Java okruženja za rad s grafičkim sučeljem

Perković, Tedi

Undergraduate thesis / Završni rad

2015

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:512575

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-24**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli

Fakultet ekonomije i turizma

"Dr. Mijo Mirković"

TEDI PERKOVIĆ

JAVA OKRUŽENJA ZA RAD S GRAFIČKIM SUČELJEM

Završni rad

Pula, 2015.

Pula, 2015.

Sveučilište Jurja Dobrile u Puli

Fakultet ekonomije i turizma

"Dr. Mijo Mirković"

TEDI PERKOVIĆ

JAVA OKRUŽENJA ZA RAD S GRAFIČKIM SUČELJEM

Završni rad

JMBAG: 0016057412, redoviti student
Studijski smjer: Informatika

Predmet: Programiranje
Mentor: doc. dr. sc. Krunoslav Puljić

Pula, kolovoz 2015.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Tedi Perković, kandidat za prvostupnika Informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student:

U Puli, __. __. 2016. _____

SADRŽAJ

Uvod 1

1 Java 2

1.1 Povijest 2

1.2 Prednosti Jave 3

1.3 Java Virtualni stroj 4

2 Integrirana razvojna okruženja 9

2.1 Eclipse 9

2.2 NetBeans 10

2.3 IntelliJ IDEA 11

2.4 Jdeveloper 12

2.5 Izabrano integrirano razvojno okruženje 13

3 Swing 14

3.1 Spremnik 15

3.2 Upravitelj razmještaja 15

3.3 Inspektor svojstva 17

4 Java projekt u IntelliJ IDEA 20

5 Swing komponente u IntelliJ IDEA 21

5.1 JPanel 21

5.2 HSpacer i VSpacer 21

5.3 JScrollPane 21

5.4 JLabel 22

5.5 JButton 23

5.6 JRadioButton 23

5.7 JCheckBox 24

5.8 JTextField 24

5.9 JPasswordField 24

5.10 JFormattedTextField 25
5.11 JTextArea 25
5.12 JTextPane 25
5.13 JEditorPane 26
5.14 JList 26
5.15 JComboBox 26
5.16 JTable 27
5.17 JTree 27
5.18 JTabbedPane 28
5.19 JSplitPane 28
5.20 JSpinner 29
5.21 JSlider 29
5.22 JSeparator 30
5.23 JProgressBar 30
5.24 JToolBar 30
5.25 JToolBar.Separator 31
5.26 JScrollBar 31
6 Izrada aplikacije kalkulatora 32
7 Zaključak 41
8 Sažetak 42
9 Summary 43
10 Literatura 44
11 Popis slika 45
12 Popis tablica 45

UVOD

U današnje vrijeme mnogi programski jezici nude dobru podršku programskih okvira za razvoj grafičkih korisničkih sučelja (eng. Graphical User Interface - GUI). U ovom radu bit će opisana okruženja za rad s grafičkim sučeljem koja koriste Javu kao temeljni programski jezik. Razlog je što je Java jedna od najpoznatijih objektno orijentiranih programskih jezika i posebno je oblikovana da što manje ovisi o operativnom sustavu (Microsoft Windows, Linux, Unix, Solaris, Android...). Zato se Java aplikacije mogu pokretati na gotovo svim operativnim sustavima.

U prvom poglavlju saznati ćemo nešto više o Javi kao programskom jeziku. Saznati ćemo tko su tvorci Jave te kako se ona počela razvijati, kako je projekt više puta mijenjao ime i tim prije nego je dobio konačan naziv. Nadalje doznati ćemo više o prednostima Jave naspram drugih programskih jezika niže razine, te njezine značajke što čine Javu močnim objektno orijentiranim programskim jezikom, zatim arhitekturu kompiliranja izvornoga koda u izvršnu datoteku te njezino izvođenje. Osim toga, u kratko ćemo se osvrnuti na primitivne podatke u Javi.

U drugom poglavlju govori se o integriranim razvojnim okruženjima, onih najčešćih i najpoznatijih sa kojima se možemo susreti. Usporedba među njih samih te kratak zaključak o tome koji je autor izabrao za korištenje i testiranje daljnjeg sadržaja u radu.

Treće poglavlje govori o Swingu - GUI programskom okviru. Objašnjava se općenita logika i mehanika rada te ono što čini bit korištenja takvog programskog okvira. U potpoglavljima se opisuju određeni prozori programskog okvira u odabranom integriranom razvojnom okružju i svojstva komponenti programskog okvira.

U četvrtom poglavlju imamo opisano kako se stvara novi projekt u odabranom integriranom razvojnom okružju, te upute za što je sve potrebno napraviti i podesiti kada se izrađuje program sa korisničkim sučeljem.

Peto poglavlje, koje je ujedno i zadnje poglavlje, sadrži detaljno opisane sve moguće komponente, jednu po jednu, koje nam nudi integrirano razvojno okružje za rad nad programom koji ima grafičko korisničko sučelje.

1 JAVA

Ovo poglavlje daje kratak uvod u povijest Jave i temeljne prednosti koje Java pruža u odnosu na neke druge programske jezike. Osim toga, govorit će se o Java virtualnom stroju te će se pobliže objasniti Java prevoditelj te njegovo izvršavanje bajt koda na virtualnoj mašini. Na kraju poglavlja bit će riječi o Java primitivnim podacima, te će biti pobliže objašnjena razlika između referenci u Javi i pokazivača u programskim jezicima niže razine. …

1.1 POVIJEST

Ideja o Javi se pojavila oko 1990. godine kada je glavni istraživač Bill Joy iz Sun Microsystems počeo zagovarati ideju o ostvarivanju složenih zadataka sa jednostavnim softverom te je osnovao Sun Aspen Smallworks.

Od izvornih članova malog tima programera okupljenih u Aspenu, James Gosling će biti zapamćen kao otac Jave. Gosling je dobio na reputaciji početkom 80tih kao autor Gosling Emacsa, prve verzije popularnog uređivača teksta Emacs koji je bio pisan u C-u i pokretao se pod Unixom. Goslingov Emacs je bio vrlo popularan ali ga je ubrzo potisnula besplatna verzija GNU Emacsa kojeg je bio napisao Emacsov originalni dizajner. U to vrijeme, Gosling je prešao na oblikovanje Sunovog NeWS, koji se povezivao sa X Window Systemom za kontrolu nad Unix GUI desktop verziji 1987. Iako dio ljudi tvrdi da je NeWS bolji od Xa, NeWS je izgubio budući da ga je Sun zadržao u vlasništvu i nije objavio izvorni kod dok su glavni programeri od Xa formirali X Konzorciji, te pokušali suprotnim pristupom.

Gosling je stečeno znanje iskoristio na Bill Joyevom početnom Aspen projektu. Rad na projektu 1992. doveo je do osnivanja Sun-ove podružnice FirstPerson, Inc. Njezina misija je bila uvođenje Sun-a u svijet potrošačke elektronike.

FirstPerson tim je radio na razvijanju softvera za mobilne telefone i dlanovnike. Cilj je bio omogućavanje prijenosa informacija i aplikacija preko infracrvene veze i tradicionalnih mreža. Ograničenja na memoriji i propusnosti podataka su diktirali da kod bude malen i efikasan. Također, aplikacije su zahtijevale da budu sigurne i robusne. Gosling i njegovi članovi tima su započeli sa programiranjem u C++, ali su ubrzo shvatili koliko jezik može biti zbunjujući, složen, nezgrapan i nesiguran za projekt na kojemu su radili. Odlučili su krenuti ispočetka, tako da je Gosling započeo raditi na nečemu što je on nazivao “C++ minus minus”.

Neuspjehom Apple Newton-a (Appleovo prvo ručno računalo), Sun je usmjerio FirstPerson-ove napore na interaktivnu televiziju (ITV). Odabrani programski jezik za ITV je

bio predak Jave, programski jezik nazvan Oak. Neovisno o sposobnosti jezika da pruža sigurnu interaktivnost i efikasnost, ITV nije uspio u to vrijeme. Sun je napustio zamisao jer ju klijenti nisu dobro prihvatili.

U to vrijeme, Joy i Gosling su se udružili da definiraju novu strategiju za njihov inovativni jezik. Veliki interes za Webom koji se javlja 1993. godine ponudio je novu priliku za Oak. Oak je bio malen, siguran, neovisan o arhitekturi računala i objektno orijentiran. Dogodilo se da su ti zahtjevi za programski jezik univerzalni kod web programera, dizajnera i naprednih korisnika. Sun se ubrzo preusmjerio prema toj grani, i sa malim izmjenama, Oak je postao Java.

1.2 PREDNOSTI JAVE

U početku, Java je bila orijentirana na izgradnju ugrađenih aplikacija za web tzv. appletima. No, u ranim danima, appleti i ostale aplikacije koje su koristile grafičko korisničko sučelje (GUI) bile su ograničene. Danas, Java ima Swing, jedan od najsofisticiranijih softverskih alata za izgradnju grafičkih korisničkih sučelja (GUI). Taj razvoj je omogućio Javi da postane popularna platforma za razvoj tradicionalnih korisničkih softvera.

U poznatim programskim jezicima kao što su C, Pascal, Fortran i sl. izvorni programski kod se temelji na algoritmu, odnosno nizu instrukcija i funkcijskih poziva. Razvojem računalne tehnologije, programski kod programa kod takvih jezika postaje složeniji, teži za izmjene i daljnji razvoj te razumijevanje u cjelini.

Kako bi se ublažio navedeni problem, prirodno se razvilo objektno orijentirano programiranje sa svojim osnovnim pojmovima: objekt, klasa, atribut, metoda. Osnovna ideja objektno orijentiranog programiranja je sljedeća:

- Izvorni kod programa se temelji na skupu objekata koji međusobno komuniciraju
- Objekti se sastoje od atributa (predstavljaju stanje objekta) i metoda (funkcije koje služe za promjenu stanja i komunikaciju sa drugim objektima)
- Objekt je instanca neke klase, odnosno klasa predstavlja tip objekta, pa izvorni kod programa čine klase

Temeljne značajke Jave su:[1]

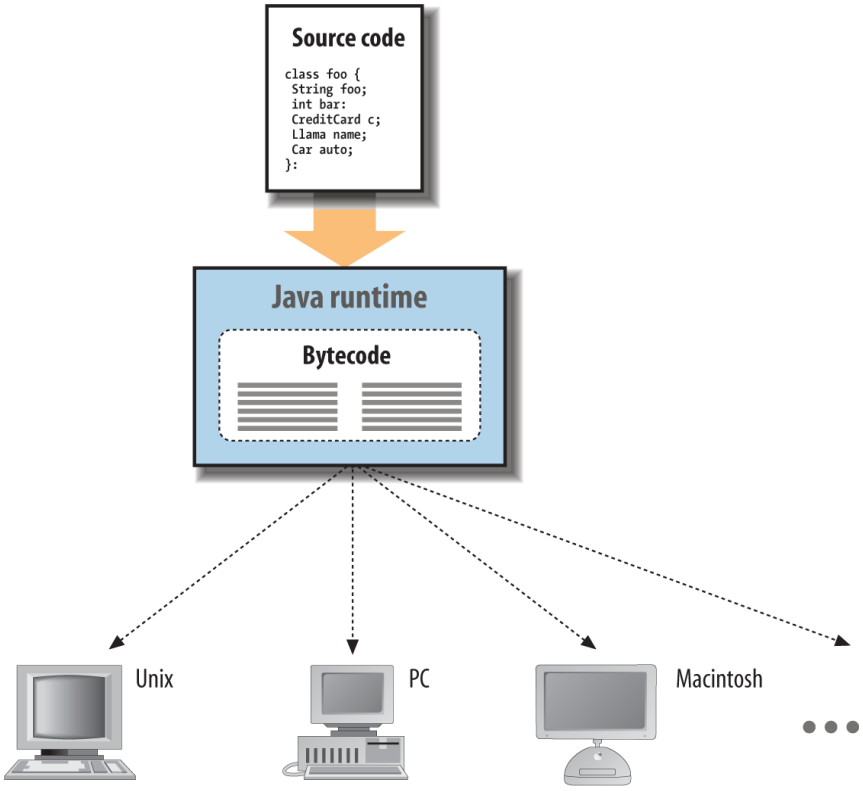
[1] Flanagan, David. Java in a Nutshell. Sebastopol: O'Reilly, 1997.

- Jednostavnost: nema potrebe za pokazivačima, datotekama zaglavlja, preopterećenjem operatora.
- Objektno orijentirani programski jezik u kojem se programski kod prevodi u bajt kod koji se pokreće putem Java virtualnog stroja (Java Virtual Machine - JVM) te time postaje neovisan o računalnoj arhitekturi i operacijskom sustavu.
- Robusnost: zahvaljujući automatskom upravljanju memorijom i skupljanju smeća (eng. Garbage collector) iz Java virtualnog stroja izbjegavamo problem alokacije i dealokacije objekata koji može dovesti do otjecanja (eng. Leak) memorije.
- Višedretvenost je podržana kao i sinkronizacija dretvi i procesa koja omogućuje pokretanje interaktivnih sustava bez potpunog zastoja (eng. Deadlock).
- Dinamičnost omogućuje da dobijemo informacije o objektima u tijeku samog izvršavanja što nam daje mogućnost da nadodamo male segmente bajt koda na sustavu koji je već pokrenut bez da prije toga ponovno pokrećemo sustav.
- Distribuiranost je omogućena kao i mnoge mrežne opcije u okviru Javine biblioteke programskih elemenata (Java API – eng. Java Aplication Programming Interface, odnosno Java aplikacijsko programsko sučelje)

1.3 JAVA VIRTUALNI STROJ

Java je jezik koji se prevodi (kompajlira) i interpretira. Njezin izvorni kod se pretvara u bajt kod, slično kao kod mikroprocesorskog strojnog koda. Međutim, za razliku od C ili C++ koda koji se reducira na izvorne instrukcije za određeni model procesora, Java kod se kompilira u univerzalni format - instrukcije za virtualni stroj.

Java bajt kod dobiven prevođenjem izvršava se preko izvedbenog interpretera. Izvedbeni sustav izvršava sve normalne funkcije procesora, ali to radi u sigurnom, virtualnom okruženju. Izvršava skup instrukcija na stogu i upravlja memorijom kao operativni sustav. Stvara i manipulira primitivnim tipovima podataka i učitava i poziva se na nove referencirane blokove koda. Najbitnije, to sve odrađuje u skladu sa strogo definiranim otvorenim specifikacijama koje može implementirati svatko tko želi proizvesti Java virtualni stroj. Zajedno, virtualni stroj i definicija jezika pružaju potpunu specifikaciju. Nema značajki u Java jeziku koje su ostavljene nedefinirane ili ovisne o implementaciji. Na primjer, Java definira veličine i matematička svojstva svih svojih primitivnih tipova podataka umjesto da ih prepusti platformi na implementaciju.



