edice
začínáme s ...

PROGRAMOVÁNÍ V JAZYKU JAVA

Učebnice pro budoucí profesionální programátory

- Přehled základních jazykových konstrukcí a principů OOP
- Návrh objektové architektury pro škálovatelné aplikace
- Tvorba základního rámce projektu a jeho API pro efektivní práci týmu
- Zefektivnění vývoje pomocí testovacích scénářů

soubory ke stažení na www.grada.cz

GRADA®

RUDOLF PECINOVSKÝ

edice
začínáme s ...

Programování v jazyku JAVA

Učebnice pro budoucí profesionální programátory

RUDOLF PECINOVSKÝ

GRADA Publishing

Rudolf Pecinovský

Programování v jazyku Java
Učebnice pro budoucí profesionální programátory

Vydala Grada Publishing, a.s.
U Průhonu 22, Praha 7
obchod@grada.cz, www.grada.cz
tel.: +420 234 264 401
jako svou 10132. publikaci

Odpovědný redaktor Petr Somogyi
Fotografie na obálce Depositphotos/KostyaKlimenko
Grafická úprava a sazba Rudolf Pecinovský
Počet stran 576
První vydání, Praha 2025
Vytiskly Tiskárny Havlíčkův Brod, a. s.

ISBN 978-80-271-8054-7 (ePub)
ISBN 978-80-271-8053-0 (pdf)
ISBN 978-80-271-5751-8 (print)

Věnováno mé ženě Jarušce

Stručný obsah

Podrobný obsah

Výpis 38.4: *Definice interfejsu `IUI`*

```
1  package game77.ck1l_user_io;
2
3  /* Interfejs {@code IUI} definuje požadované rozhraní objektů,
4   * jejichž prostřednictvím bude aplikace kounikovat s uživatelem. */
5  public interface IUI
6  {
7      /* Zobrazí uživateli odpověď hry a požádá jej o zadání dalšího příkazu. */
8      String getCommand(String answer);
9
10     /* Zobrazí zadaný string -- většinou odpověď hry. */
11     void showAnswer(String answer);
12
13     /* Zeptá se uživatele a očekává odpověď ANO/NE, nebo její ekvivalent. */
14     boolean askQuestion(String question);
15 }
```

38.4.2 Definice třídy `Main2`

Abyste mohli porovnávat změny, nebudu třídu `Main` dále upravovat, ale vytvořím její kopii nazvanou `Main2` a tu upravím tak, aby umožňovala metodám `run` a `multirun` komunikovat s uživatelem prostřednictvím zvoleného komunikačního objektu a aby bylo možné zadat, zda se součástí odpovědi stane i informace o aktuálním stavu hry.

Začneme úpravou metody `main(String[])`. V pozdějších dobách ji můžeme upravit tak, že potřebná nastavení bude možné nastavit jako argumenty příkazového řádku, jak jsme si vysvětlovali v pasáži *27.3.1 Argumenty příkazového řádku* na straně 438 a následně ukazovali ve výpisu 27.3.

V metodě definujeme následující pomocné proměnné:

- Proměnná **`ui`** bude uchovávat odkaz na objekt, jehož prostřednictvím bude aplikace komunikovat s uživatelem.
- Proměnná **`game`** bude uchovávat odkaz na hru, která se hraje.
- Hodnota logické proměnné **`state`** bude určovat, zda se má odpověď hry doplňovat o informace o aktuálním stavu hry.
- Proměnná **`details`** bude uchovávat odkaz na kód, který bude doplňovat odpověď hry o dodatečné informace o jejím stavu. Protože se tyto informace po zpracování každého příkazu mění, potřebujeme něco, co je bude na požádání operativně zjišťovat.

Zdrojový kód takto upravené metody si můžete prohlédnout ve výpisu 38.5.

V metodě by vás mohla zaujmout inicializace proměnné `details`. V závislosti na hodnotě proměnné `state` se do ní bude ukládat buď odkaz na lambda-výraz spouštějící metodu `currentState(IGame)`, anebo odkaz na lambda-výraz vracející prázdný string.

Jak vidíte na řádku 7, metodě `multirun(…)` od minula přibyly dva parametry: první obsahuje odkaz na kód, který může předat k odpovědi hry další informace, druhý obsahuje odkaz na komunikační objekt, jehož prostřednictvím bude aplikace komunikovat s uživatelem.

Výpis 38.5: Upravená definice metody `main(String[])` ve třídě `Main2`

```
1  public static void main(String[] args) {
2      IUI        ui = Console.get();
3      IGame    game = new game77.ck1k_whole.Portal().game();
4      boolean state = true;   //Budeme-li chtít vypisování aktuálního stavu
5      Supplier<String> details = (state ? () -> currentState(game)
6                                        : () -> "");
7      multirun(game, details, ui);
8  }
```

38.4.3 Úprava metod `run(…)` a `multirun(…)`

Pojďme se tedy podívat, jak bychom měli po těchto změnách upravit definice metod `run(…)` a `multirun(…)`, kterým na jednu stranu přibydou parametry, ale na druhou stranu se v nich usnadní rozhodování.

