

# Izrada programa za nadzor strojeva u tiskari

---

**Karin, Marija**

**Undergraduate thesis / Završni rad**

**2015**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Graphic Arts / Sveučilište u Zagrebu, Grafički fakultet**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:216:685098>

*Rights / Prava:* [In copyright](#)/[Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2025-02-05**



*Repository / Repozitorij:*

[Faculty of Graphic Arts Repository](#)



SVEUČILIŠTE U ZAGREBU  
GRAFIČKI FAKULTET ZAGREB

# ZAVRŠNI RAD

Marija Karin

SVEUČILIŠTE U ZAGREBU  
GRAFIČKI FAKULTET ZAGREB

Smjer: Tehničko-tehnološki

# ZAVRŠNI RAD

IZRADA PROGRAMA ZA NADZOR STROJEVA U  
TISKARI

Mentor:  
prof. dr. sc. Klaudio Pap,

Student:  
Marija Karin

Zagreb, 2015.

**SAŽETAK:**

Izrada samostalnog programa koji bi se koristio kao pomoć u tiskarskim pogonima. Program bi omogućavao brz i jednostavan nadzor rada tiskarskih strojeva u trenutku upita, za od korisnika definirani period rada. Program bi prikupljao podatke koje bi strojevi zapisivali na određenoj lokaciji i u određenim ciklusima. Implementacija će biti napravljena pomoću programskog jezika Java. Program funkcionalno predstavlja implementaciju dijela sustava kojim upravlja korisnik, a dio je kompleksnog sustava koji se u cijelosti sastoji od strojeva (koji upisuju podatke na mjesto pohrane-baza podataka) te programa koji služi za prikaz tih podataka na jednostavan i korisniku prihvatljiv način. Program neće zapisivati podatke već će samo učitavati potrebne podatke.

**Ključne riječi:**

- nadzor strojeva, program, grafičko sučelje

**ABSTRACT:**

Standalone program that will be used in printing offices. The program will enable quick and easy supervision of the current status of the machines in that moment. Program will collect data that are written by machines on specified location in specified cycles. Implementation will be performed with Java programming language. The functionality of the program represents a implementation which is a part of bigger system which is managed by the user and it is a part of a complex system that is completely made out of machines (that are writing data in data base) and program that is used for showing that data in simple and easy way. Program will not write data but load them.

**Key words:**

- machine supervision, program, graphical interface

# SADRŽAJ

1. UVOD .....	1
2. TEORIJSKI DIO.....	3
2.1 Java .....	3
2.2 Biblioteka Swing.....	6
2.3 JSON .....	7
3. EKSPERIMENTALNI DIO.....	9
3.1 <i>Login</i> prozor .....	9
3.2 Tablica s popisom raspoloživih strojeva.....	18
3.3 Korisnički prozori za strojeve .....	27
3.4 <i>JSON Parsing</i> .....	31
3.5 Dijagram klasa .....	38
4. ZAKLJUČAK.....	39
5. LITERATURA .....	41

# 1. UVOD

Mnoge tiskare, posebno one velike imaju probleme s organizacijom posla. Organizacija zahtijeva znanje i iskustvo. Dobra organizacija je nužna za uspješnost bilo koje tvrtke. Tiskarski pogoni mogu biti rasprostranjeni na velikim površinama, pa je dobra organizacija bez dobre komunikacije nemoguća. Koliko je bitna komunikacija između zaposlenika, toliko je bitna i komunikacija na relaciji zaposlenik-stroj. Zaposlenik u pogonu u svakom trenutku mora znati u kojoj je fazi posao koji se trenutno obavlja. Također je i bitno da zaposlenici u pripremi znaju koje strojeve imaju na raspolaganju da bi mogli rasporediti naručene poslove sa što manje troškova i gubitaka. U cilju je maksimalno iskoristiti svaki stroj sa što kraćim vremenom između poslova. Da bi se olakšala ta komunikacija, neophodno je imati računalne programe koji bi omogućavali jednostavan i brz pregled, bez velikih troškova dodatne edukacije zaposlenika. Takvi software-i zahtijevaju velika ulaganja koja si ne mogu priuštiti sve tvrtke, posebno one male. U ovom radu će biti opisan jednostavan program za nadzor strojeva koji bi omogućio jednostavno praćenje proizvodnje.

U prvom poglavlju biti će opisane korištene tehnologije u eksperimentalnom dijelu rada. Eksperimentalni dio se bazira na objektno-orijentiranom programiranju stoga će najviše biti riječi o Java programiranju. Obuhvatit će se i Biblioteka Swing kao i JSON.

U prvom potpoglavlju će biti opisana Java. Njezin nastanak, njeni kreatori, kome je namijenjena kao i mjesto koje trenutno zauzima u svijetu programiranja. Biti će objašnjeno što točno znači objektno programirati. Važno je napomenuti i od čega se Java program sastoji i o tome će biti riječi. Opisati će se i način na koji radi konstruktor u Javi i JAR datoteka (Java arhiv).

U drugom potpoglavlju opisati će se ukratko Biblioteka Swing koja je korištena za grafičko sučelje programa.

To je posebni dodatak Javi, *plug-in* kojega je potrebno dodati u Java okruženje. Biti će riječi i o tome što sve Swing nudi i kako se razvijaju korisnička sučelja. U trećem potpoglavlju opisan će biti JSON format zapisa, što je i na čemu se temelji. Navedeni će biti koji su to JSON elementi i vrijednosti u JSON dokumentu.

U drugom poglavlju biti će opisan sami eksperimentalni dio rada.

Dokumentacija se sastoji od detaljnog opisa kodova kao i primjera kodova i slika samog programa. Program se sastoji od tri prozora, a svaki od tih prozora će biti opisan zasebno u potpoglavlju.

u prvom potpoglavlju biti će opisan prvi korisnički prozor koji se otvara prilikom pokretanja programa. To je *Login* prozor, služi za autorizaciju korisnika i opisati će se na koji način daje korisniku dopuštenje da uđe u prozor s tablicom strojeva. Opisane će biti sve opcije i gumbi na tom prozoru drugo potpoglavlje posvećeno je opisu tablice sa strojevima i *Layoutu* tablice. Biti će ukratko pojašnjeno i način na koji radi `JTable` metoda i način na koji se iscrtavaju ćelije u tablici. Pošto se i u ovom prozoru učitavaju podatci iz JSON datoteka biti će objašnjeno kako je postignuto da se klikom na svaki određeni stroj prikažu podatci za baš taj stroj.

Treće potpoglavlje opisuje korisnički prozor s detaljima o stroju. Podatci su ponovno učitani iz JSON datoteka, a linije koda će jasno prikazati stvorene objekte.

U potpoglavlju 3.4 objasniti će se metode koje će se koristiti za učitavanje JSON podataka, a zadnje potpoglavlje (3.5) prikazati će dijagram klasa koji će prikazati odnose između klasa unutar programa.

Na kraju rada biti će izveden zaključak i kritički osvrt eksperimentalnog dijela rada.

## 2. TEORIJSKI DIO

Potrebno je osvrnuti se na tehnologije koje su korištene za izradu programa. Java je objektno orijentirani programski jezik, bitno je to naglasiti i pobliže objasniti da bi se shvatilo objektno programiranje u cijelosti. Za vizualni prikaz potrebno je koristiti razne dodatke Javi (*plug-in*) koji se posebno instaliraju u Eclipse program koji je specijaliziran za Java programiranje. Java učitava (parsira, *eng.* parsing) datoteke raznih formata, a tu će biti prikazano učitavanje datoteka JSON formata.

### 2.1 Java

Programski jezik Java dizajnirao je James Gosling 1995. godine dok je radio u tvrtki Sun Microsystems (trenutno je ona podružnica tvrtke Oracle Corporation). Cilj je bio pružiti jednostavniju alternativu jeziku C++ koja će biti neovisna o platformi.

Java je programski jezik opće namjene koji se koristi u svim industrijama i za gotovo sve vrste aplikacija. Svladavajući ga, šanse za zaposlenjem kao razvojni inženjer bit će veće nego kod specijaliziranja samo jednog programskog jezika posebne namjene.

U svijetu trenutno postoji oko šest mil. profesionalnih java programera i većina njih je spremna prenositi svoje znanje putem raznih društvenih mreža. Ukoliko se prilikom programiranja naiđe na problem, vrlo je vjerojatno da se do rješenja može doći putem Interneta.

Budući da je zajednica Java programera ogromna, voditelji projekata u malim i velikim tvrtkama vole koristiti Javu za razvoj novih projekata.



Ne samo da je Java programski jezik otvorenog izvornog koda, već postoje i milijuni projekata otvorenog izvornog koda koji se razvijaju u Javi. Pridruživanje nekom od njih najbolji je način da se upoznavanja s procesom razvoja aplikacija.

Programski jezik Java je objektno orijentiran, što omogućava lako povezivanje programske konstrukcije s objektima iz stvarnog svijeta.

Java zahtijeva da se izvorni kôd prije izvršavanja kompajlira. On se pretvara u kôd specifičan za procesor ili bajtkod koji mogu razumjeti neki izvršni ili virtualni strojevi. Java kompajler neće samo provjeriti postoje li u kodu sintaktičke pogreške, već se nakon kompajliranja programu mogu dodati (pozvati) neke druge biblioteke Java koda.

Tehnički gledano, izvorni kôd Java programa može se napisati u bilo kojem editoru teksta (Notepad, TextEdit, vi, itd.), ali će za kompajliranje biti potrebni dodatni alati i biblioteke koda uključene u Java Development Kit (JDK).

Ukoliko se željeno računalo želi koristiti za razvoj Java programa, potrebno je preuzeti i instalirati JDK, a ukoliko je namjena tog računala samo za izvršavanje Java programa (kompajliranje na drugom računalu) tada je potrebno samo izvršno okruženje Java Runtime Environment (JRE) [1].