Slika 1: Java izvršno okruženje. Izvor: Learning Java, Poglavlje 1.2

Java interpreter je relativno lagan i malen, što znači da mu je potrebno malo resursa za izvršavanje zadatka. Može biti primijenjen u bilo kojem željenom obliku za određenu platformu. Interpreter se može pozivati kao odvojena aplikacija ili može biti ugrađeni u drugi dio softvera, kao što su web preglednici. Drugim riječima, to znači da je Java programski kod implicitno prenosiv. Isti aplikacijski Java bajt kod se može pokretati na bilo kojoj platformi koja ima Java izvedbenu okolinu (eng. Java runtime), kao što je prikazano na slici 1. Stoga nije potrebno izdavati alternativne verzije aplikacije za različite platforme, i ne treba distribuirati izvorni programski kod krajnjim korisnicima.

Temeljna jedinica Java programskog koda je klasa. Kao i u drugim objektno orijentiranim jezicima, klase su komponente aplikacije koje drže izvršni kod i podatke. Java klase se nakon prevođenja distribuiraju u univerzalnom binarnom formatu koji sadrži Java bajt kod i druge informacije o klasi. Klase se mogu održavati diskretno i sačuvati u datotekama ili arhivama lokalno ili na poslužitelju. Klase se pronalaze i učitavaju dinamično prilikom izvršavanja i prema potrebi aplikacije.

Intepreteri su se nekada smatrali sporima, ali Java nije tradicionalni interpretrani jezik. Osim što ima mogućnost prevođenja izvornog programskog koda u prijenosni bajt kod, Java je pažljivo oblikovana tako da implementacija softvera u izvedbenom sustavu može dodatno

optimizirati performanse prevođenjem bajt koda u izvorni strojni kod i to sve prilikom izvršavanja. To se naziva just-in-time (JIT) ili dinamično prevođenje. Sa JIT prevođenjem, Java programski kod se može pokretati brzo kao i izvorni programski kod a da pritom zadrži prenosivost i sigurnost.

Problem sa JIT prevođenjem je što optimizacija koda troši određeno vrijeme. Zato JIT prevoditelj može proizvesti dobre rezultate, ali može patiti od izrazite latencije prilikom pokretanja aplikacije. To generalno nije problem za poslužiteljske aplikacije koje su dugoročno pokrenute, ali može biti veliki problem za klijentske softvere i aplikacije koje se pokreću na manjim uređajima sa ograničenim mogućnostima. Da bi riješila navedeni problem, Javina prevoditeljska tehnologija zvana HotSpot, koristi trik zvani *prilagodljiva kompilacija*. Ako se pogleda na što programi zaista troše vrijeme, ispadne da najviše vremena potroše na izvršavanje relativno malog dijela koda ponovno i ponovno. Komad koda koji se ponavlja prilikom izvršavanja može biti samo mali dio cijelog programa, ali njegovo ponašanje utvrđuje ukupnu učinkovitost programa. Prilagodljiva kompilacija omogućuje Javi da iskoristi nove vrste optimizacije koji se inače ne mogu primijeniti u statičkom prevoditeljskom jeziku. Otuda je tvrdnja da se Java programski kod u nekim slučajevima može izvršavati brže od C/C++.

Jedna od glavnih razlika između Jave i jezika niže razine kao što su C i C++ je upravljanje memorijom. Korištenjem skupljača smeća (eng. Garbage collector) rješava se mnogo problema oko sigurnosti, prenosivosti i optimizacije koda. Javin skupljač smeća tako pomaže programerima oko najčešćih grešaka koje se pojavljuje u programiranju, onima vezanim uz alokaciju i dealokaciju memorije. Dosta često, programeri u nižim jezicima prilikom alokacije memorije za neki objekt znaju zaboraviti napraviti i dealokaciju (otpuštanje) nakon njegovog korištenja čime se pojavljuje problem sa otjecanjem memorije (eng. *memory leak*). U Javi je taj problem riješen skupljačem smeća koji objekt nakon korštenja dereferencira i time označi skupljaču smeća da ga može obrisati.

Java ima dvije vrste tipova podataka:[2]

- primitivne - numeričke vrijednosti, vrijednosti istinitosti i znakovi
- reference - siguran “pokazivač” (eng. pointer) na objekt

[2] Čupić, Marko. Programiranje u Javi. Zagreb: FER, 2013.

Numeričke vrijednosti mogu biti cjelobrojne (byte, short, int, long, char) i decimalne (float i double). Za istinitosti definiran je tip boolean.

Tabela 1: Primitivni tipovi podataka u Javi. Izvor: Čupić, Marko. Programiranje u Javi. Zagreb: FER, 2013.

Tip	**Opis**
byte	Predstavlja cijeli broj čiji je raspon ograničen na interval od -128 do +127. Radi se o 8 bitnom cijelom broju s predznakom koji koristi dvojni komplement.
short	Predstavlja cijeli broj čiji je raspon ograničen na interval od -32768 do +32767. Radi se o 16 bitnom cijelom broju s predznakom koji koristi dvojni komplement.
int	Predstavlja cijeli broj čiji je raspon ograničen na interval od -2,147,483,648 do +2,147,483,647. Radi se o 32 bitnom cijelom broju s predznakom koji koristi dvojni komplement.
long	Predstavlja cijeli broj čiji je raspon ograničen na interval od -9,223,372,036,854,775,808 do +9,223,372,036,854,775,807. Radi se o 64 bitnom cijelom broju s predznakom koji koristi dvojni komplement.
float	Predstavlja decimalni broj sa rasponom od $1.4e^{-45}$ do $3.4e^{38}$. Odgovara specifikaciji IEEE 754 za decimalne brojeva jednostruke preciznosti (troši se 32 bita odnosno 4 okteta za svaku vrijednost).
double	Predstavlja decimalni broj sa rasponom od $4.9e^{-324}$ do $1.8e^{308}$. Odgovara specifikaciji IEEE 754 za decimalne brojeva dvostruke preciznosti (troši se 64 bita odnosno 8 okteta za svaku vrijednost).
boolean	Predstavlja tip podataka koji se koristi za prikaz istinitosti. Podatci ovog tipa mogu poprimiti jednu od dviju vrijednosti: true ili false.
char	Predstavlja tip podataka koji se koristi za prikaz jednog znaka. To je cijeli 16 bitni broj bez predznaka, tako da su legalne vrijednosti cijelih brojeva iz raspona od 0 do 65535, ili alternativno od '\u0000' do '\uffff'. Ovo je značajna razlika u odnosu na programski jezik C kod kojeg je jedan znak koristio za pohranu jedan oktet. Ova razlika uvedena je kako bi se jeziku omogućilo da transparentno radi s velikim brojem različitih znakova koristeći pri tome Unicode specifikaciju.

Reference u Javi su jedna vrsta sigurnog pokazivača. Svim objektima u Javi, osim primitivnim numeričkim tipovima, se pristupa preko referenci. Za razliku od C/C++

pokazivača, gdje se mogu izvoditi aritmetika na pokazivačima i tako mijenjati adresu u memoriji na koju pokazivač pokazuje, reference u Javi mogu pokazivati samo na specifičan objekt ili element u polju. Znači referenca je atomska stvar kojom se ne može mijenjati njezina vrijednost osim da joj se dodijeli objekt. Reference se prosljeđuju po vrijednosti i moraju pokazivati na klasni tip objekta. Osim toga, Java nema pokazivača na metode i zamjenska mehanika je korištenje klasa sučelja (eng. interface) i adaptera.

2 INTEGRIRANA RAZVOJNA OKRUŽENJA

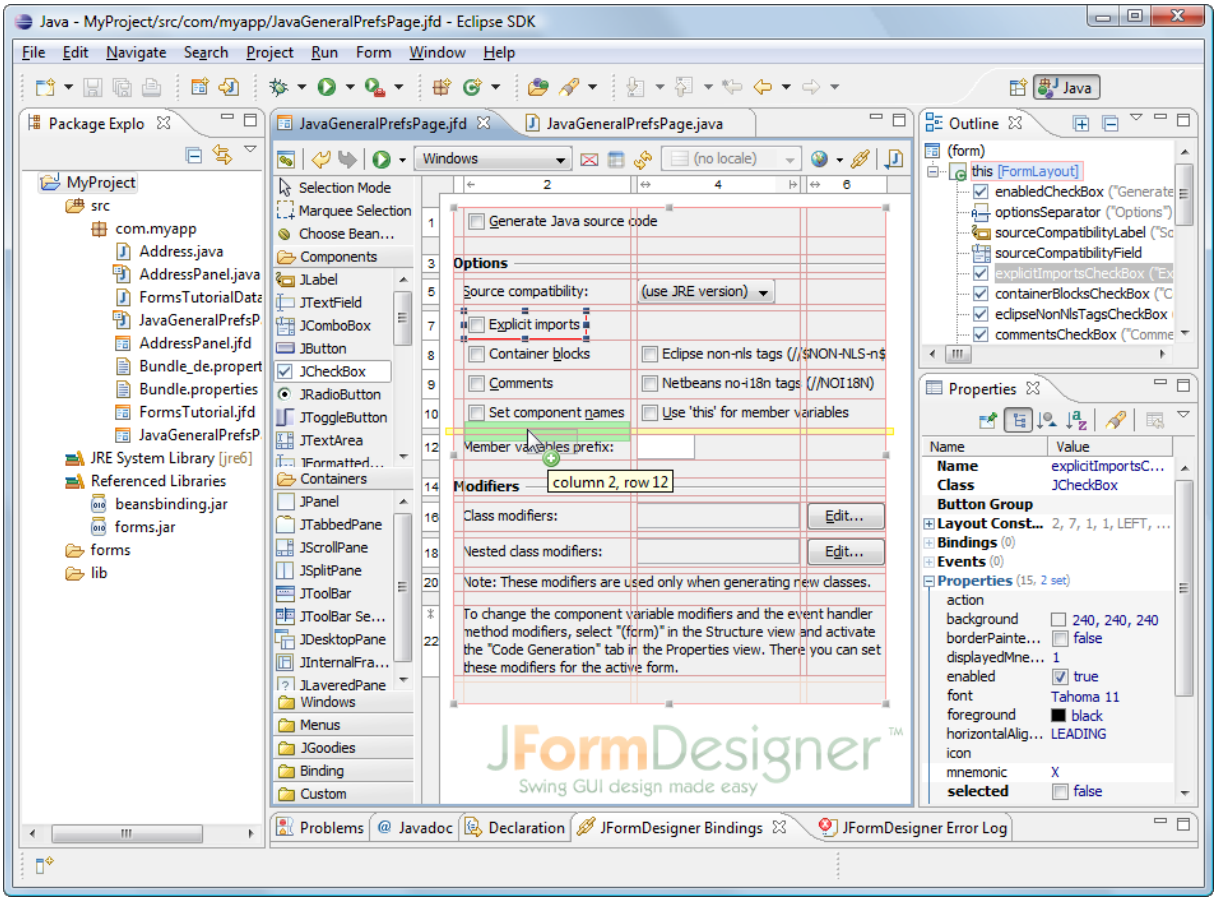
Za rad sa programskim okružjem Swingom, koje će detaljnije biti obrađeno u sljedećem poglavlju, potrebno nam je određeno integrirano razvojno okruženje (*eng. integrated development environment* - IDE). Od poznatijih IDE-a za rad sa Javom imamo Eclipse, NetBeans, IntelliJ IDEA i Jdeveloper. Sva četiri navedena IDE-a kratko će biti opisana u ovom poglavlju zajedno s iskustvima korisnika.[3] [4] [5] [6]

Svi spomenuti IDE-i su dostupni na svim poznatijim operativnim sustavima, što znači da imamo verzije za Microsoft Windows, distribucije Linuxa, Solaris, Mac OS. ...

2.1 ECLIPSE

Eclipse je besplatan IDE koji je strjelovito brzo preuzeo Java industriju. Arhitektura mu se temelji na dodacima (*eng. plugin*) čime postaje vrlo proširiv i prilagodljiv. Eclipse je izgrađen na vlastitoj SWT (eng. Standard Widget Toolkit) GUI biblioteci, odnosno programskom alatu za izradu standardnih programčića. Eclipse se ističe u refakturiranju (automatskoj promjeni atributa objekta (naziv, tip i sl.) na trenutno otvorenom projektu – što znači da će prilagoditi nove izmjene na svim datotekama unutar projekta), te podršci u Java platformi za poduzeća (*eng. Java 2 Platform, Enterprise Edition* - J2EE – okruženje za izgradnju i razvoj specifičnih Web aplikacija za poduzeća), gdje još u novoj verziji Eclipse-a dolazi i *WindowsBuilder*, alat koji je mješavina Swinga i SWT dizajnera. Budući da nova verzija Eclipsa zvana Luna zaostaje u mogućnostima za konkurentnim IDE-ima gdje su čak i izgubili podršku Googlea za dodatak koji omogućuje izradu Android aplikacija, stoga Eclipse neće biti razmatran u daljnjem tekstu. Izgled Eclipsa možemo vidjeti na Slici 2.

[3] http://faq.programmerworld.net/programming/best-java-ide-review.html
[4] http://teks.co.in/site/blog/eclipse-vs-netbeans-better-ide-android-app-development/
[5] https://dzone.com/articles/netbeans-ide-and-intellij-idea
[6] https://jaxenter.com/eclipse-netbeans-or-intellij-which-is-the-best-java-ide-107980.html

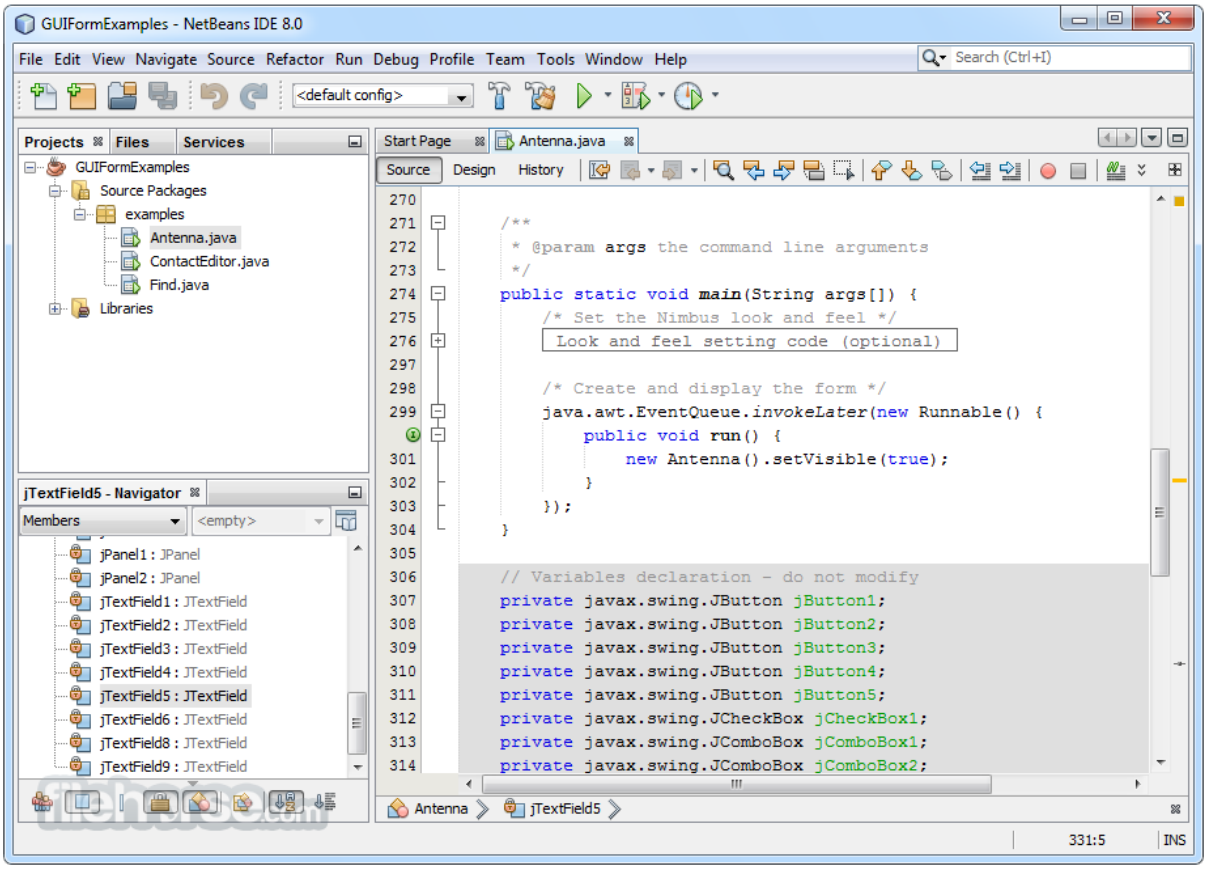


Slika 2: Eclipse. Izvor: marketplace.eclipse.org

2.2 NETBEANS

NetBeans je besplatan IDE kojeg je stvorio Sun Microsystems a kasnije ga je preuzeo od Oraclea, u kojem je postao softver otvorenoga koda. NetBeans je također izgrađen na arhitekturi dodataka kao i Eclipse, i budući da je to standardni IDE kojega preporučuje Oracle, prilično je respektabilan. Kako je potpuno modularan, i sve IDE funkcije dolaze u paketima odnosno modulima, nudi jednostavnu integraciju s kontrolnom verzijom softvera.