Jak se můžete přesvědčit ve výpisu 38.6, definice obou metod se zavedením komunikačního objektu opravdu zjednoduší. Zájemci je mohou porovnat s definicemi ve výpisech 38.1 na straně 556 a 38.2 na straně 557.

Výpis 38.6: Upravené definice metod `run(…)` a `multirun(…)` ve třídě `Main2`

```
1  private static void run(IGame game, Supplier<String> details, IUI ui) {
2      String  command = "";
3      for(;;) {
4          String answer = game.executeCommand(command);
5          if (! game.isActive()) {
6              ui.showAnswer(answer);  //Už bez přidaných informací
7              return;
8          }
9          command =  ui.getCommand(answer + details.get());
10     }
11 }
12
13 public static void multirun(IGame game, Supplier<String> details, IUI ui) {
14     for(;;) {
15         run(game, details, ui);      //Spuštění jednoho běhu hry
16         boolean answer = ui.askQuestion("Chcete si zahrát ještě jednou?");
17         if (! answer) {
18             ui.showAnswer("Ještě jednou děkuji za hru.\nNa shledanou.\n");
19             return;
20         }
21         ui.showAnswer("Dobře, zahrajeme si ještě jednou.");
22     }
23 }
```

V definici metody `run(…)` bych chtěl upozornit na použití lambda-výrazu `details` na řádku 9, kde se k odpovědi hry přičte string, jenž je návratovou hodnotou lambda-výrazu definovaného v metodě `main(String[])` a obsahuje případné informace o aktuálním stavu hry, které mohou sloužit jako částečná nápověda.

Příkaz na řádku 6 vypisující odpovědi hry už po závěrečném příkazu tyto informace o aktuálním stavu hry k odpovědi hry nepřidává, protože předpokládáme, že po skončení hry je uživatel již nepotřebuje.

Výpis 38.7: *Definice třídy `Console` v modulu `interfaces`*

```
1  package game77.ck1l_user_io;
2
3  import java.util.Scanner;
4
5  public   class Console
6      implements IUI
7  {
8      private static final Scanner SCANNER = new Scanner(System.in);
9
10     private static final Console CONSOLE = new Console();
11     public static Console get() { return CONSOLE; }
12     private Console() {}
13
14     @Override
15     public String getCommand(String answer) {
16         System.out.println(answer);
17         System.out.print("==============================\nZadáte: ");
18         String command =  SCANNER.nextLine();
19         System.out.print("------------------------------\n");
20         return command;
21     }
22
23     public void showAnswer(String answer) {
24         System.out.println(answer + "==============================\n");
25     }
26
27     public boolean askQuestion(String question) {
28         char answer;
29         for(;;) {
30             System.out.println("==============================\n"
31                                + "Chcete si zahrát ještě jednou (A/N): ");
32             String line = SCANNER.nextLine().trim();
33             if (!  line.isEmpty()
34                 && "01ANan".indexOf(line.charAt(0)) != -1) {
35                 answer = line.charAt(0);
36                 break;                      //Výskok z cyklu ---------->
37             }
38             System.out.println("------------------------------\n"
39                 + "Odpověď musí začínat některým ze znaků \"01ANan\"\n"
40                 + "Zkuste odpovědět znovu.");
41         }
42         return ("1Aa".indexOf(answer) != -1);
43     }
44 }
```

38.4.4 Definice třídy `Console`

Abychom mohli vše rozběhnout, potřebujeme alespoň jeden komunikační objekt. Definujeme třídu `Console`, která bude implementovat interfejs `IUI` a její instance bude zprostředkovávat komunikaci s uživatelem prostřednictvím standardního vstupu a výstupu.

Vzhledem k tomu, že budeme potřebovat pouze jedinou instanci této třídy, definujeme ji podle návrhového vzoru *Jedináček – Singleton*. Její definici si můžete prohlédnout ve výpisu 38.7. Doufám, že vám nebude vadit, že jsem v ní srazil k sobě definici atributu s jedináčkem, tovární metody a soukromého konstruktoru.

38.5 Vytvoření primitivního GUI

Aplikace je tedy připravena na výběr z několika verzí uživatelského rozhraní. Pojďme se nyní podívat na to, jaké nástroje nám *Java* nabízí pro návrh alespoň primitivního GUI.

Java definuje ve standardní knihovně balíček `javax.swing` obsahující sadu modulů definujících základní nástroje pro vytváření GUI. Protože se jedná o relativně ucelenou sadu, bývá často označován jako knihovna nebo framework. Tuto knihovnu nyní použijeme.

38.5.1 Modalita dialogových oken

Než se ale pustíme do návrhu našeho GUI, měli bychom si nejprve ujasnit pojem modalita. Dialogová okna, jejichž prostřednictvím uživatel komunikuje s aplikací, můžeme rozdělit do dvou skupin:

- **Modální okna** zastaví jakoukoliv další komunikaci s aplikací do doby, než je uživatel řádně ukončí. Modální bývají např. okna pro otevření či uložení souboru. Dokud nezadáte, jaký soubor chcete otevřít, resp. uložit, aplikace se s vámi „nebaví“.

