Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Banič

Pisanje programov s pomočjo igralne palice

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Tomaž Dobravec

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Tematika naloge:

Pri klasične programiranju programer ročno vpisuje programsko kodo črko-po-črki. Tak način programiranja je počasen, poleg tega pa od programerja zahteva dobro poznavanje vseh ključnih besed programskega jezika ter vsebine sistemskih in ostalih knjižnic. Kot nadomestilo za klasično programiranje obstajajo različni načini alternativnega vnosa kode, večinoma namenjeni otrokom in začetnikom za uvajanje v svet programiranja. V diplomskem delu preglejte področje alternativnega programiranja in predstavite nekaj tovrstnih javno dostopnih produktov. Razvijte tudi samostojno aplikacijo za hiter vnos javanske programske kode. Aplikacija naj programerju omogoča uporabo igralne palice za izbiro posameznih delov programske kode. Pisanje programske kode s pomočjo vaše aplikacije naj bo podobno igranju računalniške igrice: hitro, zanimivo in doživeto.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Nejc Banič, z vpisno številko **63080098**, sem avtor diplomskega dela z naslovom:

Pisanje programov s pomočjo igralne palice

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  doc. dr. Tomaža Dobravca

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani Podpis avtorja:

Zahvala.

Najlepše se zahvaljujem mentorju doc. dr. Tomažu Dobravcu za strokovnost in sprotno usmerjanje k pravemu cilju.

Zahvaljujem se tudi staršem in bližnjim sorodnikom, ki so me v času študija nenehno spodbujali in mi stali ob strani.

Kazalo

Povzetek

V diplomski nalogi smo implementirali nov način programiranja, in sicer s pomočjo igralne palice. Razlog je predvsem, da se pokaže še drugačen način v razvoju programov. Velika večina programira s pomočjo dveh vhodno/izhodnih naprav: tipkovnice in miške. Ti dve napravi sta postali standard in vsekakor bosta tudi ostali za vsakega programerja.
Za svojo implementacijo smo uporabili visoko nivojski programski jezik Java in integrirano razvojno okolje NetBeans. Program je sestavljen iz tako imenovanega vtičnika (ali modula), s pomočjo katerega lahko ob zagonu programa NetBeans že začnemo s programiranjem. Sam vtičnik omogoča razvoj preprostih programov v Javi, ki ga potem lahko tudi prevedemo in izvedemo.
Poleg samega vtičnika, smo na kratko opisali zgodovino programiranja, programski jezik Java, NetBeans in ostale načine programiranja.

Abstract

In this thesis, we implemented a way of programming by means of gaming accessories. The main reason is that to show a diferent way of developing programs, because vast majority of programers are using two input / output devices: keyboard and mouse. These two devices have become standard and will definitely remain so in the future.
For our implementation, we used high-level programming language Java and NetBeans integrated development environment. The program is actually a plug-in (or module) through which you can start programming at the start of NetBeans editor. Our plug-in allows the development of simple programs in Java, which then can be compiled and executed.
In addition to the plug-in, we briefy describe the programming language Java, NetBeans and other ways of visual programming.

Seznam uporabljenih kratic

- IDE (Integrated Development Environment) - orodje za razvoj projektov, ki vsebuje urejevalnik programske kode, prevajalnik, razhroščevalnik, itd.
- OOP (Object Oriented Programming) - objektno usmerjeno programiranje je način programiranja, ki bazira na konceptu objekta in njegovih gradnikov (npr. razred).
- JRE (Java Runtime Environment) - orodje, ki omogoča izvajanje aplikacij, napisanih v Javi.
- JDK (Java Development Kit) - orodje, ki omogoča razvoj aplikacij v Javi.
- JVM (Java Virtual Machine) - del JRE, ki omogoča izvajanje vmesne kode.
- JIT (just-in-time) - poseben prevajalnik vmesne kode, ki optimizira delovanje JVM.
- API (Application Programming Interface) - specifikacija komponent programske opreme, definira njihovo funkcionalnost.

Poglavje 1

Uvod

V poglavju bomo spoznali alternativni način programiranja, in sicer vizualno programiranje. Na kratko smo opisali tri programska okolja in njihovo uporabo.

1.1 Vizualno programiranje

Vizualno programiranje je tehnika, kjer je možno s pomočjo grafičnih elementov, napisati program. Ti grafični elementi predstavljajo določeno logiko, npr. računanje, pretok podatkov, pogoji. Uporabnik na pano nanaša te elemente in jih nato logično poveže. V tem poglavju bomo predstavili `Blockly`, `Scratch` in `Microsoft Visual Programming Language`, toda obstaja še mnogo več programov kot npr. `AppWare, LabVIEW, App Inventor`.

1.1.1 Blockly

`Blockly` je spletni grafični urejevalnik, kjer lahko uporabniki sestavljajo gradnike in s tem razvijajo aplikacijo. Razvil ga je `Google`, ki je želel prikazati stil programiranja s pomočjo logičnega razmišljanja in vizualizacije, kjer pisanje ni potrebno.

Uporaba urejevalnika zelo spominja na platformo `Scratch`. Razlika je v tem, da `Blockly` lahko vgradimo v vsak program ali spletno stran, ki

želi začetnikom omogočiti učenje programiranja. Cilj urejevalnika je uporabnikom olajšati prehod na druge programske jezike, s pomočjo ustvarjanja enostavno berljive kode. Vsebuje in podpira logiko, ter glavne koncepte programiranja za razvoj aplikacij. O njem si lahko bolj podrobno preberemo na spletni strani [8]. Spodaj je prikazan primer preprostega `IF` stavka, listing 1.1, ki je bil na narejen na podlagi vira [7].

Slika 1.1: Primer uporabe stavka `IF`.

Pozitivne lastnosti uporabe programa `Blockly`:

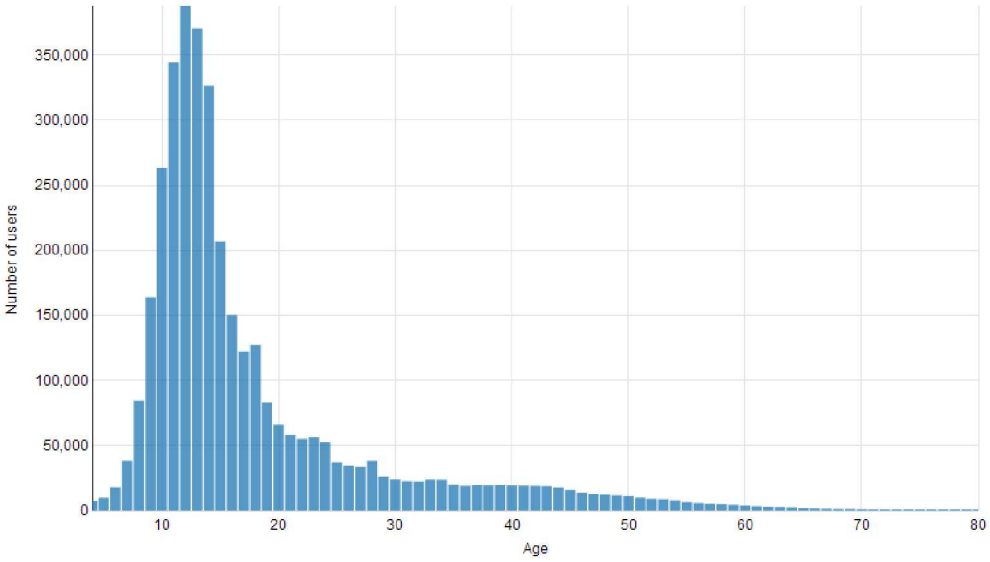
- enostaven za učenje;
- ni potrebno naprednega znanja programiranja;
- preprost in fleksibilen za uporabo;
- program lahko enostavno izvozimo v programski jezik `JavaScript`, `Python` ali `XML`;
- odprtokodni projekt.

1.1.2 Scratch

`Scratch` je orodje, namenjeno predvsem za zabavno in preprosto učenje programiranja. Vsebuje orodja za ustvarjanje različnih simulacij, iger, zgodb, animacij, glasb ali interaktivnih risb. Kot pri `Blockly`-ju, lahko uporabniki upravljajo objekte s prej definiranimi skriptami (npr. skripta ob kliku prestavi objekt naprej za 10 korakov ali reči "Hello"), ki se skupaj sestavijo kot sestavljanka.

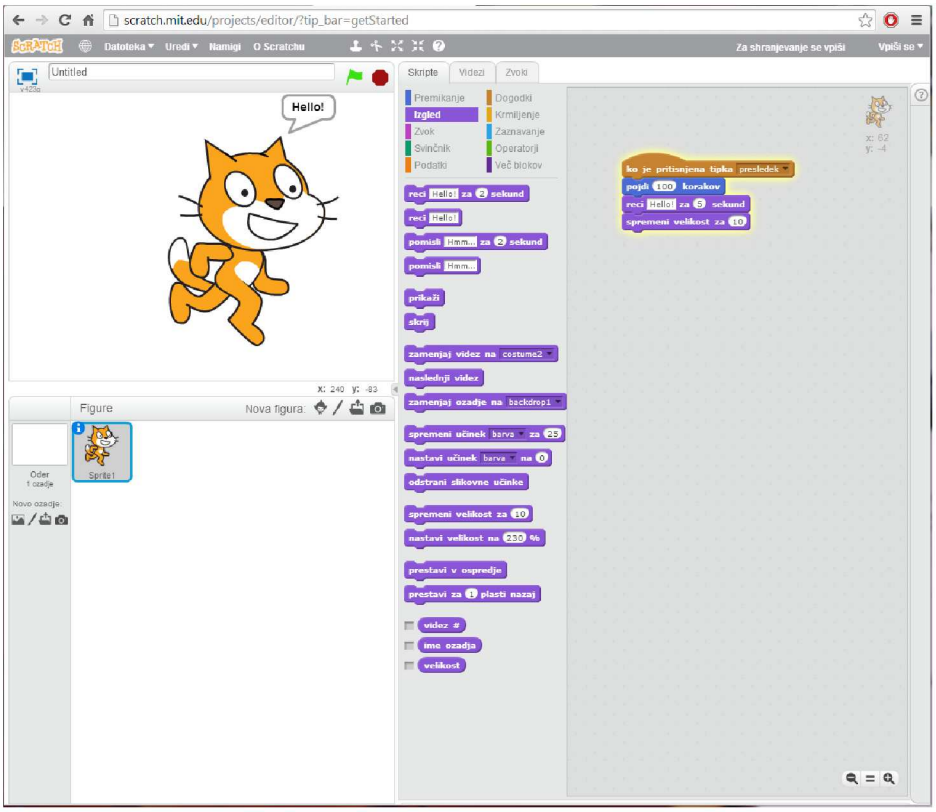
Slika 1.2: Logo in maskota Scratch.

