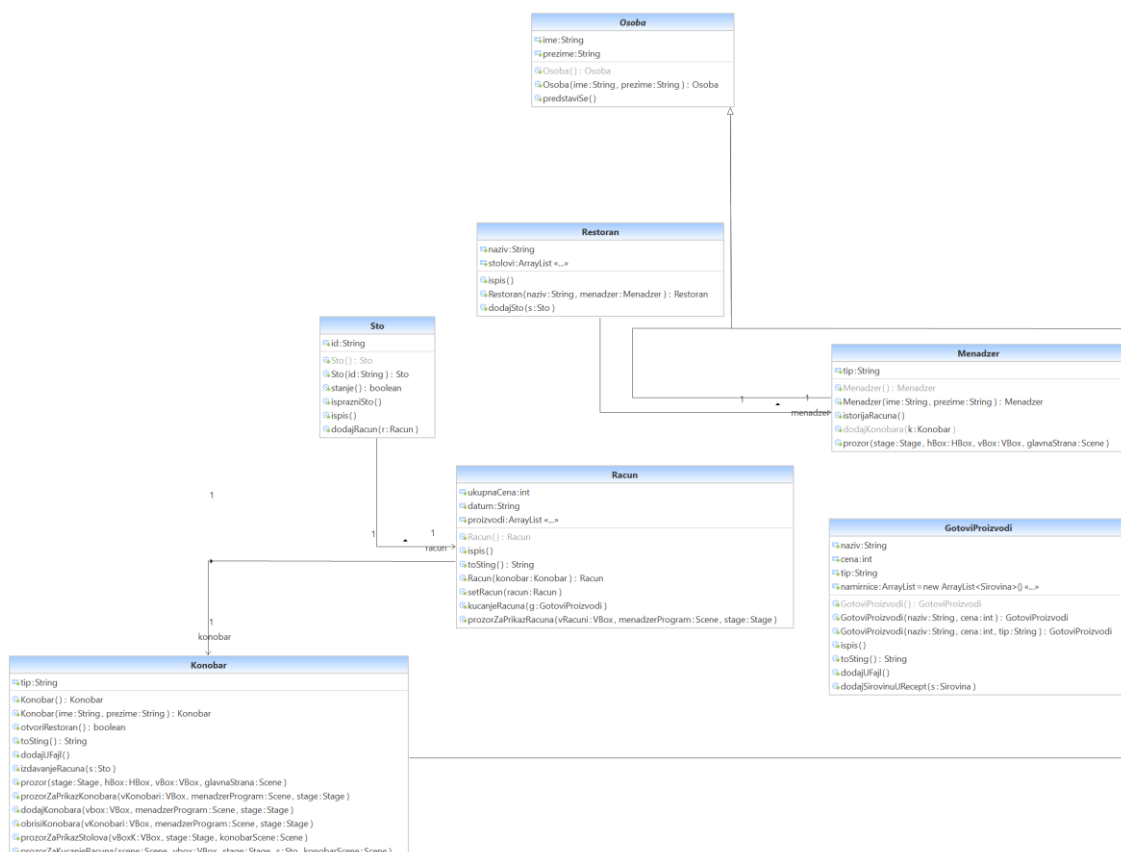
Апликација „Ресторан“ је подељена на два дела, а приказује се један од делова у зависности од тога ко има приступ у датом моменту. Први део јесте менаџерски. Менаџерски део пројекта садржи могућности измене, додавања или брисања података. Конкретно подаци који су у овој апликацији приказан и унесени у базу, јесу подаци о артиклима које ресторан поседује. Артикли су спојени и они чине ценовник. Ценовник може да се уређује, да се унесе нови подаци, бришу стари, или мењају цене већ постојећих. Сличне опције постоје за податке о конобарима. Сви конобари чине једну базу и у њој се налазе њихова имена и презимена. Менаџер има опцију да брише или уноси нове конобаре. Још једна од опција којој менаџер има приступ јесте приказ листе свих рачуна који су издати у одређеном временском периоду. Ово је битна опција ради правдања трошкова или извлачења продајних података на месечном, тј годишњем нивоу.. Други део апликације користе конобари ресторана. Они не требају да имају исте могућности као и менаџери, и због тога је апликација и подељена на два дела. Конобарима се приликом отварања апликације приказује распоред столова у ресторану. Кликом на сто отварају опцију за куцање рачуна, на управо том столу. Приликом куцања рачуна битно је да се одреди конобар који куца рачун у том моменту, а након тога бирају се артикли које је гост поручио. Када је потребно да се изврши наплата, конобар кликне опцију да изда рачун и приказује се списак свих поручених артикала, њихове цене и коначан износ рачуна. Конобар након издавања рачуна има могућност да на истом столу куца нове рачуне, али док год постоји рачун који није завршен, сто се води као заузет. Ово је укратко опис апликације „Ресторан“.

### 3. ИМПЛЕМЕНТАЦИЈА АПЛИКАЦИЈЕ

Да бисмо имплементирали апликацију у *Јави*, морамо да знамо шта је *Јава* виртуелна машина (JVM). JVM је апликација у рачунару која омогућава имплементирање и покретање *Јава* програма. Пре него што се програм покрене, он се преводи у *Јава* бајт кодове (скупове инструкција које извршава Јавина Виртуална Машина, свака инструкција је дужине једног бајта, што објашњава сам назив). Предност постојања и рада *Јава* виртуелне машине јесте и разлог популарности коју је стекао *Јава* програмски језик. Основни принцип овог језика је "*Compile once, run everywhere*" („Компајлирај једном, покрени свуда.“), када је програм једном преведен у Јава бајт код, он се може покренути на било којој платформи која садржи *Јава* виртуелну машину, без обзира на оперативни систем.

#### 3.1. UML дијаграм

Развој апликација је превише комплексан да би могли да га планирамо из главе. У циљу решавања тог проблема настали су модели. Модели су упрошћене верзије реалности. Они дефинишу шаблон који помаже касније приликом конструисања апликације, омогућава јасноћу и лакше разумевање. Објектно оријентисано моделовање је засновано на објектима и класама. *Unified Modelling Language (UML)* је стандардни графички језик за моделовање објектно оријентисаног софтвера. Основни елементи *UML*-а су ствари и релације. Ствари представљају битне концепте, тј класе програма, док су релације повезаност тих концепата. На крају настаје дијаграм који је груписање међусобно повезаних колекција ствари и релација.