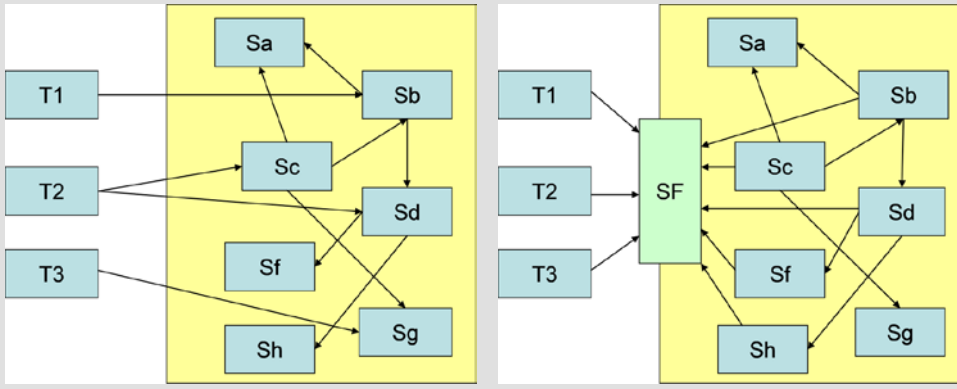
  To, že se zastaví komunikace s uživatelem, ale neznamená, že se zastaví běh celé aplikace. Ta může dále běžet včetně případného zobrazování průběžných výsledků. Můžete např. spustit nějakou animaci a otevřít dialogové okno s výzvou: „*Až se pokocháš, stiskni OK!*“ Uživatel si může například prohlížet běžící animaci a stiskem OK pak celou aplikaci zavřít.
- **Nemodální okna** sice vyskočí, ale aplikace s uživatelem dále komunikuje. Nemodální bývají v řadě aplikací okna pro vyhledávání a nahrazování textu. Poté, co zadáte příkaz, který okno otevře, můžete odskočit do aplikace, tam např. vybrat do schránky text, který chcete vyhledat, vrátit se do okna a tam jej zadat. Nemodální okno může zůstat otevřené a vy můžete dále s aplikací pracovat.

38.5.2 Třída javax.swing.JOptionPane

Knihovna swing představuje poměrně rozsáhlý komplex vzájemně provázaných tříd, který umožňuje vytvářet komplexní grafické uživatelské rozhraní. Uživatelé však v řadě případů nepotřebují využívat veškeré její schopnosti a stačí jim vytvářet jednoduchá dialogová okna. Pro tento účel je v ní definována třída JOptionPane, kterou bychom mohli označit za implementaci návrhového vzoru *Fasáda* (viz podšeděný blok).

Návrhový vzor Fasáda

Návrhový vzor *Fasáda* použijeme ve chvíli, kdy nějaký systém začíná být pro své uživatele příliš složitý vzhledem k oblasti úloh, které chtějí s jeho pomocí řešit. Doporučuje nahradit sadu rozhraní jednotlivých subsystémů sjednoceným rozhraním zastupujícím celý systém. Definuje tak rozhraní vyšší úrovně, které usnadní využívání podsystémů. Cílem je zjednodušit rozhraní celého systému a snížit počet tříd, s nimiž musí uživatel přímo či nepřímo komunikovat.



Používá se např. u grafických knihoven, kdy uživatelům, kteří nepotřebují vytvářet složitě strukturovaná okna, nabídne sadu funkcí vyvolávajících předpřipravená jednoúčelová okna.

Třída JOptionPane definuje 65 metod, z nichž nás budou zajímat následující tři, které otevírají modální okna:

showMessageDialog(Component parentComponent, Object message)
Otevře dialogové okno se zobrazeným textovým podpisem parametru message a tlačítkem **OK**. Parametr parentComponent určuje rámec, v němž se dialog zobrazí, ale může se zde zadat prázdný odkaz null.

int showConfirmDialog(Component parentComponent, Object message)
Otevře dialogové okno se zobrazeným textovým podpisem parametru message a tlačítky **Yes**, **No** a **Cancel**. Parametr parentComponent má stejný význam jako u předchozí

metody. Podle toho, které tlačítko bylo stisknuto, vrací jednu ze čtyř hodnot definovaných jako konstantní atributy třídy `JOptionPane`: `YES_OPTION`, `NO_OPTION` a při zavření okna hodnotu `CANCEL_OPTION`.

`String showInputDialog(Object message)`
Otevře dialogové okno s vypsanou výzvou a vstupním polem pro zadání údajů. Tato metoda má na rozdíl od předchozích přetíženou verzi, která nepožaduje zadání rámce, což obvykle znamená, že je okno vycentrováno na primární obrazovce.