Projekt `Scratch` se je začel leta 2003. Razvoj je potekal pri skupini Lifelong Kindergarten na interdisciplinarnem raziskovalnem laboratoriju, imenovanem MIT Media Lab, univerze Massachusetts Institute of Technology ali bolj znano MIT. Prva uradna spletna stran je zaživela v letu 2006, naslednje leto pa so širši javnosti ponudili možnost prenosa orodja (različica 1.0.2). Še v istem letu, mesecu maju, so javnosti ponudili uradno različico orodja (različica 1.1) in predstavili prenovljeno spletno stran, ki je tudi ostalim omogočila ustvarjanje in delitev lastnih projektov. Zadnja večja nadgradnja orodja in spletne strani je bila izdana v različici 2.0 leta 2012.



Slika 1.3: Graf za starostno porazdelitev uporabnikov Scratch.

Uporabniški vmesnik razvojnega okolja je sestavljeno iz več sektorjev. Leva stran je predogled začetne scene objektov in seznam uvoženih objektov slik (angl. *sprites*), na sredini so vnaprej definirane skripte po skupinah (premik, dogodek, kontrola, itd), na desni strani pa prostor za razvoj programa (slika 1.4 prikaže to porazdelitev). Skripte enostavno povlečemo na desno stran in jih združimo skupaj. Tako dobimo niz ukazov za izvršitev programa. Kot smo že omenili, orodje lahko prenesemo (podpira več različnih operacijskih sistemov) ali uporabljamo kar preko spletnega brskalnika. Uporabljajo ga lahko tako starši kot otroci, vsi željni novih znanj. Starostno porazdelitev uporabe `Scratch` lahko primerjamo na sliki 1.3.



Slika 1.4: Preprost program spletni aplikaciji Scratch, narejen na podlagi navodil na uradni strani [9].

1.1.3 Microsoft Visual Programming Language

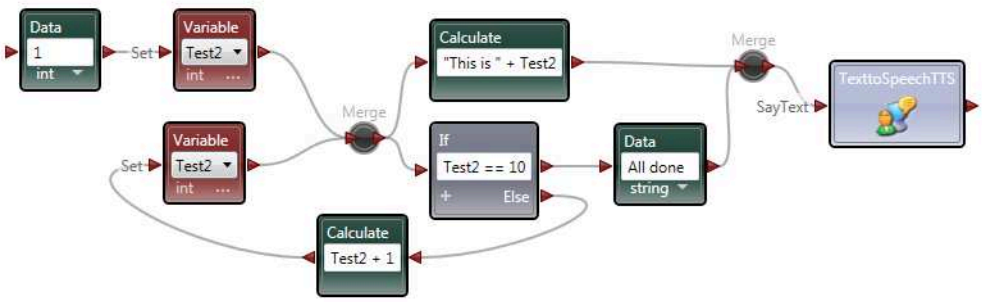
`Microsoft Visual Programming Language` je razvojno okolje, kjer programiramo z med seboj povezanimi aktivnostmi, predstavljenimi kot bloki. Skupek blokov in povezav lahko gledamo kot nekakšen diagram poteka programa.

Vsak blok ima vsaj en vhodni in izhodni zatič, vsebuje poimenovanje aktivnosti, lahko tudi grafično predstavitev namena aktivnosti ali elemente uporabniškega vmesnika (npr. vnosno polje za upravljanje podatkov), predstavlja kontrolo ali funkcijo procesiranja podatka, vnaprej definiran DSS storitev ali lastno ustvarjeno aktivnost.

Povezave služijo za prenos podatkov (ti so predstavljeni kot sporočila), ki sprožijo določeno dejavnost, npr. "preberi besedilo" (v okolju znano kot `SayText`). Ponazorili smo tudi primer (na sliki 1.5), ki smo ga sestavili na podlagi navodil na spletni strani [10].

Slika 1.5: Predstavitev podatka kot niz "Hello World" in zahteva za branje le tega.

Vhodni in izhodni zatiči predstavljajo funkcionalnost. Vhodni zatič se ponavadi nahaja na levi strani bloka (prikazan kot puščica v blok) in predstavlja vhodno sporočilo za obdelavo, izhodni zatič pa se nahaja na desni strani bloka (prikazan kot puščica iz bloka) in predstavlja odgovor bloka. Obstaja tudi izhodni zatič za obveščanje (prikazan kot krogec), ki prikazuje informacije o notranji spremembi stanja bloka. S pomočjo obveščanja lahko spremljamo oz. priredimo aktivnosti pri ustrezni spremembi bloka. Na sliki 1.6 smo prikazali preprost program v Microsoft VPL.



Slika 1.6: Predogled preprostega programa v Microsoft VPL (zanka seštevanja do števila 10, govori števila od 1-10 ali na koncu "All done").

Poglavje 2

Orodja

V tem poglavju bomo opisali vsa orodja, ki smo jih uporabili pri razvoju aplikacije. Pri izbiri operacijskega sistema nismo kolebali, izbrali smo `Microsoft Windows 7`. Ta izbira je bila optimalna, ker smo lahko uporabili in testirali veliko aplikacij, ki služijo alternativnemu načinu programiranja. Vsekakor bi bilo možno uporabiti tudi `Linux Ubuntu`, toda prevladal je `Windows 7`, ker je bil za naše potrebe povsem zadovoljiv.

2.1 Java

`Java` [2, 3] je programski jezik, ki je objektno (razredno) usmerjen, bolj znano kot jezik OOP. Glavna ideja OOP je razvoj programov s pomočjo objektov. Atributi jih opisujejo, metode pa izvajajo operacije nad objektom. Objekti so ponavadi primerki razredov, ki sodelujejo med sabo za razvoj aplikacij in programov. Ostali primeri OOP: `C++, Objective-C, Delphi, Javascript, C#, Perl, Python, Ruby, Php`...

`Java` je eden najbolj popularnih programskih jezikov, predvsem pri razvoju spletnih aplikacij odjemalec-strežnik (preko 9 milijonov razvijalcev). Programski jezik je razvil James Gosling leta 1995, zaposlen pri podjetju Sun Microsystems (kasneje so se združili z Oracle Corporation). Jezik že

Slika 2.1: Java logotip

od takrat služi kot glavna komponenta programske platforme `Java`. `Java` je za razvijalce pomembna, saj omogoča izvajanje aplikacij na različnih platformah. Znan moto "napiši enkrat, izvajaj vsepovsod" (ang. *"write once, run anywhere" (WORA)*) pomeni, da programske kode ni potrebno ponovno pretvoriti v izvršljiv program za zagon na drugih platformah. Aplikacije so prevedene v tako imenovano vmesno kodo (ang. `bytecode`), ki je, kot že prej omenjeno, neodvisna od arhitekture, operacijskega sistema ali platforme (podrobni opis v poglavju 2.1.2). Za delovanje Java aplikacij sistem potrebuje programsko opremo imenovano `JRE` (*Java Runtime Enviroment*), za razvoj aplikacij pa `JDK` (*Java Development Kit*).

2.1.1 JRE

`JRE` je postavljen na vrh operacijskega sistema naprave, platforme in arhitekture. Pri zagonu aplikacije `JRE` služi kot povezava med platformo in aplikacijo. Aplikacijo prevede v skladu s platformo, na kateri se aplikacija izvaja. Del `JRE`, ki opravi to povezavo med platformo in aplikacijo, se imenuje *Java Virtual Machine* (`JVM`), ki je zelo pomemben za `JRE`. `JVM` deluje kot virtualni

proces, ki omogoča aplikacijam zagon na lokalnem sistemu. Glavni namen je prevod formata vmesne kode, da prikaže kodo kot strojno kodo (ang. *native*).

2.1.2 Vmesna koda

Vmesna koda oz. *bytecode* je, na kratko povedano, strojni jezik Javanskega virtualnega stroja. Vsakič, ko naredimo spremembo v izvorni kodi in jo prevedemo, nam prinese novo datoteko s končnico `.class`. Ta datoteka vsebuje ukaze za `JVM`, ki ga potem `JVM` prevajalnik tudi tolmači (torej ukaz za ukazom).
Postopek za pridobitev vmesne kode je precej enostaven. Podali bomo primer na izvorni kodi, ki jo vidite v listing 2.1.

```
1 public class Test {
2     public static void main(String[] args) {
3         for (int i = 0; i < 10; i++){
4             System.out.println("Yes");
5         }
6     }
7 }
```

Listing 2.1: Primer izvorne kode, preko katere bomo dobili vmesno kodo.