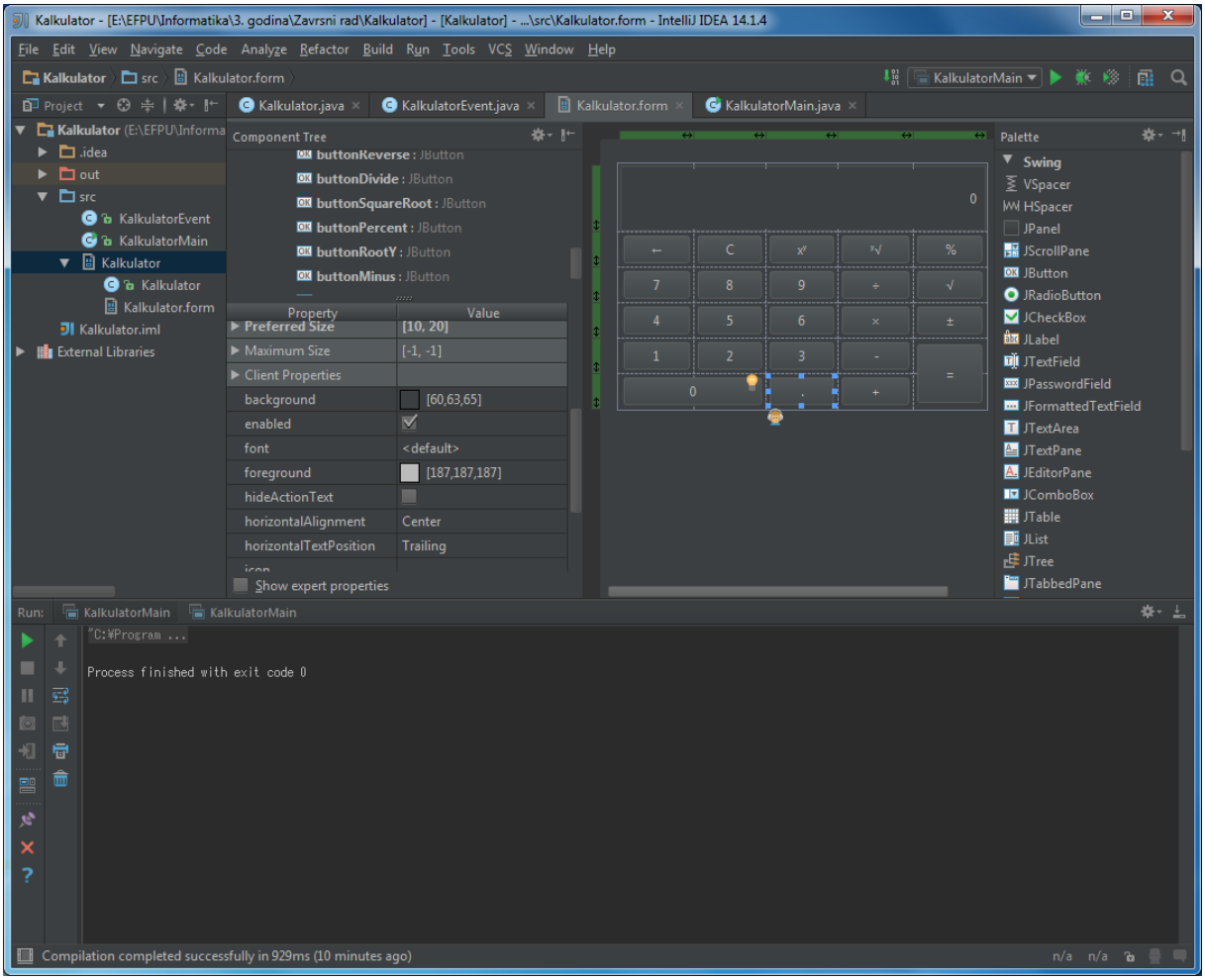
Ima potpunu podršku za sve Java platforme (*JavaFX*, *JavaME*, *JavaEE*, *JavaSE*), JSP , XML / XHTML, alat vizualnog dizajna, generatora koda i CVS podršku.



Slika 3: NetBeans. Izvor: www.filehorse.com

2.3 IntelliJ IDEA

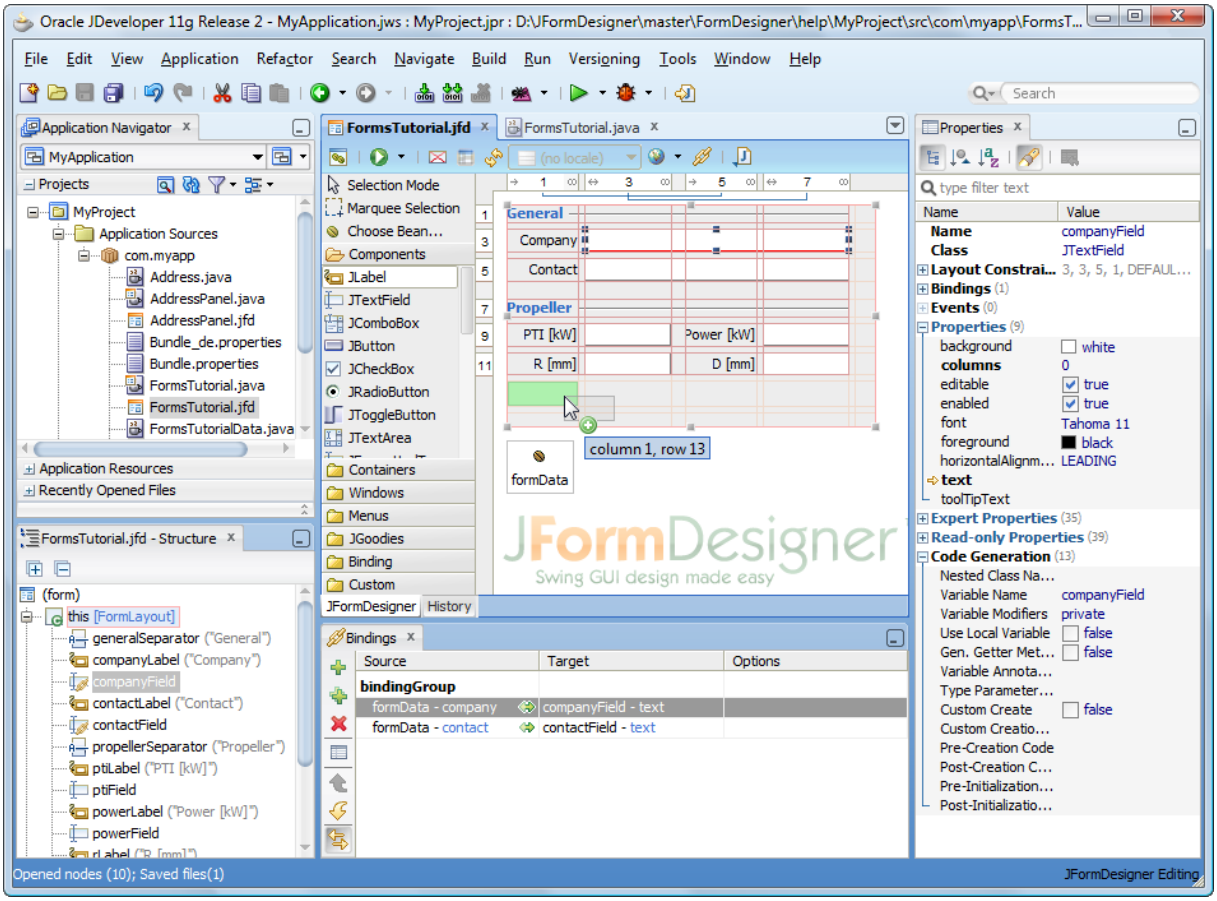
IntelliJ IDEA je komercijalni IDE sa puno vjernih i zadovoljnih korisnika. Postoji još i besplatna verzija sa smanjenim mogućnostima. Ima odličnu J2EE (*eng. Java 2 Platform, Enterprise Edition*) i GUI podršku, te je IDE još i proširiv putem dodataka. Njegova glavna značajka je izvanredno, a po tvrdnjama mnogih i najbolje refakturiranje koda. Nudi robusnu kombinaciju razvojnih alata u jednom paketu. U kompletu s inteligentnim Java editorom, koji pomaže u kodiranju i naprednim alatom za automatizaciju koda, IDEA omogućuje Java programerima da povećaju svoju produktivnost te ujedno i smanje vrijeme potrošeno na rutinske zadatke.



Slika 4: IntelliJ IDEA. Izvor: Slika zaslona autora.

2.4 JDEVELOPER

Oracle JDeveloper je IDE sa podrškom od početka do kraja (eng. end-to-end) za modeliranje, razvoj, ispravljanje pogrešaka, optimiziranje, i postavljanje Java aplikacija i web usluga. Oracle JDeveloper 10g uvodi novi pristup razvoja J2EE sa značajkama koje omogućuju vizualno i deklarativno razvijanje. Inovativno programsko okruženje OAF (Oracle Application Framework), koji nudi dodatne biblioteke za razvoj na korporacijskim aplikacijama te ujedno pojednostavljuje i komplementira razvoj sa J2EE. Ima dobru podršku i vizualizaciju za rad sa bazama podataka.



Slika 5: JDeveloper. Izvor: www.formdev.com

2.5 IZABRANO INTEGRIRANO RAZVOJNO OKRUŽENJE

Budući da svaki IDE ima svojih prednosti i mana, te se više svodi na subjektivni izbor pojedinca. Također, u izboru može presuditi i neka mogućnost koja nam je potrebna a pojedini IDE ima tu mogućnost bolju od drugih, Od velike pomoći mogu biti i različiti članci o razvojnim okruženjima kao i iskustva iskusnih korisnika sa raznih foruma.

Oko Eclipse bije glas da ima čudnih grešaka i da se ponekad neočekivano ruši, dok se Jdeveloper sporije učitava ali ima općenito dobru podršku za baze podataka koje nisu u fokusu ovog rada. Izbor se na kraju svodi između NetBeans-a i IDEA-e, ali budući da je potonja imala više pozitivnih komentara i članaka u nastavku ovog rada koristit ćemo IDEA-u. Međuostalom i Google ju je izabrao za razvoj Androida.

3 SWING

Swing je Javin programski alat za grafičko korisničko sučelje. Dio je velike API biblioteke pod nazivom Java Foundation Classes (JFC). Paket javax.swing (i njegov velik broj podpaketa) sadrži klase za sučelje kao što su prozori, tipke, kombinirani okviri, stabla, tablice i izbornici – sve što je potrebno da bi se izgradila moderna korisnička aplikacija.

Komponente u Swingu, kao što su tipke i kombinirani okviri, su implementirani u samoj Javi. To znači da neovisno o platformi koju koristimo, npr. standardna tipka u Swingu uvijek izgleda isto. Osim toga, Swing ima dodatni API koji omogućuje izgled komponenata ovisno o operativnom sustavu.

U Javi komponenta je osnovni objekt korisničkog sučelja. Sve što se vidi prikazano u Javinoj aplikaciji je komponenta. To su npr. prozori (eng. Windows), paneli (eng. Panels), tipke (eng. Buttons), potvrdni okviri (eng. Check boxes), pomični trakovi (eng. Scrollbars), liste (eng. Lists), izbornici (eng. Menus), tekst polja (eng. Text fields) i itd. Komponenta se inače postavlja u spremnik, koji se onda pomoću upravitelja razmještaja pozicionira za točan prikaz u spremniku tako da se ne preklapa sa drugim komponentama.

Glavna karakteristika komponenti je njezina reakcija na događaje pokrenutih od strane korisnika. Kada korisnik napravi akciju (kao što je pritiskanje lijeve tipke miša) unutar komponente, Swingina dretva isporuči objekt događaja koji opisuje što se dogodilo. Tada se događaj isporučuje objektima koji su registrirani kao slušatelji za taj tip događaja kod te određene komponente. Tako na primjer, kada korisnik klikne na komponentu tipke, ta tipka generira ActionEvent objekt. Da bi se takvi događaji mogli primiti, postoje objekti koji su registrirani kao ActionListener.

Događaji se isporučuju dozivanjem određenih metoda rukovatelja događaja (eng. Event Handler) unutar objekta primatelja (takozvani “slušatelj”). Objekt slušatelj prima specifični tip događaja preko metoda njegovog sučelja (na primjer preko actionPerformed() metode od ActionListener sučelja) za tipove događaja za koje je registriran. Specifični tipovi događaja pokrivaju različite kategorije korisničkih interakcija sa komponentama. Na primjer, MouseEvents opisuju aktivnosti miša unutar prostora komponente, KeyEvents opisuju pritisnute tipke dok događaji više razine (kao što je ActionEvents) upućuju da je komponenta korisničkog sučelja odradila svoj posao.

3.1 SPREMNIK

Spremnik je vrsta komponente koja sadrži i upravlja ostalim komponentama. Tri općenito najkorištenija spremnika su JFrame, JPanel i JApplet.

JFrame je prozor najviše razine koju vidimo na ekranu. On je podklasa od JWindow koji ima naslovnu traku, tipke za upravaljanje prozora (zatvori, minimiziraj, maksimiziraj) i okvir. Možemo povlačiti okvir okolo po ekranu i promijeniti veličinu korištenjem uobičajenih kontrola za prozor.

JWindow je jednostavno čisto, grafičko platno koji se prikazuje na ekranu. Prozori nemaju nabora ni rubova; oni se najčešće koriste kao skočni prozori i još u situacijama kada se padajuće komponente, kao što su izbornici i kombinirani okviri, proširuju izvan njihovog prethodnog okvira.