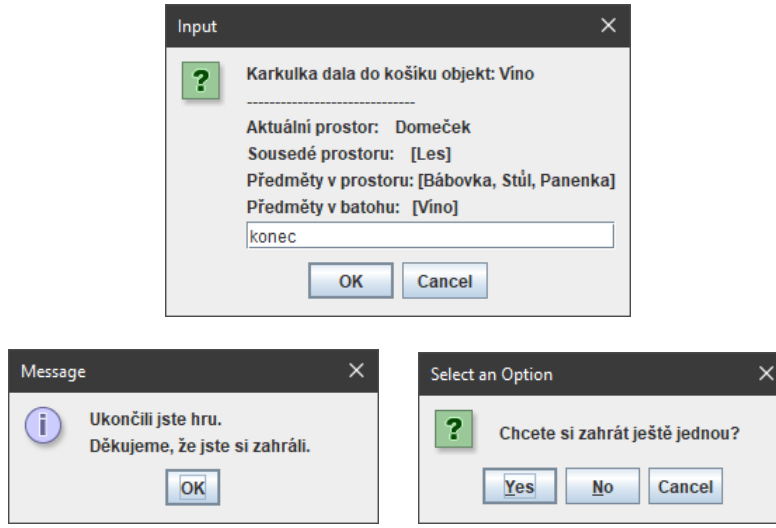
38.5.3 Třída `PrimitiveGUI`

S pomocí uvedených metod třídy `JOptionPane` nyní můžeme snadno definovat třídu `PrimitiveGUI`, které bude realizovat primitivní grafické rozhraní pro naši hru. Její definici najdete ve výpisu 38.8.

Výpis 38.8: Definice třídy `PrimitiveGUI`

```
 1 package game77.ck1l_user_io;
 2
 3 import javax.swing.JOptionPane;
 4 import static javax.swing.JOptionPane.showInputDialog;
 5 import static javax.swing.JOptionPane.showMessageDialog;
 6 import static javax.swing.JOptionPane.showConfirmDialog;
 7
 8 public   class PrimitiveGUI
 9     implements IUI
10 {
11     private static final PrimitiveGUI SINGLETON = new PrimitiveGUI();
12     public static PrimitiveGUI get() { return SINGLETON; }
13     private PrimitiveGUI() {}
14
15     @Override
16     public String getCommand(String answer) {
17         String command = showInputDialog(answer);
18         if (command == null) { return ""; }
19         return command;
20     }
21
22     public void showAnswer(String answer) {
23         showMessageDialog(null, answer);
24     }
25
26     public boolean askQuestion(String question) {
27         int answer = showConfirmDialog(null, question);
28         return answer == JOptionPane.YES_OPTION;
29     }
30 }
```

Podobu zobrazovaných dialogových oken v průběhu hry si můžete prohlédnout na obrázku 38.1.



Obrázek 38.1:
Dialogové okno s výzvou k zadání údajů

38.6 Několik dalších námětů

Hra, kterou jsme doposud vytvořili, je velmi prostoduchá. Nicméně vytvořený program by mohl být základem dokonalejších programů, na nichž by bylo možné vysvětlovat další pravidla návrhu a programátorské obraty. Pojďme si projít některé náměty.

Převod pod kvalitní grafické uživatelské rozhraní

Po primitivním GUI je to jistě to první, co vás napadne. Nicméně popis tohoto tématu by spolu s výkladem potřebných programátorských obratů vydal sám o sobě na samostatnou publikaci.

Zdokonalení h-objektů

Doposud jsme v prefixu názvu h-objektů pouze oznamovali, zda daný objekt je či není přenositelný. Mohli bychom ale zadat i jemnější dělení. Budou-li např. náš batoh představovat ruce, mohli bychom rozdělit předměty na ty, k jejichž přenesení stačí jedna ruka, a na ty těžší, k jejichž přenesení jsou potřeba obě ruce.

Vhodným nastavením prefixu můžete zadat i další charakteristiky, např. že se jedná o alkoholický nápoj, který proto smějí konzumovat pouze dospělé osoby.

H-objekty – prostory

Mezi h-objekty mohou být i takové, které jsou současně prostory, do nichž se musíte dostat speciálním příkazem. V aktuálním prostoru může být např. truhla, do níž se dostanete zadáním příkazu `otevři`. Pak se truhla stane aktuálním prostorem, z nějž můžete h-objekty odebírat, nebo je do něj naopak ukládat.

Hra může také vyžadovat, abyste k otevření truhly použili klíč, který musíte nejprve najít a případně se jej proti nějakému odporu zmocnit.

Rozhovor

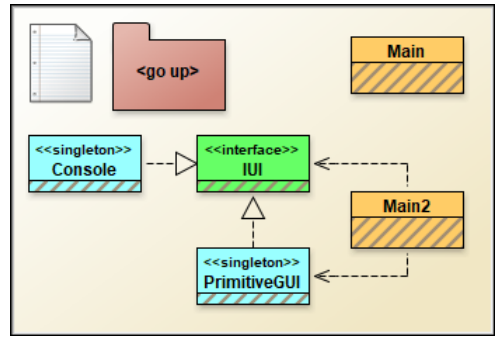
Speciálním druhem akce je taková, při níž zapředeme rozhovor s nějakým h-objektem – osobou či věcí, od níž něco potřebujeme. Můžeme např. oslovit kouzelného dědečka, od něj se dozvíme, že má hlad, my mu dáme buchtu, on nám dá píšťalku, kterou uspíme psa, jenž hlídá princeznu, apod.

