Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Macro manager v jazyce Java

2020

Gabriel Kucer

Vedoucí práce:
RNDr. Martin Trnečka, Ph.D.

Studijní obor: Aplikovaná informatika,
prezenční forma

Bibliografické údaje

Autor:	Gabriel Kucer
Název práce:	Macro manager v jazyce Java
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2020
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	RNDr. Martin Trnečka, Ph.D.
Počet stran:	34
Přílohy:	1 DVD
Jazyk práce:	slovenský

Bibliograhic info

Author:	Gabriel Kucer
Title:	Macro manager in Java language
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2020
Study field:	Applied Computer Science, full-time form
Supervisor:	RNDr. Martin Trnečka, Ph.D.
Page count:	34
Supplements:	1 DVD
Thesis language:	Slovak

Anotace

Práca predstavuje nástroj – macro manager, ktorý umožňuje vykonavanie serie uživateľom zadaných akcí ako je napríklad vstup z klávesnice alebo myši. Aplikacia umožňuje zadávať príkazy textovou formou, grafickým rozhraním a nahrávaním vstupu uživateľa. Vstup z posledných dvoch metód je automatický konvertovaný na textovu formu, ktoru je možné následne upraviť a prehrať.

Synopsis

This thesis presents a utility – macro manager, which allows execution of series of commands, like input from keyboard or mouse, which were specified by user. Application allows to specify commands in text format, from graphic user interface or from user's input recording. Input from last two methods is automaticly converted to text format, which can be edited and played.

Klíčová slova: nástroj, macro manager; nahravanie vstupu; Java, JavaFX, JNA

Keywords: utility, macro manager, recording of input, Java, JavaFX, JNA

Ďakujem RNDr. Martinovi Trnečkovi, Ph.D. za vedenie mojej práce, poradenstvo a trpezlivosť.

Čestne vyhlasujem, že som celú prácu vrátane príloh vypracoval samostatne a za použitia iba zdrojov spomínaných v texte práce a uvedených v zozname literatúry.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Úvod

1.1 Makro

Makro je v informatike definícia pravidlá, ako bude vstupná postupnosť transformovaná na výstupnú postupnosť (znakov, akcií, výpočtov a podobne). Túto transformáciu označujeme, ako substitúcia alebo expanzia makra.[1]

1.2 Program typu macro manager

Mnohokrát sa pri práci s počítačom stretávame s činnosťami, ktoré sú si podobné poprípade vykonávame jednu rovnakú činnosť niekoľkokrát. Bežným príkladom môže byť úprava textov, kde potrebujeme odstrániť určitý počet znakov z každého n-tého riadku, testovanie aplikácie alebo užívateľského rozhrania aplikácie, kde nie je možné jednoducho a rýchlo vytvoriť automatické testy. Taktiež pri hraní počítačových hier, ktoré od nás vyžadujú opakovať jednu činnosť zas a znovu.

Programy poskytujúce riešenie tohto problému sú často označované ako auto clicker, macro recorder, input recorder a podobne. Úlohou týchto programov je automatizácia repetitívnej činnosti alebo činnosti, ktoré sa riadia určitým vzorom chovania.

1.3 Rozšíriteľnosť

Program tohto typu ma teda veľké spektrum využití. Avšak mnoho z týchto využití sú veľmi špecifické a funkcie z jedného prípadu použitia nemusia byť vôbec potrebné v inom. Teda napísať program, ktorý by pokrýval všetky možnosti využitia by bolo až príliš komplikované. Preto som sa rozhodol program zostaviť tak, aby pokrýval základnú funkcionalitu pre mnohé príklady použitia, ale zároveň aby bol jednoducho rozšíriteľný.

1.3.1 Rozšíriteľnosť vlastným príkazom

Základný spôsob ako priviesť novú špecifickú funkcionalitu do programu, ktorú neobsahuje je vytvorením vlastného príkazu. Vlastný príkaz je napísaný v jazyku Java a môže byť pridaný do programu počas jeho behu. Teda nie je potreba program re-kompilovať a dokonca ani zatvoriť.

Viacej informácií o vytváraní vlastného príkazu je popísaných v sekcií 6.

1.3.2 Rozšíriteľnosť zmenou programu

Môžu existovať prípady kedy je potreba pridať väčšie množstvo funkcionality, alebo pridať funkcionalitu, ktorá ovplyvni chovanie samotného programu a teda nie je možné takto obohatiť program cez vlastný príkaz. Toto rozšírenie sa už bez kompilácie celého programu nezaobíde. A však pri písaní programu som sa snažil

kód napísať takým spôsobom, aby bolo čo najjednoduchšie integrovať do programu novú funkcionalitu bez potreby zmeny celého kódu. Z tohto dôvodu po dokončení tejto práce chcem program spolu so zdrojovým kódom umiestniť na verejný repozitár pod MIT licenciou, ktorá dovoľuje program voľne modifikovať, používať, kopírovať a distribuovať. Týmto spôsobom umožním nielen modifikáciu programu, ale poskytnem aj možnosť zostaviť úplne iný program, ktorý bude používať túto prácu ako svoje jadro prípadne modul, a zároveň táto práca môže ostatným poskytnúť informácie pri vývoji podobnej aplikácie alebo aplikácie, ktorá používa rovnaké technológie.

1.4 Podobné dostupné programy

Existuje niekoľko programov, ktoré sa snažia riešiť problém automatizácie repetitívnych úloh. Mnohé sa medzi sebou líšia vo vlastnostiach ako forma zadávania príkazov, možnosti programu alebo vo schopnosti program rozšíriť, no všetky z nich pokrývajú do istej miery základnú funkcionalitu a to automatický pohyb myšou a vykonanie stlačenia tlačidiel myši.

Nanešťastie mnohé z existujúcich programov trpia veľkým množstvom nedostatkov, ako napríklad možnosť nahrávať vstup len z myši, nefunkčnosť pri viacerých obrazovkách, prípadne veľkých rozlíšeniach. Najčastejšie neposkytujú žiadnu možnosť program rozšíriť.