Java program sastoji se od klasa (*eng.* class). Klasa je skup podataka pohranjenih u poljima kojima je pridodano ime i kodova koji su posloženi unutar imenovanih metoda koje rade s tim podacima. Polja i metode se nazivaju članovima (*eng.* members) klasa. Također, klase mogu unutar sebe sadržavati i druge klase. Članovi klasa dolaze u dva posebna tipa: članovi klasa, statični, povezani su sa samom klasom dok su članovi instance povezani s individualnim instancama klase (npr. objektima).

Postoje četiri tipa članova:

- Polje klasa
- Polje metoda
- Polje instanci
- Metode instanci

Svaka klasa u Javi ima barem jedan konstruktor (*eng.* constructor) koji je metoda koja ima isto ime kao klasa i čija je svrha provoditi potrebne inicijalizacije za nove objekte. Ukoliko se ne zada koji će se konstruktor koristiti, Java će postaviti uobičajeni konstruktor koji ne prima nikakve argumente i ne izvodi posebne inicijalizacije.

Konstruktor radi na ovaj način. Operacija `new` kreira novu neinicijaliziranu instancu klase. Tada je pozvana metoda konstruktora i objekt se implicitno predaje, kao i bilo koji argument unutar okruglih zagrada. Konstruktor može koristiti te argumente za bilo koju potrebnu inicijalizaciju [2].

Java ne bi bila tu gdje je danas bez stvaranja njenog formata arhivskih datoteka. Java arhiv, kojeg programeri obično nazivaju JAR datoteka, je način za grupiranje više datoteka, uključujući i druge JAR-ove, u samo jednu datoteku s nastavkom imena `.jar`. JAR datoteke koriste isti format za sažimanje datoteka kao i zip format. Dakle, može se otvoriti JAR datoteku u programu koji razumije obično zip sažimanje i obrađivati je dalje. To čini format JAR datoteka vrlo prenosivim na mnoge različite operacijske sustave jer ih većina razumije zip format ili ima pomoćne programe koji su stvoreni da za njih rade sa zip datotekama. JAR datoteke mogu uvelike skratiti vrijeme preuzimanja klasa, slika, audio, i drugih velikih datoteka njihovim sažimanjem. Apleti i njihovi resursi mogu biti sažeti u JAR datoteku, što znatno skraćuje vrijeme za preuzimanje apleta [3].

## 2.2 Biblioteka Swing

Java je na početku nudila vrlo jednostavnu osnovnu biblioteku klasa za korisničko sučelje pod imenom Abstract Toolkit (AWT). Nakon nekoliko godina predstavljen je novi paket alat za programčiće Swing. On nudi lakši skup komponenata grafičkog korisničkog sučelja i pritom ne mijenja osnovnu ideju – razvoj korisničkog sučelja odvojeno od specifičnosti operativnog sustava konačnog korisnika. Programeri danas pokušavaju razvijati korisnička sučelja koja izgledaju kao i operativni sustav, neovisno o tome je li riječ o Windowsima, Mac OS-u, iOS-u ili Androidu.

Swing nudi sve što je potrebno za razvoj korisničkih sučelja u Javi: kontrole za predstavljanje gumba, padajućih izbornika, mreža, vrpce za pomicanje sadržaja, stabala, mapa i tako dalje. Korisnička sučelja se razvijaju kombiniranjem kontrola u kontejnerima (kao što je `JPanel`) s predlošcima koji omogućavaju raspoređivanje kontrola na način na koje su zamišljene.

Swing klase nalaze se u paketu `javax.swing` a proces razvoja korisničkih sučelja svodi se na proširivanje nekih klasa tako da prikazuju korisničko sučelje i reagiraju na događaje koje generiraju korisnik ili sustav [1].

## 2.3 JSON

(JavaScript Object Notation) jednostavni format zapisa koji služi za razmjenu informacija. Za ljude je lako čitljiv i jednostavan je za pisanje. Za strojeve je lagan za interpretiranje i generiranje. Temelji se na podskupu JavaScript programskog jezika, Standard ECMA-263 za 3. izdanje – Prosinac 1999. JSON je tekstualni format neovisan o jeziku, ali koristi konvencije slične programima C-obitelji, uključujući C, C++, C#, Java, JavaScript, Perl, Python, itd. Ove karakteristike čine JSON idealnim zapisom za razmjenu podataka.

JSON se temelji na dvije strukture:

- Skup parova ime/vrijednost. U različitim jezicima to predstavlja različite oblike podatkovnih struktura kao npr.: objekt, zapis (*eng. record*), struktura, rječnik (*eng. dictionary*), tablica slučajnog odabira (*eng. hash table*), lista ključnih pojmova (*eng. keyed list*), ili asocijativno polje (*eng. associative array*).
- Poredane liste vrijednosti. U većini jezika, ovo je implementirano kao niz (*eng. array*), vektor, lista, ili sekvenca.

To su univerzalne podatkovne strukture. Generalno gledajući, podržavaju ih svi moderni programski jezici na ovaj ili onaj način. Razumljivo je, dakle, da format koji služi za razmjenu podataka također bude temeljen na ovim strukturama.

JSON elementi su:

*Objekt* je neuređen skup parova ime/vrijednost. *Objekt* počinje { (lijevom vitičastom zagradom) i završava } (desnom vitičastom zagradom). Iza svakog imena slijedi : (dvotočka), a parovi ime/vrijednost su odvojeni , (zarezom).

*Niz* je uređen skup vrijednosti. *Niz* počinje [ (lijevom uglatom zagradom) i završava ] (desnom uglatom zagradom). Vrijednosti su odvojene , (zarezom).

*Vrijednost* može biti *string* pod dvostrukim navodnicima, broj, *true* (istinita vrijednost), *false* (neistinita vrijednost) ili *null* (ništa), objekt ili niz. Ove strukture mogu biti ugniježdene.

Vrijednosti u JSON dokumentu:

*String* je redoslijed nula ili više Unicode znakova pod dvostrukim navodnicima, koristeći obrnutu kosu crtu (\) za isključivanje posebnih znakova.

Znak se tumači kao *string* od jednog znaka. *String* je veoma nalik *stringu* korištenom u C ili Javi.

*Broj* je veoma nalik broju korištenom u C ili Javi, s iznimkom da se ne koriste oktalni i heksadecimalni zapisi brojeva.

*Praznina* može biti umetnuta između bilo kojeg para *tokena*. Izuzev nekoliko pojedinosti kodiranja koje u potpunosti opisuju jezik [4].

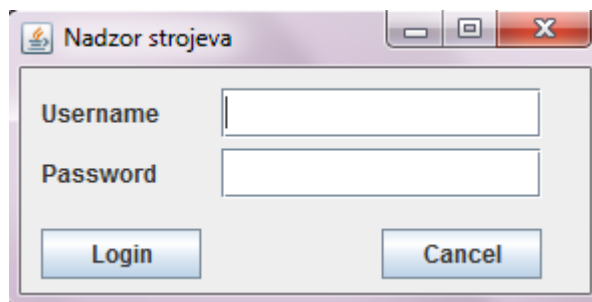
## 3. EKSPERIMENTALNI DIO

Program se sastoji od tri korisnička prozora: *Login* prozora, tablice s prikazom raspoloživih strojeva i prozora s detaljima o traženom stroju. *Login* prozor je ulazni prozor koji služi za autorizaciju korisnika, unošenjem ispravnih korisničkih podataka (prethodno kreiranih od strane administratora) korisnik ima dopuštenje dobiti uvid popis strojeva u tiskari. Ukoliko korisnik zatraži, može dobiti i detaljniji uvid svakog stroja zasebno i može ih međusobno pratiti pošto se prozori s prikazima detalja za svaki stroj međusobno ne isključuju.

### 3.1 *Login* prozor

*Login* prozor služi za autentifikaciju korisnika. Korisnik unosi svoje korisničko ime i lozinku koju je postavio preko administratora. Program ne nudi mogućnost upravljanja lozinkom ili mijenjanja korisničkog imena već isključivo za odobravanje pristupa podacima. Moguće je postaviti više korisnika koji se mogu služiti programom.

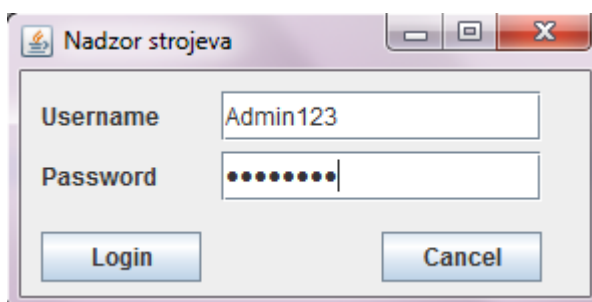
Prozor sadrži polje *Username* za upisivanje korisničkog imena i polje *Password* koje služi za upisivanje lozinke važeće za to korisničko ime. Slika 1 prikazuje prozor za prijavu korisnika. Na slici se također mogu vidjeti polje za upisivanje korisničkog imena i polje koje služi za upisivanje važeće lozinke.



Slika 1 Prozor za prijavu korisnika

Ispod polja za upisivanje korisničkog imena i lozinke nalaze se gumbi *Login* i *Cancel*. *Login* gumb služi za potvrđivanje unešenih podataka za autentifikaciju, a gumb *Cancel* služi za otkazivanje cijelog prozora i na njegov pritisak prozor se gasi.