Jednotlivé věty rozhovoru nejsou příkazy, ale na druhou stranu rozhovor může být prokládán příkazy. Program tak střídavě přepíná mezi dvěma režimy: rozhovorem a zadáváním příkazů.

38.7 Shrnutí a soubory pro opakování

Zadávané příkazy i obsah výpisů se záznamem komunikace s interpretem probíhající v této kapitole najdete v souborech nazvaných `s38__Uživatelské_rozhraní`. Aktuální podoba zdrojových kódů je zkopírována do balíčku `game77.ck1l_user_io`.

UML diagram aktuálního stavu tohoto balíčku si můžete prohlédnout na obrázku 38.2.



Obrázek 38.2:
Diagram tříd aktuálního stavu balíčku `game77.ck1l_user_io`

Literatura

[1] *Akademický slovník cizích slov*. Praha: Academia, 1995. ISBN 80-200-0497-1.

[2] BERGIN, Joseph, STEHLIK, Mark, ROBERTS, Jim, PATTIS, Richard: *Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java*. Dreamsongs Press, 2013, ISBN 978-0-9705795-1-5

[3] BERGIN, Joseph: *Beyond Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming*. Slant Flying Press, 2023, ISBN 978-1-9401130-9-8

[4] BERGIN, Joseph: *Beyond Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming, Volume 2*. Software Tools, 2013, ISBN 978-0-9851543-0-1

[5] FOOTE Brian, YODER Joseph, *Big Ball of Mud Fourth Conference on Patterns Languages of Programs* (PLoP '97/EuroPLoP '97) Monticello, Illinois, září 1997. Dostupné na adrese http://www.laputan.org/mud/.

[6] FOWLER, Martin. *UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition*. Addison-Wesley Professional, 2003. 192 s. (Přeloženo v [7]) ISBN 0-321-19368-7.

[7] FOWLER Martin. *Destilované UML.* Praha: Grada Publishing, 2009. 174 s. (Překlad [6]). ISBN 978-80-247-2062-3.

[8] FOWLER, Martin. *Refactoring – Improving the Design of Existing Code.* Addison-Wesley, 2000. 430 s. (Přeloženo v [9]) ISBN 0-201-48567-2.

[9] FOWLER, Martin. *Refaktoring – Zlepšení existujícího kódu.* Praha: Grada Publishing, 2003. 394 s. (Překlad knihy [8]) ISBN 80-247-0299-1.

[10] GAMMA E., HELM R., JOHNSON R., VLISSIDES J.: *Design Patterns: Elements of Reusable Object-oriented Software.* Boston: Addison-Wesley, 2001. ISBN 0-201-63361-2.

[11] GOSLING J., JOY B., STEELE G., BRACHA G., BUCKLEY A., SMITH D., BIERMAN G.: *The Java® Language Specification – Java SE 14 Edition.* Dostupné na adrese https://docs.oracle.com/javase/specs

[12] HAVELKA Antonín, PECINOVSKÝ Rudolf: *JUnit 5 – Jednotkové testování na platformě Java*. Praha: Grada Publishing, 2018. ISBN 978-80-271-0733-9.

[13] LEXA Ivan: *Shora dolů nebo zdola nahoru?* Programování Ostrava, 1987. Dostupné na adrese http://prog-story.technicalmuseum.cz/index.php/m-virtualni-sbirky-tm-v-brne/programovani-a-tvorba-sw-ostrava/1985-1994/1987-programovani-ostrava/2985-1987-shora-dolu-nebo-zdola-nahoru.

[14] McLAUGHLIN Brett D., POLLICE Gary WEST Dave: *Head First Object-Oriented Analysis and Design*. O'Reilly Media 2006. ISBN 978-0-596-00867-3.

[15] PATTIS Richard E.: *Karel The Robot: A Gentle Introduction to the Art of Programming*. John Wiley & Sons, 1981. ISBN 0-471-59725-2.

[16] PECINOVSKÝ R.: *Java 5.0 – Novinky jazyka a upgrade aplikací*, Brno: Computer Press, 2005, ISBN 80-251-0615-2. Dostupné na adrese http://knihy.pecinovsky.cz/java5novinky/

[17] PECINOVSKÝ Rudolf*: Návrhové vzory – 33 vzorových postupů pro objektové programování*. Brno: Computer Press, 2007. ISBN 978-80-251-1582-4.

[18] PECINOVSKÝ R.: *Myslíme objektově v jazyku Java.* Praha: Grada Publishing, 2009. ISBN 978-80-247-2653-3.

[19] PECINOVSKÝ R.: *Java 7 – Učebnice objektové architektury pro začátečníky.* Praha: Grada Publishing, 2012. ISBN 978-80-247-3665-5 (tištěná), ISBN 978-80-247-8325-3 (PDF), ISBN 978-80-247-8326-0 (epub).

[20] PECINOVSKÝ R.: *Java 8 – Učebnice objektové architektury pro mírně pokročilé.* Praha: Grada Publishing, 2014. ISBN 978-80-247-4638-8 (tištěná).

[21] PECINOVSKÝ Rudolf: *OOP a Java 8: Návrh a vývoj složitějšího projektu vyhovujícího zadanému rámci*. Praha: Nakladatelství Evy a Tomáše Brucknerů, 2015. ISBN 978-80-87924-03-7.