Pri prieskume som narazil aj na program, ktorý vynikal nad ostatnými. Jedna sa o program s názvom Macro Recorder od spoločnosti JitBit. Tento program poskytuje nie len príjemné užívateľské rozhranie, ale aj mnoho funkcií a čiastočnú možnosť rozšíriť program pomocou jazyka C#. Jedna sa o platený program, ktorý až v prémiovej verzií poskytuje tieto možnosti. Táto verzia programu v roku 2020 spolu s aktualizáciami programu na jeden rok stála $99. Po uplynutí tohto obdobia od zakúpenia užívateľovi nie sú viac k dispozícií aktualizácie. Ak užívateľ chce udržiavať program v aktuálnej verzií a využívať novú funkcionalitu musí platiť ročný poplatok $19,95.[2]

2 Použité technológie

2.1 Java

Java je programovací jazyk vyvinutý v roku 1995 ako hlavná komponenta Java platformy firmou Sun Microsystems, ktorá je v súčastnosti vlastnená firmou Oracle. Jazyk bol navrhovaný, tak aby bol jednoduchý, objektovo orientovaný, class–based a aby jedenkrát zkompilovaný program dokázal bežať na rôznych platformách bez potreby kompilovať program znovu.[3] Teda Java aplikácie sú typický kompilované do bytecode-u, ktorý môže byť spustený na akomkoľvek zariadení na ktorom beží Java virtual machine. Java je dodnes veľmi často používaný jazyk v skutočnosti podľa Tiobe indexu z Augusta 2020 sa jedna o druhý najpopulárnejší jazyk hneď po jazyku C.[4]

2.2 JavaFx

JavaFx je množina grafických a média balíčkov, ktoré dovoľujú navrhovať, vytvárať, testovať, ladiť, nasadzovať aplikácie, ktoré sú schopné fungovať na rôznych platformách. [5]

2.3 Maven

Maven je nástroj pre správu, riadenie a automatizáciu budovania programu. Nástroj môže byť využívaný pre projekty v rôznych jazykoch. Podporovaný je najmä jazyk Java. Maven je založený na koncepte „project object model“ (POM). POM je vlastné XML reprezentácia Maven projektu, ktorá je uložená v súbore pom.xml.[6] V práci je Maven použitý na sprehľadnenie a nastavenie budovacieho systému a na jednoduché integrovanie knižníc, ktoré sú popísane v sekcií 3.

3 Použité knižnice

3.1 Java Native Access

Knižnica Java Native Access (ďalej JNA) poskytuje základnú funkcionalitu na ktorej je vystavaná táto práca. JNA je knižnica vyvíjaná komunitou pod licenciou Apache Software License. Vznikla ako alternatíva na knižnicu Java Native Interface (ďalej JNI). Podobne ako JNI jej úlohou je umožniť použitie Java kódu z Java Virtual Machine, ktorý sprístupni natívne aplikácie systému. Hlavným rozdielom a teda aj motiváciou k vzniku tejto knižnice bol fakt, že používanie knižnice JNI so sebou prináša väčšiu réžiu (napríklad väčšie množstvo opakovaného kódu, potreba kódu na prepojenie rozdielnosti medzi jazykom C a jazykom JAVA a podobne). [8]

V tejto práci je knižnica použitá, ako na nahrávanie vstupu užívateľa, tak aj na vykonávanie makra. Keďže Java aplikácia nedokáže zistiť vstupy z klávesnice a myši, ak je minimalizovaná. Pôvodne som knižnicu nemal v pláne použiť na prehrávanie výstupu, keďže podobnú funkcionalitu poskytuje aj trieda `java.awt.Robot`, ktorá sa sa nachádza priamo v knižniciach jazyka Java. Pri použití triedy `Robot` som narazil najmä na problém so súradnicami, ktoré neodpovedali realite v prípade ak OS Windows má nastavenú vlastnosť „Zväčšenie textu“ na inú hodnotu ako 100 %.

3.2 Richtext

Knižnica poskytuje pamäťovo-efektívnu prácu s textovým poľom. Slúži ako náhrada za komponentu `HTMLEditor`. Narozdiel od spomínanej komponenty táto knižnica poskytuje širokú škálu funkcií vďaka ktorým je možné vytvoriť vlastný textový editor s vlastnosťami, ako číslovanie riadkov, zvýrazňovanie syntaxe, aplikovanie rôznych CSS štýlov na rôzne úseky kódu a podobne. V práci z tejto knižnice je najmä využívania komponenta `CodeArea`. Práve túto komponentu užívateľ používa na vstup textovej formy makra. Cieľom využitia tejto komponenty v práci je užívateľovi poskytnúť komfort pri písaní a čítaní už napísaného makra.

3.3 JFoenix

Keďže súčasný stav knižnice JavaFX bez potreby veľkých zásahov neodpovedá súčasným požiadavkám na moderný vzhľad aplikácie rozhodol som sa využiť knižnicu JFoenix. Knižnica je open–source, pod licenciou Apache 2.0.[7] Jej hlavný zámer je priniesť materiál design od spoločnosti Google do klasických Java aplikácií. V knižnici môžme nájsť mnohé objekty, ktoré dedia klasické JavaFX objekty ako `JFXButton`, ktorý dedí `Button`, `JFXTextField`, `JFXTabPane` a podobne. Obohacuje aplikáciu aj o moderné komponenty, ktoré JavaFX neobsahuje ako napríklad `Drawer` alebo `SnackBar` komponenta, s ktorými sa bežne stretávame v užívateľských rozhraniach mobilných aplikácií.

3.4 Reflections

Knižnica Reflections funguje ako classpath skener. Indexuje takto získané metadata a umožňuje nám ich využívať za behu programu. Medzi príklady použitia patrí napríklad nájdenie všetkých podtypov daného typu, nájdenie typov na základe anotácie a získanie metódy spolu s jej parametrami, návratovým typom a anotácie parametrov. Knižnica je využitá pre nájdenie a inicializáciu všetkých príkazov. Práca by sa zaobišla bez tejto knižnice, avšak rozhodol som sa ju použiť, aby som užívateľom poskytol jednoduchší spôsob rozriešenia programu. Konkrétne užívateľ pre vytvorenie nového príkazu nemusí riešiť žiadnu réžiu, ako inicializácia príkazu, pridanie príkazu do zoznamu dostupných príkazov. Stačí len vytvoriť novú triedu so správnou štruktúrou v balíčku `net.kucer.manager.commands` a ostatné sa postará samotný program.

3.5 Font Awesome

S knižnicou FontAwesome sa môžme najčastejšie stretnúť pri vývoji webových stránok a aplikácií. Existuje aj verzia tejto knižnice pre JavaFX. Knižnica poskytuje mnoho vektorových ikon s ktorými sa bežne stretávame pri navštevovaní internetových stránok. Práve z tohto dôvodu som sa rozhodol použiť tieto ikony v tejto aplikácií. Nie len, že ikony majú jednoduchý a zároveň profesionálny dizajn, ale zároveň užívatelia sú na tieto ikony zvyknutý, čo im uľahčí navigáciu v aplikácií.