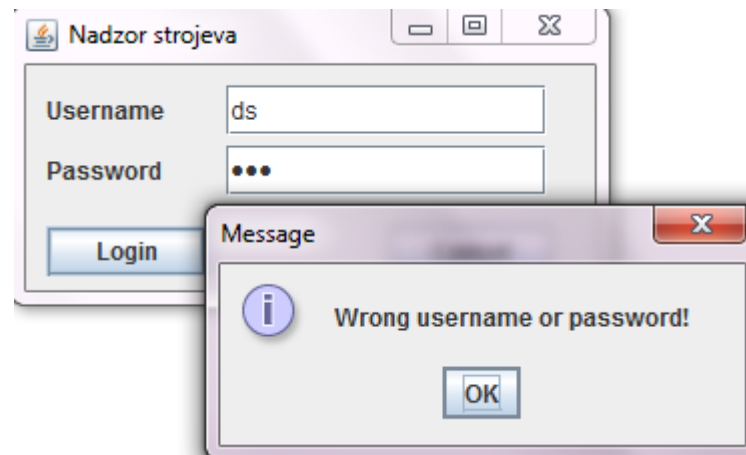
Unošenjem odgovarajućeg korisničkog imena i lozinke korisnik pritišće gumb *Login* i odobrava mu se pristup podacima, a log in prozor se gasi. Odmah nakon gašenja prozora, otvara se prozor s tablicom strojeva koji su na raspolaganju toj tvrtci. Ispravno unešeno korisničko ime i lozinka su vidljivi na slici 2, a još nisu pritisnuti *Login* ili *Cancel* gumbi.



Slika 2 Ispravno korisničko ime i lozinka

Ukoliko korisnik unese neodgovarajuće korisničko ime ili lozinku, program će tada upozoriti korisnika na jednu od te dvije mogućnosti i neće mu dozvoliti ulazak u tablicu. Na slici 3 je prikazana obavijest koju je program prikazao uslijed krivo unešene lozinke ili korisničkog imena.

Pristup tablici sa strojevima je odbijen, a poruka prikazana u prozoru *Message* glasi „Wrong username or password!“. Program ne detektira o kojoj je grešci riječ već samo obavještava da nisu zadovoljeni svi uvijeti, a to su: ispravno korisničko ime i lozinka.



Slika 3 Obavijest programa o krivo unešenim podacima

Ovaj prozor ima sadržava `main` metodu što znači da će se pri pokretanju programa on pokrenuti prvi. Kada je riječ o objektnom programiranju, potrebno je prvo objekte kreirati. Kreirani objekti se tada mogu upotrebljavati u akcijama. Slika 4 prikazuje `main` metodu koja je dodijeljena *Login* prozoru, jer se on pokreće prvi.

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

Slika 4 `main` metoda dodijeljena prozoru koji se prvi pokreće

U Javi je potrebno imati `main` metodu u barem jednoj klasi [5]. U toj klasi se kreiraju objekti koji će se koristiti tijekom izvođenja programa. Ona je metoda klasa i ona kao argumente prima parametre iz komande linije.



Za izradu ovog prozora je bilo potrebno koristiti nekoliko Java objekata. Za kreiranje strukture koja će sadržavati kontrole korisničkog sučelja korišten je `JFrame` objekt. `JFrame` najčešće sadrži još jedan kontejner u kojem su smještene kontrole poput `JBUTTON`, `JTABLE` i `JLIST`[1].

Primjer toka posla za izradu jednog `JFrame` koji sadrži `JPanel` je sljedeći:

1. Izrada `JPanel`.
2. Pridruživanje upravljača rasporeda (*layout manager*).
3. Instanciranje neke Swing kontrole i dodavanje njih na paletu.
4. Dodavanje palete u vršni kontejner – `JFrame` – pozivanjem `setContentPane()`.
5. Zadavanje veličine okvira i uključivanje njegove vidljivosti[1].

Klasa `JFrame` se u Java programima koristi za konstrukciju prozora najviše razine s rubom i naslovom, kao i za stvaranje gumba za smanjivanje, uvećavanje i zatvaranje prozora[3]. Slika 5 prikazuje kreirani `JFrame` koji sadrži `JPanel`.

```
19 public static void main(String[] args) {
20     frame = new JFrame("Nadzor strojeva");
21     frame.setSize(300, 150);
22     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23
24     JPanel panel = new JPanel();
25     frame.add(panel);
26     placeComponents(panel);
27
28     frame.setVisible(true);
29
30 }
```

Slika 5 Kreirani `JFrame` koji sadrži `JPanel`

`JFrame`-u je pridodano ime kao i veličina prozora. Postavljen je tako da se pritiskom na gumb za zatvaranje u gornjem desnom kutu cijeli program zatvori.

To kontrolira `setDefaultCloseOperation` metoda sa `EXIT_ON_CLOSE` operacijom. To je prikazano na slici 6.

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

**Slika 6** Prikaz `setDefaultCloseOperation` metode

Kreiranjem `JLabel` objekta postavljena su *Username* i *Password* polja. `TextField` objektom postavljeno je polje upisivanja korisničkog imena, a `PasswordField` polje upisivanja lozinke. `TextField` i `PasswordField` su jedna od elemenata Swing korisničkog sučelja. Postoje još i drugi elementi: `JLabel`, `Button`, `JToggleButton`, `JCheckBox`, `JTable`, itd. Za *Login* i *Cancel* gumbе korišteni su `Button` objekti, za svaki gumb po jedan.

Svi ti objekti su stvoreni unutar `JPanel`. `JPanel` klasa je kontejner komponenti. Paneli uobičajeno ne dodaju nikakve osobine, pa ni boju osim boje njihove pozadine, ali jednostavno se mogu dodati granice (eng. *borders*) i tako na drugi način urediti njihovo obojenje. U velikom broju slučajeva, paneli su neprozirni (eng. *opaque*). Neprozirni paneli služe kao dobri okviri za sadržaje i uspješno mogu pomoći s obojenjem. Moguće je promijeniti transparentciju panela pozivanjem `setOpaque` metode. Transparentni panel ne iscrtava pozadinu, pa se svaka komponenta može vidjeti kroz njega [6].

Objektima su pridodane osnovne karakteristike kao što su veličina i pozicija. Nakon što je svaki objekt stvoren, metodom `panel.addComponent()` je on i dodan u panel kojem nije postavljen raspored (eng. *Layout*). Upravo zbog toga što raspored nije postavljen, nužno je bilo ručno odrediti pozicije pojedinog objekta. Slika 7 prikazuje upravljač rasporeda `FlowLayout`, koji raspoređuje vodoravno sve komponente dodane u kontejner.

Kada više nema mjesta za komponentu, `FlowLayout` prelazi u sljedeći red i nastavlja vodoravno dodavati komponente.

Vrijednost `null`, koja je prikazana na slici 7, nije valjana instanca objekta, dakle nije joj dodijeljena memorija. To je jednostavno vrijednost koja pokazuje da se referenca objekta trenutno ne referencira na objekt [7].

Tom vrijednošću, u ovom slučaju, nije dodijeljen nikakav uobičajeni raspored.

```
34     panel.setLayout(null);  
35
```

**Slika 7** Metoda `setLayout` kojom je panelu zadano ručno podešavanje rasporeda objekata

Slika 8 prikazuje primjer korištenja metode `panel.add`. kada se dodaju komponente na paletu, koristi se `add` metoda. U ovom slučaju, ta metoda dodaje `TextField` na paletu. *Panel* je argument koji je specificiran za ovu metodu.

```
40     final JTextField userText = new JTextField(20);  
41     userText.setBounds(100, 10, 160, 25);  
42     panel.add(userText);
```

**Slika 8** Primjer korištenja metode `panel.add`

Nakon što su stvoreni svi objekti potrebni za ovaj program, njima se dodjeljuju funkcionalnosti. Upravo je na taj način dodijeljena funkcionalnost `loginButton` objektu. Kada se *Login* gumb klikne tada se metodom `JsonParsing` dohvaćaju podatci iz JSON datoteke i provjeravaju s unešenim vrijednostima.

`ActionListener` je opisan ovako: Sučelje slušatelja za primanje događaja. Klasa koja je zainteresirana za obradu događaja implementira sučelje, a objekt izrađen tom s tom klasom registriran je komponentom korištenjem metode `addActionListener` te komponente. Kada se događaj odigra, poziva se metoda `actionPerformed` tog objekta. Ako se događaj odigra, virtualni stroj je taj koji poziva metodu `actionPerformed()` [1].

Slika 9 prikazuje dodijeljivanje funkcionalnosti *Login* gumbu. Prikazuje i metodu `JsonParsing` kojom se dohvaćaju podatci iz JSON datoteka, a JSON format je jednostavan za učitavanje i generiranje.

```
 JButton loginButton = new JButton("Login");
 loginButton.setBounds(10, 80, 80, 25);
 panel.add(loginButton);

 // Definiranje funkcionalnosti kada se klikne login gumb
 loginButton.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        //Dohvaćanje podataka iz JSON-a i provjera s unesenim podacima
        JsonParsing jsonParsing = new JsonParsing();

        Map<String, String> values = jsonParsing.getUserData();
```

**Slika 9** Dodijeljivanje funkcionalnosti *Login* gumbu

*If* izrazom postignuto je to da se otvori prozor tablice ukoliko se unesene vrijednosti poklapaju s onima iz JSON datoteke, a ukoliko se ne poklapaju pokreće se *else* funkcija u izrazu i program izbacuje upozoravajuću poruku. Slika 10 prikazuje korišten *if* izraz u korisničkom prozoru *Login*.

```

// Definiranje funkcionalnosti kada se klikne login gumb
loginButton.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {

        //Dohvaćanje podataka iz JSON-a i provjera s unesenim podacima
        JsonParsing jsonParsing = new JsonParsing();

        Map<String, String> values = jsonParsing.getUserData();

        if(values.get("username").equals(userText.getText())
            && values.get("password").equals(String.valueOf(passwordText.getPassword()))){
            //Otvaranje prozora tablice
            new Table();

            frame.setVisible(false);
        } else {
            JOptionPane.showMessageDialog(frame, "Wrong username or password!");
        }
    }
}

```

**Slika 10** *If* izraz korišten u prozoru *Login*

*If* izraz kaže računalu da izabere jedan od dvaju različitih tokova izvršavanja programa, u ovisnosti o vrijednosti zadanog logičkog izraza. *If* izraz grananja ili odlučivanja ima oblik:

```

If - izraz
else – izraz

```

Računalo pri izvršavanju *if* izraza procjenjuje logički izraz koji vraća vrijednost *true* ili *false*. Ako je vrijednost *true*, računalo izvršava prvi izraz, a preskače izraz nakon, "else". Ako je vrijednost *if* izraza *false*, računalo preskače prvi izraz i izvršava drugi. U svakom slučaju, samo jedan od tih dvaju izraza unutar *if* izraza će biti izvršen. Dva izraza predstavljaju alternativne tokove programa; računalo se odlučuje za jedan od ovih tokova programa na osnovi vrijednosti logičkog izraza.