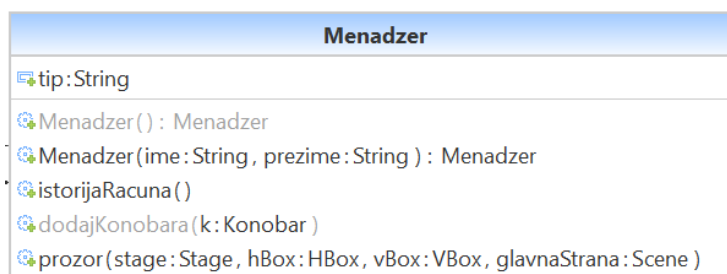
Слика 1 - UML дијаграм апликације "Ресторан"

### 3.2. Главне класе „Menadzer.java“ и „Konobar.java“

Класе „Menadzer.java“ и „Konobar.java“ наслеђују класу „Osoba.java“, она садржи атрибуте име и презиме и функцију која омогућава представљање, тако што исписује име и презиме тражене особе. У класама „Menadzer.java“ и „Konobar.java“ од додатних атрибута садрже и тип. Он нам кроз програм омогућава лакшу проверу и у односу на тип додељује различите функционалности.

#### 3.2.1. „Menadzer.java“

„Menadzer.java“ садржи различите функције, које нам омогућавају отварање менаџерског дела апликације. Функција „prozor()“ служи за отварање предходно наведеног дела апликације. Да бисмо у класи „App.java“ били у могућности да направимо инстанцу особе која ће бити типа менаџер неопходно нам је да постоје конструктори ( нпр „Menadzer()“ ).



Слика 2 – Дијаграм објекат МЕНАѢР

У функцији „prozor()“ су реализоване могућности попут, додавања пића, хране, конобара у базу података, као и приказ историје рачуна. Један од примера јесте функција „istorijaRacuna()“ у којој се рачун уписује у датотеку „racuni.txt“. Рачун долази у облику текста који је писан у више линија, да бисмо тај текст уписали у датотеку, неопходно је да поделимо тај текст на делове. Као параметар за поделу узимамо *enter*, и уписујемо једну по једну линију у датотеку. По сличном принципу раде и функције за додавање конобара и артикала у ценовник.

```

public void istorijaRacuna() {
    StringBuilder sb = new StringBuilder();
    try (BufferedReader br = Files.newBufferedReader(Paths.get("racuni.txt"))) {
        // read line by line
        String line;
        while ((line = br.readLine()) != null) {
            sb.append(line).append("\n");
        }
    } catch (IOException e) {
        System.err.format("IOException: %s\n", e);
    }
    System.out.println(sb);
}
  
```

Слика 3 – Функција „istorijaRacuna()“ у класи „Menadzer.java“

#### 3.2.2. „Konobar.java“

„Konobar.java“ садржи различите функције, које нам омогућавају отварање конобарског дела апликације. Ова класа садржи 2 конструктора, који омогућавају каснију реализацију инстанци у „App.java“. Функција „prozor()“ омогућава кориснику коришћење тог дела апликације. У њој се налази део кода који пружа прилику за куцање и издавање рачуна. Функција је подељена на више споредних, које се користе

кроз код. Друге функције попут „*prozorZaPrikazStolova()*“, која служи за распоређивање столова на главној страници и омогућава даље позивање функције „*prozorZaKucanjeRacuna()*“ кликом на неки од понуђених столова, затим отварањем тог прозора отвара се могућност за куцање рачуна, где имамо опције за бирање конобара која се реализује преко функције „*prozorZaPrikazKonobara()*“, наредна опција приликом куцања рачуна је избор артикала. Куцање рачуна се завршава кликом на дугме „издај рачун“. Уколико конобар приликом куцања рачуна није завршио поруџбину, има могућност да се врати назад, и подаци који су унешени приликом куцања рачуна на одређени сто остају сачувани, све до издавања рачуна.

Konobar
tip:String
Konobar(): Konobar
Konobar(ime:String, prezime:String): Konobar
otvoriRestoran(): boolean
toString(): String
dodajUFajl()
izdavanjeRacuna(s: Sto)
prozor(stage:Stage, hbox:HBox, vbox:VBox, glavnaStrana:Scene)
prozorZaPrikazKonobara(vKonobari:VBox, menadzerProgram:Scene, stage:Stage)
dodajKonobara(vbox:VBox, menadzerProgram:Scene, stage:Stage)
obrisiKonobara(vKonobari:VBox, menadzerProgram:Scene, stage:Stage)
prozorZaPrikazStolova(vBoxK:VBox, stage:Stage, konobarScene:Scene)
prozorZaKucanjeRacuna(scene:Scene, vbox:VBox, stage:Stage, s:Sto, konobarScene:Scene)

Слика 4 – Дијаграм објекат КОНОБАР

Функције за додавање и брисање конобара се такође налазе у класи „*Konobar.java*“. Додавање ради по веома сличном принципу као и додавање рачуна у текстуалну датотеку. Брисање функционише мало другачије. Правимо једну променљиву у коју стављамо конобара ког је означио корисник апликације. Затим пролазимо кроз текстуалну датотеку „*osobe.txt*“ све док не пронађемо подударање са променљивом која је предходно направљена. Када се догоди подударање, цео текст, изузев линије код које се догодило подударање се копира у променљиву, текстуална датотека се празни, и поново се уписују подаци из променљиве. На овај начин у датотеци више не постоји конобар који треба бити обрисан.

```
obrisi.setOnMouseClicked(e -> {
    ObservableList<Konobar> izabrano = FXCollections.observableArrayList();
    izabrano = tableView1.getSelectionModel().getSelectedItem();
    System.out.println(izabrano.get(0).getTime());
    String ime = izabrano.get(0).getTime();

    izabrano.forEach(gp1::remove);

    BufferedReader reader1;
    try {
        reader1 = new BufferedReader(new FileReader("osobe.txt"));
        ArrayList<String> line = new ArrayList<String>();
        line.add(reader1.readLine());
        while (line.get(line.size()-1) != null) {
            line.add(reader1.readLine());
        }
        line.remove(line.size()-1); // na ovom mestu je ushvacen null i stavljen u niz

        PrintWriter writer = new PrintWriter("osobe.txt");
        for (int l = 0; l < line.size(); l++) {
            String[] prim = line.get(l).split("@");
            if(prim[0].compareTo(ime) != 0) {
                writer.print(line.get(l));
                writer.println();
            }
        }
    }
})
```