JApplet klasa je vrsta spremnika koji služi kao temelj za aplete koji se pokreću unutar web preglednika. Kao i drugi spremnici, JApplet može sadržavati druge komponente korisničkog sučelja. Osim toga, može se upotrijebiti JComponent klasu direktno koja je najtemeljniji objekt u Swingu gdje su sve komponente izvedene iz njega, odnosno kažemo da ga nasljeđuju. To znači da je JComponent korijen hijerarhije Swingovih komponenti. Osim JComponent-a možemo upotrijebiti Jpanel koji je detaljnije opisan u nastavku rada, da zadrži komponente unutar drugog spremnika.

Spremnik održava listu "djece" komponenti kojima upravlja i ima metode koje mu to omogućuju. Treba naglasiti da se ta povezanost odnosi na vizualnu hijerarhiju a ne na hijerarhiju tipa podklasa i nadklasa. Većina komponenti nisu korisne same za sebe dok se ne dodaju u spremnik i ne prikažu. Metoda add() u spremnik klase dodaje komponentu u spremnik. Zatim se ta komponenta može prikazati u prikaznom području spremnika i pozicionirati preko upravitelja razmještaja. Komponenta se može ukloniti iz spremnika metodom remove(). Naravno metode add() i remove() se koriste ako komponente u spremnik dodajemo programskim putem a ne preko dizajnera, jer inače tamo možemo vizualno označiti komponentu u spremniku i izbrisati je.

3.2 UPRAVITELJ RAZMJEŠTAJA

Upravitelj razmještaja je objekt koji kontrolira položaj i veličinu komponente unutar prikaznog prostora spremnika. On je kao upravitelj prozora u sustavu prikaza; kontrolira gdje komponente idu i koliko su velike. Svaki spremnik ima vlastitog zadanog upravitelja

razmještaja, ali se može instalirati i neki drugi pozivom spremnikove metode setLayout() ili podešavanjem atributa na željeni izbor unutar dizajnera.

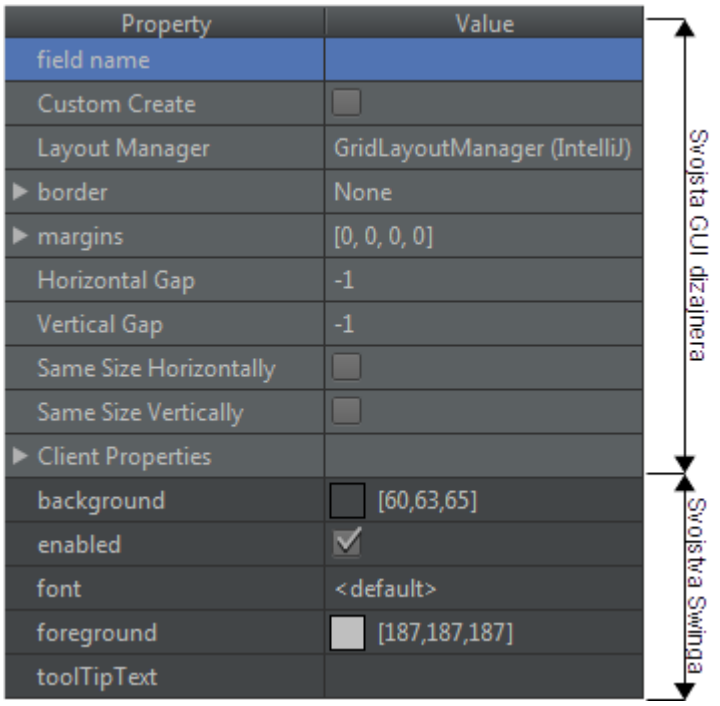
Swing dolazi sa nekoliko upravitelja razmještaja koji implementiraju uobičajene sheme razmještaja. Zadani upravitelj razmještaja za JPanel je FlowLayout, koji pokušava postaviti objekte na njihovu željenu veličinu s lijeva na desno i od gore prema dolje unutar spremnika. Zadani upravitelj razmještaja za JFrame je BorderLayout, koji postavlja objekte na specifičnu lokaciju unutar prozora, kao što su sjever, jug i centar. Još jedan upravitelj razmještaja je GridLayout koji raspoređuje komponente u pravokutnu mrežu. Najopćenitiji (i najteži za korištenje) upravitelj razmještaja je GridBagLayout koji dozvoljava korisniku da definira razmještaj kao što se to radi sa HTML tablicama.

Kada programski dodajemo komponentu u spremnik korištenjem jednostavnog upravitelja razmještaja, koristit ćemo verziju metode add() koji prima jednu komponentu kao argument. Ali ako koristimo upravitelj razmještaja koji koristi "ograničenja", odnosno prima više argumenata, kao što je BorderLayout ili GridBagLayout, moramo dodatno definirati informacija o tome gdje ćemo staviti novu komponentu. Za to možemo upotrijebiti verziju koja koristi objekt ograničenja. Npr. da bi se postavila komponenta na vrh spremnika koji koristi BorderLayout upravitelj, trebalo bi izvršiti:

```
myContainer.add(myComponent, BorderLayout.NORTH);
```

U ovom slučaju, objekt ograničenja je statični atribut NORTH. GridBagLayout koristi mnogo složenije objekte ograničavanja za određivanje pozicije komponente.

3.3 INSPEKTOR SVOJSTVA



Slika 6: Inspektor Svojstva u IDEA-i. Izvor: Slika zaslona autora.

Postoje dvije grupe svojstava: svojstva GUI dizajnera i svojstva Swinga. Prva su na prethodnoj slici 2 označena svjetlo sivom bojom, a druga tamno sivom. Preko svojstva GUI dizajnera komponentu možemo dodatno urediti i pozicionirati da se ne preklapa sa drugim komponentama ili da napravimo određeni razmak među njima zbog ljepšeg izgleda. Ova svojstva na slici vrijede za spremnike, a druge komponente mogu imati ista ili neka druga svojstva. Zbog jednostavnosti i preglednosti navest ćemo sva svojstva moguća za prvu i drugu grupu svih komponenti. Svojstva koja pronalazimo kroz IDEA dizajner su sljedeća:

- Field name - služi za imenovanje komponente tako da joj možemo kroz programsku logiku pristupiti direktno kao objektu te napraviti nešto s njom.
- Custom Create - omogućuje nam da zaobiđemo standardni izgled ili ponašanje komponente te da ju implementiramo kroz vlastiti kod i logiku.
- Layout Manager - definira sheme razmještaja komponenti u spremniku. U ovom slučaju JPanel ima mrežni oblik razmještaja.
- Border - definira rubove, odnosno granice oko spremnika; možemo izabrati bez granice, linija, udubljeno, udubljena donja i desna, udubljena gornja i lijeva.
- Margins – definira margine oko spremnika, odnosno koliko će biti razmaka između spremnika i druge komponente ili spremnika.

- Horizontal Gap - dodavanje horizontalnog razmaka između spremnika i drugih komponenti, s time da za razliku od margine dodaje razmak po cijeloj horizontalnoj duljini spremnika; -1 označuje da nema razmaka.
- Vertical Gap - dodavanje vertikalnog razmaka između spremnika i drugih komponenti, s time da za razliku od margine dodaje razmak po cijeloj vertikalnoj duljini spremnika; -1 označuje da nema razmaka.
- Same Size Horizontally - razvlači komponentu horizontalno maksimalno koliko upravitelj razmještaja dopusti.
- Same Size Vertically - razvlači komponentu vertikalno maksimalno koliko upravitelj razmještaja dopusti.
- Horizontal Size Policy - pravila za razvlačenje komponente horizontalno. Pravila su sljedeća: može se smanjiti (Can Shrink), može rasti (Can Grow), želi rasti (Want Grow). Razlika između može rasti i želi rasti je da potonja forsira širenje komponente do kraja za razliku od može rasti gdje se to ne mora dogoditi.
- Vertical Size Policy - vrijedi kao i za Horizontal Size Policy samo što se odnosi na vertikalno.
- Horizontal Align - horizontalno poravnavanje komponente. Izbor je lijevo, desno, centar.
- Vertical Align - vertikalno poravnavanje komponente. Izbor je gore, dolje, centar.
- Indent - definiramo razmak brojčano sa lijeve strane komponente što je pomiče u desno.
- Minimum/Preferred/Maximum Size - definiramo veličinu komponente tako da brojčano izkažemo njezinu duljinu i širinu (primjer [50,50]). Uvijek će se prvo gledati vrijednosti pod Preferred Size. Zadana vrijednost je [-1,-1] što označuje automatsku veličinu ovisnu o upravitelju razmještaja.

Druga svojstva u tamnijoj sivoj boji su svojstva Swinga. Budući da klasa komponente nasljeđuje svojstva od bazičnog objekta komponente, sve komponente će imati određena zajednička svojstva. Svojstva Swinga u IDEA-i su sljedeća:

- Background - definira pozadinske boje komponente ili spremnika. Boja se definira brojčano u obliku RGB (Red, Green, Blue) od 0 do 255 ili heksadecimalno. Također ponuđena je i paleta sa bojama na kojoj možemo klikom izabrati onu koja nam

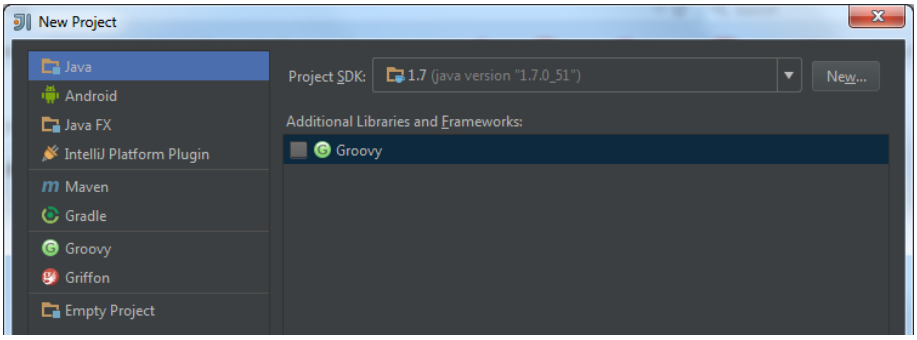
odgovara. Ako je komponenta ili spremnik iza nekog drugog onda učinak mijenjanja boje spremnika ili komponente nećemo vidjeti.

- Enabled - jednostavan način za uključivanje ili isključivanje objekta da se prikazuje iz dizajnera i programske logike. Na primjer, ako vidimo da nam neki objekt radi problema ili ga jednostavno želimo privremeno ugasit, imamo vrlo jednostavan način da to učinimo tako da maknemo kvačicu iz kvadratića čime se objekt deaktivira.
- Font - definira željeni font sa liste mogućih fontova, te njegovu veličinu i dodatni efekt. Zadani font u InteliJ IDEAi Segoe UI.
- Foreground - definira boju fonta na objektu. Boja se definira brojčano u obliku RGB (Red, Green, Blue) od 0 do 255 ili heksadecimalno; vrijede ista pravila i izbor boja kao kod backgrounda.
- ToolTipText - tekst koji nam se prikazuje kada držimo kursor miša na komponenti.
- Icon - po izboru se može dodati ikona koja će se prikazati na komponenti.
- Text – upisuje se tekst koji se prikazuje na komponenti.
- Columns - kod komponenti za tekst možemo definirati broj stupaca koji nam razdvaja tekst.
- Editable - možemo učiniti da je tekst kod tekstne komponente izmjenjiv ili da nije.
- DropMode - Ako uključimo padajući izbor na komponenti, moramo odrediti na koji način da se izbor interpretira. Zadani početni način rada je USE_SELECTION. Izabrana stavka u listi se pomiče za potencijalno puštanje na izabranu lokaciju, gdje prilikom puštanja izabrana stavka se pomiče na to mjesto. Način rada ON, također pomiče izabranu stavku ako za potencijalno puštanje na izabrano mjesto, samo za razliku od prvoga ništa se ne događa na puštanju izabrane stavke. Način rada INSERT, korisnik je ograničen na izbor slobodnog mjesta između stavaka, prije ili nakon, odnosno nije dozvoljeno označavanje već postojeće stavke u listi. Način rada ON_OR_INSERT je jednostavno kombinacija tih istih.

4 JAVA PROJEKT U INTELLIJ IDEA

Za početak moramo provjeriti da li na računalu imamo instaliran Java razvojni alat (*Java Developer Kit* - JDK) koji nam je neophodan za razvijanje aplikacija u Javi.

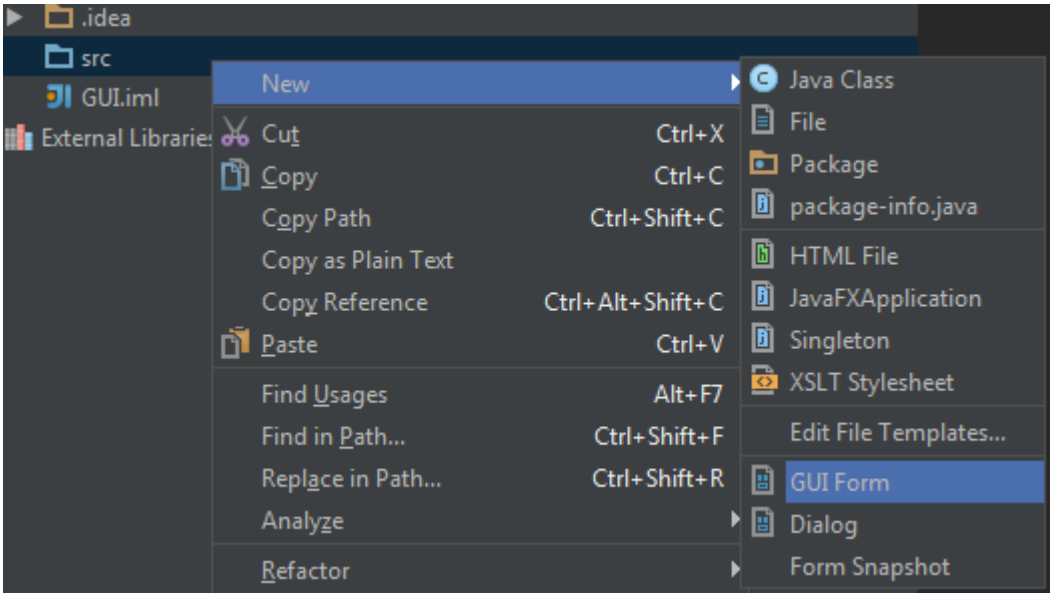
Pokretanjem IDEA otvara nam se početni prozor u kojem za stvaranja novog projekta biramo “Create New Project”.



Slika 7: Stvaranje novog projekta u IDEA-i. Izvor: Snimka zaslona autora.

Zatim u sljedećem prozoru, označimo da ćemo imati Java projekt, te dodamo pod “Project SDK:” direktoriji sa instaliranim JDKom. Ostale postavke ćemo ostaviti kakve jesu te odlazimo na iduće korake dok nas ne pita za ime projekta, gdje ga i upišemo i odredimo direktorij u kojemu ćemo sačuvati projekt na disku.