Ko naredimo prevod zgornje izvorne kode, dobimo v ustrezni datoteki `Test.class` vmesno kodo. Sedaj imamo zagotovljeno izvajanje programa na katerikoli platformi. Če želimo izvedeti, kakšni strojni ukazi sestavljajo naš program, lahko v ukazni vrstici `cmd` (angl. *Command Prompt*) izvedemo naslednji ukaz:
`javap -c Test.class`
Ukaz sestavljajo naslednji parametri:

- `javap`: javin inverzni prevajalnik (angl. `disassembler`), ki izpiše razultat glede na uporabljena stikala;
- `-c`: stikalo, ki izpiše vmesne kode ustrezne datoteke;

- <ime_datoteke>.

```
Compiled from "Test.java"
public class Test {
  public Test();
    Code:
       0: aload_0
       1: invokespecial #1 // Method java/lang/Object."<init>"
          :()V
       4: return

  public static void main(java.langl.String[]);
    Code:
       0: iconst_0
       1: istore_1
       2: iload_1
       3: bipush        10
       5: if_icmpge     22
       8: getstatic     #2 // Field java/lang/System.out:Ljava/
          io/PrintStream;
      11: ldc           #3 // String Yes
      13: invokevirtual #4 // Method java/io/PrintStream.println
          :(Ljava/lang/String;)V
      16: iinc          1, 1
      19: goto          2
      22: return
}
```

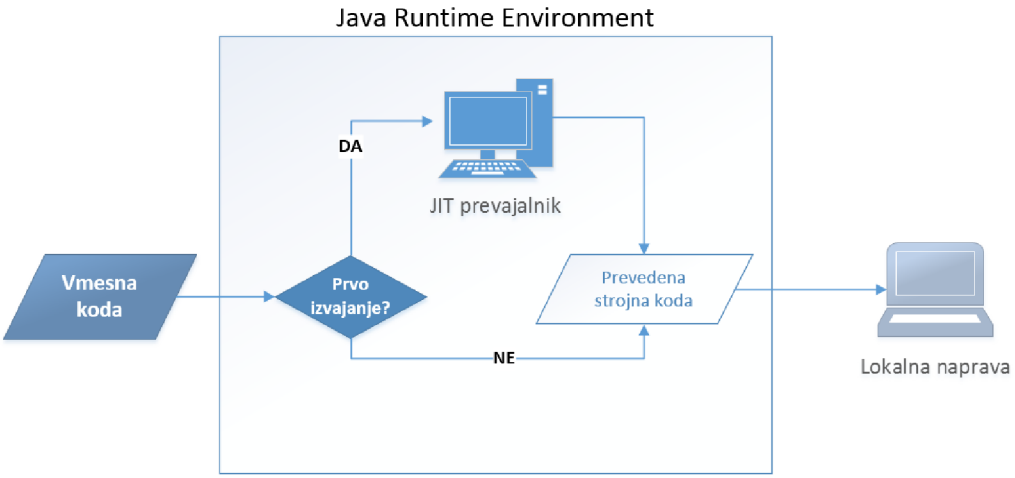
Listing 2.2: Primer vmesne kode.

V podrobnosti se ne bomo spuščali, ker smo samo želeli pokazati delovanje ukaza in njegov izpis, primer pod listing 2.2.

2.1.3 JIT

Čeprav je JVM pomembne del JRE, je šele z uvedbo posebnega prevajalnika doživel spremembo, ki je dodatno izboljšalo delovanje. Ta prevajalnik se imenuje JIT oz. *just-in-time* [5].

Pred uvedbo JIT je JVM deloval tako, da je za vsako izvajanje tolmačil vmesno kodo. To je pomenilo, da ni bilo narejene optimizacije za izvajanje programov. Toda z uvedbo JRE verzije 1.2 se je vse spremenilo. Dodal se je nov prevajalnik, ki je vmesno kodo (ali del vmesne kode) takoj prevedel v strojno kodo (ang. *native code*). To je v bistvu prevedena koda, narejena za točno določen procesor in njegove ukaze. JIT lahko prevede celotno kodo, določeno funkcijo ali pa samo manjši del (odvisno od sprememb na vmesni kodi). Ta koda je lahko prevedena takrat, ko se bo izvajala (od tod ime *just-in-time*) in se potem tudi shrani.
Spodnja slika (2.2) na kratko ponazori delovanje JIT in je bila narejena na podlagi vira [4].



Slika 2.2: Prikaz delovanja JIT.

2.2 NetBeans

Za razvoj aplikacije sem uporabil program NetBeans [1], ki je integrirano razvojno okolje (oz. bolj znana kratica IDE). Omogoča razvoj na več platformah, kot so npr. Windows, Mac OS X, Linux in ostalih, ki imajo podporo za kompatibilni JVM. Njegova značilnost (in podobnih IDE-jev) je razvoj

kompleksnih aplikacij v različnih programskih jezikih, kot so `Java, C, C++, Python`.
Za njega smo se odločili, ker ima možnost razvoja vtičnikov za sam `NetBeans IDE` na dokaj preprost način. Dodatna prednost je tudi preprosta gradnja grafičnih vmesnikov, saj vsebuje vse potrebne gradnike knjižnice `Awt` in `Swing`.

Slika 2.3: NetBeans logo.

2.2.1 Zgodovina

Zgodovina `NetBeans IDE` [6] sega v leto 1996, bolj znano kot `Xelfi` (besedna igra programskega jezika `Delphi`). Začelo se je kot študentski projekt pod vodstvom Fakultete za matematiko in fiziko na Karlovi univerzi v Pragi. Dalje so se študenti (na sliki 2.4) odločili za razširitev projekta na komercialni produkt.

Podjetnik Roman Stanek je iskal idejo za investiranje in odkril `Xelfi`. Skupaj z ustanovitelji so ustvarili `NetBeans IDE`. Leta 1999 je podjetje Sun Microsystems odkupilo le tega in ga naslednje leto ponudil kot odprtokodni produkt. To je povzročilo njegovo bliskovito rast, saj so ljudje uporabljali `NetBeans IDE` platformo za razvoj aplikacij z lastnimi vtičniki. Oracle je odkupil Sun leta 2010, prav tako tudi NetBeans. Podjetje vidi produkt `NetBeans IDE` kot uradni `IDE` za `Java` platformo.

Kot ostali odprtokodni projekti, je tudi ta potreboval čas za pridobitev ustreznih razvijalcev in njihovih prispevkov. Rast nekaterih različic `NetBeans IDE`:

- Na začetku so se največ ukvarjali s temo, kaj deluje in kaj ne. Vse se je poplačalo z različico 3.5, saj so bili opravljeni veliki koraki v

Slika 2.4: Študentje, ki so delali na projektu.

učinkovitosti in preprečevanju nazadovanja. Pri 3.6 so ponovno implementirali lastnosti objektov in spremenili ter poenostavili grafični vmesnik.

- Različica 4 je prinesla spremembe delovanja `IDE`. Novi sistem projektov je prenovila, tako uporabniško izkušnjo kot možnost zamenjave infrastrukture. Z 4.1. so pridobili nove posebnosti in polno podporo `J2EE`.

- Različica 5 je predstavila celovito podporo za razvoj `IDE` modulov, aplikacij na strani klienta, inovativno orodje za gradnjo grafičnih vmesnikov (`Matisse`), nova in predelana podpora `CVS`, podpora za `Sun ApplicationServer 8.2, Weblogic 9` in `JBoss 4`.

- Pri različici 6 je bil fokus predvsem za izboljšanje produktivnosti razvijalca. To je omogočil hitrejši, pametnejši in predelan urejevalnik.

- Različico 7 so ponudili javnosti skupaj z `JDK 7`. V 7.1 različici so dodali orodje za delo s konstruktorji jezika `JDK 7`, dalje pri 7.2 izboljšali

učinkovitost pri IDE in pri 7.3 dodali podporo aplikacijam tehnologije HTML5.

- Različica 8 je omogočila podporo uporabe JDK 8, nagradila razvoj HTML5 z novimi orodji (npr. AngularJS) in izboljšala podporo uporabe Maven, C/C++ in PHP.

Potrebno je omeniti, da je mnogo prvotnih "arhitektov" projekta NetBeans še vedno prisotnih, ki skrbijo za rast le tega. Leta 2010 je število aktivnih uporabnikov programske opreme presegla število 1,000,000.

Poglavje 3

Implementacija

V tem poglavju se bomo osredotočili na implementacijo programa. Omenili bomo pomembne metode v Javi, ki smo jih uporabili oz. sprogramirali Ker je za delovanje programa potrebnih več gradnikov, se bomo osredotočili na vsakega posebej. Potrebno je še omeniti, da smo za svoje namene uporabili igralni plošček (angl. *gamepad*) `Saitek P2500 Rumble` (na sliki 3.1) in ne igralne palice, saj vseeno omogoča normalno testiranje našega programa. Ima sledeče lastnosti:

- USB povezava s računalnikom,
- 6 gumbov,
- dve analogni palici,
- ostalo (osem smerni D-pad, dva prožilca, itd.).

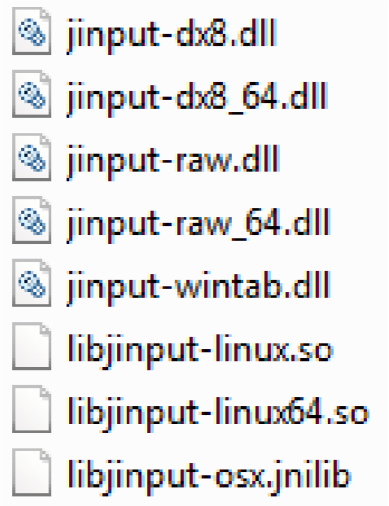
3.1 Premik kazalca

3.1.1 JInput

Za kontrolo igralne palice smo pri svoji implementaciji uporabili knjižnico `JInput`. Omogoča upravljanje igralnih palic več vrst in pridobivanja različnih

Slika 3.1: Igralni plošček Saitek P2500 Rumble.

podatkov med samim delovanjem. Njen prvotni namen je ustvarjalcem omogočiti razvoj visoko kakovostne igre v programskem jezika `Java`.
Knjižnica je odprtokodna in deluje na več platformah, kar je tudi opisano na spletni strani [11]. Če knjižnice ne bi uporabili, bi izgubili veliko časa za implementacijo zaznavanja in premik vhodno-izhodne naprave.
Na začetku smo se obrnili na navodila za namestitev vseh potrebnih datotek, ki je na voljo na spletni strani [12]. Najprej smo prenesli vse potrebne datoteke na računalnik in jih kasneje prenesli v ustrezno mapo projekta. Za popolno delovanje smo projektu sklicevali tudi `jinput.jar`, ki nam omogoča uporabo vseh razredov in metod, ki spadajo v `JInput` knjižnico (slika 3.2). Po končani namestitvi, smo se osredotočili na programski del. Najprej je potrebno pridobiti seznam vseh igralnih pripomočkov, ki jih operacijski sistem prepozna, in jih shraniti v seznam. Za ta namen smo ustvarili posebno metodo, ki na lokalnem računalniku najprej pridobi seznam vseh naprav (s pomočjo ukaza `ControllerEnvironment.getDefaultEnvironment()` `.getControllers()`) ter jih shrani v objekt razreda `Controller[]`. Ker



Slika 3.2: Namestitev potrebnih datotek za razvoj aplikacije.

smo potrebovali le igralne palice in ploščke, je bilo potrebno preveriti tip samega objekta. Vsak objekt je bilo možno preveriti z ukazom `getType()`, za naše namene smo shranjevali naprave `STICK` ali `GAMEPAD`. Vsak tak tip smo potem shranili v seznam razreda `ArrayList`. Ta omogoča dinamično dodajanje elementov (v našem primeru tip `Controller`). Na koncu smo vrnili seznam za nadaljno uporabo v programu. Spodaj je izvorna koda, listing 3.1, ki prikazuje takšno pridobivanje in dodajanje v seznam:

```
1  import java.util.ArrayList;
2  import net.java.games.input.Controller;
3  import net.java.games.input.ControllerEnvironment;
4
5  public class ControllerJoystick {
6
7      public ArrayList<Controller> contList;
8
9      public ArrayList<Controller> getControllers(){
10
11         Controller[] allCon = ControllerEnvironment.
               getDefaultEnvironment().getControllers();
```

```
12          contList = new ArrayList<>();
13          for (Controller allCon1 : allCon) {
14            if (allCon1.getType() == Controller.Type.STICK ||
                  allCon1.getType() == Controller.Type.GAMEPAD)
15              {
16                  contList.add(allCon1);
17              }
18          }
19
20         return contList;
21        }
22 }
```

Listing 3.1: Primer izvorne kode, preko katere bomo seznam vseh igralnih palic in ploščkov.