[22] PECINOVSKÝ Rudolf*: Java 9 – JShell*. Praha: Grada Publishing, 2017. ISBN 978-80-271-9785-9 (pdf), ISBN 978-80-271-9786-6 (epub).

[23] PECINOVSKÝ Rudolf: *Java 14 – Kompletní příručka jazyka*. Praha: Grada Publishing, 2020. ISBN 978-80-271-1369-9.

[24] PECINOVSKÝ R.: *Java 21 – Kompletní příručka jazyka.* Grada Publishing 2023. ISBN 978-80-271-7041-8 (pdf), ISBN 978-80-271-7042-5 (ePub), ISBN 978-80-247-0599-6 (print).

[25] POST E.: *Real Programmers Don't Use PASCAL*. Datamation 1983. Dostupné na adrese https://www.ee.ryerson.ca/~elf/hack/realmen.html, český překlad *Opravdoví programátoři nepoužívají Pascal* dostupný na adrese http://www.logix.cz/michal/humornik/Pojidaci.Kolacu.xp.

[26] PRAVDOVÁ Markéta, SVOBODOVÁ Ivana: *Akademická příručka českého jazyka, 2., rozšířené vydání*. Praha: Academia, 2019. ISBN 978-80-200-2947-8.

[27] VIRIUS Miroslav: *Java – programování podprocesů (vláken)*. Praha: Grada Publishing, 2021. ISBN 978-80-271-3266-9 (print), 978-80-271-4253-8 (pdf), 978-80-271-4254-5 (ePub)

[28] VERMEER Brian: *IntelliJ IDEA Dominates the IDE Market with 62% Adoption among JVM Developers*. publikováno 5. 2. 2020 na https://snyk.io/blog/intellij-idea-dominates-the-ide-market-with-62-adoption-among-jvm-developers/.

Rejstřík

L

M

N

O

P

R

Ř

S

Š

T

U

V

X

Y

Z

Část F
Přílohy

Tato část se vyskytuje pouze v elektronické verzi učebnice. Podrobněji vás seznámí s postupem při instalaci prostředí *JShell* v systému *Windows* včetně představení poněkud opomíjené možnosti definice substituovaných disků. Následující přílohy jsou pak určeny pro naprosté začátečníky, kteří občas zápasí s termíny a někteří i s angličtinou.

Příloha A Příprava spuštění JShell pod Windows

A.1 Problémy s klávesnicí

Už od svého prvního uvedení ve verzi 9 mělo prostředí *JShell* problém s některými jinými rozloženími kláves než US. V českém prostředí bylo možné až do verze 12 tyto problému řešit používáním rozložení označovaného jako *České (QWERTY)*. To používá řada programátorů, protože znaky, které se nevyskytují na klávesnici psacího stroje, má toto rozložení umístěny na stejných klávesách jako rozložení US, jenom se aktivují s jiným nastavením přeřaďovačů.

Bohužel se pak někdo rozhodl prostředí *JShell* „vylepšit“. Od té chvíle sice funguje rozložení označované jako *České*, ale pro změnu zase přestalo chodit rozložení *České (QWERTY)*, které nám nyní neumožňuje zadat některé důležité znaky – např. složené závorky (`{}`) a „svislítko“ (`|`), a to ani tak, že je do textu vložíte ze schránky.

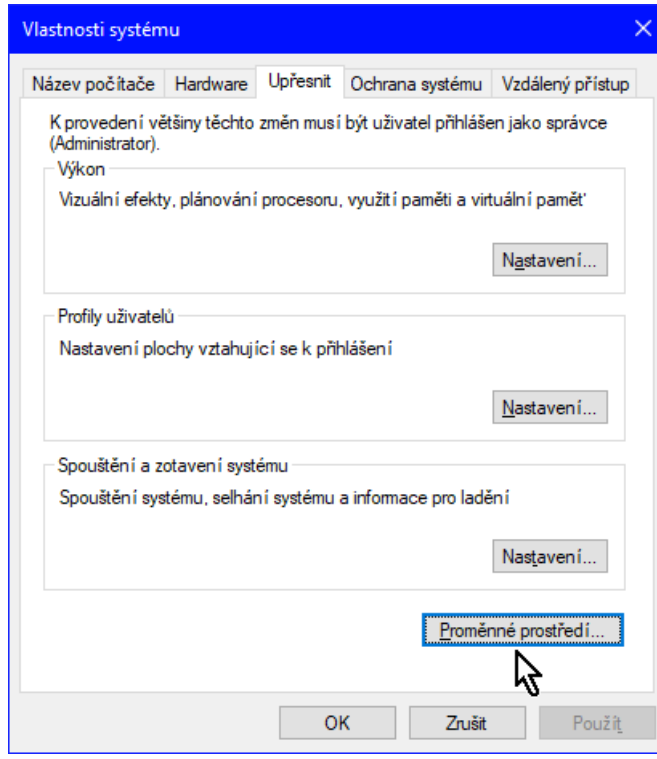
Vložit je můžete buďto přímým zadáním jejich kódu (`{=123, |=124, }=125`), anebo použitím jiného rozložení kláves.