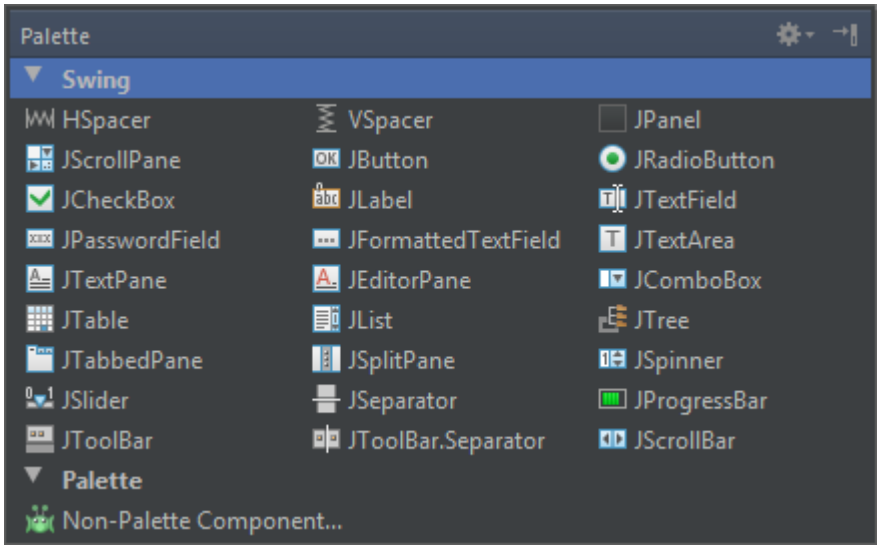
Nakon što je generiran Java projekt, potrebno je u “src” direktorij dodati novu GUI formu, klikom na desnu tipku miša na isti direktorij. U sljedećem koraku je bitno da bude označena opcija na “Create bound class”, te da imenujemo novu formu.



Slika 8: Stvaranje nove GUI Form-e u IDEA-i. Izvor: Snimka zaslona autora.

5 SWING KOMPONENTE U INTELLIJ IDEA

Na desnoj strani IDEA-e nalazi se lista svih Swingovih komponenti. U nastavku slijedi kratak opis komponenti te nekih njihovih atributa



Slika 9: Swing komponente. Izvor: Snimka zaslona autora.

5.1 JPANEL

Prilikom stvaranja forme grafičkog korisničkog sučelja (eng. *GUI Forms*) automatski nam se stvori i komponenta JPanel koja služi kao spremnik koji grupira druge komponente unutar JFrame ili drugim JPanelima.

5.2 HSPACER I VSPACER

HSpacer nam služi da pozicionira komponentu na prvo početno horizontalno vizualno mjesto koje upravitelj razmještaja dopusti. Za Vspacer vrijedi isto samo što namješta vertikalno. Kad preko dizajnera premjestimo komponentu na neko drugo mjesto, to možemo učiniti samo po pravilima upravitelja razmještaja. Da bi napravili razmak između komponenta ili ih jednostavno grupirali, dizajner će nam iskoristiti HSpacer i VSpacer da natjera upravitelja razmještaja na ponovnu rekalkulaciju razmještaja sa dodatnim ograničivaćem gdje se komponenta pozicionira po zahtjevu korisnika.

5.3 JSCROLLPANE

JScrollPane je spremnik koji može držati jednu komponentu. Drugim riječima, on omotava drugu komponentu. Inače po zadanoj vrijednosti, ako je omotavana komponenta

veća nego JScrollPane onda nam se pojavljuju pomični trakovi. JScrollPane upravlja događajima pomičnih trakova i prikazuje adekvatni dio sadržane komponente.

JScrollPane ima svoj upravitelj razmještaja, koji se ne može mijenjati i smješta samo jednu komponentu. To nije baš neko ograničavanje jer ako želimo staviti više komponenti u njega, jednostavno prije toga sve komponente ubacimo u JPanel, izaberemo upravitelj razmještaja koji želimo, i na kraju sve ubacimo u JScrollPane.

Kada stvorimo JScrollPane, odredimo uvjete po kojima će se pomični trakovi prikazivati. To se naziva *scrollbar display policy*; odvojena polica se koristi za horizontalne i vertikalne pomične trakove. Možemo izabrati sljedeće konstante koje definiraju ponašanje police za svaki pomični trak:

- HORIZONTAL_SCROLLBAR_AS_NEEDED, VERTICAL_SCROLLBAR_AS_NEEDED - prikazuje pomični trak samo ako omotana komponenta ne stane
- HORIZONTAL_SCROLLBAR_ALWAYS, VERTICAL_SCROLLBAR_ALWAYS - uvijek prikazuje pomični tak, neovisno o sadržanoj veličini komponente.
- HORIZONTAL_SCROLLBAR_NEVER, VERTICAL_SCROLLBAR_NEVER - nikad ne prikazuje pomični trak, čak i ako sadržajna komponenta neće stati. Ako koristimo ovu policu, trebali bi odrediti neki drugi način upravljanja JScrollPaneom.

Po zadanim vrijednostima, police su HORIZONTAL_SCROLLBAR_AS_NEEDED i VERTICAL_SCROLLBAR_AS_NEEDED.

5.4 JLABEL

Komponenta koja prikazuje tekst koji napišemo. Kao što njezin naziv to govori, oznaku možemo iskoristiti ako želimo dodatno pojasniti neku drugu komponentu ili jednostavno označiti dodatnim tekstom. Nema nikakvih specijalnih događaja vezanih za oznake; sve što se može učiniti je da se odredi centriranje teksta, koji kontrolira poziciju teksta unutar prikaznog prostora oznake.

Dodatno što nam nudi Swing svojstva oznake za razliku od standardne komponente je da po izboru možemo definirati za koju komponentu se ta oznaka odnosi (svojstvo labelFor).

5.5 JBUTTON

JButton generira ActionEvent kada ga korisnik pritisne. Da možemo primiti te događaje, program nam mora registrirati ActionListener, koji mora implementirati metodu actionPerformed(). Argument proslijeđen toj metodi je objekt događaja.

Još jedna stvar koja vrijedi sa sve komponente koji generiraju događaj akcije. Java nam dopušta da definiramo string komande akcije za tipke, i druge komponente kao što su članovi izbornika, koji mogu generirati događaje akcije. Komanda akcije je String koji služi za identifikaciju komponente koja je poslala događaj. Po zadanim vrijednostima, komanda akcije JButtona je ista kao i njezina oznaka (tekst koji upišemo koji nam se pojavi na komponenti); dodan je u događaje akcije tako da možemo lakše raspoznati sa koje je tipke došao događaj.

Slušač akcije možemo dodati kroz dizajner tako da označimo komponentu te pritisnemo Ctrl+O i izaberemo ActionListener. Evo i primjera koda kako to izgleda:

```
button1.setActionCommand("Da");
button1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Da")) {
            //kod koji ce se izvrsavati kad se pritisne button1 "Da"
        }
    }
});
```

Slika 10: Generirani kod dizajnera IDEA-e. Izvor: Snimka zaslona autora.

Da dobijemo komandu akcije iz događaja akcije moramo pozvati metodu događaja getActionCommand(). Možemo promijeniti komandu akcije tako da pozovemo metodu tipke setActionCommand(). Dobra je praksa da se komanda akcije postavi eksplicitno; to pomaže na način da se kod ne sruši iz razloga što je netko promijenio naziv tipke.

5.6 JRADIOBUTTON

Radijska tipka je dio grupe tipaka gdje izborom jednu od tipki označava tu i ujedno gasi druge. Radijske tipke imaju više smisla kada su povezane zajedno korištenjem instance klase ButtonGroup. Dodajemo tipke metodom add() u grupu tipki (ButtonGroup) da ih napravimo međusobno isključive. Neobična stvar kod objekta ButtonGroup je što bi očekivali da je ili komponenta ili spremnik, međutim nije; već je pomoćni objekt koji omogućuje da je samo jedna radijska tipka označena istovremeno.

5.7 JCHECKBOX

Izborni kvadratić je označena preklopna tipka. Svaki put kada korisnik klikne na njega, njegovo stanje se prebacuje između označenog i neoznačenog. Swing implementira izborni kvadratić kao specijalnu vrstu tipke. JCheckBox šalje događaje jedinica (ItemEvents) kada je pretisnut. Budući da je izborni kvadratić vrsta tipke, također šalje događaj akcije kada je označen. Za nešto kao što je izborni kvadratić, možemo si priuštiti kasnije provjere za kada korisnik izvrši akciju. Na primjer, kada se ispunjuje forma nas samo zanimaju odgovori korisnika na kraju kada se pritisne tipka za slanje i forma pošalje.

5.8 JTEXTFIELD

JTextField je komponenta koja omogućuje korisniku da upiše malu količinu teksta, točnije jednu liniju teksta. Kada korisnik završi sa unosom teksta (najčešće pritiskom tipke Enter), tekstno polje okine događaj akcije.

JTextField ima metodu da poveže string komandne akcije sa događajem akcije koji se pokrenuo. Točnije, iskoristiti će komandni string sa metodom setActionCommand ako nije null, inače će iskoristiti tekst polja.

Horizontalno poravnavanje može biti postavljeno na lijevo poravnato, poravnato, centrirano, desno poravnato. Desno poravnavanje je korisno ako je potrebna veličina tekstnog polja manja nego veličina koja joj je dodijeljena. To se može postaviti pomoću metoda setHorizontalAlignment i getHorizontalAlignment.

Okidanje događaja VK_ENTER (pritiskanjem tipke Enter) ovisi o tome da li teksto polje ima registrirane slušatelje događaje. Ako ima, onda VK_ENTER rezultira na slanju događaja akcije na strani slušatelja.[7]

5.9 JPASSWORDFIELD

JPasswordField se ponaša kao JTextField (u biti je podklasa), samo što se svaki utipkani znak pretvara u sakriveni znak koji je tipično zvjezdica. Stvaranje i korištenje JPasswordField je isto kao i kod JTextField. Ako nam se zvjezdica ne sviđa možemo to promijeniti tako da pozovemo metodu setEchoChar() i postavimo na željeni znak.

Inače bi koristili getText() da dobijemo upisani tekst, ali kod JPasswordField umjesto toga bi trebali koristiti metodu getPassword(). Metoda getPassword() vraća polje znakova

[7] http://docs.oracle.com/javase/7/docs/api/javax/swing/JTextField.html

umjesto objekt String. To je napravljeno iz razloga što su polja znakova teža za otkrivanje pomoću raznih memorijskih sniffera za lozinke i mogu se obrisati direktnije i lakše nego String objekt.

5.10 JFORMATTEDTEXTFIELD

JFormattedTextField komponenta omogućuje eksplicitnu podršku za obrađivanje složenih formatiranih vrijednosti kao što su brojevi i datumi. Komponenta se ponaša na neki način kao JTextField, samo što prihvaća objekt koji je specifično oblikovan i upravlja složenom tipom objekata, kao što su Date, Integer, Float i sl., preko metoda setValue() i getValue().

5.11 JTEXTAREA

JTextArea je jednostavan višelinijski uređivač teksta. Model za tekstne komponente je objekt Document. Kada dodajemo ili brišemo tekst iz JTextField ili JTextArea, odgovarajući Document se mijenja. To znači, ono što stvara događaje vezane za tekst kada netko nešto izmjeni u polju uređivača je modul dokument a ne komponenta koju vidimo. Također, može se jednostavno napraviti da imamo više od jedne vizualne komponente koje koriste isti podatkovni model dokumenta.

Po zadanim postavkama, JTextField i JTextArea su izmjenjivi; može se tipkati i mijenjati tekst u oba dvije komponente. Može se izmijeniti da samo prikazuju tekst na način da se pozove metoda setEditable(false). Obje komponente podržavaju selekcije. Selekcija je raspon teksta koji je istaknuti za kopiranje, izrezivanje ili zaljepljivanje.

5.12 JTEXTPANE

JTextPane je podklasa od JTextComponent koja nam omogućuje da radimo sa tekstom što god hoćemo. Bazične tekstne komponente, JTextField i JTextArea, su ograničene na jedan font u jednom stilu i boji. Ali JTextPane može prikazivati više različitih fontova i više različitih stilova na jednoj komponenti. Također imamo podršku za isticanje, umetanje slike i druge napredne mogućnosti.

Da napravimo stilirani tekst, trebamo jednostavno povezati grupu svojstva teksta sa različitim dijelovima teksta dokumenta. Swing dodaje klase i metode za manipuliranje grupom svojstva nad tekstom, kao što je određivanje masnog otiska ili drukčije boje. Svojstva

se nalaze u klasi SimpleAttributeSet; te grupe svojstva se upravljaju sa statičnim metodama u klasi StyleConstants.

5.13 JEditorPane

JEditorPane, također podklasa od JTextComponent koja može prikazivati HTML i RTF (Rich Text Format) dokumente i omogućuje dodatni programski okvir za potporu drugih tipova sadržaja. Ima dodatno još jedan tip događaja više za razliku od drugih komponenti, a to je HyperlinkEvent. Podtipovi ovog događaja se opaljuju kada miš uđe, izađe ili klikne na hiperlink. U kombinaciji sa komponentinim prikaznim mogućnostima HTMLa, prilično je lagano za napraviti jednostavan web preglednik.

5.14 JList

JLists i JComboBoxes su korak dalje u evolucijskom lancu od JButtons i JLabels. Liste omogućuju korisniku da izaberu iz grupe alternativa. Mogu se namjestiti da forsiraju jednostruku selekciju ili da se dozvoli višestruki izbori. Uobičajeno je da se prikazuje samo mala grupa izbora; pomični trak omogućuje korisniku da se pomiče izbore koji još nisu vidljivi. Korisnik može označiti stavku klikanjem na nju. Može proširiti raspon izbora stavki držajući Shift tipku pritisnutom i klikanjem do druge stavke. Za diskontinuirane izbore, korisnik može držati tipku Control umjesto Shifta.

Kao druge komponente u Swingu, liste i kombinirani okviri imaju podatkovni model koji je različit od vizualne komponente. Lista još ima i izborni model koji kontrolira kako se izbori mogu napraviti na listi podataka.

Liste i kombinirani okviri su slični zato što imaju slične podatkovne modele. Svaki od njih je jednostavno polje sa mogućim izborima. Ta sličnost je reflektirana u Swingu; tip podatkovnog modela JComboBox je podklasa od tipa podatkovnog modela korištenog kod JList.

5.15 JComboBox

Kao što smo već rekli, JComboBox i JLists su slični jer koriste isti podatkovni model. Točnije, kombinirani okvir je ukrštanje između tekstnog polja i liste. Prikazuje jednu liniju teksta (mogućnost i sa slikom) i strelicu za dolje na jednoj strani. Ako kliknemo na strelicu, kombinirani okvir otvori padajući izbornik i prikaže listu izbora. Možemo selektirati izbor klikom na njega. Kada napravimo selekciju, padajući izbornik se zatvori; lista nestane i novi

selektirani izbor je prikazan u tekstnom polju. Ako postavimo da je kombinirani izbornik izmjenjiv pozivanjem metode setEditable(true) onda kombinirani okvir nudi izmjenjivo polje gdje korisnik može upisati vrijednost.