S tem smo pokrili en del, ostalo je še pridobivanje podatkov igralnega ploščka in premikanje miške. Pri razvoju nam je pomagal program, ki je na voljo na spletni strani [13].
Za pridobivanje podatkov igralnega ploščka smo uporabili JInput, medtem ko za premik miškinega kazalca razred `java.awt.Robot`. V naslednjem odstavku, ga bomo na kratko opisali, preden nadaljujemo s našim programskim delom.

3.1.2 java.awt.Robot

`java.awt.Robot` je razred, katerega glavni namen je prevzem nadzora miške in tipkovnice. Podrobneje o njegovih lastnostih lahko preberete na strani [14]. Razred vsebuje več zanimivih metod:

- premik miške (`mouseMove`),
- pritisk miškinih tipk (`mousePress` in `mouseRelease`),
- premik koleščka (`mouseWheel`),
- pritisk tipk na tipkovnici (`keyPress`),

- zakasnitev za določeno število milisekund (`delay`),
- zajem slike (`createScreenCapture`),
- itd.

Delovanje razreda je preprosto. Za prikaz si bomo izmislili primer: izpiši "Dober dan!" v tekstovno datoteko. Najprej je potrebno ustvariti objekt razreda `Robot`. Preko tega objekta lahko potem manipuliramo tako miško kot tipkovnico. Najprej bi želeli imeti dovolj časa, da odpremo preprost tekstovni program. Zato je potrebno narediti preprosto zakasnitev za nekaj sekund. To storimo z metodo `delay (int stMilisekund)`, kjer v oklepaju določimo dolžino zakasnitve. Ker metoda sprejema milisekunde, je potrebno upoštevati pretvorbo v sekunde. S to zakasnitvijo imamo dovolj časa, da po zagonu programa odpremo tekstovni program (npr. `notepad`). Nato lahko v preprostem zaporedju kličemo metodo `keyPress(crka)`, kjer v oklepajih določimo črko kot dogodek tipkovnice (vir [15]). Lahko se držimo zaporedja `keyPress(crka)` metod, če pa želimo pravilno prikazati velike črke in posebne zanke, je potrebno znak `SHIFT` tudi prekiniti. To storimo s `keyRelease(crka)` metodo. Ta primer je prikazan spodaj v izvorni kodi (listing 3.2):

```
1  import java.awt.AWTException;
2  import java.awt.Robot;
3  import java.awt.event.KeyEvent;
4
5  public class TestRobot {
6
7      public static void main(String[] args) throws AWTException {
8
9              Robot robot = new Robot();
10             robot.delay(3000);
11             robot.keyPress(KeyEvent.VK_SHIFT);
12             robot.keyPress(KeyEvent.VK_D);
13             robot.keyRelease(KeyEvent.VK_SHIFT);
14             robot.keyPress(KeyEvent.VK_O);
```

```
15                robot.keyPress(KeyEvent.VK_B);
16                robot.keyPress(KeyEvent.VK_E);
17                robot.keyPress(KeyEvent.VK_R);
18                robot.keyPress(KeyEvent.VK_SPACE);
19                robot.keyPress(KeyEvent.VK_D);
20                robot.keyPress(KeyEvent.VK_A);
21                robot.keyPress(KeyEvent.VK_N);
22                robot.keyPress(KeyEvent.VK_SHIFT);
23                robot.keyPress(KeyEvent.VK_1);
24                robot.keyRelease(KeyEvent.VK_SHIFT);
25        }
26 }
```

Listing 3.2: Kratek program, ki prikazuje delovanje nekaterih metod razreda Robot.

3.1.3 Premik in kontrola

V prejšnjem poglavju smo na kratko opisali delovanje razreda `java.awt.Robot`, ki nam pomaga pri implementaciji. Sedaj bomo opisali, kako smo sprogramirali premik in kontrolo miške s pomočjo igralnega ploščka.

Za normalno istočasno delovanje modula in kontrole kazalca, smo ustvarili novo `nit` [16]. Niti (angl. *thread*) omogočajo istočasno delovanje programske kode z glavnim programov. V našem primeru imamo glavni program (t.j. modul) in premik kazalca s pomočjo igralnega ploščka. Ker je za naše potrebe potrebno preverjati stanje v zanki, je za pravilno delovanje programa nujno ustvariti nit. Če je ne bi ustvarili, bi v našem primeru program zmrznil. Lahko bi sicer kontrolirali kazalec, toda modul zaradi zanke ne bi deloval. Spodaj je primer iz naše izvorne kode, ki prikazuje ustvarjanje niti:

```
1 new Thread(new Runnable() {
2   public void run(){
3     while (isRunningThread) {
4       if(selectedCont.poll()){
5       /* sledi pridobivanje in manipuliranje komponent */
6       ...
```

```
7         }
8       }
9     }
10 }).start();
```

Listing 3.3: Izvorna kode za ustvarjanje in zagon niti.

Na kratko: nit bo normalno delovala, dokler bo izbrana igralna palica (ali plošček), delovala normalno (lahko se zgodi nepričakovano prenehanje delovanja). To preverjamo s `selectedCont.poll()`, ki bere podatke naprave. Dodali smo tudi pogoj `isRunningThread`, ki lahko ustavi delovanje v takšnih primerih in tudi takrat ko uporabnik onemogoči delovanje. Možno je s pritiskom na gumb ustaviti delovanje igralne palice, ne da bi popolnoma ustavili program. Niti so bile torej prva težava, ki smo jo rešili.
Ko smo to rešili, smo se lotili na pridobivanju vseh igralnih pripomočkov tipa `STICK` in `GAMEPAD`. Pridobiti je bilo potrebno vse komponente (gumbi in smeri premika igralnega ploščka). Omeniti moramo, da ima naš modul spustni meni z vsemi igralnimi palicami in ploščki, zato moramo izbranega poiskati, preden zaženemo nit. Postopek je sledeč:

1. Poiskati izbrani igralni pripomoček.
2. Izbrati komponenti za smeri X (`x_axis`) in Y (`y_axis`).
3. V ustrezen seznam spraviti vse gumbe, ki jih bomo kasneje uporabili.

Spodaj je obsežen in pomemben del izvorne kode, listing 3.4, ki ponazori to iskanje:

```
1 if(toggleJoystick.isSelected()){
2   String t = controllerList.getSelectedItem().toString();
3
4   for (Controller selectedCont: newsticks){
5     if(selectedCont.getName().equals(t)){
6       try{
7         components = selectedCont.getComponents();
8       }
```

```
9         catch(NullPointerException e){
10          System.exit(1);
11        }
12
13        buttons= new HashMap<>();
14        for (Component component : components) {
15          if (component.getName().equalsIgnoreCase("x axis") &&
                component.isAnalog()) {
16            x_axis = component;
17          }else if (component.getName().equalsIgnoreCase("y axis")
                && component.isAnalog()) {
18            y_axis = component;
19          }else if(component.getName().contains("Button")){
20            buttons.put(component.getName(),component);
21          }
22        }
23          }
```

Listing 3.4: Izvorna kode za pridobivanje pravega igralnega pripomočka in njegovih komponent.

Potem je bilo potrebno znotraj niti uporabiti knjižnico `JUnit` in razred `java.awt.Robot`. V sami knjižnici obstaja način za pridobivanje informacij, ki smo jih lahko na preprost način tudi uporabili, izvorna koda listing 3.5. Postopek je sledeč:

1. Pridobiti informacijo o položaju miškinega kazalec.
2. Shraniti trenutno vrednost položaja.
3. Pridobiti podatke o smernih komponentah X in Y (`getPollData()`).
4. Nastaviti nov položaj X in Y, kjer trenutnemu položaju seštejemo nov podatek in ga zmnožimo z neko fiksno vrednostjo. Ta vrednost določa končno hitrost premika.
5. Če smo pritisnili na ustrezen gumb igralnega pripomočka, se je izvršila ustrezna akcija (v našem primeru se je odprl zavihek z rezerviranimi

besedami - pogoj `getPollData()` = 1.0 predstavlja pritisk na gumb).

6. Če je prišlo do premika analogne palice na igralnem pripomočku, se je naredil ustrezen premik na nov položaj. Bolj podroben opis izvorne kode, je prikazan pri listing 3.6.

```
1  mousePosition = MouseInfo.getPointerInfo().getLocation();
2  currX = mousePosition.x;
3  currY = mousePosition.y;
4
5  float xData = x_axis.getPollData();
6  float yData = y_axis.getPollData();
7  newX = (int) (xData*speedCon) + currX;
8  newY = (int) (yData*speedCon) + currY;
9
10 if (buttons.get(reservedWordsCB.getSelectedItem().toString()).
       getPollData()==1.0){
11   jTabbedPane1.setSelectedComponent(rwPane);
12 }
13 /* ostala izvorna koda */
14 ...
15 if (newX != currX || newY != currY) {
16   moveMouseCursor(r, currX,currY,newX,newY,10,5);
17 }
```

Listing 3.5: Manipuliranje komponent.