3.6 Groovy

Groovy je objektovo orientovaný programovací jazyk, ktorého syntax je kompatibilná so syntaxou jazyku Java. Môže byť použitý ako programovací jazyk, ale aj ako skriptovací jazyk. Integrovanie tejto knižnice do programu poskytuje možnosť načítať a použiť nekompilovanú triedu, čo je dôležité pri rozširovaní programu počas behu o vlastné príkazy. Zároveň pri načítaní príkazov pomocou Groovy, sú taktiež načítané závislosti na knižnice tretích strán. To znamená, že užívateľ môže implementovať vlastné príkazy, ktoré využijú knižnice napríklad na rozpoznávanie obrázkov, komunikáciu so serverom a podobne.

4 Užívateľská dokumentácia

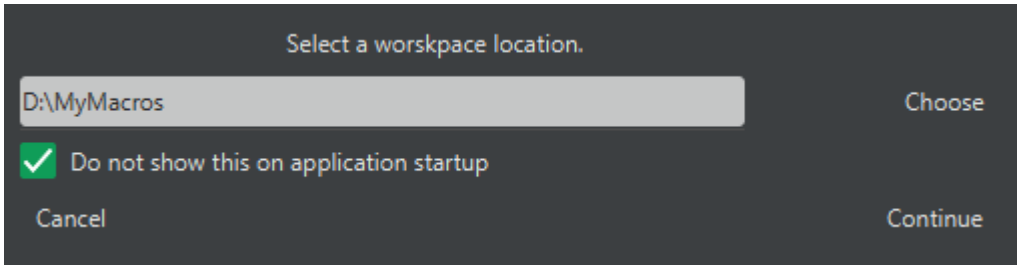
V tejto časti sa budeme zaoberať užívateľským aspektom používania aplikácie od spustenia programu cez vytvorenie makra všetkými spôsobmi, ktoré aplikácia dovoľuje až po spustenie užívateľom vytvoreného makra.

4.1 Workspace a jeho nastavenie

Workspace alebo pracovní priestor je definovaný zložkou. Z tejto zložky sa načítajú všetky súbory, ktoré končia na príponu „.macro“. Tieto súbory sú nasledovne zobrazené v ľavom menu pod záložkou Macros. Pri ukladaní súboru je ako prvá ponúknutá cesta do workspace zložky.

Pri prvom otvorení programu je užívateľovi ponúknutá možnosť vybrať si lokáciu pre workspace. Predvolená lokácia je `%appdata%`, v ktorej je vytvorená zložka MacroManager. V tejto zložke sa nachádzajú uložené nastavenia programu, a teda aj predvolene sa do nej budú ukladať a z nej načítať makra.

Tento dialóg je možné zakázať pri nasledujúcom spustení a to označením políčka „Do not show this on application startup“. Zmenu workspace-u a aj zmenu zobrazovania tohto okna je možné zmeniť aj počas behu samotnej aplikácie a to pomocou menu položky Workspace, ktorá sa nachádza v menu pod položkou Settings.



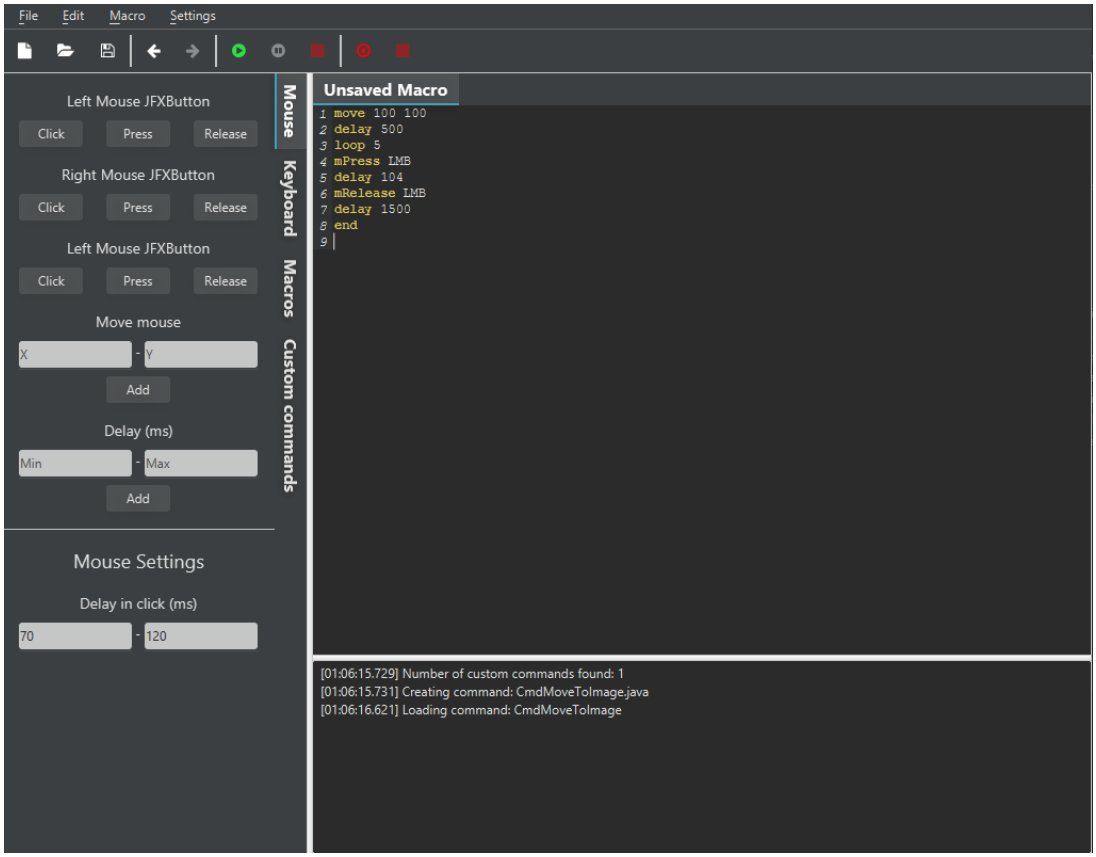
Obrázek 1: Okno s nastavením workspace

4.2 Popis užívateľského rozhrania

Po nastavení workspace je užívateľovi zobrazené hlavné okno aplikácie. Užívateľské rozhranie môžme rozdeliť do piatich kategórií

1. Menu
2. Toolbar
3. Rozhranie pre vkladanie príkazov klikom
4. Textový editor
5. Konzola pre výstup

Medzi komponentami 3 a 4, respektívne 4 a 5 existuje posuvník, ktorý umožňuje zmeniť pomer zobrazenia týchto komponent.