Jedna od mogućnosti primjene ove naredbe je odlučivanje samo da li će neka naredba biti izvršena ili ne. U tom slučaju *if* izraz nema *else* dio [8].

Na kraju je dodana i funkcionalnost *Cancel* gumbu. Klikom na njega prozor se zatvara, a s njime i cijeli program. Metodom `addAction` se dodaje željena funkcija tj. akcija gumbu. Slika 11 prikazuje programski kod kojim je dobivena funkcionalnost *Cancel* gumba. Tu su također dodane metoda `addActionListener` i metoda `actionPerformed` kojom se izvršava zatvaranje prozora i gašenje programa.

```
        frame.setVisible(false);
    } else {
        JOptionPane.showMessageDialog(frame, "Wrong username or password!");
    }
}
});|
JButton cancelButton = new JButton("Cancel");
cancelButton.setBounds(180, 80, 80, 25);
panel.add(cancelButton);

//Definiranje funkcionalnosti kada se klikne cancel gumb
cancelButton.addActionListener(new ActionListener() {

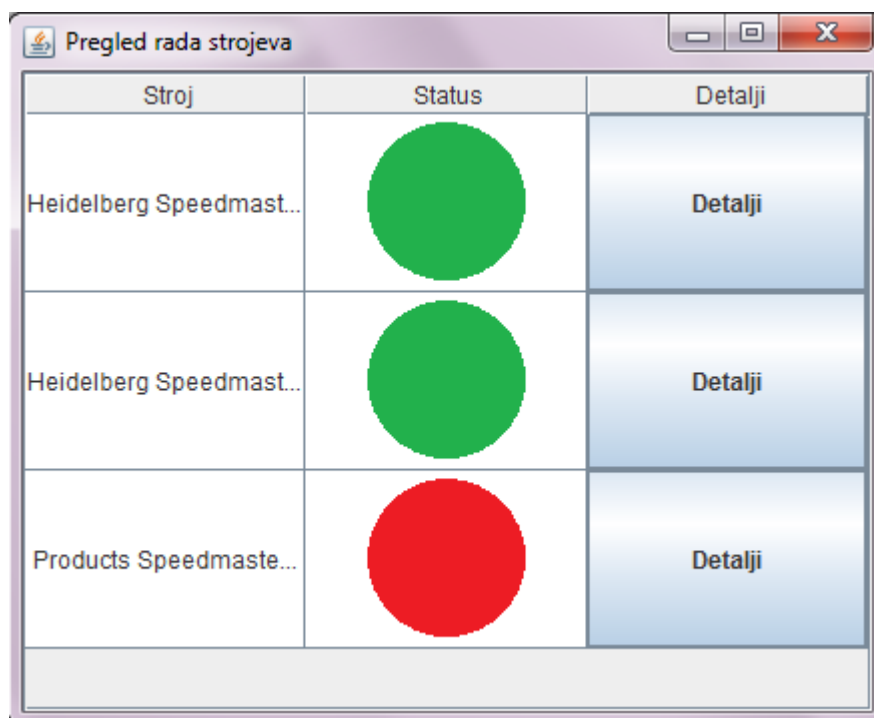
    @Override
    public void actionPerformed(ActionEvent e) {
        //Zatvaramo prozor i gasimo program
        frame.setVisible(false);
        frame.dispose();
    }
});
}
```




**Slika 11** Korištenje metode `addActionListener` i `actionPerformed` na gumbu *Cancel*

### 3.2 Tablica s popisom raspoloživih strojeva

Prozor koji se prikazuje nakon *Login* prozora je zapravo tablica s popisom strojeva koji su u pogonu. Daje jasan pregled trenutnog stanja strojeva s mogućnošću detaljnijeg pregleda stanja svakog pojedinog stroja. Tablica opisuje tri karakteristike stroja: ime stroja, status i detalje.

Slika 12 prikazuje korisnički prozor s tablicom raspoloživih strojeva u tiskari. Prikazane su tri rubrike: rubrika za ime stroja, za status rada stroja i detalji. Tablica prikazuje da stroj Products Speedmaster CX 102 trenutno nije u funkciji, jer status pokazuje crveni znak.



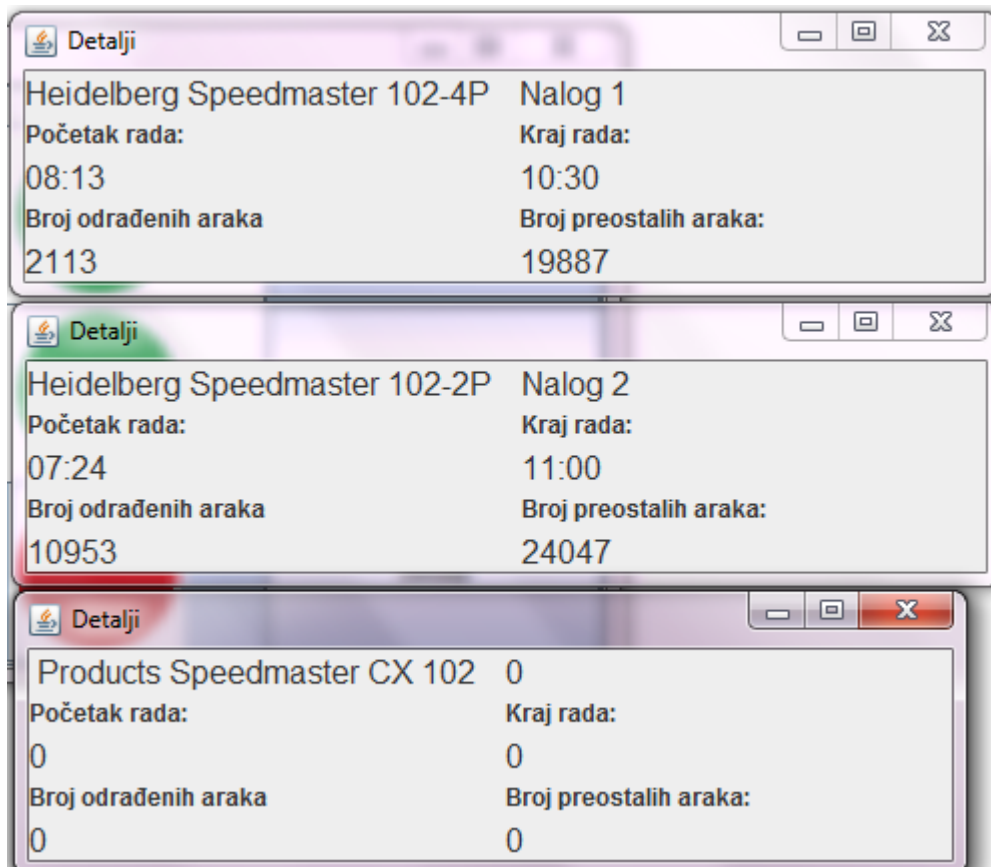
Stroj	Status	Detalji
Heidelberg Speedmast...		Detalji
Heidelberg Speedmast...		Detalji
Products Speedmaste...		Detalji

Slika 12 Tablica s popisom raspoloživih strojeva

Status stroja daje informaciju da li je stroj u pogonu (zelena lampica) ili izvan pogona (crvena lampica). Informacija koju daje program je obavještajna, program neće ustanoviti koji je razlog zbog kojega je došlo do eventualnog zastoja.

Rubrika za detalje je zapravo gumb na čiji se pritisak otvara novi prozor u kojem je prikazano stanje zabilježeno u trenutku kada je gumb stisnut tj. naredba izvršena. Dok je taj prozor otvoren u pozadini i dalje je otvoren prozor s tablicom, a moguće je imati otvoreno i nekoliko prozora s detaljima u isto vrijeme jer se oni međusobno ne isključuju.

Slika 13 prikazuje istovremeno otvorene sve prozore s detaljima. Međusobno se ne isključuju, pa je moguće otvoriti istovremeno sve prozore s detaljima.