Kot ste najbrž opazili, obstaja metoda `moveMouseCursor`, ki naredi premik na nov položaj. Metoda razdeli razdaljo med dvema položajema na `n` delov in naredi premik s pomočjo metode `mouseMove`. V vsaki iteraciji se kazalec premakne za odsek poti. Da dosežemo gladek premik, je bilo potrebno uporabiti metodo `delay`, kjer naredimo zakasnitev za fiksno število milisekund. Tako `mouseMove` kot `delay`, sta metodi razreda `java.awt.Robot`, ki je bolj podrobno opisan v poglavju 3.1.2. [17].

```
1  public static void moveMouseCursor(Robot robot, int x_1, int y_1
       , int x_2, int y_2, int t, int n) {
2    double dx = x_2 - x_1 ;
```

```
3     double dy = y_2 - y_1;
4     for (int i = 1; i <= n; i++) {
5                 robot.delay((int)(t/(double)n));
6                 robot.mouseMove((int) (x_1 + dx/n * i), (int) (y_1 +
                      dy/n * i));
7     }
8  }
```

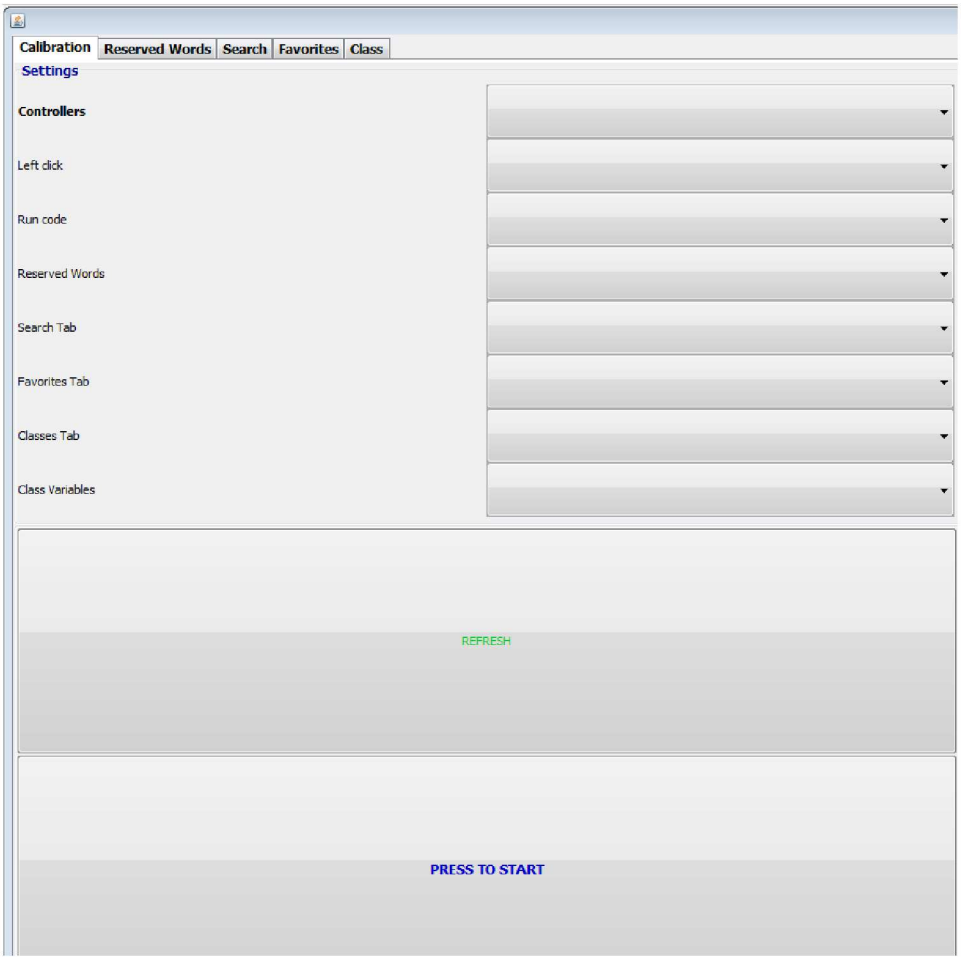
Listing 3.6: Premik kazalca s pomočjo `java.awt.Robot`.

3.2 Nastavitve igralnega pripomočka

Zavihek *Calibration* (prikaz na sliki 3.3) je namenjen nastavitvam igralnega pripomočka. Preden uporabnik začne z delom, mora najprej izbrati svoj igralni pripomoček (palico ali plošček). Ko je pripomoček priključen v računalnik in je namestitev gonilnikov končana, se lahko začne uporabljati modul. Za uporabo večjega nabora pripomočkov smo uporabili spustni seznam vseh, ki jih sistem zazna. Za osvežitev seznama, obstaja gumb *Refresh*, ki naredi ravno to: osveži seznam igralnih pripomočkov in gumbov. Za namen testiranja, smo implementirali določene funkcionalnosti:

- klik miške,
- zagon programa,
- hitri dostop do rezerviranih besed,
- hitri dostop do zavihka iskanja razredov in tipkovnice,
- hitri dostop do zavihka priljubljenih stvari,
- hitri dostop do zavihka lastnosti razreda,
- hitri dostop do zavihka lastnosti razreda, kjer se izpišejo spremenljivke programa, ki ga pišemo.

Uporabnik lahko za vsako možnost preko spustnega seznama določi svoj gumb. Vse kar je potrebno, je da se določijo vsi gumbi, ki se bodo uporabili. Poleg tega morajo biti vsi različni. S tem bo delovanje optimalno. Ko je uporabnik zadovoljen z nastavitvami, lahko požene igralni pripomoček s pritiskom na gumb *Press to start*. Od tega trenutka naprej, bodo funkcije omogočene. Seveda lahko tudi preneha z izvajanjem, s pritiskom na isti gumb (kjer se spremeni tekst v *Stop*). Dodatno pojasnilo o premiku miškinega kazalca, smo opisali v poglavju 3.1.



Slika 3.3: Videz zavihka *Calibration*.

3.3 Rezervirane besede

Grafični vmesnik za programiranje omogoča izbiro vnaprej definiranih ukazov, funkcij ali lastnosti. Ti so podani v navadni tekstovni datoteki. S pomočjo podprograma, listing 3.7, preberemo datoteko, razločimo in "predelamo" vrednosti, rezultat shranimo v slovar ter ga posredujemo programu. Datoteko odpremo s pomočjo ukaza `File` in dalje beremo z ukazom `Scanner`. Pred branjem definiramo še slovar za hrambo podatkov. Ključi slovarja so funkcionalnost nabora, predstavljeni kot niz. Vrednost posameznega ključa je predstavljena kot tabela oznak. Definiranje slovarja je prikazano v prikazani kodi pod ukazom `Map` oz. `HashMap`. Datoteko smo definirali tako, da ločilo podpičje loči vrednosti ključa in nabora. Ukaz `Scanner` samodejno s pomočjo ločila podpičja shrani vrednosti v vsako vrstico posebej. Tako lahko prvo vrstico uporabimo kot ključ in naslednjo kot njene vrednosti. Funkcija `while` nam omogoči avtomatski preskok na ustrezno vrstico, dokler le ta ni pri koncu. Vrstici shranimo v spremenljivki kot niz, kjer jima odstranimo odvečne znake. Spremenljivko, kjer so shranjene vrednosti, uporabimo za ustvarjenje tabele vrednosti. S pomočjo funkcije `for` dinamično dodajamo nize v tabelo `ArrayList`. To tabelo dodamo v slovar pod ustrezen ključ. Za konec slovar posredujemo glavnemu programu za uporabo le tega.

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.ArrayList;
4  import java.util.HashMap;
5  import java.util.Map;
6  import java.util.Scanner;
7  import javax.swing.JLabel;
8
9  public class ReservedWords{
10
11     private final String contentName;
12     public ReservedWords(String contentName){
13         this.contentName = contentName;
14     }
```

```
15      public Map<String, ArrayList<JLabel>> retList(){
16
17              // Iscemo v datoteki
18          File file = new File(contentName+".txt");
19          Map<String, ArrayList<JLabel>> dict = new HashMap<>();
20          try {
21              try (Scanner scanner = new Scanner(file).
                    useDelimiter(";")) {
22
23                  while (scanner.hasNext()) {
24                      String keyword = scanner.next().replace("\r\
                            n", " ").replace("\n", " ");
25                      String resWords = scanner.next().replace("\r
                            \n", " ").replace("\n", " ");
26                      String splitSpace[]= resWords.split(" ");
27                      ArrayList<JLabel> arrayLabel = new ArrayList
                            <>();
28                      for(String t:splitSpace){
29                          if (!"".equals(t)){
30                              arrayLabel.add(new JLabel(t));
31                          }
32                      }
33                      // vstavi v slovar razred JLabel
34                      dict.put(keyword, arrayLabel);
35                  }
36              }
37          } catch (FileNotFoundException e) {}
38      return dict;
39      }
40  }
```

Listing 3.7: Izvorna koda za pridobivanja rezerviranih besed.

Za lažjo predstavo si lahko pogledamo videz zavihka na spodnji sliki 3.4. Ko uporabnik pritisne na eno od prikazanih rezerviranih besed, se bo avtomatsko izpisala na oknu za urejanje kode. Beseda se bo izpisala na položaju kazalca.

Slika 3.4: Videz zavihka *Reserved Words*.

3.4 Navidezna tipkovnica in iskalnik razredov

Za naše namene smo se odločili, da bomo na našem modulu prikazali tudi navidezno tipkovnico in iskalno okno, ki sta sestavni del našega iskanja razredov. Za to smo se odločili, da pospešimo delovanje našega programa in olajšamo delo programerjev. Najprej je bilo potrebno poiskati način, da pridobimo seznam vseh razredov. Za iskanje lokacije razredov, smo uporabili

metodo `System.getProperty`, ki poišče lastnosti sistema glede na vhodni parameter. Za naše namene smo uporabili `sun.boot.class.path`, ki v bistvu pomeni lokacijo razredov [18]. Vsi pomembni razredi se nahajajo v arhivu `rt.jar`, ki smo ga morali rekurzivno pregledati. Algoritem je torej sledeč (na kratko opisan v kodi, listing, 3.8):

1. Poiščemo lokacijo razredov, ki so zbrani v arhivu.

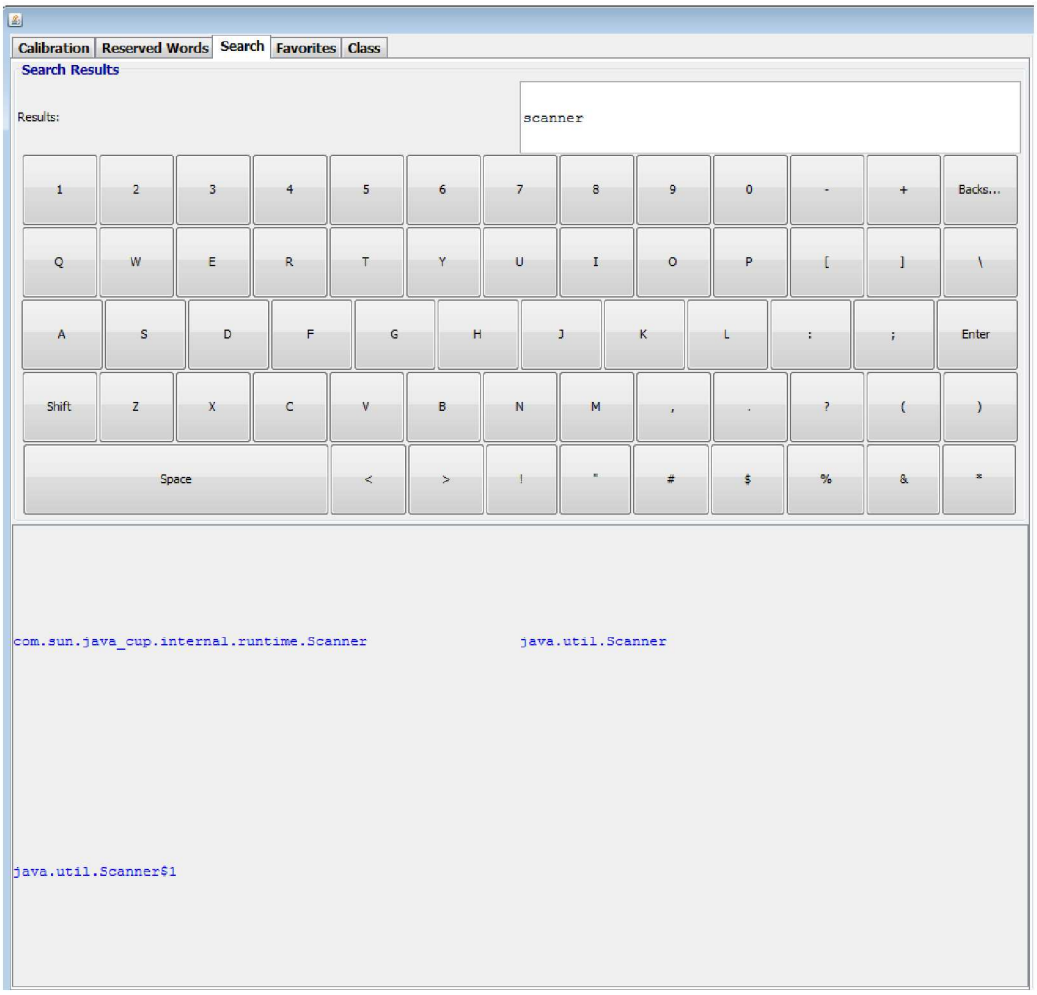
2. Rekurzivno poiščemo vse arhive (če jih obstaja več).

3. Pogledamo vsebino arhivov (`contentOfJar`), kjer zbiramo samo imena vseh s končnico `.class` [19].