A.2 Příprava spuštění pod Windows

Jak již bylo řečeno, jednou z velkých nevýhod *Windows* je, že mají pro různé skupinky států nastavená různá kódování. Všechny doprovodné programy k tomuto kurzu ale budou kódovány v celosvětově jednotném kódování UTF-8.

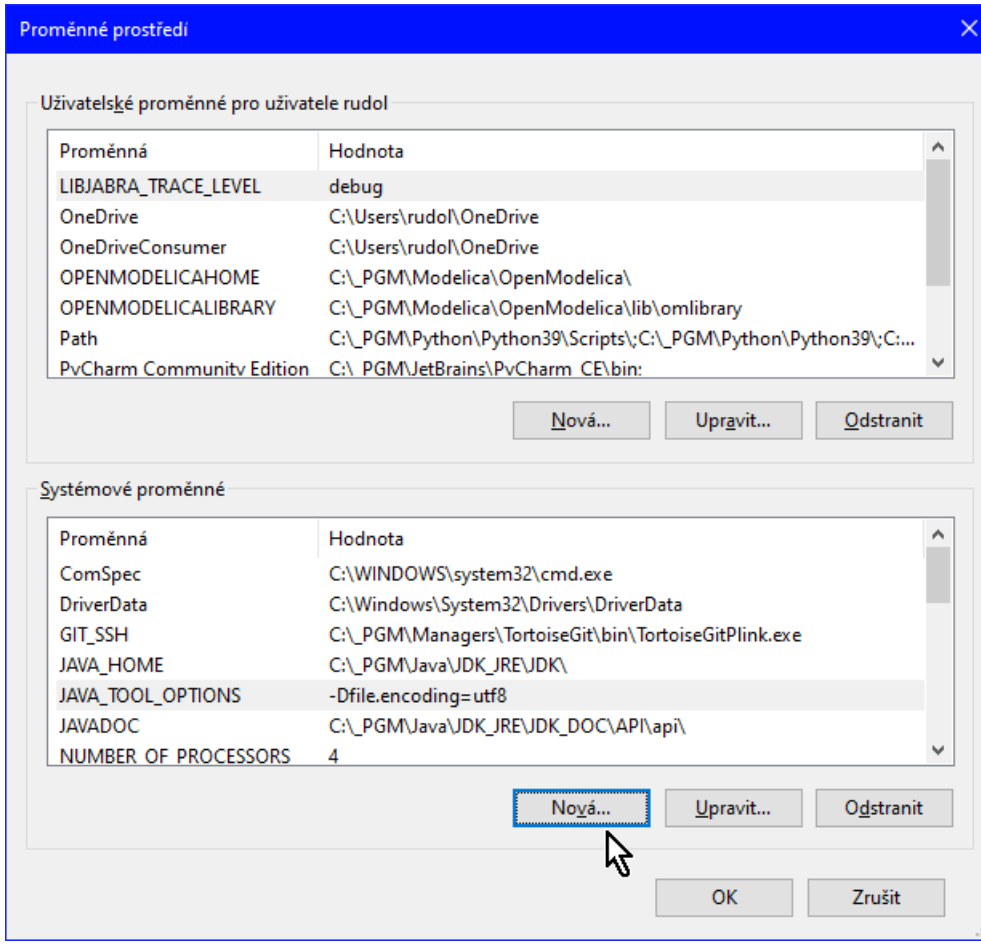
Budete-li chtít toto kódování používat ve všech svých programech v jazyku *Java*, bude nejlepší, když definujete proměnnou prostředí `JAVA_TOOL_OPTIONS`. Dosáhnete toho následovně:

1. V nastaveních systému požádáte o nastavení proměnných prostředí.
2. Otevře se okno **Vlastnosti systému**, v němž stisknete tlačítko **Proměnné prostředí** (viz obrázek A.1).



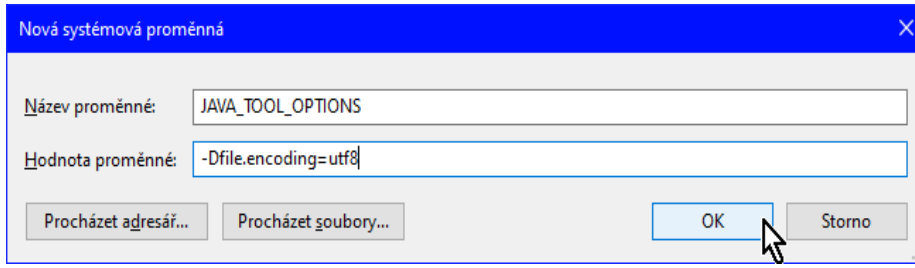
Obrázek A.1:
Dialogové okno ***Vlastnosti systému***

3. Tím otevřete stejnojmenné okno. V něm požádáte o vytvoření nové proměnné (viz obrázek A.2).



Obrázek A.2:
Dialogové okno ***Proměnné prostředí***

4. V následně otevřeném dialogovém okně **Nová systémová proměnná** zadáte název `JAVA_TOOL_OPTIONS` a hodnotu `-Dfile.encoding=utf8`. (viz obrázek A.3).