Obrázek 2: Náhľad užívateľského rozhrania

4.2.1 Menu

Menu obsahuje nasledujúce položky:

1. File
   (a) New – Vytvorenie novej záložky
   (b) Open – Otvorenie existujúceho súboru
   (c) Save – Uloženie práve otvorenej záložky
   (d) Exit – Ukončenie programu
2. Edit
   (a) Undo – Vykonanie operácie „späť“, ak je k dispozícií v aktuálnej záložke

(b) Redo – Vykonanie operácie „opakuj“, ak je k dispozícií, v aktuálnej záložke

(c) Copy – Uloženie označeného textu do schránky

(d) Cut – Uloženie označeného textu do schránky a jeho nasledovné odstránenie

(e) Paste – Vloženie textu zo schránky

(f) Delete – Vymazanie označeného textu

3. Macro

(a) Play – Spustenie vykonávania makra z aktuálnej záložky

(b) Pause playing – Pozastavenie prehrávania makra

(c) Stop playing – Zastavenie prehrávania makra

(d) Record – Začiatok nahrávania užívateľského vstupu

(e) Stop recording – Ukončenie nahrávania užívateľského vstupu a jeho prevedenie do textovej formy

4. Settings

(a) Workspace – Nastavenia workspace

4.2.2 Toolbar

Všetky položky, ktoré sa nachádzajú v komponente toolbar sú prístupné aj z menu. Toolbar obsahuje tieto položky znovu z dôvodu rýchlejšieho prístupu a zároveň rýchlym pohľadom na stav zapnutých, alebo vypnutých tlačidiel užívateľ dokáže rýchlo určiť stav aplikácie.

Komponenta je rozdelená na štyri časti ich obsah je nasledovný

1. Správa súborov

(a) Vytvorenie nového súboru

(b) Otvorenie existujúceho súboru

(c) Uloženie súboru

2. Správa histórie textového polia

(a) Akcia späť

(b) Akcia opakovať

3. Správa prehrávania makra

(a) Prehranie makra

(b) Pozastavenie prehrávania makra

(c) Zastavenie prehrávanie makra

4. Správa nahrávania makra

   (a) Nahrávanie makra

   (b) Zastavenie nahrávania makra

Funkcia jednotlivých akcií je totožná s akciami menu popísaných v sekcií 4.2.1.

4.2.3 Rozhranie pre vkladanie príkazov klikom

Rozhranie sa nachádza v ľavej časti obrazovky. Pozostáva zo štyroch záložiek:

1. Mouse
2. Keyboard
3. Macros
4. Custom commands

Záložka Mouse poskytuje možnosť vkladania príkazov pre ovládanie myši. Umožňuje vloženie piatich rôznych operácií

1. Kliknutie, stlačenie a uvoľnenie ľavého tlačidla myši
2. Kliknutie, stlačenie a uvoľnenie pravého tlačidla myši
3. Kliknutie, stlačenie a uvoľnenie stredného tlačidla myši
4. Presunutie kurzoru na špecifikované miesto
5. Pozastavenie vykonávania programu na náhodne číslo z uvedeného intervalu

Pri použití príkazu click je medzi operáciami stlačenia a uvolnenia tlačidla myši vložené pozastavenie vykonávanie príkazov o náhodnej dĺžke z intervalu uvedenom v poslednom nastavení s názvom Delay in click.

Záložka Keyboard umožňuje vkladanie príkazov pre ovládanie klávesnice a obsahuje nasledujúce operácie

1. Získanie názvu klávesy pomocou tlačidla Record a následného stlačenia klávesy
2. Kliknutie, stlačenie a uvoľnenie klávesy
3. Pozastavenie vykonávania programu na náhodne číslo z uvedeného intervalu

Podobne ako záložka Mouse, aj táto záložka obsahuje možnosť nastavenie čakania pri použití funkcie click.

Záložka Macros obsahuje zoznam príkazov, ktoré boli nájdene vo workspace. Pri pridaní alebo odobraní súboru je potrebné tento zoznam obnoviť tlačidlom Refresh. Po vybraní položky zo zoznamu môžme makro použiť pomocou tlačidla Use, otvoriť v novej záložke editoru pomocou tlačidla Open alebo makro vymazať po stlačení tlačidla Delete. Tieto tlačidla sú k dispozícií aj v kontextovom menu, ktoré sa zobrazí po pravom kliknutí na názov položky.

Záložka Custom Commands je podobná ako záložka Macros, ale obsahuje zoznam nájdených príkazov zo zložky Commands. V tomto prípade máme k dispozícií len tlačidlo Refresh a Use. Tlačidlo Use vloží do textového editoru názov makra a uživateľ musí vyplniť jeho argumenty. Tlačidlo Refresh nahrá znovu všetky príkazy zo zložky Commands a aktualizuje ich zoznam. Po dokončení nahrávania je v konzole užívateľovi zobrazená informácia o počte nahraných príkazov a o prípadných chybách.

4.2.4 Textový editor

Textový editor slúži na vkladania príkazov pre makro. Pre uľahčenie čítania a porozumenia makra, kľúčové slová sú farebne zvýraznené. Taktiež na ľavej strane sa nachádza číslovanie riadkov, ktoré pomáha pri ladení nevalidného makra.

4.2.5 Konzola pre výstup

Konzola sa nachádza v dolnej časti okna. Slúži najmä na informovanie užívateľa stave programu a neslúži pre vstup. Najdôležitejšiu rolu má pri písaní makra a vlastného príkazu. V prípade písania makra užívateľ môže zistiť, v ktorej časti makra má chybu a o aký typ chyby sa jedná. V prípade vlastných príkazov je užívateľovi pri zapnutí programu oznámené koľko a akých vlastných príkazov bolo nájdených. V prípade chyby pri načítaní vlastného príkazu je do konzole vložená chybová hlášku spolu so stack trace-om.

4.3 Vytvorenie makra

Makro je možné vytvoriť spôsobmi, ktoré sú popísane v nasledujúcich podkategóriach. Najlepší spôsob pri vytváraní makier je skombinovať všetky tieto tri možnosti. Využitie programu a postup tvorby sa môže veľmi líšiť prípad od prípadu. Vo väčšine použití by tvorba mala začať spustením nahrávania užívateľského vstupu. Nasledovne v textovom editore by mal užívateľ doladiť a optimalizovať príkazy, ktoré mu nemusia vyhovovať – dĺžka čakania, zbytočne kliky a podobne. Ako posledný krok, ak by užívateľ potreboval vložiť ešte ďalšie príkazy, ktoré nebolí vytvorené počas nahrávania je najlepšie využiť grafické rozhranie pre vkladanie príkazov, ktoré sa nachádza v ľavej časti okna.

4.3.1 Zoznám príkazov

Príkazy programu sú case-insensitive. Dôležitá podmienka pre makra je, že na jednom riadku programu sa smie vždy nachádzať len jeden príkaz. Bez tejto podmienky by sa makro stalo veľmi ťažko čitateľným. Taktiež u všetkých základných príkazov platí, že príkaz je tvorený názvom a po názve nasledujú argumenty oddelené jednou medzerou. Táto podmienka nemusí platiť u makrách vytvorených užívateľom.