5.16 JTABLE

JTable klasa predstavlja vizualnu komponentu tablice. JTable je temeljen na tabličnom modelu. JTable ima jedan konstruktor koji stvara zadani model tablice iz dodijeljenog polja podataka. Treba samo nadodati imena zaglavlja stupaca i 2D polje objekata koji reprezentiraju tablične podatke. Prvi indeks označuje tablični red a drugi indeks označuje stupce.

Imamo sljedeća zadana ponašanja sa interakcijom nad tablicom:

- *Zaglavlja stupaca* - JTable ima automatsko formatiranje zaglavlja stupaca drugačije nego ćelije u tablici. Vidljivo je da nisu dio prostora tabličnih podataka.
- *Pretek ćelije* - Ako je podatak u ćeliji predugačak da stane u nju, automatsko se smanjuje i prikazuje sa tri točke (...).
- *Izbor retka* - Može se kliknuti na bilo koju ćeliju i označiti cijeli redak. To ponašanje je upravljivo gdje možemo označiti jednu ćeliju, cijeli redak, cijeli stupac ili neku kombinaciju tih istih. Da se namjesti ponašanje selekcije tablice koristimo metode setCellSelectionEnabled(), setColumnSelectionAllowed() i setRowSelectionAllowed().
- *Editiranje ćelije* - Dvoklik na ćeliju otvara nam se za editiranje; pojavljuje se kursor u ćeliji. Možemo tipkati direktno u ćeliju da mijenjamo njezine podatke.
- *Određivanje veličine stupaca* - Ako pozicioniramo kursor miša između dva zaglavlja stupca, kursor nam se mijenja u posebnu vrstu strelice. Kliknemo i povučemo da promijenimo veličinu stupca. Ovisno o tome kako je tablica izkonfigurirana, veličina drugih stupaca se može također promijeniti. Ponašanje promjene veličine možemo podesiti sa metodom setAutoResizeMode().
- *Razmještaj stupaca* - Ako kliknemo na zaglavlje stupca i povučemo, možemo pomicati cijeli stupac na neku drugu stranu tablice.

5.17 JTREE

Jedan od Swingovih naprednih komponenti je upravo JTree. Stabla su korisna kod predstavljanja hijerarhijskih informacija, kao na primjer što je sadržaj sa jedinice diska ili organizacijskih dijagrama poduzeća. Kao sa svim Swingonovim komponentima, podatkovni

model je različit od vizualnog modela. To znači da možemo raditi stvari kao što su ažuriranja podatkovnog modela i biti na miru da će nam se vizualna komponenta prikladno ažurirati.

JTree je snažan i složen. Dovoljno je velik da za razliku od tekstnih alata, klase koje podržavaju JTree, imaju svoji paket javax.swing.tree. Međutim, ako prihvatimo za većinu stvari temeljno zadane postavke, JTree je vrlo jednostavan za korištenje.

5.18 JTABBEDPANE

Ako se prisjetimo kako izgleda kada kliknemo desnom tipkom miša na radnu površinu da postavimo postavke prikaza u Windows XPu, onda već znamo kako izgleda JTabbedPane. To je spremnik sa označenim karticama (na primjer Teme, Zaštitnik zaslona, Izgled). Kada kliknemo na karticu, prikazuje se novi set kontrola u JTabbedPane. U Swingu, JTabbedPane je specijalna vrsta spremnika.

Svaka kartica ima svoj naziv. Da bi dodali novu karticu u JTabbedPanejednostavno pozovemo metodu addTab(). Metoda prima dva argumenta, gdje jednim moramo odrediti naziv kartice a drugim komponentu koja opskrbljuje kartični sadržaj. Tipično, to je spremnik koji sadrži druge komponente.

Iako JTabbedPane može prikazivati samo jednu grupu komponenti istovremeno, treba paziti da su sve komponente na svim stranicama zapisane u memoriji. Ako imamo komponente koje zauzimaju cijelo procesorsko vrijeme, treba ih staviti u stanje “spavanja” onda kada se ne prikazuju.

Prema početnim zadanim postavkama, kada ima previše kartica da bi se prikazalo u jednom redu, JTabbedPane ih automatski umota u dodatne redove. To ponašanje je prikladno za ideju o karticama, koje podsjećaju na popunjavanje ormarića sa mapama, ali također zahtijeva kada selektiramo karticu iz stražnjeg reda da se kartice premještaju da bi se dovukla selektirana kartica na prednji plan. Većina korisnika smatra to zbunjujuće i krši najvažniji dizajn korisničkog sučelja koji kaže da kontrole trebaju uvijek ostati na istome mjestu.

5.19 JSPLITPANE

JSplitPane je specijalna vrsta spremnika koji drži dvije komponente, svaku u svome pod oknu. Razdijeljena traka namješta veličine tih dviju pod okna. Na primjer, u pregledniku dokumenata moglo bi se koristiti razdjelno okno da se prikaže sadržaj pokraj stranice teksta.

Možemo mijenjati orijentaciju okna, gdje može biti postavljeno horizontalno ili vertikalno. Osim toga, imamo atribut klase oneTouchExpandable gdje kada ga postavimo u true onda nam se pojave dvije strelice, u lijevom i desnom smjeru, koje kada se kliknu pomaknu razdijeljenu traku do kraja u jednom smjeru te time dobijemo na prikazu samo jedno okno dok je drugo sakriveno.

5.20 JSPINNER

JList i JComboBox su dva načina gdje korisnik može izabrati iz skupa vrijednosti. JComboBox ima dodatnu fleksibilnost kada se postavi da je promjenjiv, ali općenito obje komponente su ograničene u tome da samo mogu od korisnika zatražiti izbor iz fiksnog skupa izbora. Swing je dodao komponentu JSpinner koji nam koristi za velike ili otvorene sekvence vrijednosti kao što su brojevi i datumi. JSpinner je rođak JComboBox gdje prikazuje vrijednost u polju, ali umjesto da pruža padajuću listu sa mogućim izborima, daje korisniku mali par tipki sa strelicama gore i dolje za kretanje po rasponu vrijednosti. Kao i kombinirani okvir, JSpinner se također može postaviti da je promjenjiv čime omogućujemo korisniku da upiše direktno ispravnu vrijednost u polje.

Swing pruža tri temeljna tipa Spinnera, predstavljen od tri različita podatkovna modela za komponentu JSpinner: SpinnerListModel, SpinnerNumberModel i SpinnerDateModel.

- *SpinnerListModel* - ponaša se kao kombinirani okvir koji ima skup fiksnih objekata
- *SpinnerNumberModel* - prikazuje numeričke vrijednosti. Može biti postavljen sa početnim minimalnim i maksimalnim vrijednostima.
- *SpinnerDateModel* - ponaša se kao SpinnerNumberModel samo što radi sa objektima datuma i koristi specijalne Kalendar konstante za inkrementaciju.

5.21 JSLIDER

JSlider je komponenta koja omogućuje korisniku da grafički selektira vrijednost tako da pomiče gumb unutar ograničenog intervala. Gumb se uvijek pozicionira na mjesta koja odgovaraju cijelo brojnim vrijednostima unutar određenoga intervala.

Klizač može prikazati veće oznake kvačica i manje koji su između većih oznaka kvačica. Broj vrijednosti između oznaka kvačica se podešava metodama setMajorTickSpacing za veće kvačice i setMinorTickSpacing za manje kvačice. Bojanje oznaka kvačica se podešava pozivom metode setPaintTicks.

Klizači također mogu ispisati tekstualne oznake na uobičajenim intervalima (ili na proizvoljnim mjestima) uzduž klizačeve trake. Bojanje oznaka se izvršava pomoću metoda setLabelTable i setPaintLabels.[8]

5.22 JSEPARATOR

JSeparator omogućava komponenti opće namjene za primjenu razdjeljničkih linija - najčešće korišteno kao razdjelnik između stavki izbornika koji ih dijeli u logične grupe. Oni su slični rubovima, samo za razliku od njih one su prave komponente koji se crtaju unutar spremnika a ne okolo krajeva ili na rubovima određene komponente. Razdjelnici se mogu još koristiti drugdje u GUI gdje su potrebni vizualni razdjeljivači.[9]

5.23 JPROGRESSBAR

Komponenta koja vizualno prikazuje napredak određenog zadatka. Kako se zadatak izvršava i napreduje prema završetku, grafički indikator prikazuje postotak dovršenosti zadatka. Taj postotak je tipično predstavljen kao pravokutnik koji je na početku prazan i postepeno se puni kako se zadatak izvršava. Dodatno, grafički indikator može i prikazati tekstualni prikaz tog postotka.

JProgressBar koristi BoundedRangeModel kao svoj podatkovni model, sa vrijednosti svojstva predstavlja "sadašnje" stanje zadatka, dok minimalna i maksimalna svojstva predstavljaju početak i kraj točaka.

Da prikažemo da se zadatak nepoznate duljine izvršava, možemo staviti grafički indikator u neodređeni način rada. Dok je u neodređenom načinu rada, ima konstantnu animaciju koja pokazuje da se zadatak obavlja. Čim prije se odredi duljina i količina napretka zadatka, trebalo bi se opet ažurirati vrijednost grafičkog indikatora i prebaciti u određeni način rada.[10]

5.24 JTOOLBAR

JToolBar je spremnik koji grupira nekoliko komponenti, najčešće tipke i ikone, u redak ili stupac. Prema početnim zadanim postavkama, korisnik može povući alatnu traku na drugi kraj svog spremnika ili izvan njega u svoj vlastiti prozor (ako je svojstvo floatable postavljeno u true) pod nazivom kako je definirano u konstruktoru JToolBar. Da bi

[8] http://docs.oracle.com/javase/7/docs/api/javax/swing/JSlider.html
[9] http://docs.oracle.com/javase/7/docs/api/javax/swing/JSeparator.html
[10] http://docs.oracle.com/javase/7/docs/api/javax/swing/JProgressBar.html

povlačenje alatne trake radilo ispravno, preporučeno je da se doda instanca JToolBar na jednu od četiri "strane" spremnika čiji je upravitelj razmještaja postavljen na BorderLayout, te se ne smije dodavati djeca alatne trake na bilo koju drugu preostalu "stranu" spremnika.[11]

5.25 JTOOLBAR.SEPARATOR

Kao što je već spomenuto za komponentu JSeparator, JToolBar.Separator je praktički isto to samo što je specifičan za alatne trake te je već implementiran u sam objekt JToolBar gdje automatski se prilagodi za dijeljenje alatne trake te dodaje vertikalnu ili horizontalnu liniju, ovisno o postavkama orijentacije, za podjelu. Znači ako, na primjer, želimo na alatnoj traci razdvojiti međusobno više tipki, koristit ćemo upravo JToolBar.Separator kao razdjelnik koji će na napraviti male grupe tipki.[12]

5.26 JSCROLLBAR

JScrollBar je implementacija pomičnog traka. Kao što smo već spominjali, korisnik pomiče gumb na pomičnom traku da bi vidio ostatak sadržajnog prostora. Program tipično namjesti prikaz tako da kraj pomičnog traka predstavlja kraj na prikazu sadržaja, odnosno 100% od sadržaja. Početak pomičnog traka je početak prikaza sadržaja, odnosno 0% sadržaja. Pozicija gumba unutar tih granica se prevodi na odgovarajući postotak prikazivanog sadržaja.

U pravilu, pozicija gumba u pomičnom traku mijenja odgovarajuće promjene napravljene na poziciji objekta JViewport na temeljnom prikazu, mijenjajući sadržaj objekta JViewport.

Stvaramo JScrollBar tako da određujemo orijentaciju, da li horizontalno ili verticalno. Možemo također odrediti vrijednosti minimuma i maksimuma za komponente, kao i inicijalnu vrijednost. JScrollBar podržava jedan dodatni parametar pod nazivom *extent* (odnosno granica). Granica se odnosi na raspon vrijednosti koji je predstavljen sa klizaćem na pomičnom traku. Na primjer, na pomičnom traku koji se pomiče od 0 do 255, granica od 128 znači da će klizač biti na pola širine pomičnog traka.[13]

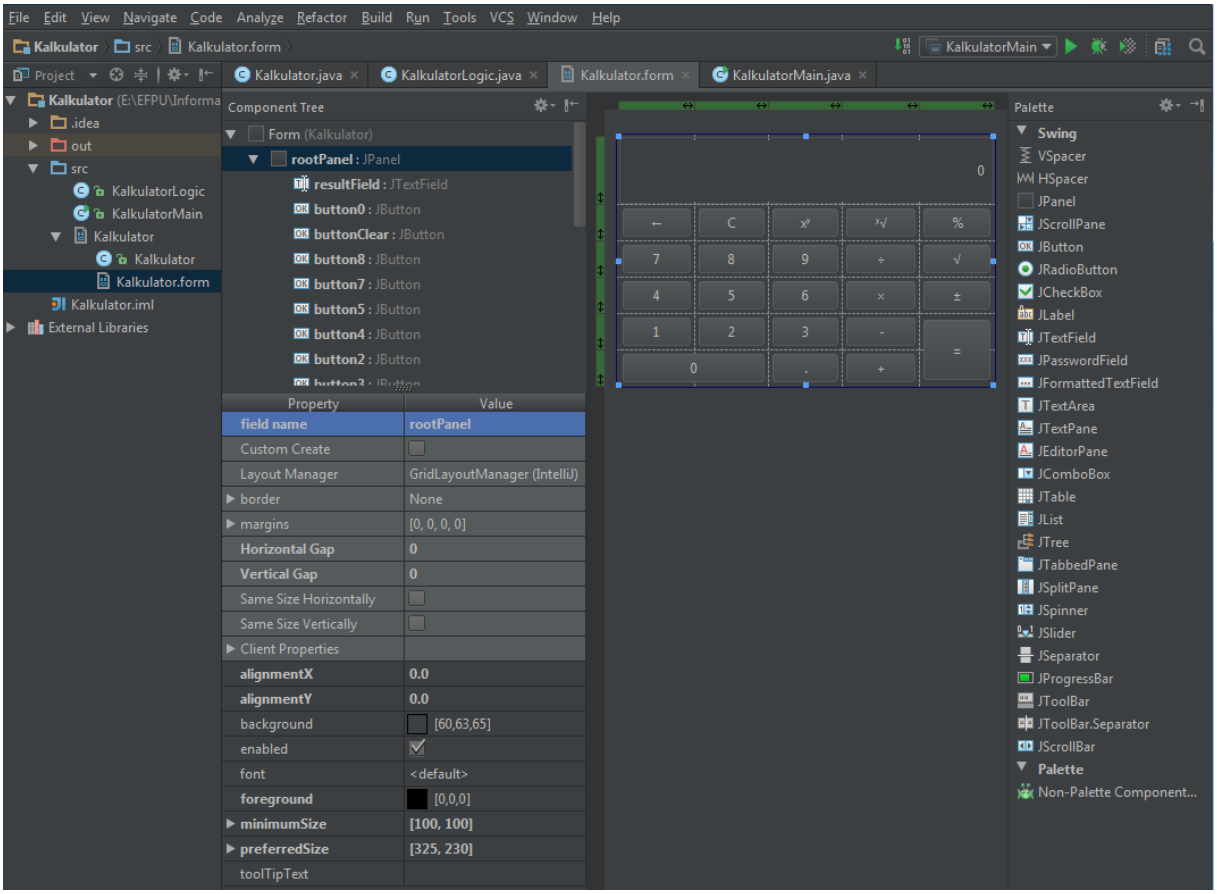
[11] http://docs.oracle.com/javase/7/docs/api/javax/swing/JToolBar.html
[12] http://docs.oracle.com/javase/7/docs/api/javax/swing/JToolBar.Separator.html
[13] http://docs.oracle.com/javase/7/docs/api/javax/swing/JScrollBar.html

6 IZRADA APLIKACIJE KALKULATORA

Detaljnije testiranje pojedinih Swing komponenti u IntelliJ IDEA-i može se izvršiti izradom određene GUI aplikacije. U ovom slučaju to će biti izrada kalkulatora.