Слика 5 - Функција „*obrisiKonobara()*“ у класи „*Konobar.java*“



### 3.3. Споредне класе

#### 3.3.1. „GotoviProizvodi.java“

Класа „*GotoviProizvodi.java*“ је класа која представља све артикле који постоје у овој апликацији. Сваки артикал има назив, цену и одређени је тип (храна или пиће). Ове особине представљају и атрибуте класе. Функције које су нам битне за даље коришћење артикала у класи „*Cenovnik.java*“ су претварање артикала у обичан текст и уписивање тог текста у датотеку „*cenovnik.txt*“. Прва функција је „*toString()*“, у њој постоји променљива у коју се уписује сав текст артикла. Текст се спаја тако што се између два различита податка (нпр назив и цена), ставља карактер „@“. Када променљива садржи текст који се генерише по формули назив@цена@тип ова функција враћа исту ту променљиву.

```
public String toString() {
    String proizvod;

    proizvod = naziv + "@" + String.valueOf(cena) + "@";
    if(getTip() == "pice") {
        proizvod = proizvod + ("pice");
    }else {
        proizvod = proizvod + ("hrana");
    }

    return proizvod;
}
```

Слика 6 – Функција „*toString()*“ у класи „*GotoviProizvodi.java*“

Наредна функција под називом „*dodajUFajl()*“ омогућава да се артикли додају у датотеку „*cenovnik.txt*“. Прво следи провера да ли датотека у коју желимо да упишемо податке постоји, ако постоји функција наставља да се извршава. Даље пролази се кроз текст који постоји у датотеци пре уписа нових података, а када се дође до краја, на крај се додаје нова линија у коју ћемо да сместимо артикал који је предходно претворен у обичан текст (уз помоћу већ објашњене функције „*toString()*“).

```
public void dodajUFajl() {
    StringBuilder sb = new StringBuilder();
    try (BufferedReader br = Files.newBufferedReader(Paths.get("cenovnik.txt"))) {
        // read line by line
        String line;
        while ((line = br.readLine()) != null) {
            sb.append(line).append("\n");
        }
    } catch (IOException e) {
        System.err.format("IOException: %s\n", e);
    }

    try (FileWriter writer = new FileWriter("cenovnik.txt");
        BufferedWriter bw = new BufferedWriter(writer)) {

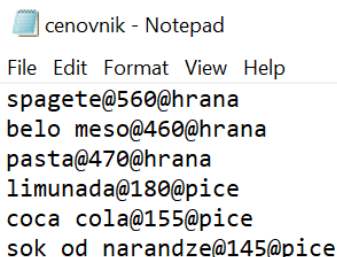
        bw.write(sb.toString());
        bw.write(this.toString());
        bw.newLine();

    } catch (IOException e) {
        System.err.format("IOException: %s\n", e);
    }
}
```

Слика 7 – Функција „*dodajUFajl()*“ у класи „*GotoviProizvodi.java*“

Након што се текст дода у „*cenovnik.txt*“ она изгледа као на слика у наставку. Карактери „@“ се користе за одвајање података који су нам битни, и касније омогућава раздвајање ових података, и њихово поновно коришћење. Могуће је користити и обичан размак али називи артикала могу да се састоје и из више речи, из овог разлога користимо

алтернативу, тј карактер по избору који се не налази у тексту који уписујемо у датотеку. Ово наравно не би било добро решење да један од података јесте мејл адреса.



```

File Edit Format View Help
spagete@560@hrana
belo meso@460@hrana
pasta@470@hrana
limunada@180@pice
coca cola@155@pice
sok od narandze@145@pice

```

Слика 8 – Изглед датотеке „cenovnik.txt“

### 3.3.2. „Cenovnik.java“

Класа „Cenovnik.java“ садржи функције које класа за менаџера користи да би приказивала артикле из текстуалне датотеке. Функције попут приказа ценовника, и оних које омогућавају измену истог, као на пример додавање или брисање артикала. Једна од функција јесте „dodajPice()“.

```

TextField nazivArtikla = new TextField();
TextField cenaArtikla = new TextField();
nazivArtikla.setPromptText("Naziv Artikla");
cenaArtikla.setPromptText("Cena Artikla");

Button dodaj = new Button("dodaj");
dodaj.setOnMouseClicked(e -> {

    GotoviProizvodi product = new GotoviProizvodi();
    product.setNaziv(nazivArtikla.getText());
    product.setCena(Integer.parseInt(cenaArtikla.getText()));
    product.setTip("pice");

    nazivArtikla.clear();
    cenaArtikla.clear();

    product.dodajUFajl();

    Text text = new Text("Uspesno ste dodali novi artikal!");
    text.setX(50);
    text.setY(50);
    text.setFont(Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR, 10));
    text.setUnderline(true);

    vCenovnik.getChildren().add(text);
});

```

Слика 9 – Функција „dodajPice()“ у класи „Cenovnik.java“

У овој функцији постоје 2 поља која омогућавају унос текста и дугме „додај“. Приликом клика на дугме додај, подаци из ова два поља се повлаче у променљиве и прави се нова инстанца типа „GotoviProizvodi“. Постављају се атрибути, тј назив, цена и тип. Након тога текст из ових поља се брише а променљива се додаје у фајл (за ово такође постоји функција, која ће инстанце овог типа претварати у текст и уписивати у фајл). Уколико се артикал успешно додао у текстуалну датотеку, на екрану се појављује и текст који обавештава корисника о томе.