```
1  String loc = System.getProperty("sun.boot.class.path");
2
3  for (String path:loc.split(";")){
4     File file = new File(path);
5     if (file.isDirectory() && file.exists())
6       recursiveSearch(path);
7     else
8       content.add(file);
9  }
10 for (File file_temp: content) {
11   if (file_temp.getPath().endsWith(".jar") && file_temp.exists()
         ){
12     contentOfJar(file_temp.getPath());
13   }
14 }
```

Listing 3.8: Iskanje vseh razredov

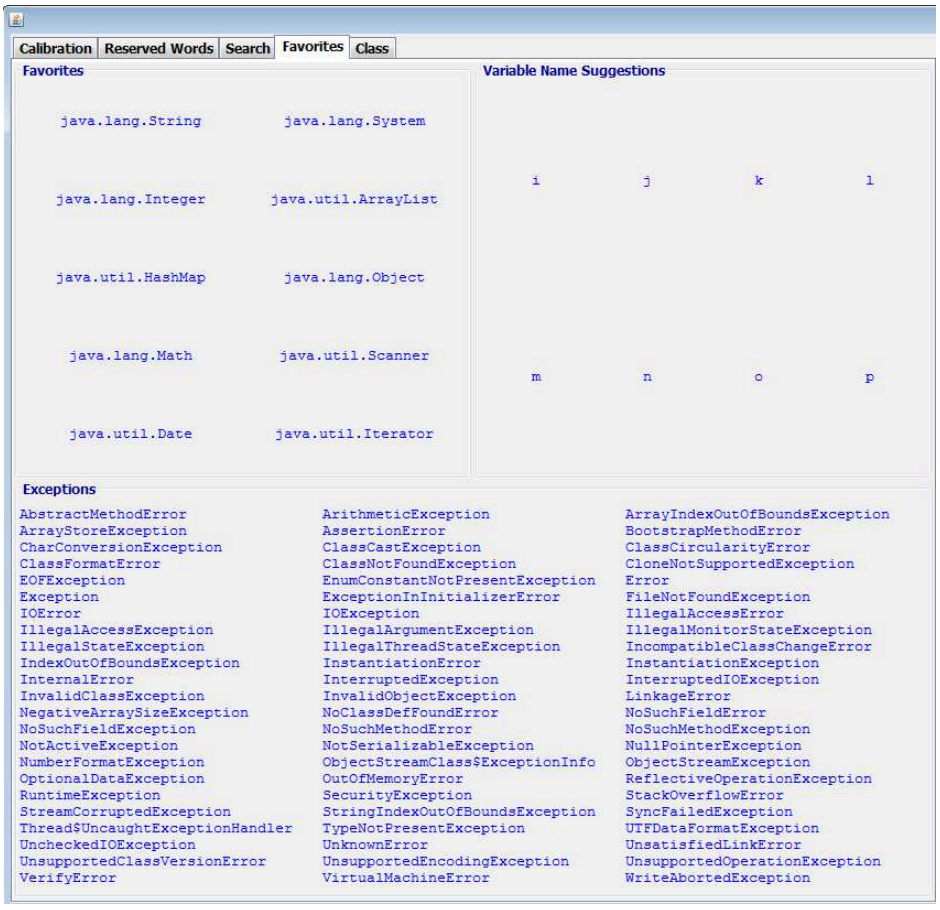
Na sliki 3.5 lahko vidimo prikaz in delovanje iskalnika razredov. Uporabnik natipka vsako črko posebej in v okno *Results* se prikaže zaporedje znakov. Iskanje se zažene dinamično, torej ni potrebna nobena potrditev s strani uporabnika. S tem sta zagotovljena večja fleksibilnost in hitrost. Ko uporabnik klikne na posamezno območje, se razred avtomatsko pokaže na oknu.



Slika 3.5: Videz zavihka Search.

3.5 Priljubljene

Zavihek *Favorites* vsebuje programske lastnosti `Jave`, ki bi programerju omogočala lažje programiranje. Za ta zavihek smo se odločili, da bi lahko uporabnik še hitreje končal svoj program. Potrebno je omeniti, da je za naše potrebe povsem dovolj seznam najbolj priljubljenih razredov in izjem, toda dodali smo tudi predloge spremenljivk, ki se največ uporabijo. Zavihek je prikazan na sliki 3.6.



Slika 3.6: Videz zavihka *Favorites*.

3.6 Lastnosti razreda

Zavihek *Class* vsebuje lastnosti razredov, ki se pojavijo v času programiranja. Tako kot zavihek *Favorites*, v poglavju 3.5, je ta namenjen hitrejšemu programiranju. Ko uporabnik izbere razred v iskalniku razreda ali med priljubljenimi, se ta zapiše v urejevalniku kode. Po zapisu se prikažejo lastnosti dodanega razreda (glej sliko 3.7), kot npr.:

- metode,
- možni parametri konstruktorja,
- spremenljivke (angl. *field*) dodanega razreda,
- spremenljivke, ki jih pišemo v program.

Da smo lahko to omogočili, je bilo potrebno uporabiti *Reflection API*. Za razlago in pomoč pri implementaciji, smo si pomagali s podanim primerom [21]. V našem programu smo ga večkrat uporabili, zato bomo njegovo implementacijo na kratko opisali. Najprej je potrebno ustvariti razred. To storimo z ukazom `Class.forName(arg)`, kjer kot argument navedemo polno ime (npr. `java.util.Scanner` - ime paketa). Ukaz nam vrne razred (npr. `Scanner`). Nato lahko dobimo njegove lastnosti. V našem primeru smo uporabili naslednje ukaze:

- `getDeclaredMethods()` - metode razreda,
- `getConstructors()` - konstruktorji razreda,
- `getFields()` - spremenljivke razreda.

Primer pridobivanja podatkov razreda je prikazan v izvorni kodi, 3.9. Najprej je potrebno naložiti nov razred, nato pa lahko nad njim uporabimo enega od ukazov (v našem primeru `getFields()`). Ker ukaz vrne seznam objektov tipa `Field`, je potrebno ta seznam shraniti v novo spremenljivko. Na koncu se sprehodimo po vseh objektih, kjer lahko manipuliramo s podatki.