Za početak, potrebno je dizajnirati kako da izgleda aplikacija preko IDEA dizajnera za GUI forme. To se radi tako da se sa desne strane IDEA prozora (početna zadana pozicija), iz Swing palete, izabere JButton i povlaći na željeno mjesto u JPanel spremniku. Osim JButton-a trebati će i JTextField za prikaz rezultata kalkulatora. Da bi definirali veličine JButton-a i JTextField-a, mijenja se vrijednost Preferred Size sa -1 (početna zadana vrijednost koja označava da se komponenta može povećavati ili smanjivati ovisno o postavkama JPanel spremnika) na željenu veličinu.

Na slici 11 možemo vidjeti JButton-e 0 i = da su veći od ostalih. U ovom primjeru JButton 0 ima Preferred Size postavljen na [20, 20] (širina i dužina), dok JButton = ima postavljen na [10, 60]. Svi ostali JButton-i imaju Preferred Size postavljen na [10, 20], dok je kod JTextField postavljeno na [-1, 60]. Osim toga, JPanel ima također postavljen Preferred Size na [325, 230] tako prilikom pokretanja programa da uvijek isto izgleda.



Slika 11: InteliJ IDEA GUI dizajner. Izvor: Snimka zaslona autora.

Nakon dizajniranja sučelja kalkulatora prelazimo na implementaciju logike. Na slici 12 se vidi da je dizajner stvorio klasu i napravio definiciju objekata komponenti.

```
/**
 * Created by Tedi on 19.9.2015..
 */
public class Kalkulator extends JFrame {
    private JTextField resultField;
    private JButton button0;
    private JButton button1;
    private JButton button2;
    private JButton button3;
    private JButton button4;
    private JButton button5;
    private JButton button6;
    private JButton button7;
    private JButton button8;
    private JButton button9;
    private JButton buttonDecimal;
    private JButton buttonPlus;
    private JButton buttonResult;
    private JButton buttonMinus;
    private JButton buttonMultiply;
    private JButton buttonReverse;
    private JButton buttonDivide;
    private JButton buttonSquareRoot;
    private JButton buttonBackspace;
    private JButton buttonClear;
    private JButton buttonPowerY;
    private JButton buttonRootY;
    private JButton buttonPercent;
    private JPanel rootPanel;
    private JMenu file, help;
    private JMenuItem itemExit, itemAbout;

    public Kalkulator()
    {
        super("Kalkulator");
        setContentPane(rootPanel);
        pack();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);

        //JMenu
        file = new JMenu("Datoteka");
        file.setMnemonic(KeyEvent.VK_D);
        itemExit = new JMenuItem("Izlaz");
        itemExit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X, ActionEvent.CTRL_MASK));
        file.add(itemExit);
        help = new JMenu("Pomoc");
        help.setMnemonic(KeyEvent.VK_P);
        itemAbout = new JMenuItem("O Kalkulatoru");
        help.add(itemAbout);
        JMenuBar mb = new JMenuBar();
        mb.add(file);
        mb.add(help);
        setJMenuBar(mb);
```

Slika 12: Definicija objekata komponenti Swing-a. Izvor: Slika zaslona autora.

Da se ne stvara još jedna klasa u kojoj se definira okvir aplikacije, odnosno JFrame, proširiti ćemo trenutno stvorenu klasu Kalkulator sa naredbom extends JFrame. Sada možemo preko konstruktora klase Kalkulator() pozvati metode i atribute koji su dio klase JFrame te time dodatno podesiti okvir aplikacije po potrebi. U ovom slučaju, poziva se metoda super() sa argumentom String koja postavlja naziv okvira na definirani String. Pozivom metode setContentPane(objekt spremnik) postavljamo određeni spremnik da bude dio temeljnog okvira aplikacije. Metoda pack() postavlja JFrame da bude promjenljive veličine ovisno o broju i veličini komponenti unutar spremnika a postavljen od strane upravitelja razmještaja. Metoda setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) definira ponašanje okvira prilikom zatvaranja prozora, u ovom slučaju prozor se zatvara, dok setVisible(true) omogućuje da se okvir prikazuje, gdje se u suprotnom ne bi vidio.

JMenu koji inače nije dio palete komponenti IntelliJ IDEA-e, ali je dio Swing paketa, te se može dodati samo programskim putem. JMenu predstavlja izbornik koji pritiskom na jednog od izbora se pojavljuje padajući izbornik sa dodatnim izborima. U ovom slučaju su definirani izbori „Datoteka“ sa podizborom „Izlaz“ i „Pomoc“ sa podizborom „O Kalkulatoru“. Zanimljivost kod JMenu je da se može dodati mnemonik, odnosno slovo ili riječ koje nas asocira, kao što je u ovom primjeru kod Datoteke slovo „D“ a kod Pomoći slovo „P“, te zbog toga su i početna slova podcrtana. Također se mogu dodati i povezati prečaci na sa tim slovima na tipkovnici. Metoda setAccelerator(), koja se poziva na objektu podizbora, sa definiranom kombinacijom tipki prikazuje tu kombinaciju tipki pokraj naziva podizbora.

Nakon dodavanja JMenu-a, prelazi se na dodavanje ActionListener-a, odnosno u ovom slučaju AbstractListener koji je na hijerarhiji iznad ActionListener-a, tj. roditelj mu je. Korištenje AbstractListener-a umjesto ActionListener-a omogućuje stvaranje slušača koji nisu usko povezani sa određenom radnjom, gdje istovremeno može biti slušač za tipke, miš ili nešto treće , odnosno puno globalnije za razliku od specijaliziranijeg ActionListener-a. Na slici 13 se vidi definiranje i implementacija slušača za sve komponente. Na primjer, vidi se dodavanje slušača za JMenu podizbor „O Kalkulatoru“ koji prilikom klikom na njega se stvara događaj koji se izvršava u metodi actionPerformed(ActionEvent). U ovom primjeru, stvara se JOptionPane prozor sa podacima o nazivu, verziji i autoru aplikacije. Nadalje svi registrirani slušači događaja su u skraćenom obliku (AbstractAction) (e) → {*//programski kod funkcije*} da zauzimaju manje mjesta, ali valja znati da je uvijek dio registriranog slušača metoda actionPerformed() koja se izvršava.

```
itemAbout.addActionListener(new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String message = "Kalkulator ver. 1.0\n\nAutor: Tedi Perkovic";
        JOptionPane.showMessageDialog(null, message, "O Kalkulatoru", JOptionPane.INFORMATION_MESSAGE);
    }
});

Action itemExitAction = (AbstractAction) (e) -> { System.exit(0); };
itemExit.addActionListener(itemExitAction);

final ActionListener event = new KalkulatorLogic(resultField);
button0.addActionListener(event);
button1.addActionListener(event);
button2.addActionListener(event);
button3.addActionListener(event);
button4.addActionListener(event);
button5.addActionListener(event);
button6.addActionListener(event);
button7.addActionListener(event);
button8.addActionListener(event);
button9.addActionListener(event);
buttonPlus.addActionListener(event);
buttonMinus.addActionListener(event);
buttonMultiply.addActionListener(event);
buttonDivide.addActionListener(event);
buttonDecimal.addActionListener(event);
buttonResult.addActionListener(event);
buttonPercent.addActionListener(event);
buttonPowerY.addActionListener(event);
buttonRootY.addActionListener(event);

Action backspaceAction = (AbstractAction) (e) -> {
        new KalkulatorLogic(resultField).backspace();
};
buttonBackspace.addActionListener(backspaceAction);

Action clearAction = (AbstractAction) (e) -> { new KalkulatorLogic(resultField).clear(); };
buttonClear.addActionListener(clearAction);

Action reverseAction = (AbstractAction) (e) -> { new KalkulatorLogic(resultField).reverse(); };
buttonReverse.addActionListener(reverseAction);

Action squareRootAction = (AbstractAction) (e) -> {
        new KalkulatorLogic(resultField).squareRoot();
};
buttonSquareRoot.addActionListener(squareRootAction);
```

Slika 13: Definiranje i implementacija slušaća. Izvor: Snimka zaslona autora.

Kao što se vidi iz primjera sa slike 13, vidi se da ima registriranih slušača za razne vrste akcija kalkulatora (tipka za brisanje zadnjeg utipkanog znaka, tipka za brisanje, tipka za predznak, tipka za izračun korijena od broja, te sve ostale tipke na kalkulatoru).

Nadalje na slici 14 imamo registrirane prečace tipki koje se prilikom pritiskanja istih, dok je aplikacija u fokusu, poziva definirana akcija. Na primjer, ako pritisnemo tipku ESC pozvati će se akcija clearAction koja će izvršiti poziv metode clear() definiranu u klasi KalkulatorLogic. Ovo je moguće iz razloga što su slušači komponenata definirani kao AbstractAction, gdje inače to se ne bi moglo napraviti bez da se odvojeno stvori za svaku komponentu slušač tipa KeyListener. Korištenje AbstractAction-a smanjilo se redundantnost

programskog koda, a time i čitljivost te lakše proširivanje za neke dodatne tipke ili slične akcije.

```
        button0.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_0, 0), "0_pressed");
        button0.getActionMap().put("0_pressed", (Action) event);
        button1.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_1, 0), "1_pressed");
        button1.getActionMap().put("1_pressed", (Action) event);
        button2.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_2, 0), "2_pressed");
        button2.getActionMap().put("2_pressed", (Action) event);
        button3.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_3, 0), "3_pressed");
        button3.getActionMap().put("3_pressed", (Action) event);
        button4.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_4, 0), "4_pressed");
        button4.getActionMap().put("4_pressed", (Action) event);
        button5.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_5, 0), "5_pressed");
        button5.getActionMap().put("5_pressed", (Action) event);
        button6.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_6, 0), "6_pressed");
        button6.getActionMap().put("6_pressed", (Action) event);
        button7.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_7, 0), "7_pressed");
        button7.getActionMap().put("7_pressed", (Action) event);
        button8.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_8, 0), "8_pressed");
        button8.getActionMap().put("8_pressed", (Action) event);
        button9.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_9, 0), "9_pressed");
        button9.getActionMap().put("9_pressed", (Action) event);
        buttonPlus.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_PLUS, 0), "+_pressed");
        buttonPlus.getActionMap().put("+_pressed", (Action) event);
        buttonMinus.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_MINUS, 0), "-_pressed");
        buttonMinus.getActionMap().put("-_pressed", (Action) event);
        buttonMultiply.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_MULTIPLY, 0), "*_pressed");
        buttonMultiply.getActionMap().put("*_pressed", (Action) event);
        buttonDivide.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_DIVIDE, 0), "/_pressed");
        buttonDivide.getActionMap().put("/_pressed", (Action) event);
        buttonResult.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_EQUALS, 0), "=_pressed");
        buttonResult.getActionMap().put("=_pressed", (Action) event);
        buttonDecimal.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_PERIOD, 0), "._pressed");
        buttonDecimal.getActionMap().put("._pressed", (Action) event);
        buttonBackspace.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(8, 0), "backspace_pressed");
        buttonBackspace.getActionMap().put("backspace_pressed", backspaceAction);
        buttonClear.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0), "ESC_pressed");
        buttonClear.getActionMap().put("ESC_pressed", clearAction);

        buttonClear.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW).put(KeyStroke.getKeyStroke(KeyEvent.VK_X, ActionEvent.CTRL_MASK), "Ctrl+X_pressed");
        buttonClear.getActionMap().put("Ctrl+X_pressed", itemExitAction);
    }
}
```

Slika 14: Registracija tipaka za prečac na tipkovnici. Izvor: Snimka zaslona autora.

Za sada se vidjelo definiranje i registriranje objekata komponenti i njihovih slušača, a nadalje će se prijeći na logiku kalkulatora. Iz slike 15 možemo vidjeti potrebne import-ove paketa koji su potrebni za rad.

Klasa KalkulatorLogic koja je proširena sa AbstractAction omogućuje univerzalno definiranje slušača za preostale tipke u kalkulatoru (tipke 0-9, tipke matematičkih operacija, tipka =, itd.). Napravljeno je iz razloga da programski kod ne bude redundantan i da bude čitljiv. Preko konstruktora klase postavlja se JTextField koji se koristi za ispisivanje pritisnutih tipki i rezultata.

Osim toga, koristi se enumeracija za matematičke operacije koja je u Javi implementirana kao tip klase. Stoga, može se definirati preko konstruktora enumeracije željene vrijednosti ili simbole. Na primjer, kroz programski kod nadalje se umjesto simbola + može koristiti član enumeracije ADDITION. Enumeracija još ima i implementiranu metodu getMembersByValue(String value) koja nadalje služi da dobijemo ovisno o proslijeđenom Stringu, odnosno simbolu, točno taj član enumeracije. Na primjer ako proslijedimo toj metodi simbol „+“, metoda će nam vratiti objekt enumeracije ADITTION.

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.lang.*;

/**
 * Created by Tedi on 21.9.2015..
 */
public class KalkulatorLogic extends AbstractAction {
    private JTextField resultField;
    private static Operators operand;
    private enum Operators {
        ADDITION("+", "\\+"), SUBTRACTION("-", "\\-"), MULTIPLICATION("\u00D7", "\u00D7"), DIVISION("\u00F7", "\u00F7"),
        PERCENT("%", "%"), EXPONENT("\u0078\u02B8", "\u0078\u02B8"), NTH_ROOT("\u02B8\u221A", "\u02B8\u221A");

        private String value;
        private String regexValue;
        Operators(String value, String regexValue)
        {
            this.value = value;
            this.regexValue = regexValue;
        }

        public static Operators getMemberByValue(String value) {
            for (Operators o : Operators.values())
                if (o.value.equals(value)) return o;
            return operand;
        }
    }

    public KalkulatorLogic(JTextField resultField)
    {
        this.resultField = resultField;
    }
```

Slika 15: Programska logika kalkulatora. Izvor: Snimka zaslona autora.