### 3.3.3. „Racun.java“

У класи „Racun.java“ омогућава се опција куцања рачуна за столом. Функција која омогућава ово јесте „kucanjeRacuna()“. Како сам рачун као своје атрибуте садржи конобара, датум издавања, листу артикала и укупну цену, у овој функцији се додају

артикли у листу, и мења укупна цена рачуна, тако што се на стару укупну цену додаје цена производа који се у том моменту додаје.

```
public void kucanjeRacuna(GotoviProizvodi g) {
    proizvodi.add(g);
    int i = proizvodi.size();
    ukupnaCena = ukupnaCena + proizvodi.get(i-1).getCena();
}
```

Слика 10 – Функција „kucanjeRacuna()“ у класи „Racun.java“

Друга битна функција служи са испис самог рачуна. Сви атрибути се исписују на конзолу, редом. Пре исписа, следи провера да ли рачун постоји, ако не постоји, исписује се порука која обавештава корисника о томе. У случају да рачун постоји прво што се исписује је име и презиме конобара, позивом функције „predstaviSe()“. Након тога исписује се датум и време издавања рачуна. Следећи део рачуна су артикли. Кроз петљу пролазимо онолико пута колико је и листа артикала дугачка, у сваком проласку кроз петљу, ми исписујемо назив и цену производа. Последња ставка која се исписује на рачуну је укупна цена, која постоји као атрибут ове класе.

```
public void ispis() {
    if(proizvodi.size() == 0) {
        System.out.println("Racun jos uvek nije iskucan!");
    }else {
        System.out.println("Konobar: " );
        konobar.predstaviSe();
        System.out.println("Datum i vreme: " + datum);
        System.out.println("Porudzbina: ");
        for (int i = 0; i < proizvodi.size(); i++) {
            System.out.println((i+1) + ". proizvod: " + proizvodi.get(i).getNaziv() + " cena: " +proizvodi.get(i).getCena() );
        }
        System.out.println("Ukupna cena: " + ukupnaCena);
    }
}
```

Слика 11 – Функција „ispis()“ у класи „Racun.java“

## 4. КОРИСНИЧКИ ИНТЕРФЕЈС

Кориснички интерфејс је простор где се остварују интеракције између људи и машина. Смисао дизајна корисничког интерфејса је да произведе себи кориснички интерфејс који омогућава ефективну и једноставнију контролу машине од стране човека, у циљу добијања траженог резултата.

### 4.1. Увод у „javaFX“

*JavaFX* је софтверска платформа за креирање десктоп апликација или неких интернет апликације које имају могућности да раде на великом броју уређаја, независно од њиховог оперативног система (*Windows, Linux и Mek*). *Swing* је вицет алатка у Графичко корисничком интерфејсу (*GUI*), за *Java*. *JavaFX* је дизајниран тако да замени *swing*, као подразумевана *GUI* библиотека, али је кроз праксу одлучено да ће оба бити коришћена у будућности. Све до верзије 2.0 *JavaFX* програмери су користили статички куцан језик који се звао *JavaFX Script* за креирање апликација. Овај језик је компајлиран у *Java bytecode*, па је тако омогућено истовремено коришћење обичног *Java* кода. *JavaFX 2.0* и касније верзије су имплементирани као оригинална *Java* библиотека, и апликације које користе *JavaFX* су писане углавном у *Java*. *JavaFX* је веома компатибилан језик, могуће га је покренути на било ком десктопу који може да покрене *Java SE*.

### 4.2. Имплементација „javaFX“ у апликацији

*JavaFX* представља делове кода који су у овој апликацији названи прозорима. Кликом на дугме отвара се одређени прозор. Ово можемо видети у класи „App.java“. *JavaFX* садржи *box*-еве у које се смештају сви елементи да би кориснику приликом покретања апликације они били видљиви. Постоје вертикални и хоризонтални *box*. У овој апликацији подаци се прво смештају у хоризонталне (може их бити и више), а након тога те хоризонталне *box*-еве стављамо у вертикални.

```
public void start(Stage stage) throws FileNotFoundException{
    VBox vbox = new VBox();
    HBox hbox = new HBox();
    Scene glavnaStrana = new Scene(vbox, 600, 600);

    Image image3 = new Image(new FileInputStream("slike/blue.png"));
    ImageView pravougaonik = new ImageView(image3);
    pravougaonik.setScaleX(300);
    pravougaonik.setFitHeight(80);
    pravougaonik.setFitWidth(80);

    Image image1 = new Image(new FileInputStream("slike/menager.png"));
    ImageView menadzerDugme = new ImageView(image1);

    Image image2 = new Image(new FileInputStream("slike/waiter.png"));
    ImageView konobarDugme = new ImageView(image2);

    Konobar konobar = new Konobar();
    Menadzer menadzer = new Menadzer();

    menadzerDugme.setOnMouseClicked(e -> menadzer.prozor(stage, hbox, vbox, glavnaStrana));
    konobarDugme.setOnMouseClicked(a->konobar.prozor(stage, hbox, vbox, glavnaStrana));

    menadzerDugme.setFitHeight(80);
    menadzerDugme.setFitWidth(80);
    konobarDugme.setFitHeight(80);
    konobarDugme.setFitWidth(80);
}
```

Слика 12 – Функција „start()“ у класи „App.java“

Још неки од елемената које нам *javaFX* омогућава да користимо су слике. Можемо да контролишемо величину слика, и да користимо слике као дугме у програму, па тако кликом на слику „image1“ корисник отвара менаџерски прозор апликације. Приликом коришћења слика неопходно је да будемо сигурни да ли та слика постоји, и у случају да

се деси неки проблем, да би апликација остала у функцији без обзира на слику која у том тренутку из неког разлога није могла бити пронађена. Функција проверава путању на којој се налази слика, ако она постоји, користи се у даљем програму, ако није пронађена на конзолу се испишује порука кориснику.

```
try {
    Image image;
    image = new Image(new FileInputStream("slike/logo.jpg"));
    ImageView imageView = new ImageView(image);

    imageView.setFitHeight(455);
    imageView.setFitWidth(600);

    ImageView imageView1 = new ImageView(image);

    imageView1.setFitHeight(455);
    imageView1.setFitWidth(500);
    vbox.getChildren().add(imageView);

} catch (FileNotFoundException e1) {
    e1.printStackTrace();
}
```