Príkazy programu môžeme rozdeliť do dvoch kategórií a to príkazy pre ovládanie vstupu a príkazy na kontrolu vyhodnocovania makra.

4.3.1.1 Príkazy pre ovládanie vstupu

Následujúce príkazy slúžia k simulovaniu pohybu myši a stlačenie, respektívne uvolnenia stlačenia klávesy klávesnice a tlačidla myši.

Tabulka 1: Príkazy pre ovládanie vstupu

Názov	**Argumenty**	**Popis**
move	Celé číslo arg1: X pozícia na obrazovke arg2: Y pozícia na obrazovke	Presunie kurzor na pozíciu X a Y
mPress	Výčtový typ – slovo : lmb – pre ľavé tlač. myši mmb – pre stredné tlač. myši rmb - pre pravé tlač. myši	Vykoná stlačenie tlačidla myši. Tlačidlo zostane stlačené kým užívateľ nezavolá mRelease s rovnakým argumentom.
mRelease	Výčtový typ – slovo : lmb – pre ľavé tlač. myši mmb – pre stredné tlač. myši rmb – pre pravé tlač. myši	Vykoná uvolnenia stlačeného tlačidla myši.
kPress	Slovo Názov klávesy	Vykoná stlačenie tlačidla klávesnice. Tlačidlo zostane stlačené kým užívateľ nezavolá kRelease s rovnakým argumentom.
kRelease	Slovo Názov klávesy	Vykoná uvolnenia stlačeného tlačidla klávesnice.
writeText	Text Text ktorý je vložený	Vloží text, ktorý zadaný v argumente.
delay	Celé číslo Dĺžka čakania v ms	Pozastaví vykonávanie makra na čas, ktorý je zadaný v argumente.

4.3.1.2 Príkazy pre ovládanie vyhodnocovania

Následujúce príkazy slúžia k manipulácií procesu vyhodnocovania makra. Makro sa vždy vyhodnocuje smerom z vrchu nadol a vždy po vykonaní príkazu sa okamžite vykoná následujúci príkaz. Toto chovanie môžu ovplyvniť tieto príkazy:

Tabulka 2: Príkazy pre ovládanie vyhodnocovania

Názov	Argumenty	Popis
loop	arg1: počet opakovania	Začiatok loop bloku. Blok sa vykoná toľkokrát, koľko je špecifikované v jeho argumente.
end	-	Ukonči loop blok, ktorý bol otvorený ako posledný.
set	arg1: text, názov premennej arg2: celé číslo, hodnota premennej	Nastaví premenne hodnotu. Ak premenná neexistuje, tak bude vytvorená.
add	arg1: text, názov premennej arg2: celé číslo, hodnota o ktorú je premenná zväčšená	Navýši hodnotu premennej o zadané číslo. Premenná musí existovať.
sub	arg1: text, názov premennej arg2: celé číslo, hodnota o ktorú je premenná zmenšená	Zníži hodnotu premennej o zadané číslo. Premenná musí existovať.
times	arg1: text, názov premennej arg2: celé číslo, hodnota, ktorou je premenná vynásobená	Vynásobí hodnotu premennej zadaným číslo. Premenná musí existovať.
div	arg1: text, názov premennej arg2: celé číslo, hodnota, ktorou je premenná vydelená	Celočíselné videli hodnotu premennej zadaným číslo. Premenná musí existovať.

4.3.2 Použitie textového editoru

Textový editor predstavuje základný spôsob vytvárania makra. V skutočnosti výstupy z ostatných dvoch spôsobov sú zkonvertované práve do tejto formy. V textovom editore je makro popísané ako séria príkazov, kde každý nasledujúci príkaz sa nachádza na novom riadku. Prehľad príkazov sa nachádza v sekcií 4.3.1.

Textový editor sa nachádza v panely so záložkami. Každá záložka obsahuje práve jeden editor. Záložku môžme pridať pomocou tlačidla „New Tab“ alebo stlačením kombinácie kláves Ctrl+N. Každá nová vytvorená záložka je pomenovaná „New Tab“. Po uložení sa názov záložky zhoduje s názvom uloženého súboru.

4.3.3 Použitie grafického rozhrania

Grafické rozhranie má za úlohu šetriť užívateľovi čas pri vytváraní makra a taktiež poskytnúť špeciálnu funkcionalitu, ako je napríklad náhodná hodnota z intervalu pri volaní príkazu delay. Zároveň poskytuje prehľad o základných funkci-

ách, vlastných funkciách a už existujúcich makrách. Pri použití väčšiny tlačidiel z tohto menu je automatický vložený príkaz na označené miesto v textovom editore spolu s nastavenými parametrami. V niektorých prípadoch je vložené aj viacero príkazov napríklad tlačidlo click v záložke Mouse vloží štyri príkazy – stlačenie tlačidla, pozastavenie, uvoľnenie tlačidla a pozastavenie.

4.3.4 Použitie nahrávania vstupu

Spustenie nahrávania makra je možné stlačením tlačidla Start recording alebo stlačením klávesy F8. Po spustení nahrávania bude nahrávaný vstup z klávesnice a myši. Pre zastavenie nahrávania je potreba stlačiť tlačidlo Stop recording alebo klávesu F9. Zastavenie makra stlačením klávesy je vhodnejšie, keďže tento spôsob negeneruje zbytočné príkazy použitia myši vo výsledku nahrávania. Po zastavení nahrávania je otvorená nová záložka, v ktorej sa nachádza nahraný vstup. Je vhodné tento výstup v textovom editore optimalizovať – odstrániť nežiaduce príkazy, zmeniť dĺžku čakania a podobne.

4.4 Spustenie makra

Makro je možne spustiť stlačením tlačidla Play makro. Po spustení prehrávania makra je okno okamžite minimalizované, tak ako pri nahrávaní. V momente spustenia program začne simulovať užívateľom zadefinované chovanie. Užívateľ môže stále využívať myš a klávesnicu, ale jeho vstup bude narušovaný vykonávaním makra.

Vykonávané makro je možné pozastaviť tlačidlom Pause macro. Program najprv dokončí príkaz, ktorý začal vykonávať a následne sa pozastaví. Program zotrvá v tomto stave kým nieje znovu spustený, alebo kým nieje úplne zastavený.

Zastavenie makra je možné vyvolať stlačením tlačidla Stop macro. Po stlačení je súčasne vykonávaný príkaz dokončený a potom je makro zastavené. Po úplnom zastavení narozdiel od pozastavenia nieje možne pokračovať v mieste, kde ku zastaveniu došlo, ale je možné spustiť makro znovu prípadne spustiť iné makro.