```
private Boolean doCalculation(String symbol)
{
    operand = Operators.getMemberByValue(symbol);
    if (operand == null && symbol == "=") return true;
    if (operand != null && symbol == "=") {
        String[] split;
        switch (operand) {
            case ADDITION:
                split = resultField.getText().split(Operators.ADDITION.regexValue);
                resultField.setText(Double.toString(Double.parseDouble(split[0]) + Double.parseDouble(split[1])));
                break;
            case SUBTRACTION:
                String removeNegative = resultField.getText();
                if (removeNegative.startsWith("-"))
                    removeNegative = resultField.getText().replaceFirst("\\=", "N");
                split = removeNegative.split(Operators.SUBTRACTION.regexValue);
                resultField.setText(Double.toString(Double.parseDouble(split[0].replace("N", "-")) - Double.parseDouble(split[1])));
                break;
            case MULTIPLICATION:
                split = resultField.getText().split(Operators.MULTIPLICATION.regexValue);
                resultField.setText(Double.toString(Double.parseDouble(split[0]) * Double.parseDouble(split[1])));
                break;
            case DIVISION:
                split = resultField.getText().split(Operators.DIVISION.regexValue);
                resultField.setText(Double.toString(Double.parseDouble(split[0]) / Double.parseDouble(split[1])));
                break;
            case PERCENT:
                split = resultField.getText().split(Operators.PERCENT.regexValue);
                resultField.setText(Double.toString(Double.parseDouble(split[0]) * Double.parseDouble(split[1])/100));
                break;
            case EXPONENT:
                split = resultField.getText().split(Operators.EXPONENT.regexValue);
                resultField.setText(Double.toString(Math.pow(Double.parseDouble(split[0]), Double.parseDouble(split[1]))));
                break;
            case NTH_ROOT:
                split = resultField.getText().split(Operators.NTH_ROOT.regexValue);
                resultField.setText(Double.toString(Math.pow(Double.parseDouble(split[1]), 1 / Double.parseDouble(split[0]))));
                break;
        }
        operand = null;
        return true;
    }
    return false;
}
```

Slika 16:Metoda doCalculation(). Izvor:Snimka zaslona autora.

Na slici 16 vidi se metoda koja je zadužena za izračun i prikaz rezultata. Ovisno o postavljenom objektu operand (da li je neka valjana matematička operacija ili običan broj; tipa Operators ili null) izvršava se i njegova logika. Kao što smo već rekli ako je operand postavljen na ADDITION u switch case-u će se i isti izvršiti. Na primjer, JTextField će imati String „1+1“, gdje će se zatim podijeliti na dva Stringa po operatoru „+“. Ta dva Stringa nakon konverzije u Double se zbrajaju te ponovno konvertiraju nazad u String da bi se postavio tekst u JTextField-u i prikazao kao rezultat. Slična priča vrijedi i za druge matematičke operacije kao što to možemo vidjeti iz programskog koda.

```
    public void backspace()
    {
        if (resultField.getText().isEmpty()) return;
        String newText = resultField.getText().substring(0, resultField.getText().length() - 1);
        resultField.setText(newText);
    }

    public void clear()
    {
        operand = null;
        resultField.setText("0");
    }

    public void reverse()
    {
        if (resultField.getText().isEmpty()) return;
        String newText = Double.toString(Double.parseDouble(resultField.getText()) * -1);
        resultField.setText(newText);
    }

    public void squareRoot()
    {
        if (resultField.getText().isEmpty()) return;
        String newText = Double.toString(Math.sqrt(Double.parseDouble(resultField.getText())));
        resultField.setText(newText);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (resultField.getText().startsWith("0") && e.getActionCommand() != "." && resultField.getText().length() <= 1)
            resultField.setText("");
        if (doCalculation(e.getActionCommand())) return;
        resultField.setText(resultField.getText() + e.getActionCommand());
    }
}
```

Slika 17: Implementacija ostalih metoda. Izvor: Snimka zaslona autora.

Implementaciju preostalih metoda koje se koriste u Kalkulatoru, kao što je tipka za brisanje ili promjenu predznaka, možemo vidjeti sa slike 17.

Glavni dio klase KalkulatorLogic je metoda actionPerformed() koja se izvršava svaki put kada kliknemo na neki registrirani JButton ili kad pritisnemo njegovu tipku na tipkovnici. Sve pritisnute tipke i njihove akcije se nadodaju u JTextField pozivom resultField.setText(resultField.getText() + e.getActionCommand()), gdje je objekt e sa metodom getActionCommand() definirani tekst na tipki. Prije toga vidi se da se svaki put poziva metoda doCalculation() koja provjerava da li tekst resultField-a ima valjani oblik za izračun rezultata.

```
import javax.swing.*;

/**
 * Created by Tedi on 19.9.2015..
 */
public class KalkulatorMain {
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        new Kalkulator();
    }
}
```

Slika 18: Kalkulatorova main funkcija. Izvor: Snimka zaslona autora.

Sa slike 18 se vidi metoda main() koja je uvijek potrebna da bi se aplikacija pokrenula. Imamo testiranje promjene korisničkog sučelja sa standardno zadane postavke „Metal“ na „Nimbus“ unutar try-catch u slučaju da određena platforma ne može promijeniti korisničko sučelje na „Nimbus“ te da se aplikacija ne sruši ako je to nemoguće.

Metoda setLookAndFeel(), kao što je već spomenuto, služi da bi promijenili kompletan izgled aplikacije, uključujući i oblike komponenti i njihove boje.

Na kraju stvaramo novi objekt klase Kalkulator pozivom njegovog konstruktora te time je omogućeno da se aplikacija i pokrene.

7 ZAKLJUČAK

Rad u Javi je potvrdio pozitivna mišljenja koja sam imao vezana za nju. Nakon programiranja u C++, Java je pokazala svoju veliku jednostavnost sa upravljanjem memorijom naspram C++. Strukturiranje programa tako da na svaku klasa možemo gledati kao objekt uvelike ubrzava i pojednostavljuje pisanje koda te ga čini organiziranim i modularnim, odnosno lakšim za daljnja poboljšavanja i unapređivanja.

Izrada GUI aplikacije nikad nije bila lakša. Pozivanjem već gotovih metoda, te IDEA-inim odličnim automatskim generiranjem koda, vrlo jednostavno možemo napraviti aplikaciju sa korisničkim sučeljem. Swing zaista nudi veliku količinu gotovih komponenti koje samo nadodajemo i slažemo u dizajneru IDEA-e po izboru, te na kraju samo popunimo i zalijepimo logiku na komponente u .java datoteci. Ona se na kraju izkompilira te nakon pokretanja odmah možemo vidjeti prozor koji smo stvorili prethodno u dizajneru.

Proučavanjem komponentu po komponentu, te prakticiranjem iste kroz IDEA-u naučio sam dosta detaljno o tim komponentama te kako napraviti jednostavnu aplikaciju, kao što je kalkulator. Ima veoma puno informacija o svim tim komponentama, te još i nekim komponentama koji nisu dio originalnog okvira IDEA-e ali su dio paketa Swinga te se mogu programskim putem nadodati. Shodno tome, sve te informacije mi ne stanu u ovaj rad zato sam selektivno izabrao one koje su najkorištenije i dio originalnog okvira IDEA-e, te jasno zato i postoje knjige na tu temu sa više od nekoliko stotina stranica.

8 SAŽETAK

Ideja o Javi je nastala 1990. od strane glavnog istraživača Bill Joya u Sun Microsystems. James Gosling je bio dio programerskog tima koji se pridružio Bill Joyu te će biti zapamćen kao otac Jave. 1993. se počeo povećavati interes za Web te time je Sun iskoristio priliku sa svojim pretkom Jave da dodatno uskladi projekt sa zahtjevima Weba, gdje su ju nakon toga izdali.

Za razliku od programskih jezika temeljenih na algoritmu koji bi svakom nadogradnjom na kodu postajali sve složeniji, time kompleksniji i teži za izmjene i daljnji razvoj te razumijevanje u cjelini, Java kao objektno orijentirani programski jezik ublažava taj problem korištenjem objekata (klasa sa atributima i metodama). Osim toga, Javina virtualna mašina služi da kompilirani bajtkod prevodi na operacije razumljive glavnom računalu odnosno operacijskom sustavu na tom računalu. Zato se kaže da je Java neovisna o platformi.

Swing je Javin programski okvir za izradu grafičkih korisničkih sučelja. Osnovni objekt korisničkoga sučelja je komponenta. Komponente u Swingu, kao što su tipke i izborni kvadratići, su implementirane u samoj Javi. To znači da, neovisno na kojem smo operacijskom sustavu ili na kojem će se program koristiti, nam grafičko korisničko sučelje na kojemu radimo uvijek isto izgleda. Komponenta se inače postavlja u spremnik, koji se onda pomoću upravitelja razmještaja namješta po pravilima definiranim u upravitelju. Karakteristika komponenti je njezino ponašanje na događaje pokrenutih od strane korisnika, slanjem objekata događaja i primanja tih objekata preko registriranih slušača za takav specifičan tip događaj.

Spremnik je vrsta komponente koja zadržava i upravlja ostalim komponentama, možemo povući usporedbu sa knjigom polica; gdje je spremnik sama polica, a komponente su knjige na polici. Komponente koje stavljamo u spremnik ovise o upravitelju razmještaja čije se pravilo razmještaja može definirati na samom spremniku, te da se komponente automatski poslažu po tom pravilu.

Kao što je već spomenuto, upravitelj razmještaja je objekt koji kontrolira položaj i veličinu komponente unutar prikaznog prostora spremnika. Definiramo razmještaj komponenti pravilima koje se primjenjuju na cijelom spremniku. Različiti spremnici mogu imati različita pravila upravljanja razmještajem komponenti.

9 SUMMARY

The concept of Java appeared in 1990 by the chief researcher from Sun Microsystems. James Gosling was a part of development team who joined Bill Joy, and will be remembered as father of Java. In 1993, there has been increasing interest in Web and therefore the Sun took the opportunity with his ancestor of Java to further update the project with the requirements of the Web, where they would release it afterwards.

Unlike programming languages based on an algorithm that any upgrade to the code would make it more composed, more complex and difficult to maintain it, and understanding it as the whole. Java object-oriented programming language alleviates this problem by using objects (classes with attributes and methods). In addition, Java's virtual machine is used for compiled bytecode which is converted to operations understandable to main computer or operating system on that computer. Therefore it is said that the Java is platform independent.

Swing is the Java programming framework for creating graphical user interfaces. The basic object of the user interface is component. The components in Swing, such as buttons and checkboxes, are implemented in the Java. This means that, regardless of the operating system we are using on which will the program run, the graphical user interface we are developing will always look the same. The component is normally set up in the container, which is then adjusted using the layout manager by the rules defined in the manager. The characteristic of components are their behavior on the events initiated by the user, sending and receiving event objects via registered listeners for a specific type of event.

The container is a type of component that maintains and manages other components, we can draw a comparison with the book shelves; where the container is the shelf and components are books on the shelf. The components that are placed in the container depends on the layout manager which rule of layout we may define on the container, where the components are automatically arranged by this rule.

As already mentioned, the layout manager is an object that controls the position and the size of the components within the display area of the container. We define rules of layout on the components that apply to the entire container. Different containers can have different rules on the components of the layout manager.

10 LITERATURA

Čupić, Marko. *Programiranje u Javi.* Zagreb: FER, 2013.

Fakhruddin, Hussain. *Teks mobile.* 22. 1 2015. http://teks.co.in/site/blog/eclipse-vs-netbeans-better-ide-android-app-development/ (pokušaj pristupa 17. 8 2015).

Flanagan, David. *Java in a Nutshell.* Sebastopol: O'Reilly, 1997.

Hamilton, Coman. *Jaxenter.* 30. 7 2014. https://jaxenter.com/eclipse-netbeans-or-intellij-which-is-the-best-java-ide-107980.html (pokušaj pristupa 17. 8 2015).

Katamreddy, Sivaprasadreddy. *dzone.com.* 14. 10 2014. https://dzone.com/articles/netbeans-ide-and-intellij-idea (pokušaj pristupa 17. 8 2015).

Mangla, Sudhir. *ProgrammerWorld.* n.d. http://faq.programmerworld.net/programming/best-java-ide-review.html (pokušaj pristupa 17. 8 2015).

Niemeyer, Patrick, i Daniel Leuck. *Learning Java.* Sebastopol: O'Reilly Media, 2013.

Oracle. *docs.oracle.com.* n.d. http://docs.oracle.com/javase/7/docs/api/javax/swing/JToolBar.Separator.html (pokušaj pristupa 4. 9 2015).

—. *docs.oracle.com.* n.d. http://docs.oracle.com/javase/7/docs/api/javax/swing/JToolBar.html (pokušaj pristupa 4. 9 2015).

—. *docs.oracle.com.* n.d. http://docs.oracle.com/javase/7/docs/api/javax/swing/JScrollBar.html (pokušaj pristupa 4. 9 2015).

—. *docs.oracle.com.* n.d. http://docs.oracle.com/javase/7/docs/api/javax/swing/JToolBar.html (pokušaj pristupa 4. 9 2015).

—. *docs.oracle.com.* n.d. http://docs.oracle.com/javase/7/docs/api/javax/swing/JProgressBar.html (pokušaj pristupa 4. 9 2015).

—. *docs.oracle.com.* n.d. http://docs.oracle.com/javase/7/docs/api/javax/swing/JSlider.html (pokušaj pristupa 3. 9 2015).

—. *docs.oracle.com.* n.d. http://docs.oracle.com/javase/7/docs/api/javax/swing/JSeparator.html (pokušaj pristupa 3. 9 2015).

—. *docs.oracle.com.* n.d. https://docs.oracle.com/javase/tutorial/uiswing/dnd/dropmodes.html (pokušaj pristupa 2. 9 2015).

—. *docs.oracle.com.* n.d. http://docs.oracle.com/javase/7/docs/api/javax/swing/JTextField.html (pokušaj pristupa 2. 9 2015).

—. *docs.oracle.com.* n.d. http://docs.oracle.com/javase/7/docs/api/javax/swing/JSlider.html (pokušaj pristupa 3. 9 2015).

11 POPIS SLIKA

Slika 1: Java izvršno okruženje. Izvor: Learning Java, Poglavlje 1.2 5
Slika 2: Eclipse. Izvor: marketplace.eclipse.org 10
Slika 3: NetBeans. Izvor: www.filehorse.com 11
Slika 4: IntelliJ IDEA. Izvor: Slika zaslona autora. 12
Slika 5: JDeveloper. Izvor: www.formdev.com 13
Slika 6: Inspektor Svojstva u IDEA-i. Izvor: Slika zaslona autora. 17
Slika 7: Stvaranje novog projekta u IDEA-i. Izvor: Snimka zaslona autora. 20
Slika 8: Stvaranje nove GUI Form-e u IDEA-i. Izvor: Snimka zaslona autora. 20
Slika 9: Swing komponente. Izvor: Snimka zaslona autora. 21
Slika 10: Generirani kod dizajnera IDEA-e. Izvor: Snimka zaslona autora. 23
Slika 11: InteliJ IDEA GUI dizajner. Izvor: Snimka zaslona autora. 32
Slika 12: Definicija objekata komponenti Swing-a. Izvor: Slika zaslona autora. 33
Slika 13: Definiranje i implementacija slušaća. Izvor: Snimka zaslona autora. 35
Slika 14: Registracija tipaka za prečac na tipkovnici. Izvor: Snimka zaslona autora. . 36
Slika 15: Programska logika kalkulatora. Izvor: Snimka zaslona autora. 37
Slika 16:Metoda doCalculation(). Izvor:Snimka zaslona autora. 38
Slika 17: Implementacija ostalih metoda. Izvor: Snimka zaslona autora. 39
Slika 18: Kalkulatorova main funkcija. Izvor: Snimka zaslona autora. 40

12 POPIS TABLICA

Tabela 1: Primitivni tipovi podataka u Javi. Izvor: Čupić, Marko. Programiranje u Javi. Zagreb: FER, 2013. 7