Слика 13 – Провера путање слике у класи „App.java“

Главни део целе апликације је *stage* у који се смешта *scene* која је одређене висине и ширине, а у њој се налазе *box*-еви у који садрже податке. *Stage*-у додељујемо име, што је заправо назив саме апликације.

```
hBox.getChildren().addAll(pravougaonik,menadzerDugme, konobarDugme);
vBox.getChildren().addAll(hBox);
stage.setTitle("Projekat - Restoran");
stage.setScene(glavnaStrana);
stage.show();
```

Слика 14 – Коначан приказ stage-а у класи „App.java“

Приликом клика на „*image2*“ у класи „App.java“ отварамо конобарску страницу. Она се налази у класи „Konobar.java“, и садржи нову сцену у коју ће бити смештени *box*-еви са подацима који се приказују приликом отварања овог прозора. У *box*-у постоји и дугме „одјави се“ који враћа корисника на предходну сцену, тј на главну страницу.

```
public void prozor(Stage stage, HBox hBox, VBox vbox, Scene glavnaStrana) {
    VBox vboxK = new VBox();

    Scene konobarProgram = new Scene(vboxK, 610, 650);
    stage.setScene(konobarProgram);

    Button odjava1 = new Button("ODJAVI SE");
    odjava1.setOnMouseClicked(e->stage.setScene(glavnaStrana));
    prozorZaPrikazStolova(vboxK, stage, konobarProgram);
    vboxK.getChildren().add(odjava1);
}
```

Слика 15 – Функција „prozor()“ у класи „Konobar.java“

На конобарској страници позива се и функција „*prozorZaPrikazStolova()*“ која омогућава приказ 9 столова, тј слика. Постоји 5 различитих слика, али 9 променљивих типа *Image*. У овој сцени имамо 3 хоризонтална *box*-а у којој се налазе ове променљиве, а они су смештени у главни, вертикални *box*. Свака слика повезана је са променљивом типа *Sto* који садржи *id*. То нам омогућава куцање више рачуна у исто време, без губитка података. Кликком на неку од слика отвара се нови прозор, који служи за куцање рачуна.

```

Image image = new Image(new FileInputStream("slike/table.png"));
Image image1 = new Image(new FileInputStream("slike/table1.png"));
Image image2 = new Image(new FileInputStream("slike/table2.png"));
Image image3 = new Image(new FileInputStream("slike/table3.png"));
Image image4 = new Image(new FileInputStream("slike/table4.png"));
ImageView imageView1 = new ImageView(image2);
ImageView imageView2 = new ImageView(image1);
ImageView imageView3 = new ImageView(image3);
ImageView imageView4 = new ImageView(image);
ImageView imageView5 = new ImageView(image4);
ImageView imageView6 = new ImageView(image);
ImageView imageView7 = new ImageView(image2);
ImageView imageView8 = new ImageView(image);
ImageView imageView9 = new ImageView(image1);

VBox vb1 = new VBox();

Sto sto1 = new Sto("1");
Scene s1 = new Scene(vb1, 500, 500);
imageView1.setOnMouseClicked(e -> stage.setScene(s1));
prozorZaKucanjeRacuna(s1, vb1, stage, sto1, konobarScene);

```

Слика 16 – Функција „prozorZaPrikazStolova()“ у класи „Konobar.java“

Менаџерска страница садржи опцију падајућег менија. Мени се састоји *MenuItem*-а.

```

//opcije koje se pojavljuju u padajucem meniju
MenuItem prikazCenovnika = new MenuItem("prikazi cenovnik");
MenuItem prikazPica = new MenuItem("prikazi pice");
MenuItem dodajPice = new MenuItem("dodaj pice");
MenuItem prikazHrane = new MenuItem("prikazi hranu");
MenuItem dodajHranu = new MenuItem("dodaj hranu");
MenuItem prikazKonobara = new MenuItem("prikazi listu konobara");
MenuItem dodajNovogKonobara = new MenuItem("dodaj novog konobara");
MenuItem obrisiKonobara = new MenuItem("obrisi konobara");
MenuItem prikazRacuna = new MenuItem("prikazi istoriju izdatih racuna");

```

Слика 17 – променљиве типа *MenuItem* у класи „Menadzer.java“

Приликом клика на неки *MenuItem* отварамо други прозор, тј они се понашају као и обично дугме. На пример, у менију садржимо 3 основне опције, ценовник, конобари и историја рачуна, код ценовника постоји падајући мени и приликом клика на опцију приказ ценовника отвара се прозор за приказ истог, тј на *javaFX* језику, отвара се друга сцена.

```

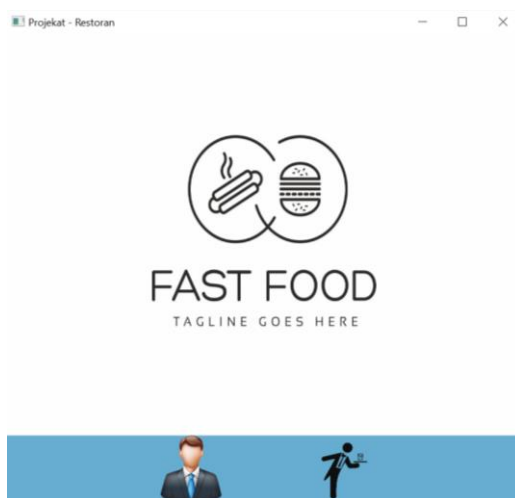
Menu cenovnik = new Menu("Cenovnik");
cenovnik.getItems().add(prikazCenovnika);
cenovnik.getItems().add(prikazPica);
cenovnik.getItems().add(dodajPice);
cenovnik.getItems().add(prikazHrane);
cenovnik.getItems().add(dodajHranu);
prikazCenovnika.setOnAction(e -> {
    stage.setScene(cenovnikScene);
    Cenovnik c = new Cenovnik();
    c.prozorZaPrikazCenovnika(vCenovnik, menadzerProgram, stage);
});
prikazPica.setOnAction(e -> {
    stage.setScene(cenovnikScene);