V prípade ak by prvý príkaz nebol vykonaný je potrebné pred jeho vykonanie vložiť príkaz pozastavenia – delay. Taktiež ak za sebou nasleduje niekoľko príkazov bez výrazného pozastavenia je možné, že systém ich nestíha spracovávať. V tomto prípade je potreba predĺžiť dĺžku pozastavenia.

5 Programátorská dokumentácia

V nasledujúcich sekciách 5.1 – 5.5 sa budeme zaoberať významom jednotlivých balíčkov a popisom najdôležitejších tried, ktoré obsahujú.

Logiku programu môžme rozdeliť na päť hlavných časti, respektívne balíčkov. Každý z týchto balíčkov plní dôležitú úlohu pri fungovaní programu. Jednotlivé časti medzi sebou komunikujú a sú na seba navzájom závisle.

5.1 Balíček core

Už názov balíčku napovedá, že sa jedna o hlavný a teda najdôležitejší balíček. Tento balíček spája všetky ostatné balíčky a teda vytvára funkčný celok.

5.1.1 Trieda Macro

Zdrojový kód 1: Náhľad triedy Macro

```
public Macro(String content){...}

@Override
public String toString() {
   return getContent();
}

public boolean isValid() {...}

void setValid(boolean valid) {...}

public String getContent() {...}

public void setContent(String content) {...}

public List<Command> getCommands() {...}

void addCommand(Command cmd){...}

void clearCommands(Command cmd){...}
```

Úlohou triedy je reprezentácia makra. Konštruktor makra akceptuje len jeden parameter, ktorý reprezentuje makro v textovej forme. Túto textovú formu môže neskôr zmeniť akákoľvek iná trieda. Až pri validácií makra dôjde k transformácií textovej podoby makra na zoznam príkazov. O túto transformáciu sa stará trieda `Validator`. Pri vytvorení makra sa makro považuje za nevalidné. Nevalidné makro neposkytne zoznam príkazov a teda nie je ho možné spustiť. Funkcia, ktorá nastavuje validitu makra a funkcia pre manipuláciu zoznamu transformovaných príkazov zo zrejmých bezpečnostných dôvodov je prístupná, len triedam

v rovnakom balíčku. V balíčku sa nachádza len jedna trieda a to `Validator`. Makro sa stáva nevalidným ak dôjde k manipulácií s textovou formou makra alebo so zoznamom príkazov.

5.1.2 Trieda Validator

Trieda `Validator` obsahuje len statické metódy. Trieda nemá žiaden vnútorný stav a poskytuje len jednu verejnú metódu a to `validate(Macro macro)`. Proces validácie je následovný:

1. Ak je makro validné skonči.
2. Vytvor zoradené pole s názvom lines, kde každá položka obsahuje jeden riadok z makra.
3. Pre každý riadok vykonaj 4–9.
4. Ak je riadok prázdny pokračuj na ďalší.
5. Zober prvé slovo z riadku a nájdi príkaz s týmto názvom. Ak príkaz nebol najedený, skonči.
6. Zistí či argumenty príkazu sa nachádzajú v zozname premenných, ak áno a zároveň príkaz nemá nastavený príznak, že sa tento argument nevyhodnocuje, vyhodnoť premennú.
7. Porovnaj syntax tohto príkazu voči obsahu riadku s vyhodnoteným obsahom, ak syntax nieje zhodná skonči.
8. Ak makro vytvára premennú, tak si ju ulož do zoznamu premenných.
9. Pridaj príkaz do zoznamu príkazov v danom makre.
10. Označ makro za validné a skonči.

5.1.3 Trieda Settings

Podobne ako trieda `Validator`, táto trieda obsahuje len statické funkcie. Práve cez túto triedu je možné získať a nastaviť hodnoty ovplyvňujúce chod programu, ako sú: cesta na zložku obsahujúcu workspace, cesta na zložku obsahujúce knižnice a podobne. Pri spustení programu je táto trieda inicializovaná podľa properties súboru, ktorý sa nachádza v `%appadata%/MacroManager`. Ak súbor neexistuje trieda použije prednastavené hodnoty a súbor uloží.

Pri akejkoľvek zmene nastavení je súbor `settings.properties` aktualizovaný.

5.1.4 Trieda Executor

Úlohou triedy je vykonávanie makra. Trieda beží vo vlastnom vlákne. Najdôležitejšie vlastnosti a metódy tejto triedy sú následovné:

Zdrojový kód 2: Náhľad triedy Executor

```
...
    public List<Command> commandList = new ArrayList<>();
    public int currentIndex = 0;
    public final HashMap<String, String> variables =
        new HashMap<>();

    ...

    @Override
    public synchronized void start() { ... }

    @Override
    public void run() { ... }

    public void stopExecution() { ... }

    public void pauseExecution() {
        isPaused = true;
    }

    public void resumeExecution() {
        isPaused = false;
    }

    public boolean isPaused() {
        return isPaused;
    }
...
```

Premenná `commandList` obsahuje zoznám všetkých príkazov, ktoré sa majú vykonať. Jedna sa o zoznam a nie zásobník z dôvodu, aby užívateľom vytvorené makra mohli meniť proces vykonávania – pridať príkazy, odobrať, zopakovať niektoré príkazy a podobne.

Premenná `currentIndex` označuje index príkazu v zozname `commandList`, ktorý sa práve vykonáva .

Mapa `variables` obsahuje premenné a ich hodnoty. Mapa je verejne dostupná z rovnakého dôvodu ako `commandList`, teda aby užívateľ s ňou mal možnosť manipulovať.

Funkcia `start()` spustí vlákno. Pomocou funkcie `pauseExecution`, je možné

pozastaviť makro a pomocou funkcie `resumeExecution` je ho možné znovu spustiť. Funkcia `stopExecution` natrvalo zastaví vykonávanie makra.

Samotné vykonávanie prebieha vo funkcií `run()`. Kde vždy v zozname príkazov je nájdený príkaz na základe premennej `currentIndex` a je vykonaný. Následne hodnota premennej `currentIndex` je navýšena o 1. Ak je hodnota indexu väčšia než je množstvo prvkov liste príkazov, zmenšene o hodnotu 1, vykonávanie skončí.

5.2 Balíček ui

Balíček obsahuje triedy pre prácu s užívateľským rozhraním. Jedná sa najmä o kontrolórov užívateľského rozhrania a o implementáciu vlastných buniek zoznamu pre zoznam makier a zoznám príkazov.

Keďže celá aplikácia pozostáva z dvoch vlastných okien nachádzajú sa tu práve dva ovládače a to `ApplicationController` a `WorkspaceController`. `WorkspaceController` ovláda okno nastavovania workspace.
`ApplicationController` spracováva vstupy z hlavného okna. Až na pár výnimiek tieto triedy nevytvárajú žiaden vzhľad aplikácie, ale riešia len volanie funkcií pri udalostiach, ako je napríklad stlačenie tlačidla. Samotný vzhľad aplikácie je implementovaný pomocou `FXML` a to v súboroch `Workspace.fxml` a `Application.fxml`. Na štýlovanie aplikácie sú využité kaskádové štýly umiestnené v súbore `MainTheme.css`.