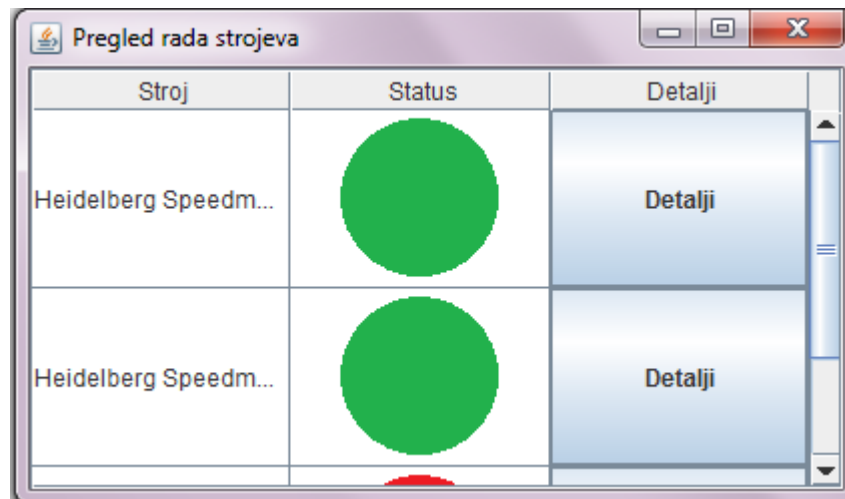


Slika 13 Prikaz svih istovremeno otvorenih prozora s detaljima



Kao i za *Login* prozor tako je i za ovaj korišten `JFrame`, ali u kombinaciji sa `JTable` kreatorom. Prozoru je dodana i opcija `JScrollPane` koja omogućava klizač sa strane za lakši pregled velikog popisa atributa.

Slika 14 prikazuje tablicu s popisom strojeva s klizačem. Dakle, ukoliko tablica ima više strojeva koje tablica ne može prikazati, moguće je koristiti klizač sa desne strane prozora za lakši pregled.



Slika 14 Tablica s klizačem

Podaci koji su prikazani u tablici su učitani iz vanjske datoteke koja predstavlja podatke koje je stroj zapisao i koje se na zahtjev prikazuju. Te datoteke su spremljene u JSON formatu koji je lako čitljiv svim programskim jezicima. Za prikazivanje lampica radnog stanja stroja upotrebljene su slike .png formata, a za gumb detalji `JButton` objekt.

Slike prvo moraju biti uvedene u klasu, od njih se mora prvo stvoriti objekt s kojim će se kasnije manipulirati.

Na slici 15 prikazan je programski kod za pozivanje slika zelenog i crvenog gumba koji prikazuju status rada stroja u tom trenutku.

```

public ImageIcon okImage = new ImageIcon("Slike\\zeleni_gumb.png");
public ImageIcon nokImage = new ImageIcon("Slike\\crveni_gumb.png");

```

**Slika 15** Programski kod za pozivanje slika

Ova klasa je deklarirana kao `public`, u tom slučaju klasa je vidljiva svim klasama posvuda. Ukoliko klasa nema uređivač (eng. *modifier*) (uobičajeni, također poznat kao `private`), tada je klasa vidljiva unutar svojih paketa (paketi su imenovane grupe povezane s klasom).

Slika 16 prikazuje programski kod u kojem su dodani objekti `JFrame`, `JTable` i `JScrollPane`. Visinu određuje veličina slike, a to je definirano `setRowHeight()` metodom. `setRowHeight` metoda ima `table` kao argument.

```

public List<Map<String,String>> machinePropertyList = new ArrayList<Map<String, String>>();
public Object[][] jsonMainData;
public Table(){
    JFrame frame = new JFrame("Pregled rada strojeva");
    JTable table = new JTable(new MyTableModel());
    table.getColumnModel().getColumn(2).setCellRenderer(new ButtonRenderer());
    table.getColumnModel().getColumn(2).setCellEditor(new ButtonEditor(new JCheckBox()));
    table.setRowHeight(okImage.getIconHeight());
    JScrollPane pane = new JScrollPane(table);
    frame.getContentPane().add(pane);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
    frame.pack();
}

```

**Slika 16** Prikaz programskog koda za kreiranje tablice

Tablice kreirane u Swingu nisu kreirane na uobičajen način. Radi performansi tablice su implementirane drugačije, jedna ćelija ne mora nužno biti jedna komponenta.

Dakle, *cell renderer* se koristi za iscrtavanje ćelija koje sadržavaju isti tip podataka. *Renderer* se može figurativno prikazati kao pečat promjenjive konfiguracije koji tablica koristi za otiskivanje pravilno formatiranih podataka u svakoj ćeliji. Kada korisnik krene uređivati podatke u ćeliji, *cell editor* tada preuzima i kontrolira ćeliju.

Kao primjer može poslužiti stupac u kojem svaka ćelija sadrži broj, točnije `Integer`. *Cell renderer* za ćelije koje sadrže brojeve koristi jednu `JLabel` instancu za iscrtavanje prikladnih brojeva koji su poravnati desno u odnosu na ćeliju. Ukoliko korisnik počne uređivati jednu od ćelija, zadani *editor* koristiti će desno poravnati `JTextField` za kontroliranje uređivanja ćelija.

Da bi tablica izabrala *renderer* koji će prikazati ćelije, ona prvo utvrđuje da li je korisnik već odabrao *renderer*. Ukoliko nije, tablica poziva `getColumnClass` metodu koja utvrđuje o kojem je tipu podataka riječ. Tada tablica uspoređuje podatke stupca s listom vrsta podataka za koje je *renderer* ćelija registriran. Listu podataka inicijalizira tablica, ali se ona može i izmijeniti. Trenutno, liste tablica sadrže ove tipove [9]:

- `Boolean`
- `Number`
- `Double, Float`
- `Date`
- `ImageIcon, Icon`
- `Object`

Kod implementacije ove tablice koristio se zadani(default) renderer kod prikaza slike (ImageIcon), a kod prikaza gumb se koristio vlastiti koji nasljeđuje zadani renderer. Također za prikaz gumba se koristio i vlastiti editor koji nasljeđuje zadani editor tablice i dodaje potrebne funkcionalnosti otvaranja prozora s detaljima.

Slika 17 prikazuje programski kod rendera korištenog za ovu tablicu.

```
class ButtonRenderer extends JButton implements TableCellRenderer {  
  
    public ButtonRenderer(){  
        //Postavke svojstvava gumba  
        setOpaque(true);  
    }  
  
    @Override  
    public Component getTableCellRendererComponent(JTable table,  
        Object value, boolean isSelected, boolean hasFocus, int row, int col) {  
        //Postavi proslijeđeni objekt value kao tekst gumba  
        setText((value == null)? "":value.toString());  
  
        return this;  
    }  
}
```

Slika 17 Prikaz programskog koda za renderer tablice

Princip rada *rendera* je korišten i za kreiranje ove tablice. Postavljeno je da u stupcu broj 2 budu postavljeni gumbi. To je u stvarnosti 3. stupac, ali program počinje stupce brojati o nulte (0) pozicije. To prikazuje slika 18.

```
table.getColumnModel().getColumn(2).setCellRenderer(new ButtonRenderer());  
table.getColumnModel().getColumn(2).setCellEditor(new ButtonEditor(new JCheckBox()));  
table.setRowHeight(okImage.getIconHeight());
```

Slika 18 Korištenje *rendera* za 3. Stupac

Gumbu je potrebno pridodati funkcionalnost. Dodana mu je ta funkcionalnost da ovisno o tome koji red se klikne tj. ovisno o tome koji se gumb u kojem redu klikne da se otvori taj stroj tj. njegovi podatci. Također definirano je i to da pritiskom na gumb, gumb se ne mijenja. I dalje ostaje isti naziv na gumbu. Dodjeljivanje funkcionalnosti gumbu prikazano je na slici 19.

```

// Poziva se funkcionalnost kliknutog gumba
@Override
public Object getCellEditorValue() {
    new Detalji(machinePropertyList.get(clickedRow));
    //Mora se vratiti vrijednost kako bi ostao isti naziv na gumbu.
    return new String("Detalji");
}

```

**Slika 19** Dodjeljivanje funkcionalnosti gumbu Detalji

Tablica se popunjava redak po redak pomoću *for* petlje. Petlja vrti brojeve od 0 i po ID-u dohvaća podatke iz JSON datoteka. Po ID-u stroja se također dodaje ime stroja u detaljnije informacije o stroju. Također se i dodaju detaljne informacije za stroj u listu s detaljnim informacijama.

Slika 20 prikazuje popunjavanje tablice *for* petljom.

```

//Popunjavanje tablice redak po redak
for(int i = 0; i < jsonMainData.length; ++i){
    //Dohvaćanje detaljnih podataka o stroju po ID-u
    Map<String, String> machineProperty = jsonParsing.getDetailData((String)jsonMainData[i][0]);
    //Dodaje se ime u detaljne informacije
    machineProperty.put("name", (String)jsonMainData[i][1]);
    //Detaljne informacije za stroj se dodaju u listu detaljnih informacija
    machinePropertyList.add(machineProperty);
}

```

**Slika 20** Popunjavanje tablice pomoću *for* petlje

Svaka *for* petlja je slična *while* petlji i ne donosi nove sposobnosti programskom jeziku, ali je za neke namjene prikladnija od odgovarajuće *while* petlje.

Inicijalizacija, uvjet nastavljanja i promjena vrijednosti su objedinjeni u prvoj liniji *for* petlje. Na ovaj način su svi činitelji *for* petlje na jednom mjestu što olakšava čitanje i razumijevanje. *For* i izvorna *while* petlja izvršavaju se jednako.

Uvjet nastavljanja mora biti logički izraz, dok inicijalizacija i promjena vrijednosti mogu biti bilo kakvi izrazi. Slika 21 prikazuje izgled *for* petlje.

For petlja ima oblik:

```
for (inicijalizacija; uvjet nastavljanja; promjena vrijednosti )
    izraz
```

ili korištenjem blokova:

```
for (inicijalizacija; uvjet nastavljanja; promjena vrijednosti) {
    izrazi
}
```

Slika 21 Prikazani oblik for petlje

Na slici 22 prikazan je dio *editora* koji uređuje tablicu. *super* je ključna riječ koja se referencira na instancu roditeljske (eng. *parent*) klase trenutnog objekta. To je korisno kada se želi pisati preko metode u podklasi, ali se i dalje želi pozvati metoda definirana u roditeljskoj klasi [10].