```

Слика 18 – Опције из падајућег менија у класи „Menadzer.java“

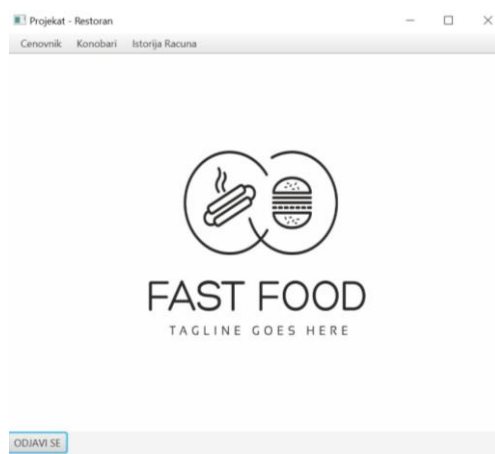
## 5. ПРИКАЗ КОНАЧНОГ ИЗГЛЕДА АПЛИКАЦИЈЕ

Главна страница апликације је прозор који се отвара приликом покретања апликације. Страница нуди две опције, кликом на лево дугме (човек у оделу), отвара се менаџерски део апликације, док се кликом на десно (човечуљак који држи тацну) отвара се конобарски део.



Слика 19 – Главна страница

Менаџерски део апликације је сличан главној страници, што се изгледа тиче, разлика је у опцијама које постоје. Мени садржи опције за ценовник, конобаре и рачуне. У углу странице за менаџера се налази и дугме за одјаву. Кликком на то дугме враћамо се на главну страницу.



Слика 20 – Менаџер страница

Опције које има ценовник су приказ ценовника, пића и хране, као и додавање пића и хране. Приликом приказа пића и хране постоје и опције за брисање истих из листе.



Слика 21 – Падајући мени за опције уређивања ценовника

Прозор који се отвара приликом клика на опцију „прикажи ценовник“, приказује табелу у којој се налазе сви производи који су предходно унети у датотеку „*cenovnik.txt*“.

Projekat - Restoran		
Naziv	Cena	Tip
coca cola	125	pice
spagete	670	hrana
burger	233	hrana
pica	750	hrana
limunska trava	185	pice
nazad		

**Слика 22 – Приказ артикала који се налазе у ценовнику**

Приликом избора опције „прикажи пиће“ сортирају се артикли из ценовника који одговарају одређеном типу, у овом случају пиће. Приликом приказа пића постоји и опција за брисање. Кликком на одређени напитака у табели, означимо пиће које желимо да обришемо, након клика на дугме „обриши пиће“ тај напитака се брише из табеле и текстуалне датотеке у којој се налазе сви артикли.

Projekat - Restoran	
Naziv	Cena
coca cola	125
šumunska trava	185
obrisi pice	nazad

### Слика 23 – Брисање пића из базе података

Изглед табеле након брисања „лимунске траве“ :

Naziv	Cena
coca cola	125

**Слика 24 – Брисање пића из базе података**

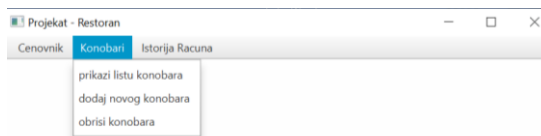
Још једна од опција јесте додавање артикала у ценовник. Битно је да ли додајемо пиће или храну, а након тога уносимо назив производа и цену. Након попуњавања тражених информација кликом на дугме „додај“ артикал се смешта у датотеку „*cenovnik.txt*“.



**Слика 25 – Додавање пића у базу података**

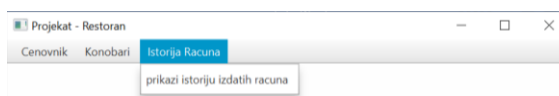


Конобари у падајућем менију има опције за приказ конобара, додавање и брисање конобара. Ове опције се раде по истом принципу као и све опције које постоје за ценовник. Сви подаци се додају у текстуалну датотеку „*osobe.txt*“.



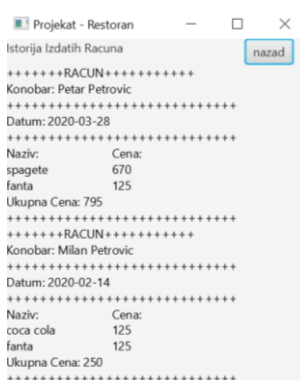
Слика 26 – Падајући мени за опције уређивања конобара

Историја рачуна у падајућем менију нуди опцију приказа листе рачуна који су издати у неком предходном периоду.



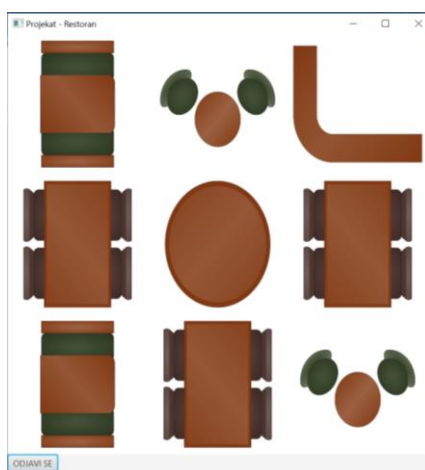
Слика 27 – Падајући мени за приказивање историје рачуна

Рачуни се учитавају из текстуалне датотеке „*racuni.txt*“. Прозор који се отвара приликом клика на ову опцију садржи ове рачуне и дугме „назад“, које омогућава повратак на предходну, тј менаџерску страницу.



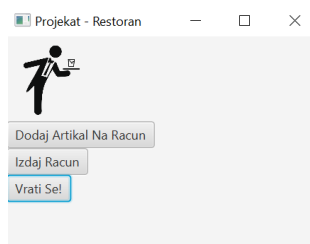
Слика 28 – Страница за приказ историје рачуна

Конобарски део апликације омогућава конобарима да куцају рачуне. Почетна страница приказује изглед ресторана и поставку столова у истом. Кликом на сто отвара се нови прозор из ког нам се отварају опције које су неопходне да би се искуцао рачун.