5.3 Balíček jna

Knižnica JNA, ktorá je popísaná v sekcií 3.1 je implementovaná práve v tomto balíčku. V balíčku sa nachádzajú rôzne triedy, ktoré združujú hodnoty získane volaním tejto knižnice. Najväčšiu funkcionalitu v sebe skrýva práve trieda `JnaRecordThread`, ktorá sa stará o registráciu hook-ov do systému a spracovania ich výsledkov.

5.3.1 Trieda JnaRecordThread

Jedna sa o triedu ktorá poskytuje možnosť zaznamenať vstup užívateľa. Trieda, presnejšie jej rodič, dedí triedu `Thread`. Vykonávanie nahrávania prebieha vo svojom vlastnom vlákne. Pri inicializácií triedy sú vytvorené Hook-y, ktoré využívajú triedy `MouseHookProc` a `KeyboadHookProc`. Obe tieto triedy obsahujú funkciu `callback`, ktorá je zavolaná systémom, ak dôjde k užívateľskému vstupu. Túto správu zaznamenám spolu s priloženým časom v milisekundách. Po dokončení nahrávania je tento zoznam poskytnutý na spracovanie a zobrazenie v užívateľskom rozhraní. Priložený čas sa využíva na výpočet príkazu čakania.

Vytvorenie hook-u pomocou knižnice JNA pozostáva z nasledujúcich časti:

1. Vytvorenie triedy, ktorá implementuje rozhranie `WinUser.HOOKPROC`, ktorú pomenujeme napríklad `HookProc`.

2. Vytvorenie triedy `MyHookStructure`, ktorá dedí triedu `Structure`.

3. Vytvorenie funkcie public `WinDef.LRESULT callback(int nCode, WinDef.WPARAM wParam, MyHookStructure hookStruct)`, kde `nCode` je kód na základe ktorého sa hook procedúra rozhodne ako spracovať správu. `wParam` je identifikátor správy napríklad u hook-u myši obsahuje informácie, ktoré tlačidlo bolo stlačené, respektívne uvolnené. Posledný argument `hookStruct` obsahuje vyplnenú nami poskytnutú štruktúru, ktorá obsahuje informácie, ako napríklad bod, kde bola akcia myši vykonaná.

4. Posledný krok je zavolanie funkcie `User32.INSTANCE.SetWindowsHookEx` s našou triedou `HookProc`. Táto funkcia zaregistruje hook do systému a vráti jeho odkaz.

Po dokončení činnosti je dôležité zaregistrované hook-y odobrať zo systému a to pomocou funkcie `User32.INSTANCE.UnhookWindowsHookEx(hook)`, kde hook predstavuje ukazovateľ na zaregistrovaný hook ktorý sme získali v kroku 4.

Viacej informácií je možne získať v oficiálnej dokumentácií Windows API, ktorú som pri tejto implementácií využíval, keďže knižnica JNA dodržuje rovnaké názvy funkcií a parametrov, ako sú práve popísane v tejto dokumentácií.

5.4 Balíček commands

Tento balíček obsahuje všetky základné príkazy. Taktiež obsahuje abstraktnú triedu `Command`, ktorá ma dôležitú rolu pri vytváraní vlastného alebo základného príkazu. Tuto triedu musia dediť všetky príkazy, či užívateľom vytvorene alebo už existujúce ktoré sú uložene v tomto balíčku.

5.5 Balíček logger

Tento balíček poskytuje funkcionalitu pre zaznamenávanie činností užívateľa a výstup do komponenty `Console`. Pre použitia logovania je potrebné vytvoriť instanciu triedy `Logger`.

5.5.1 Popis triedy Logger

Konštruktor triedy obsahuje jeden parameter typu String, ktorý špecifikuje názov triedy spolú s balíčkom, ktorá bude túto triedu využívať.

Premenná `logAllClasses` typu boolean rozhoduje či sa budú logovať všetky triedy alebo len triedy zo zoznamu `classList`.

Logovanie ma štyri úrovne:

1. ERROR

2. WARN

3. INFO

4. DEBUG

Úrovne sa využívajú pri výpise správ, kde sú vypísané len tie správy, ktoré majú rovnakú alebo menšiu úroveň. Výpis prav prebieha pomocou funkcií

- `error(String error)`
- `warn(String warn)`
- `info(String info)`
- `debug(String debug)`

Ak užívateľ chce využiť túto triedu, je ju vhodné vytvoriť na statickej úrovni.

6 Vytvorenie vlastného príkazu

Nasledujúce sekcie popisujú štruktúru a spôsob ktorým sa vlastné príkazy vytvárajú.

6.1 Štruktúra vlastného príkazu

Nový príkaz je reprezentovaný novou triedou v jazyku Java. Každý názov tejto novej triedy musí začínať prefixom `Cmd`. Táto trieda musí dediť abstraktnú triedu `manager.commands.Command`. V novovytvorenej triede je potreba vytvoriť konštruktor bez parametrov, v ktorom budú nastavené následujúce hodnoty:

- `name` – typ premennej je `String`. Špecifikuje názov makra a je povinný.
- `pattern` – typ premennej je `String`. Špecifikuje syntax príkazu. Jedna sa vlastne o regulárny výraz. Vždy musí začínať názvom príkazu. Pre rýchlejší a prehľadnejší zápis odporúčam používať statické premenné a funkcie z triedy `Command`. Nastavenie tejto premennej je povinné.
- `dontEvalArgList` – typ premennej je `List<String>`. Pôvodne je tento list prázdny. Obsahuje indexy argumentov, ktoré sa nevyhodnotia. To znamená, ak argument je reťazec a existuje premenná s rovnakým názvom. Za hodnotu argumentu sa bude považovať tento názov a nie hodnota premennej. Číslovanie argumentov je od čísla 0.

6.2 Použitie knižníc tretích strán

Ak chceme vo vlastnom príkaze použiť iné knižnice než základné je potreba tieto knižnice, ako súbory s príponou `.jar`, vložiť do zložky „lib“, ktorá sa nachádza v zložke „Commands“ v používanom workspace. Následne knižnice stačí bežným spôsobom naimportovať v našom vlastnom príkaze.