Isprogramirano je da je metodom `setOpaque` dodana neprozirnost gumbu i dodan mu je tekst „Detalji“ metodom `setText`.

Metodom `addActionListener` je ponovno dodana funkcija gumbu.

```
class ButtonEditor extends DefaultCellEditor {
    protected JButton button;
    private int clickedRow;

    //postavlja vrijednosti za gumb
    public ButtonEditor(JCheckBox checkbox){
        super(checkbox);

        button= new JButton();
        button.setOpaque(true);
        button.setText("Detalji");

        button.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent arg0) {
                fireEditingStopped();
            }
        });
    }

    //Poziva se kad se gumb klikne
    @Override
    public Component getTableCellEditorComponent(JTable table, Object value,
        boolean isSelected, int row, int column) {
```

Slika 22 Prikazuje dio editora koji dodjeljuje funkciju gumbu za Detalje

Sljedeća slika prikazuje drugi dio koda *editora*. U programskom kodu pozivaju se funkcionalnosti gumba.

Korištena metoda `override` ima pogodnost. Pogodnost joj je ta da ima mogućnost definirati ponašanje specifično tipu podklase što znači da podklasa može implementirati roditeljsku metodu klase ovisno o zahtjevima. U uvjetima objektno-orijentiranog programiranja dotjerivanje znači dotjerati funkcionalnost postojeće metode [11].

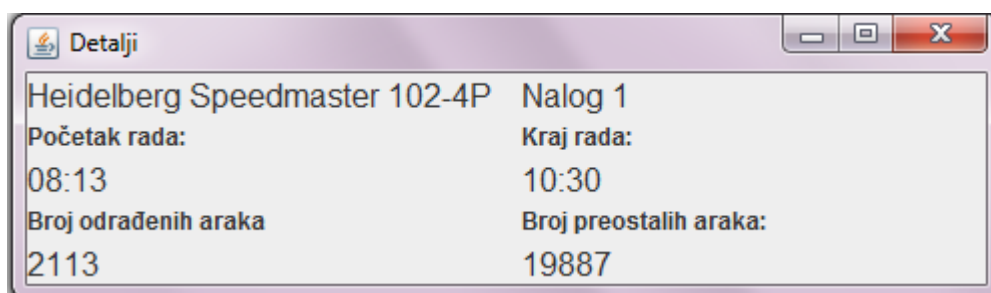
Metodom `return` vraća se onaj objekt koji će se prikazati, nakon što je akcija izvršena. Taj objekt je, u ovom slučaju, `gumb`.

```
button.addActionListener(new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent arg0) {  
        fireEditingStopped();  
    }  
});  
}  
  
//Poziva se kad se gumb klikne  
@Override  
public Component getTableCellEditorComponent(JTable table, Object value,  
    boolean isSelected, int row, int column) {  
  
    clickedRow=row;  
    return button;  
}  
  
// Poziva se funkcionalnost kliknutog gumba  
@Override  
public Object getCellEditorValue() {  
    new Detalji(machinePropertyList.get(clickedRow));  
    //Mora se vratiti vrijednost kako bi ostao isti naziv na gumbu.  
    return new String("Detalji");  
}  
}
```

Slika 23 Prikazuje drugi dio koda *editora* tablice

### 3.3 Korisnički prozori za strojeve

Prozor koji se otvara na pritisak gumba Detalji na prozoru s tablicom. Služi za detaljniji prikaz stanja rada za jedan stroj. Daje brzi pregled stanja u trenutku kada je to zatraženo za taj stroj. Prozor ne učitava nove podatke dok je otvoren već učitava jedan paket podataka prilikom otvaranja i prikazuje ga sve dok se prozor ne ugasi. Slika 24 prikazuje korisnički prozor s detaljima za stroj Heidelberg Speedmaster 102-4P. Mogu se očitati početak rada i kraj rada stroja. U tom korisničkom prozoru se može pratiti i broj odrađenih araka kao i broj preostalih araka.



Slika 24 Prozor s detaljima za traženi stroj

Na prozoru su jasno prikazani ime stroja i ime naloga koji se trenutno odrađuje. Također se može iščitati i početak rada tog naloga na stroju, kao i predviđeno vrijeme kraja. Vrijeme kraja rada stroja na tom nalogu ne određuje sam program, već je to prethodno predvidio tehnolog. Predviđeno vrijeme kraja stroja ne mora nužno biti i pravo tj. realno vrijeme kraja rada, ali daje uvid korisniku o planovima.

Pod rubrikom „broj odrađenih araka“ je prikazan broj otisnutih araka. Postoji mogućnost modificiranja te rubrike. Ukoliko je riječ o novinskom ili revijalnom tisku tada će biti prikazani metri otisnute role. Isto vrijedi za prikazani broj araka koje treba otisnuti kao i za metre papira koje treba otisnuti.



Raspored ovog prozora je postavljen pomoću `GridLayout` klase. Ta klasa radi raspored mreže ili matrice. U ovom slučaju su postavljena 2 stupca u 5 redova. Korištenje `GridLayout` rasporeda prikazano je na slici 25.

Komponente u ovom programu su raspoređene u mrežu od pet redova i 2 stupca.

```
14      GridLayout gl = new GridLayout(5, 2);
15      gl.setHgap(15);
16      windowContent.setLayout(gl);
--
```

**Slika 25** Definiranje rasporeda `GridLayout` metodom

Klasa `java.awt.GridLayout` omogućava da se komponente rasporede u redove i stupce mreže. Komponente će se dodavati u njene zamišljene ćelije. Ako se veličina kontejnera promijeni, ćelije mogu postati veće ili manje, ali će relativni položaj komponenata unutar kontejnera ostati isti [1].

Instancom `JLabel` stvoreni su objekti, dio njih ručno je popunjen s nazivima u zagradama dok je za ostatak samo otvoren prostor da se podatci mogu napuniti onim podacima u JSON datotekama. Ono što je ručno uneseno biti će stalno za svaki prozor koji se otvori dok će se podatci iz JSON-a (predstavljaju podatke koje stroj šalje) učitavati ovisno o kojem je stroju riječ što je definirano u korisničkom prozoru s tablicom strojeva. Prilikom stvaranja objekata definirani su korišteni fontovi i veličina pisma. Postavljeni font je *Plain*, a veličina pisma glavnim rubrika je 16 tipografskih točaka.

Objekt `JLabel` može prikazivati tekst, sliku ili oboje. Postavljanjem vertikalnog i horizontalnog poravnavanja u labeli može se odrediti kako će biti poravnat sadržaj labele. Područje prikazivanja labele je obično vertikalno centrirano. Labele koje sadrže samo tekst su obično poravnate od ruba do ruba, a one koje sadrže samo slike su centrirane horizontalno [11].

Slika 26 prikazuje kreirane JLabel objekte koji će se popuniti podacima iz JSON dokumenata.

```
JLabel machineName = new JLabel(info.get("name"));
machineName.setFont(new Font(machineName.getFont().getName(), Font.PLAIN, 16));

JLabel imeNaloga = new JLabel(info.get("order"));
imeNaloga.setFont(new Font(machineName.getFont().getName(), Font.PLAIN, 16));

JLabel pocetakRadaNaziv = new JLabel("Početak rada:");

JLabel pocetakRadaBroj = new JLabel(info.get("time1"));
pocetakRadaBroj.setFont(new Font(machineName.getFont().getName(), Font.PLAIN, 16));

JLabel krajRadaNaziv = new JLabel("Kraj rada:");

JLabel krajRadaBroj = new JLabel(info.get("time2"));
krajRadaBroj.setFont(new Font(machineName.getFont().getName(), Font.PLAIN, 16));

JLabel brojOdradenihNaziv = new JLabel("Broj odradenih araka");

JLabel brojOdradenihBroj = new JLabel(info.get("number1"));
brojOdradenihBroj.setFont(new Font(machineName.getFont().getName(), Font.PLAIN, 16));

JLabel brojPreostalihNaziv = new JLabel("Broj preostalih araka");

JLabel brojPreostalihBroj = new JLabel(info.get("number2"));
brojPreostalihBroj.setFont(new Font(machineName.getFont().getName(), Font.PLAIN, 16));
```

**Slika 26** Kreirani objekti koji će biti popunjeni ručno podacima iz JSON-a

Nakon što su objekti stvoreni, potrebno ih je iskoristiti tj. dodati prozoru da budu vidljivi. Ukoliko se to ne bi napravilo, prozor bi i dalje izgledao kao da je prazan. Dodavanje se izvršava metodom `windowContent.add()`. Potrebno je na taj način dodati svaki objekt pojedinačno za koji želimo da bude viđen u korisničkom prozoru.

Slika 27 prikazuje dodavanje svakog objekta pozivanjem metode `windowContent.add()`.

```
windowConent.add(machineName);  
windowConent.add(imeNaloga);  
windowConent.add(pocetakRadaNaziv);  
windowConent.add(krajRadaNaziv);  
windowConent.add(pocetakRadaBroj);  
windowConent.add(krajRadaBroj);  
windowConent.add(brojOdradenihNaziv);  
windowConent.add(brojPreostalihNaziv);  
windowConent.add(brojOdradenihBroj);  
windowConent.add(brojPreostalihBroj);
```

**Slika 27** Dodavanje svakog objekta posebno u prozor metodom `windowContent.add`

Nakon što su objekti stvoreni i dodani prozoru, potrebno je i postaviti okvir prozora vidljivim. To se postiže `setVisible` metode. `set` metoda prikazuje ili sakriva komponentu, ovisno o vrijednosti parametra. Parametar je u ovom slučaju `Visible`, pa će ova metoda prikazivati komponentu. Korištenje te metode prikazano je na sljedećoj slici:

```
frame.setContentPane(windowConent);  
frame.pack();  
frame.setVisible(true);
```

**Slika 28** Prikazivanje prozora pomoću `setVisible` metode

### 3.4 JSON Parsing

Ova klasa kreirana je da bi se učitale i parsirale informacije iz JSON dokumenata. Koristi se u *Login* prozoru da se učitaju korisnički podatci, u glavnom prozoru (prozor s tablicom) koristi se za učitavanje općih podataka o strojevima (ime stroja), a u korisničkom prozoru s detaljima služi za učitavanje detaljnih podataka o pojedinom stroju.