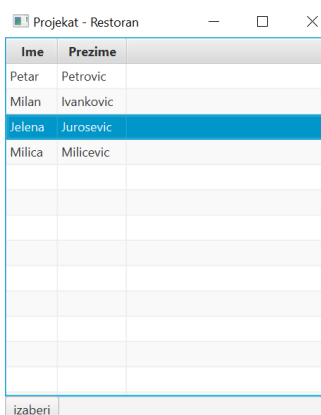
Слика 29 – Конобар страница

Ако конобар не жели да изда рачун али хоће да започне куцање рачуна за другим столом, потребно је да кликне на дугме „врати се“ и вратиће се на страницу са столовима, и отвориће се могућност за куцање нових рачун. Сто за којим је започето куцање рачуна остаће заузет, све док се не изда рачун.



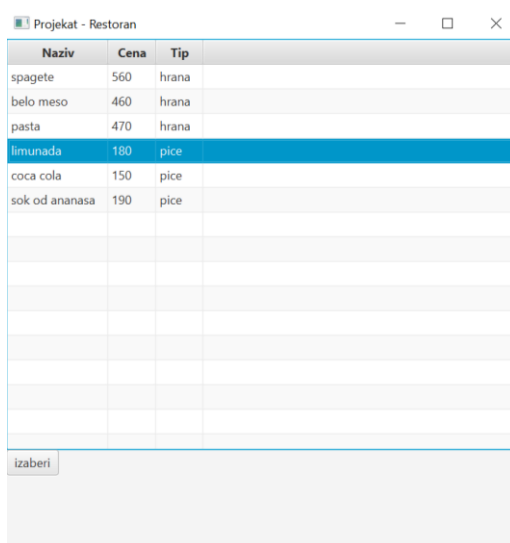
Слика 30 – Приказ странице за куцање рачуна

Приликом куцања рачуна, конобар бира своје име и презиме из листе понуђених. Како је у току једне смене могуће да више конобара ради у исто време, битно је да се зна ко издаје рачун у том моменту.



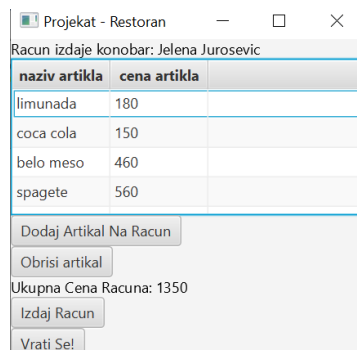
Слика 31 – Бирање конобара приликом куцања рачуна

Након бирања конобара, следи бирање артикала које гости поручују. Када год се на листу поручених артикала додаје нови, на рачуну се мења укупан износ, тј додаје се цена артикла који је поручен.



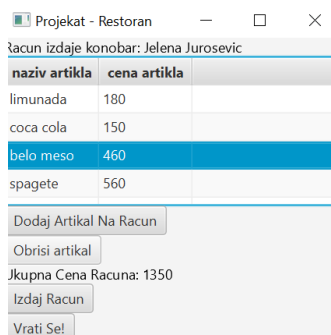
Слика 32 – Бирање артикала приликом куцања рачуна

Додавањем конобара и артикала, мења се изглед прозора. Имамо опције за брисање артикла, у случају грешке. Дугме за издавање рачуна, и повратак на конобарску страницу.



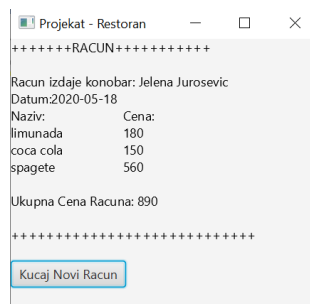
Слика 33 – Куцање рачуна

Брисање артикла са рачуна одвија се слично као и брисање артикала из ценовника. Кликом на артикал ми бирамо шта је потребно обрисати, и када желимо коначно да обришемо, притиснемо дугме „обриши артикал“. Након тога артикал се уклања из табеле, и укупна цена се смањује за цену уклоњеног артикла.



Слика 34 – Брисање артикла са рачуна

Када конобар жели да заврши са куцањем рачуна потребно је да кликне дугме „издај рачун“ и појавиће нам се изглед коначног рачуна. На њему се налази име и презиме конобара који је издао поруџбину, сви артикли који су поручени и њихове цене, као и укупна цена рачуна. Избором опције „куцај нови рачун“ отварају нам се поново опције за бирање конобара, артикала, као и дугме за повратак на предходну страницу, у случају да је потребно да се рачун куца за другим столом.



Слика 35 - Издавање рачуна

## 6. ЗАКЉУЧАК

Када смо прошли кроз основне концепте објектно оријентисаног програмирања и упознали се са апликацијом „Ресторан“ схватамо зашто је она рађена баш у *Јави*, тј језику који користи објектно оријентисану парадигму. Реализација апликације је тежња да се сам софтвер направи тако да представља модел реалног света, ово је уједно и основни концепт објектно оријентисаног програмирања. Олакшавајућа је сама чињеница да је овакав начин размишљања уједно и врло сличан људском, тј начину на који људи решавају реалне проблеме.

Подела апликације на два дела у зависности од права која корисник поседује, реализована су много лакше коришћењем објектно оријентисаног програмског језика. Све што је било потребно јесте тип у надкласи „*Osoba.java*“, који ће бити проверен приликом „пријаве“. Ово би било теже реализовати у програму који је рађен у неком функционалном програмском језику.

Особине објектно оријентисаног програмирања доста су помогле приликом имплементације самог пројекта. Битна ствар у објектно оријентисаном програмирању је што је све објекат. Програми представљају скуп објеката који међусобно задају послове једни другима преко слања порука. Тако ми остварујемо повезаност између класа „*App.java*“, „*Menadzer.java*“, „*Konobar.java*“.. Свака класа је ту да би проширила неку другу, да ли као артикли који ће бити део класе „*Cenovnik.java*“, или као класа „*Osoba.java*“ коју касније наслеђује „*Menadzer.java*“ што и јесте једна од битнијих функција које објектно оријентисано програмирање нуди. У овим примерима видимо повезаност свих класа, коју смо такође објаснили током разраде овог матурског рада.

Апликација „Ресторан“ нам је приказала основне концепте објектно оријентисаног програмирања, и то на практичан начин. Широка примена језика *Java* у ком је и рађен програм показала се као одличан избор за имплементацију овакве апликације.