6.3 Ukážka postupu na vlastnom príkaze findImage

V tejto sekcií si ukážeme konkrétny postupu ako vytvoriť vlastný príkaz. Príkaz sa bude volať `moveToImage` a bude mať jeden argument, cestu k obrázku. Príkaz vytvorí snímok obrazovky, ak v ňom nájde obrázok, ktorý bol poskytnutý v argumente tak presunie kurzor myši na stred nájdeného obrázku. Inak vypíše do konzoly hlášku o neúspešnom vykonaní príkazu.

Na implementáciu tohto príkazu využijeme dve knižnice: `MarvinFramework` a `MarvinPlugins`. Tieto knižnice sa zaoberajú spracovávaním obrazu. Knižnice vložíme do `Workspace/Commands/lib`. Následne vytvoríme v zložke `Workspace/Commands` súbor `CmdMoveToImage.java`. V súbore najprv naimportujeme všetky závislosti, ktoré budeme potrebovať.

```
// Java imports
import java.awt.*;
import java.awt.image.BufferedImage;

// MacroManager imports
import manager.commands.Command;
import manager.commands.base.CmdMove;
import manager.core.CommandManager;
import manager.core.macro.Executor;
import manager.logger.Logger;

// Imports from lib folder
import marvin.image.MarvinImage;
import marvin.image.MarvinSegment;
import marvin.io.MarvinImageIO;
import marvinplugins.MarvinPluginCollection;
```

Následne vytvoríme základnú štruktúru príkazu.

Zdrojový kód 3: Základná štruktúra príkazu

```
...
class CmdMoveToImage extends Command {

   public CmdMoveToImage() {
         // Name of command
         name = "moveToImage";

         // Syntax of command
         pattern = commandWord(name) + SPACE
               + "(" + ".+" + ")";
      }

   @Override
   public void execute(Executor executor) {
      super.execute(executor);
   }
}
```

Taktiež si vytvoríme jednoduchú funkciu, ktorá nám vráti prvý parameter ako typ `String`. Tiež vytvoríme a inicializujeme `logger`.

```
    public static final Logger logger =
        new Logger(CmdMoveToImage.class.getName());

    // Function to retrieve first parameter as string
    public String getText(Executor executor) {
          return String.valueOf(getArg(executor, 0));
    }
```

V tomto momente je príkaz validný a spustiteľný, no nič nevykoná, keďže jeho execute funkcia je prázdna. Ostáva nám len doplniť logiku hľadania obrázku v obrázku. Pre uľahčenie práce využijeme vyššie spomenutú knižnicu.

Zdrojový kód 4: Funkcia na nájdenie obrázku

```
...
@Override
public void execute(Executor executor) {
    super.execute(executor);

    BufferedImage screenshot = null;
    // Create screenshot
    try {
        screenshot = new Robot().createScreenCapture(
            new Rectangle(
                Toolkit.getDefaultToolkit().getScreenSize()
            ));
    } catch (AWTException e) {
        logger.info("Cannot create screenshot.");
        return;
    }

    if (screenshot == null) {
        logger.info("Cannot create screenshot.");
        return;
    }

    // Create Marvin Image from screenshot
    MarvinImage source = new MarvinImage();
    source.setBufferedImage(screenshot);

    MarvinImage findImage = null;
    String imagePath = getText(executor);
    try {
        // Load image from argument
```

```
      findImage = MarvinImageIO.loadImage(imagePath);
    }catch(Exception e){
       logger.info("Image on path " + imagePath
          + " has not been found.");
       return; // Image location has not been found - end
    }

    if(findImage == null)
       return;

    // Find subimage
    MarvinSegment result = MarvinPluginCollection
      .findSubimage(findImage, source, 0, 0);

    // If no subimage location has been found - end
    if(result == null) {
       logger.info("Image " + imagePath
          + " has not been found on screen.");
       return;
    }

    // Extract location from result to Array
    int x = (result.x1 + result.x2) / 2;
    int y = (result.y1 + result.y2) / 2;
    Integer[] args = new Integer[] { x, y };

    // Create move command
    CmdMove cmdMove = new CmdMove();
    // Set parameters for move command
    cmdMove.setUnresolvedArgs(Arrays.asList(args));
    // Execute
    cmdMove.execute(executor);
}
...
```

Po implementovaní logiky nájdenia obrázku stačí obnoviť zoznam vlastných príkazov a príkaz môžme začať využívať.

Závěr

Výsledok práce predstavuje nástroj na vytváranie, upravovanie, ukladanie a prehrávanie užívateľom vytvorených makier. Vytvorenie makra je možné vykonať cez nahrávanie vstupu užívateľa, užívateľské rozhranie a pomocou textovej formy, tak ako bolo určené v zadaní práce. Aplikácia bola vytvorená v programovacom jazyku Java pre operačný systém MS Windows. Aplikáciu je možne rozšíriť o vlastné príkazy a tým si ju prispôsobiť k vlastným účelom.

Conclusions

The result of this thesis presents utility for creation, editation, saving and playing macros created by users. Creation of macro can be done through recording of user's input, user interface and with usage of text form, as it was specified in assignment of this work. Application has been created in programming language Java for operating system MS Windows. Application can be enriched with custom commands and thus customized for user's specific usage.

A Obsah priloženého DVD

`bin/`
: Súbory potrebné k spusteniu programu.

`doc/`
: Text práce vo formáte PDF spolu so všetkými prílohami, a všetky súbory potrebné pre bezproblémové vygenerovanie PDF dokumentu textu.

`workspace/`
: Príklad zložky workspace, s ukážkou makier a vlastného príkazu.

`readme.txt`
: Návod na spustenie aplikácie.

Literatura

[1] WIKIPEDIA (ed.). *Makro (software)* [online] [cit. 2020-08-10]. Dostupné z: https://cs.wikipedia.org/wiki/Makro_(software)

[2] JITBIT (ed.). *Macro Recorder pricing* [online] [cit. 2020-08-03]. Dostupné z: https://www.jitbit.com/macro-recorder/purchase/

[3] ORACLE (ed.). *What is Java technology and why do I need it ?* [online] [cit. 2020-08-08]. Dostupné z: https://java.com/en/download/faq/whatis_java.xml

[4] TIOBE (ed.). *TIOBE Index for August 2020* [online] [cit. 2020-08-08]. Dostupné z: https://www.tiobe.com/tiobe-index/

[5] PAWLAN, Monica. *What Is JavaFX?* [online] [cit. 2020-08-08]. Dostupné z: https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm

[6] MAVEN APACHE (ed.). *Apache Maven Project* [online] [cit. 2020-08-03]. Dostupné z: https://maven.apache.org/

[7] JFOENIX (ed.). *JFoeniX official website* [online] [cit. 2020-08-03]. Dostupné z: http://www.jfoenix.com/

[8] WIKIPEDIA (ed.). *Java Native Access (JNA)* [online] [cit. 2020-08-12]. Dostupné z: https://en.wikipedia.org/wiki/Java_Native_Access