Slika 29 prikazuje programski kod koji služi za učitavanje generalnih podataka koji će biti iskorišteni u tablici strojeva. U kodu su korišteni `try` i `catch` blokovi, a unutar `try` bloka je `for` petlja.

```
/**
 * Učitava generalne podatke o strojevima
 * @return Vraća tablicu sa podacima iz JSON datoteke
 */
public Object[][] getMachineData(){

    Object[][] data = new Object[0][0];
    try {
        JSONObject jsonObject = new JSONObject(readFile("JSON\\glavni.json"));

        JSONArray jsonArray = jsonObject.getJSONArray("strojevi");

        data = new Object[jsonArray.length()][2];
        JSONObject machine = null;
        for(int i = 0; i < jsonArray.length(); ++i){
            machine = jsonArray.getJSONObject(i);
            data[i][0] = machine.getString("id");
            data[i][1] = machine.getString("name");
        }

    } catch (JSONException e) {

        e.printStackTrace();
    } catch (IOException e) {
        |
        e.printStackTrace();
    }
}
```

Slika 29 Programski kod za učitavanje podatke o strojevima za tablicu

Prvi korak konstruiranja rukovoditelja iznimki je ograđivanje koda koji može izvesti iznimku unutar `try` bloka. Postavljanje rukovatelja iznimki postiže se `try` blokom tako da se napišu jedan ili više `catch` blokova odmah iza `try` bloka. Ne smije biti kodova između kraja `try` bloka i početka prvog `catch` bloka. Svaki `catch` blok je rukovatelj iznimkom koji upravlja tipom iznimke određene argumentom. Tip argumenta određuje tip iznimke kojom rukovatelj može upravljati.

Svakom iteracijom `for` petlje se popuni jedan red tablice s podacima iz JSON datoteka.

Na slici 30 prikazana je datoteka JSON formata sa općim podacima o strojevima koji se prikazuju u tablici. Svakom stroju dodijeljen je `id` i ime stroja. Ovaj dokument predstavlja jedan JSON objekt koji sadrži niz više manjih objekata.

```
<script>
{
  "strojevi": [
    {
      "id": "01",
      "name": "Heidelberg Speedmaster 102-4P",
    },
    {
      "id": "02",
      "name": "Heidelberg Speedmaster 102-2P",
    },
    {
      "id": "03",
      "name": "Products Speedmaster CX 102",
    }
  ]
}
</script>
```

Slika 30 JSON datoteka sa općim podacima o stroju koji će se prikazati u tablici

Slika 31 prikazuje metodu za učitavanje detaljnih informacija iz JSON datoteke za pojedini stroj. Na osnovu ulaznog argumenta odlučuje se koja će datoteka biti učitana. Ulazni argument je `machineId`, on predstavlja identifikacijsku oznaku stroja čiji će se podaci učitati. Učitani podaci se pohranjuju u podatkovnu strukturu `Map`, odnosno njenu implementaciju `HashMap`, a ona označava strukturu parova ime/vrijednost. `Try` i `catch` blokovi se koriste kako bi uhvatili iznimke vezane uz format JSON datoteke.

```
/**
 * Učitava detaljne informacije iz JSON datoteke za pojedini stroj
 * @param machineId oznaka za stroj čiji se detalji učitavaju
 * @return Parove Ime/Vrijednost s detaljima o pojedinom stroju
 */
public Map<String, String> getDetailData(String machineId){
    Map<String, String> value = new HashMap<String, String>();

    try {
        JSONObject jsonObject = new JSONObject(readFile("JSON\\stroj" + machineId + ".json"));

        value.put("status", jsonObject.getString("status"));
        value.put("order", jsonObject.getString("order"));
        value.put("time1", jsonObject.getString("time1"));
        value.put("time2", jsonObject.getString("time2"));
        value.put("number1", jsonObject.getString("number1"));
        value.put("number2", jsonObject.getString("number2"));

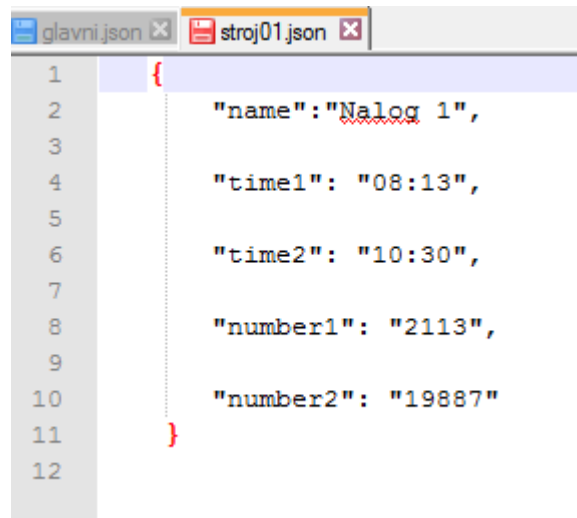
    } catch (JSONException e) {

        e.printStackTrace();
    } catch (IOException e) {

        e.printStackTrace();
    }
}
```

**Slika 31** Metoda koja učitava detaljne informacije o pojedinom stroju

Slika 32 prikazuje JSON zapis detalja za stroj s identifikacijskim brojem 01. Podatci su zapisani kao parovi ime/vrijednost i učitavaju se u program metodom `getDetailData`.



```
1 {  
2     "name": "Nalog 1",  
3  
4     "time1": "08:13",  
5  
6     "time2": "10:30",  
7  
8     "number1": "2113",  
9  
10    "number2": "19887"  
11 }  
12
```

**Slika 32** JSON zapis detalja za stroj s identifikacijskom 01

Slika 33 prikazuje metodu koja učitava korisničke podatke iz JSON datoteke u *Login* prozor. Iznimke su uhvaćene blokovima `try` i `catch`.

```

public Map<String, String> getUserData(){
    Map<String, String> value = new HashMap<String, String>();

    try {
        JSONObject jsonObject = new JSONObject(readFile("JSON\\login.json"));

        value.put("username", jsonObject.getString("username"));
        value.put("password", jsonObject.getString("password"));

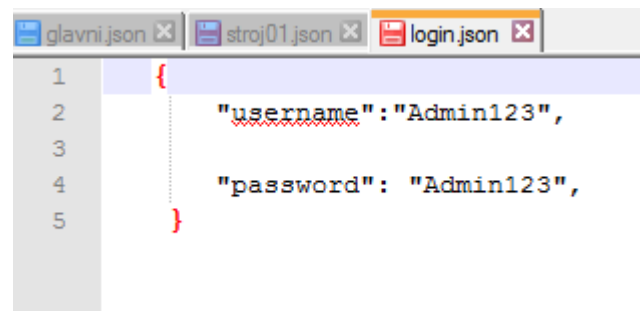
    } catch (JSONException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return value;
}

```

Slika 33 Metoda koja učitava korisničke podatke u *Login* prozor

Korisnički podatci zapisani u JSON formatu prikazani su na slici 34. Podatci imaju strukturu ime/vrijednost.



```

1 {
2     "username": "Admin123",
3
4     "password": "Admin123",
5 }

```

Slika 34 Korisnički podatci u JSON formatu



Slika 35 prikazuje metodu koja otvara JSON datoteku, učitava njen sadržaj red po red i kada ga učita red spaja na prethodno učitani red.

```
/**
 * Učitava datoteku red po red i vraća kao jedan String --> sve redove spaja u jedan
 * @param fileName ime datoteke
 * @return sve redove datoteke u jednom redu
 * @throws IOException
 */
private String readFile(String fileName) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(fileName));
    try {
        StringBuilder sb = new StringBuilder();
        String line = br.readLine();

        while (line != null) {
            sb.append(line);
            sb.append("\n");
            line = br.readLine();
        }
        return sb.toString();
    } finally {
        br.close();
    }
}
```

Slika 35 Metoda koja otvara JSON datoteku i učitava njen sadržaj red po red

While petlja se koristi za uzastopno ponavljanje jednog izraza. Točnije, while petlja ponavlja izjavu dok je zadani uvjet istinit. Kad računalo dođe do while izjave, procjenjuje logički izraz koji vraća vrijednost true ili false. Ako je vrijednost izraza false, računalo preskače ostatak while petlje i nastavlja s izvršenjem programa. Ako je vrijednost true, računalo izvršava izraze unutar petlje, zatim se vraća na početak petlje i ponavlja postupak tj. ponovno procjenjuje logički izraz, te završava petlju ako je vrijednost false, a nastavlja ako je vrijednost true. Ovo se nastavlja dok izraz ne poprimi vrijednost false; slučaj u kojem se to nikada ne dogodi naziva se beskonačna petlja [12].