## ЛИТЕРАТУРА

1. <https://wiki.openjdk.java.net/display/OpenJFX/Main>
2. <https://www.java.com/en/>
3. <https://cubes.edu.rs/sr/30/obuke-i-kursevi/sta-je-objektno-orijentisano-programiranje>
4. <http://www.oracle.com/technetwork/java/overview-141217.html>
5. <http://docs.oracle.com/javase/6/docs/technotes/guides/rmi/index.html>
6. <http://docs.oracle.com/javase/6/tutorial/doc/bnafd.html>
7. <http://docs.oracle.com/javase/6/tutorial/doc/bnagb.html>
8. <http://docs.oracle.com/javase/6/tutorial/doc/bnaph.html>
9. <https://developer.ibm.com/languages/java/>
10. <https://www.defmacro.org/2006/06/19/fp.html>
11. <https://curlie.org/Computers/Programming/Methodologies/Object-Oriented>
12. <http://java.sun.com/docs/books/tutorial/java/concepts/index.html>
13. <http://www.slideshare.net/rajeshlal/evolution-of-user-interface-26414802>
14. <https://web.archive.org/web/20040309225129/http://www.sitepoint.com/article/real-history-gui>

## Слике

Слика 1 - UML дијаграм апликације "Ресторан"	6
Слика 2 – Дијаграм објекат МЕНАѢЕР	7
Слика 3 – Функција „istorijaRacuna()“ у класи „Menadzer.java“	7
Слика 4 – Дијаграм објекат КОНОБАР	8
Слика 5 - Функција „obrisiKonobara ()“ у класи „Konobar.java“	8
Слика 6 – Функција „toString ()“ у класи „GotoviProizvodi.java“	9
Слика 7 – Функција „dodajUFajl ()“ у класи „GotoviProizvodi.java“	9
Слика 8 – Изглед датотеке „cenovnik.txt“	10
Слика 9 – Функција „dodajPice()“ у класи „Cenovnik.java“	10
Слика 10 – Функција „kucanjeRacuna()“ у класи „Racun.java“	11
Слика 11 – Функција „ispis()“ у класи „Racun.java“	11
Слика 12 – Функција „start()“ у класи „App.java“	12
Слика 13 – Провера путање слике у класи „App.java“	13
Слика 14 – Коначан приказ stage-а у класи „App.java“	13
Слика 15 – Функција „prozor()“ у класи „Konobar.java“	13
Слика 16 – Функција „prozorZaPrikazStolova()“ у класи „Konobar.java“	14
Слика 17 – променљиве типа MenuItem у класи „Menadzer.java“	14
Слика 18 – Опције из падајућег менија у класи „Menadzer.java“	14
Слика 19 – Главна страница	15
Слика 20 – МенаѢер страница	15
Слика 21 – Падајући мени за опције уређивања ценовника	15
Слика 22 – Приказ артикала који се налазе у ценовнику	16
Слика 23 – Брисање пића из базе података	16
Слика 24 – Брисање пића из базе података	16
Слика 25 – Додавање пића у базу података	16
Слика 26 – Падајући мени за опције уређивања конобара	17
Слика 27 – Падајући мени за приказивање историје рачуна	17
Слика 28 – Страница за приказ историје рачуна	17
Слика 29 – Конобар страница	17
Слика 30 – Приказ странице за куцање рачуна	18
Слика 31 – Бирање конобара приликом куцања рачуна	18
Слика 32 – Бирање артикала приликом куцања рачуна	18
Слика 33 – Куцање рачуна	19
Слика 34 – Брисање артикла са рачуна	19
Слика 35 - Издавање рачуна	19

## БИОГРАФИЈА

Душица Трбовић је рођена 20. августа 2002. године у Бачкој Тополи. У Црвенки завршава Основну школу „Вук Караџић“, 2017. године. Још кроз основну школу исказује интересовања према природним наукама. У седмом разредну, похађала је и Архимедесову математичку школу и успешно је завршила 2015/16. године. Учествовала је на многобројним такмичењима, и освајала преко 30 диплома на различитим нивоима. Хемија, 2017. године, 1. место на републичком такмичењу са истраживачким радом на тему „Испитивање квалитета воде у Великом Бачком Каналу“. Још само нека од такмичења на којима је учествовала су математика, физика, техничко образовање, и информатика. Одувек је привлачи технологија. Како у основној школи, поготово у мањој средини, нема много могућности за развијање, начин на који се тада изражавала је развој мултимедијалних презентација, на такмичењима попут „Енергија је свуда око нас“, „Филмић 2017.“ осваја друго, тј треће место на републичком нивоу. На међународном такмичењу „Дабар“ осваја места 3 године за редом, од којих једном сребро. Након 4 године, уз сугестије тадашњег разредног старешине, долази на идеју да упише „Смер за ученике са посебним способностима за рачунарство и информатику“ у Гимназији „Јован Јовановић Змај“ у Новом Саду, што јој и полази за руком. Тако у септембру 2017. године, са 15 година одлази за Нови Сад и наставља своје школовање. У периоду од 2017. до 2021. године борави у Средњошколском дому ученика „Бранково Коло“. Поред образовања 10 година се бавила каратеом. Када је положила за браон појас, прекинула је са тренирањем због пресељења у Нови Сад, али спорт је свакако један неизоставан део њеног живота. Иако релативно млада, има и радног искуства. Од четрнаесте године сваког лета ради у кафићу, а преко школске године држи часове математике и програмирања млађим ђацима. Воли контакт са људима и драго јој је када неке може да помогне. Такође има искуства и у изради сајтова, на пример Рукометном клубу „Црвенка“, када је ушао у Суперлигу Србије. У Гимназији „Јован Јовановић Змај“ потврђује своје интересовање за техничке науке, прецизније програмирање, и доноси одлуку да своје образовање жели да настави на Факултету техничких наука. Ово је још увек жеља, која би у скоријој будућности требала бити реализована.



Датум предаје: \_\_\_\_\_

Комисија:

Председник \_\_\_\_\_

Испитивач \_\_\_\_\_

Члан \_\_\_\_\_

Коментар:

Датум одбране: \_\_\_\_\_

Оцена \_\_\_\_\_ ( )