```
1 Class newClass = Class.forName("java.util.Scanner");
2 Field[] fields = newClass.getFields();
3 for (Field field:fields)
4 {
5   System.out.println(field.getName());
6 }
```

Listing 3.9: Primer uporabe razreda `newClass`.s

Slika 3.7: Videz zavihka `Class`.

Za naše namene, smo pridobili tudi spremenljivke, ki jih uporabljamo v programu. V tem primeru, je potrebno najprej prevesti napisan program, nato pa ga s pomočjo razreda `URLClassLoader` naložimo (primer na spletni strani [20]). Sledi izvorna koda, ki pokaže uporabo razreda (listing 3.10):

```
1 try {
2   classLoader = URLClassLoader.newInstance(new URL[] { root.
        toURI().toURL() });
3 } catch (MalformedURLException ex) {
4   Exceptions.printStackTrace(ex);
5 }
6 //ime razreda, parameter inicializacije razreda, loader
7 Class varClass = Class.forName("Variables", true, classLoader);
```

Listing 3.10: Primer uporabe razreda `URLClassLoader`.

3.7 Urejevalnik kode v aplikaciji

V tej sekciji bomo opisali še okolje, ki ga programer uporablja za svoje delo. Ponazorili bomo uporabo poslušalcev miške, prevajanja in izvajanja programa ter prikazali način preprostega programa.

3.7.1 Poslušalec miške

Da so se lahko stvari pravilno prenesle na programersko okno, je bilo potrebno sprogramirati poslušalca miške (ang. *MouseListener*). Potrebno je omeniti, da naš program vsebuje več različnih poslušalcev za različne namene, toda podrobno bomo opisali samo enega. Za ponazoritev bomo uporabili poslušalca miške pri programu rezerviranih besed (prikaz kode listing 3.11).

V Javi obstaja razred `MouseAdapter`, ki je namenjen prejemanju dogodkov miške. Dogodkov je lahko več, v našem primeru smo uporabili `mouseClicked` (klik na miškin gumb), `mouseEntered` (dogodek ob vhodu miške v območje objekta) in `mouseExited` (dogodek ob izhodu miške v

območje objekta). Podroben opis je na voljo na spletni strani o razredu `MouseAdapter`, [22]. Obstaja jih še več:

- `mouseDragged` (dogodek ob pritisku miškinega gumba in njenega premika);
- `mouseMoved` (dogodek ob premiku miške na posamezni komponenti);
- `mousePressed` (dogodek ob pritisku miškinega gumba na komponenti);
- `mouseReleased` (dogodek ob sprostitvi miškinega gumba na komponenti);
- `mouseWheelMoved` (dogodek ob rotiranju miškinega koleščka).

Prvi korak je bil definiranje dogajanja ob dogodku `mouseClicked`. Nato je bilo potrebno dobiti objekt tiste komponente, na katero uporabnik vpliva. Ta objekt je v našem primeru razred `JLabel`, na katerega je uporabnik kliknil. V urejevalnik kode se izpiše tekst, ki predstavlja ime objekta. Če bi uporabnik kliknil na objekt `int`, bi se na trenutni položaj kazalca v urejevalniku prikazal tekst. To je določeno s ukazom `insertString(caretPos, jc.getText()+‘‘ ’’, null)`, kateremu najprej podamo položaj kazalca (`javaEditor.getCaretPosition()` in nato tekst izvornega objekta. Zadnji argument je prazen. Za dogodke `mouseEntered` in `mouseExited` smo se odločili, da bosta namenjena definiranju robov objektov (v našem primeru robovi razreda `JLabel`). Če se miška nahaja v območju objekta (`mouseEntered`), je potrebno narisati robove, sicer pa odstraniti (`mouseExited`). Primer izvorne kode spodaj, listing 3.11.

```
1 mouseListenerRW = new MouseAdapter(){
2
3  public void mouseClicked(MouseEvent e){
4   JLabel jc = (JLabel)e.getSource();
5   int caretPos = javaEditor.getCaretPosition();
6   try {
7    if ("main".equals(jc.getText())){
```

```
8       javaEditor.getDocument().insertString(caretPos, "public
            static void main (String[] args){\n}", null);
9     }else if("try".equals(jc.getText())){
10      javaEditor.getDocument().insertString(caretPos, "try{\n\n}
            catch( ){}", null);
11      jTabbedPane1.setSelectedIndex(3);
12    }else
13      javaEditor.getDocument().insertString(caretPos, jc.getText
            ()+" ", null);
14    } catch (BadLocationException ex) {
15        Exceptions.printStackTrace(ex);
16      }
17  }
18
19  public void mouseEntered(MouseEvent e){
20   Object source = e.getSource();
21   if (source primerkiof JLabel){
22    JLabel sourceLabel = (JLabel)source;
23    sourceLabel.setBorder(BorderFactory.createLineBorder(Color.
          BLACK));
24   }
25  }
26
27  public void mouseExited(MouseEvent e){
28   Object source = e.getSource();
29   if (source instanceof JLabel){
30    JLabel sourceLabel = (JLabel)source;
31    sourceLabel.setBorder(null);
32   }
33  }
34 };
```

Listing 3.11: Uporaba poslušalca rezerviranih besed.

3.7.2 Prevod in izvajanje programa

Za testiranje pravilnosti naših programov je bilo potrebno tudi implementirati prevajanje in izvajanje izvorne kode. Uporabnik piše izvorno kodo v posebno tekstovno polje. To polje je razred `JEditorPane`, ki je namenjen za pisanje navadnega teksta in nam omogoča označevanje jezika kot npr. `HTML`, `Java`, itd (prikaz na sliki 3.8). Opis razreda je na voljo na spletni strani [24]. S preprostim zaporedjem dveh ukazov lahko določimo označevanje sintakse programskega jezika, v našem primeru `Java` (primer izvorne kode se nahaja spodaj pod listing 3.12).

```
1 EditorKit kit = CloneableEditorSupport.
2 getEditorKit("text/x-java");
3 javaEditor.setEditorKit(kit);
```

Listing 3.12: Označevanje sintakse v `JEditorPane` tekstovnem polju.

Del izvorne kode našega prevajalnika je prikazano na listing 3.13. Najprej smo se odločili, da bomo izpisali vse napake v izvorni kodi. Potrebno je bilo definirali poseben razred `OutputStream` (podrobnosti o razredu je na voljo na spletni strani [23]), ki nam napake pri prevodu izpiše v izhodno okno (ukaz `insertString(doc.getLength(), string, null)`). Izpis je prikazan na sliki 3.8. Objekt `out` potem dodamo kot argument za tok podatkov prevajalnika. V našem modulu se izpišejo vsi rezultati (tako uspešni kot neuspešni) v izhodno okno, ki je preprost `JTextPane` razred.

Nato je bilo potrebno izvorno kodo shraniti v datoteko in jo prevesti, primer v kodi, listing 3.13. To lahko dosežemo z ukazom `compiler.run(null, null, out, ‘‘-nowarn’’, fileToCompile)`, ki vsebuje sledeče parametre:

- Prvi trije parametri so namenjeni toku podatkov, kjer je prvi standardni vhod, drugi standardni izhod in tretji standardni izhod napak.
- `-nowarn` ignorira opozorila (angl. *warnings*).
- Vhodna datoteka, ki jo je potrebno prevesti.

```
1  private int compileCode() throws IOException{
2    OutputStream errOut = new OutputStream() {
3
4      public void write( int b) throws IOException {
5          updateTextPane(String.valueOf((char) b));
6      }
7
8      private void updateTextPane( String string) {
9        SwingUtilities.invokeLater(() -> {
10         Document doc = outputWindow.getDocument();
11         try {
12       doc.insertString(doc.getLength(), string, null);
13         } catch (BadLocationException ex) {
14             Exceptions.printStackTrace(ex);
15         }
16       });
17     }
18   };
19
20   ...
21
22   JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
23   int compileRes = compiler.run(null, null, errOut,"-nowarn",
       fileToCompile);
24   return compileRes;
```

Listing 3.13: Prevajanje izvorne kode.

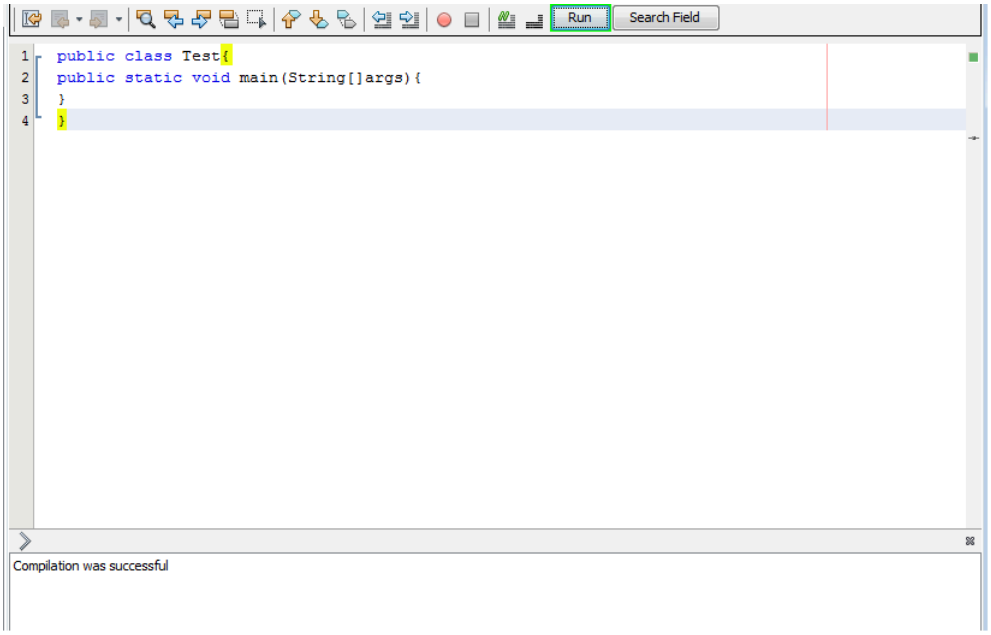
Rezultat ukaza `compiler.run` je celoštevilska vrednost. Če je rezultat enak 0, je bilo prevajanje uspešno, sicer pa ne. S pomočjo tega rezultata, lahko potem določimo, ali se bo naš program tudi izvedel (prikaz izvorne kode v listing 3.14). Za izvajanje programa smo ustvarili in izvedeli proces `java <ime programa>` (primer izvajanja je na voljo na spletni strani [25]), ki lahko zahteva vhodne podatke (`stdin` - primer razred `Scanner`), izpiše rezultat (`stdout`) ali izpiše napake (`stderr`). Če hoče uporabnik programirati s pomočjo `Scanner` razreda, se mora držati določenih pravil. V tekstovno po-

lje `Results` (t.j. `searchField`) mora ob izvajanju zapisati svoj željeni vhod oz. vhode, če je več zahtev (ločilo je podpičje). Za izpis podatkov programa beremo in izpisujemo tok podatkov procesa (isto velja za napake - `stderr`).