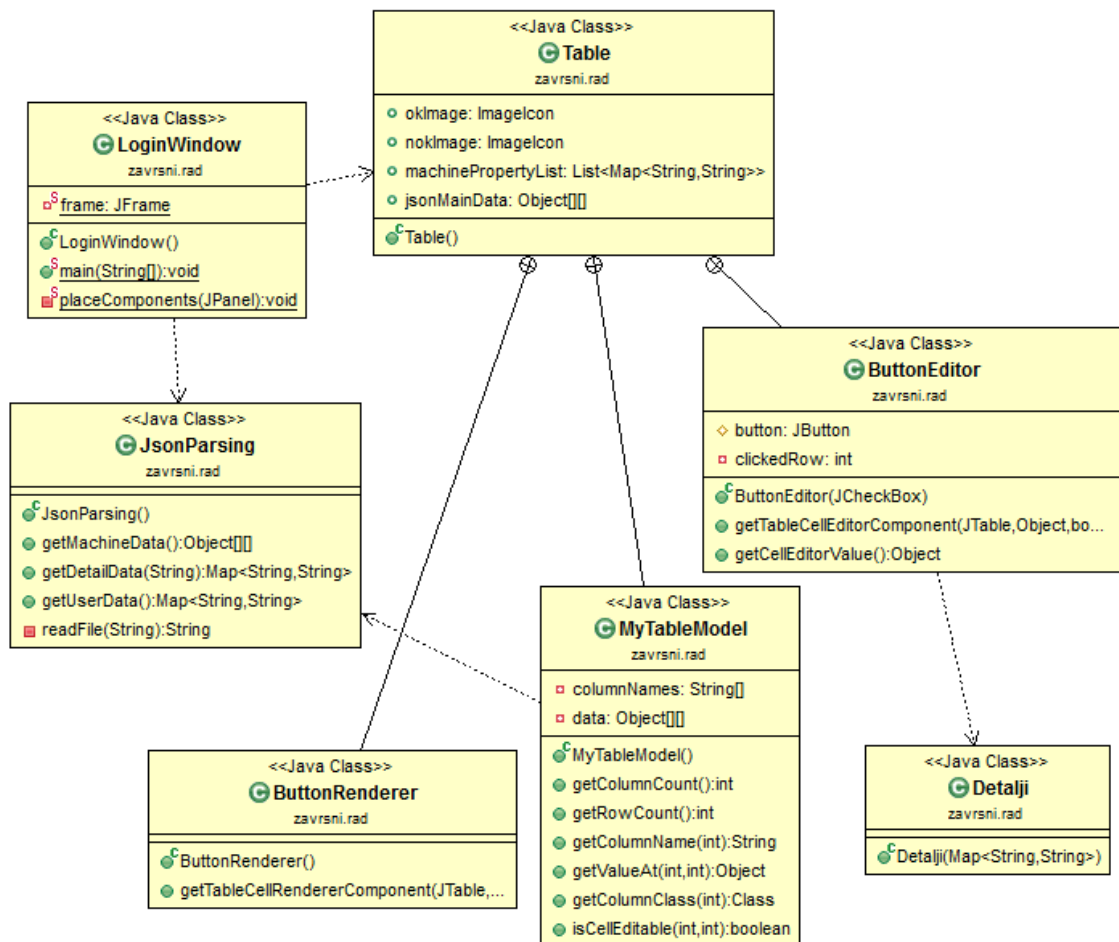
While petlja u ovoj metodi služi da se učitavaju podatci sve dok se ima što za učitati.

Klauzula `finally` se izvršava bez obzira na to da li je izuzetak bačen ili ne. Bez `finally` klauzule, blok iza `try-catch` sekvence se ne mora izvršiti ako se baci izuzetak iz `catch` bloka. Svrha `finally` je sređivanje internog stanja i/ili oslobađanje neobjektnih resursa (kao što su otvorene datoteke).

`BufferedReader` je omotač (eng. *wrapper*) oko ostalih `Reader`-a koji čitaju karaktere i on služi za ubrzano učitavanje. Da ne bi čitali znak po znak, on će pročitati više znakova odjednom i sačuvati ih u svojoj memoriji. `BufferedReader` sam ne radi ništa. Njemu je potreban objekt koji će "vršiti čitanje".

### 3.5 Dijagram klasa

Na slici 36 prikazan je dijagram klasa. Prikazuje odnose između klasa koje su korištene za izradu ovog programa.



Slika 36 Dijagram klasa

Dijagram klasa je vrsta strukturnog dijagrama u softwareskom inženjeringu koji opisuje strukturu sustava objašnjavajući klase unutar sustava, njihove atribute i odnose. Pokazuje postojanje klasa i njihovih međusobnih odnosa prilikom logičkog oblikovanja sustava.

## 4. ZAKLJUČAK

Program je napravljen sa svrhom olakšavanja kontrole rada strojeva i tiskare. Njegova implementacija olakšala bi rad voditeljima proizvodnje kao i tehnolozima koji izračunavaju proračune novih narudžbi. Program omogućava laki pristup popisu strojeva u tiskari i jednostavan pristup detaljnijem uvidu svakog od strojeva. Pokazalo se da program ima visoki potencijal uz daljnje investicije i razvoj programa. Izgled programa je jednostavan i korisniku prihvatljiv. Jednostavan je za upotrebu i intuitivan je. Program ne zahtijeva ulaganja u edukaciju kadra za upotrebu.

Program se može koristiti u svim tiskarskim pogonima, bez obzira o kojoj je tiskarskoj tehnici riječ. Prikazani podatci se mogu se modificirati ukoliko nije riječ o tisku na arke.

U prvom poglavlju opisivale su se korištene tehnologije za implementaciju programa. Svrha opisivanja tehnologija je bila ta da čitatelj lakše shvati što je objektno programiranje i način na koji se programira u Javi.

U prvom potpoglavlju prvog poglavlja opisana je Java i njezin nastanak kao i njeni kreatori. Navedeni su principi rada kao i objektno programiranje u Javi. Konstruktor u Javi je također vrlo bitan, pa je i on opisan kao i stvaranje Java arhivskih dokumenata. U potpoglavlju 2.2 riječ je bilo o Biblioteci Swing koja je dio Java programskog okruženja. Kako se razvijaju korisnička sučelja u Biblioteci Swing je neizostavan dio da bi se dobio uvid u to. Potpoglavlje 2.3 posvećeno je JSON formatu zapisa podataka.

Korištene tehnologije su se mogle i zamijeniti s nekim drugima koje imaju više mogućnosti, ali to nije bilo potrebno. S ovim jednostavnim tehnologijama, koje se brzo uče, su se ispunila sva očekivanja kod izrade programa.

Drugo poglavlje opisuje postupak kreiranja i stvaranja samog programa. Izdvojeni su bitniji redovi kodova koji su ključni da bi se shvatilo objektno programiranje i razlika između objektnog programiranja i strukturnog programiranja. U prvom dijelu poglavlja, potpogavlje 3.1 opisan je prvi korisnički prozor s *Login* mogućnošću. Sljedeće potpogavlje obuzelo je tablicu sa strojevima i opisalo njenu funkcionalnost dok je potpogavlje 3.3 opisalo jednostavni prozor s detaljima koji služi samo za uvid. Ukoliko bi kupac zatražio neke preinake na programu kao npr. dodavanje još korisnika koji bi imali pristup podacima, to bi se moglo adaptirati bez dodatnih troškova. Također su moguća i naknadna dodavanja novih strojeva za praćenje u program, ukoliko bi se tiskara proširivala ili mijenjala strojeve. Potpogavlje 3.4 je objasnilo metode koje su se koristile da bi se uveli podatci iz JSON datoteka u program. U zadnjem potpoglavljju prikazan je dijagram klasa. Taj dijagram je jednostavno opisao na koji su način povezane klase korištene u programu.

Ubuduće, bilo bi dobro napraviti i obavještanje alarmom da je stroj otisnuo cijelu naknadu i na računalu voditelja odjela kako bi on znao da je taj stroj gotov sa radom i raspoložen za nove naloge. Također bi se moglo alarmirati i ukoliko stroj stane s radom tako da voditelj proizvodnje ima s jednog mjesta u tiskari uvid u sva događanja u njoj.

## 5. LITERATURA

1. Fain Y. (2011). *Programiranje Java*, Dobar plan, Zagreb
2. Flanagan D. (1999). *Java in a Nutshell, Third Edition*, O'Reilly, Sebastopol
3. Clay Richardson W., Avondolio D, Vitale J., Schrage S., Mark W. Mitchell, Scanlon J. (2006). *Professional Java JDK 5 Edition*, Dobar Plan, Zagreb
4. \*\*\* <http://json.org/> - Introducing Java, 30.08.2015.
5. \*\*\* <http://csis.pace.edu/~bergin/KarelJava2ed/ch2/javamain.html> – Csis Pace, 02.09.2015
6. \*\*\* <https://docs.oracle.com/javase/tutorial/uiswing/components/panel.html> – Oracle/javase/tutorial, 02.09.2015
7. \*\*\* <http://www.programcreek.com/2013/12/what-exactly-is-null-in-java/> - Program Creek, 02.09.2015
8. \*\*\* [http://laris.fesb.hr/java/blokovi\\_petlje.htm](http://laris.fesb.hr/java/blokovi_petlje.htm) – Laris FESB/java, 02.09.2015
9. \*\*\* <https://docs.oracle.com/javase/tutorial/uiswing/components/table.html#editor> – Oracle/Java Documentation, 30.08.2015.
9. \*\*\* <http://stackoverflow.com/questions/5141344/meaning-of-super-keyword> – Stackoverflow/questions, 25.08.2015
10. \*\*\* [http://www.tutorialspoint.com/java/java\\_overriding.htm](http://www.tutorialspoint.com/java/java_overriding.htm) – Tutorials Point/java, 25.08.2015
11. \*\*\* <http://docs.oracle.com/javase/7/docs/api/javax/swing/JLabel.html> – Oracle/javase, 29.08.2015
12. \*\*\* [http://laris.fesb.hr/java/blokovi\\_petlje.htm](http://laris.fesb.hr/java/blokovi_petlje.htm) Laris FESB/java 02.09.2015