```
1  Process process = Runtime.getRuntime().exec("java "+javaEditor.
       getText().split("class")[1].split("[^a-zA-Z0-9']+")[1]);
2
3  OutputStream stdin = process.getOutputStream ();
4  InputStream stderr = process.getErrorStream ();
5  InputStream stdout = process.getInputStream ();
6
7  String input = searchField.getText();
8  try {
9    for(String tempIntput:input.split(";")){
10     line=tempIntput+"\n";
11     stdin.write(line.getBytes() );
12     stdin.flush();
13   }
14     stdin.close();
15 } catch (IOException ex) {
16     Exceptions.printStackTrace(ex);
17   }
18
19 try {
20   BufferedReader bufferedReaderEditor = new BufferedReader (new
         InputStreamReader (stdout));
21   try {
22     while ((line = bufferedReaderEditor.readLine ()) != null) {
23       try {
24         doc.insertString(doc.getLength(), line+"\n",null );
25       } catch (BadLocationException ex) {
26         Exceptions.printStackTrace(ex);
27       }
28     }             } catch (IOException ex) {
29       Exceptions.printStackTrace(ex);
30     }
31   bufferedReaderEditor.close();
32
```

```
33  /* ponovimo za standardno napako */
34  ...
```

Listing 3.14: Izvajanje izvorne kode.



Slika 3.8: Videz urejevalnika kode in izhod.

Poglavje 4

Testiranje

Za testiranje našega programa, smo si izmislili preprost problem: s pomočjo igralnega pripomočka mora uporabnik izpisati vrednost spremenljivke. Najprej je potrebno nastaviti našo napravo. Pritisnemo gumb `Refresh` in dobimo seznam igralnih pripomočkov. Za izbrano napravo moramo določiti, kateri gumb bo izvajal določeno operacijo. Ko smo s tem končali, lahko zaženemo naš program. S pritiskom na gumb `Run`, lahko v trenutku začnemo s programiranjem.

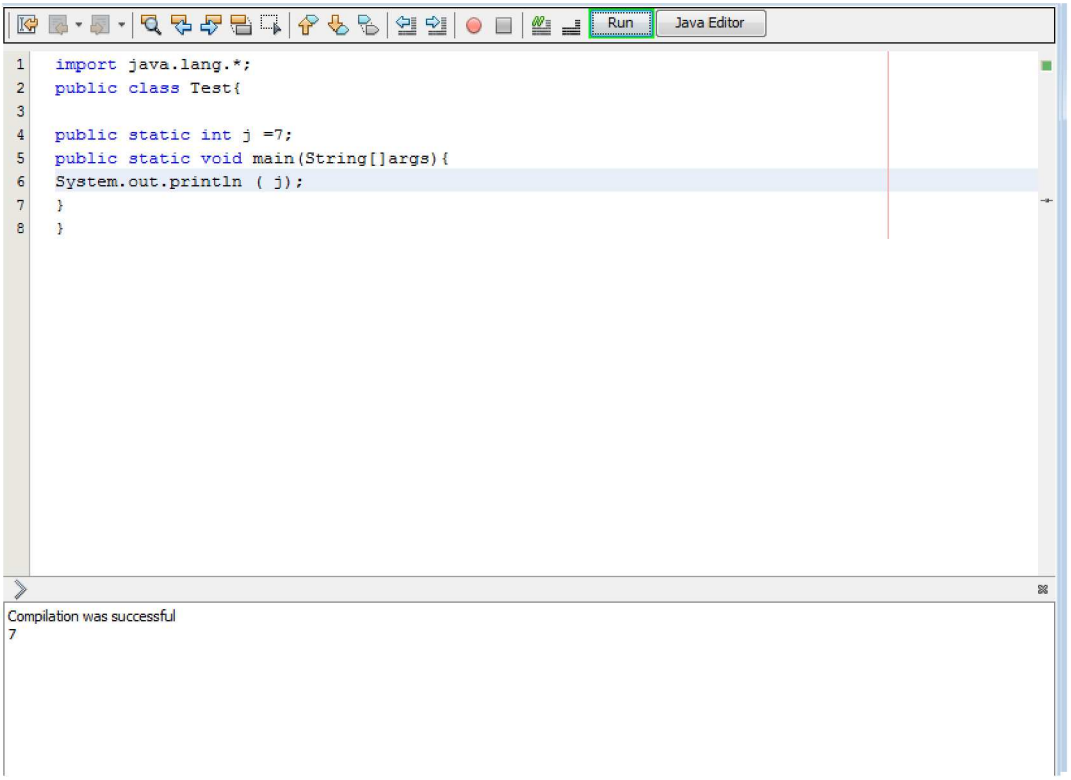
Za izpis vrednosti potrebujemo razred `System`. Lahko ga poiščemo v zavihku `Search` ali `Favorites`. Ob kliku na razred, se bo ta prikazal na urejevalniku izvorne kode. Avtomatsko se bo dodal tudi paket, kateremu pripada naš razred (npr. za razred `Scanner` bi se dodal stavek `import java.util.*;`).

Ko smo dodali naš `System`, se prikažejo vse njegove lastnosti, npr. metode in konstruktorji (s parametri). Za zapis stavka `System.out.println()`, je potrebno najprej klikniti na polje `out` (rezultat lahko vidimo na sliki 4.1), pritisniti gumb za pisanje v drugo polje (v tem primeru gumb z zapisom `Search Field`), v zavihku `Search` izbrati znak piko in na koncu še izbrati ustrezno metodo (z ustreznim parametrom). Ob kliku na metodo se ta izpiše, poleg tega pa se doda nova spremenljivka `public static <tip_spremenljivke>`. Lahko jih je tudi več, odvisno od izbora metode.

Zadnji korak je samo še nastavitev vrednosti spremenljivke in zagon pro-

grama (prikazan na sliki 4.2).

Slika 4.1: Primer zavihka `Class` ob kliku na `out`.



Slika 4.2: Izvajanja našega testnega programa.

Poglavje 5

Zaključek

Glavni namen diplomske naloge je implementacija modula, s pomočjo katerega je možno programirati v Javi. Skozi diplomo nalogo smo opisali orodja, ki so nam pomagali pri njegovem razvoju. Vsakemu smo namenili nekaj besed, kjer smo opisali tiste stvari, ki so se nam zdele pomembne.
Opisali smo tudi tri različne programe za vizualno programiranje. Vsak od njih je bil za nas zanimiv, ker predstavljajo drugačen stil, kot smo ga navajeni. Predvsem Blockly in Scratch sta nas zabavala s svojim preprostim konceptom sestavljanje programa.
V našem programu smo razvili več zavihkov, ki pomagajo in izboljšajo celotno izkušnjo. Vsak zavihek doda neko vrednost programu. Prednost našega koncepta je predvsem v možnosti uporabe igralnega pripomočka, s pomočjo katerega lahko pospešimo programiranje. Potrebno je tudi omeniti, da delovanje ni pogojeno z uporabo igralnega pripomočka. Vsekakor ga je za naše namene bolje uporabiti. Zadane cilje smo osvojili, toda vseeno obstaja možnost izboljšave.
Če bi hoteli omeniti slabosti programa, bi se morali vprašati: kaj lahko še izboljšamo in kaj manjka? Vsekakor bi lahko v programu dodali možnost razvoja večjega projektov, ne samo enega programa. Poleg tega bi lahko preverili možnost razvoja razhroščevalnika. Izboljšali bi lahko tudi vizualno podobo, vendar smo se bolj osredotočili na sam razvoj koncepta pridobivanja

podatkov in pridobivanju dobrih programskih izkušenj. Izvorna koda je na voljo na spletni strani [26].

Literatura

[1] Heiko Böck, *The Definitive Guide to NetBeans Platform 7*, Apress, 2011

[2] Joshua Bloch, *Effective Java, Second Edition*, Addison-Wesley, 2008

[3] *Java* Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Java_(programming_language)

[4] *Java*, Wikibooks. Dostopno na:
http://en.wikibooks.org/wiki/Java_Programming

[5] *Java JIT*, Wikipedia. Dostopno na:
http://en.wikipedia.org/wiki/Just-in-time

[6] *A Brief History of NetBeans*. Dostopno na:
https://NetBeans.org/about/history.html

[7] *Blockly primer*. Dostopno na:
http://bigknol.com/open-blog/2013/07/google-blockly-tutorial/

[8] Cade Metz, *Blockly*. Dostopno na:
http://www.wired.com/2012/06/google-blockly/

[9] *Scratch*. Dostopno na:
http://scratch.mit.edu/
http://en.wikipedia.org/wiki/Scratch_(programming_language)

[10] *Microsoft Visual Programming Language*. Dostopno na:
http://msdn.microsoft.com/en-us/library/bb483088.aspx

[11] *JInput*. Dostopno na:
https://apps.ubuntu.com/cat/applications/precise/libjinput-java/

[12] *JInput namestitev*. Dostopno na:
http://theuzo007.wordpress.com/2012/09/02/
joystick-in-java-with-jinput/

[13] *Program za premik kazalca s pomočjo igralne palice*. Dostopno na:
https://code.google.com/p/joystick-to-mouse-java/

[14] *Rezred java.awt.Robot*. Dostopno na:
http://docs.oracle.com/javase/1.5.0/docs/api/java/awt/Robot.html

[15] *KeyEvent dogodki*. Dostopno na:
http://docs.oracle.com/javase/7/docs/api/java/awt/event/
KeyEvent.html

[16] *Java niti*. Dostopno na:
http://www.tutorialspoint.com/java/java_multithreading.htm

[17] *Primer premika iz izvorne točko v novo točko v Javi*. Dostopno na:
http://stackoverflow.com/questions/9340483/
moving-directly-from-point-a-to-point-b

[18] *Iskanje sistemskih lastnosti s pomočjo Jave*. Dostopno na:
http://docs.oracle.com/javase/7/docs/technotes/tools/
findingclasses.html

[19] *Iskanje vsebine po JAR datotekah*. Dostopno na:
http://www.rgagnon.com/javadetails/java-0665.html

[20] *URLClassLoader razred*. Dostopno na:
http://docs.oracle.com/javase/7/docs/api/java/lang/Class.html

[21] Pankaj Kumar, *Primer uporabe Reflection API*. Dostopno na:
http://www.journaldev.com/1789/java-reflection-tutorial-for-
classes-methods-fields-constructors-annotations-and-much-more

[22] *MouseAdapter dogodki.*. Dostopno na:
http://docs.oracle.com/javase/7/docs/api/java/awt/event/
MouseAdapter.html

[23] *OutputStream razred.*. Dostopno na:
http://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html

[24] Edgar Silva, *JEditorPane označevanje teksta.*. Dostopno na:
https://weblogs.java.net/blog/edgars/archive/2007/01/
using_the_java.html

[25] *Primer izvajanja zunanjih programov*. Dostopno na:
http://www.rgagnon.com/javadetails/java-0014.html

[26] Nejc Banič, *Izvorna koda našega programa*. Dostopno na:
https://github.com/nejcbanic/diplomska_naloga