Obrázek A.3:
Popisek obrázku

5. Zadání potvrdíte a okna postupně pozavíráte.

Pro vlastní spuštění pak doporučujeme definovat dávkový soubor, který nejprve nastaví kódovou stránku 65001 (číslo stránky s kódováním UTF-8) a teprve pak spustí *JShell*, jak naznačuje obrázek 2.1 na straně 46.

A.3 Definice substituovaných disků

V operačním systému *Windows* můžete používat 26 logických disků – pro každé písmeno abecedy jeden. Většina uživatelů však používá pouze zlomek tohoto počtu. *Windows* umožňují použít volná písmena pro tzv. **substituované disky**, což jsou složky, které se rozhodnete vydávat za logický disk. Protože o této možnosti většina uživatelů neví, a přitom je to funkce velice užitečná, ukážu vám, jak ji můžete využít. Substituované disky se definují pomocí příkazu:

```
SUBST název_disku substituovaná_složka
```

Nejjednodušší způsob, jak definovat ve *Windows* např. substituovaný disk `P:`, je vložit do složky, kterou budete chtít substituovat jako disk `P:`, dávkový soubor s příkazem k substituci. (Písmeno `P` se pro *Python* hodí nejlépe, ale můžete si vybrat jakékoliv jiné, které je na vašem počítači volné.)

Pokud jste ještě nepracovali s dávkovými soubory, tak věřte, že to jsou obyčejné textové soubory, do nichž zapisujete příkazy pro operační systém. Jejich název může být libovolný, ale musí mít příponu bat (zkratka ze slova batch – dávka). V dávkovém souboru budou následující příkazy (na velikosti písmen nezáleží):

```
SUBST P: /d
SUBST P: .
```

První příkaz má za úkol zrušit případnou doposud nastavenou substituci disku `P:` (není-li v daném okamžiku označený disk substituován, systém vypíše chybovou zprávu, ale jinak se nic nestane), druhý příkaz pak substituuje aktuální složku jako disk `P:`.

Soubor umístíte do složky, z níž budete chtít udělat substituovaný disk. Kdykoliv pak tento dávkový soubor spustíte, dávka substituuje složku, v níž je umístěna, jako příslušný disk. Dávka se přitom spouští obdobně jako aplikace – např. poklepáním na ikonu jejího souboru v *Průzkumníku*.

Kdykoliv od této chvíle budete pracovat s diskem `P:`, budete ve skutečnosti pracovat s obsahem substituované složky. A naopak: cokoliv uděláte s obsahem substituované složky, uděláte zároveň s obsahem disku `P:`.

Budete-li chtít mít danou substituci nastavenu trvale, můžete umístit zástupce dávkového souboru do položky Po spuštění ve startovní nabídce. Protože je v každé verzi operačního systému jinde, bude nejlepší, když ji ve startovní nabídce najdete, klepnete na ni pravým tlačítkem a v následně otevřené místní nabídce zadáte **Otevřít**. Tím otevřete okno průzkumníka s touto složkou. Pak v druhém okně průzkumníka

otevřete složku s příslušným dávkovým souborem, uchopíte jeho ikonu PRAVÝM tlačítkem myši, přesunete ji do složky nabídky a pustíte. Otevře se místní nabídka (ta se otevře, pouze pokud přesouváte soubor pravým tlačítkem myši), ve které zadáte, že zde chcete vytvořit zástupce, a tím celý proces končí.

Abyste mohli složku substituovat jako nějaký disk, nesmí váš operační systém používat disk označený tímto písmenem. Písmeno může být použito nejvýše pro jiný substituovaný disk, protože tuto substituci můžete před nastavením nové substituce zrušit (pro tento případ je v dávkovém souboru první příkaz s parametrem `/d` – delete).

Používáte-li operační systém *Windows*, můžete urychlit budoucí vyhledání složky s projekty právě tím, že pro ni zřídíte zvláštní substituovaný disk. Kdykoliv se pak obrátíte na příslušný disk, obrátíte se ve skutečnosti k příslušné složce. Pomocí substituce si tak můžete zkrátit cestu k často používaným složkám.

Příloha B
Význam cizích slov

Jak jsme se zmínili v úvodu, knihu jsme se snažili psát tak, aby ji mohli používat i studenti středních škol a víceletých gymnázií, z nichž mnozí si ještě s řadou relativně běžně používaných cizích slov „netykají". Snažili jsme se proto do tohoto slovníku zařadit všechna cizí slova, která byla v textu použita. Pokud jsme na nějaké zapomněli, napište, abychom je mohli do příštího vydání doplnit.

alokovat	přidělit zdroje – v OOP se používá nejčastěji v souvislosti s pamětí: alokovat paměť = vyhradit ji a přidělit danému objektu.
arita	počet operandů operátoru (operátory nulární, unární, binární, ternární)
aspekt	hledisko, stanovisko uplatňované při posuzování, zorný úhel
averze	odpor
architektura	uspořádání, skladba nějakého celku
definice	vytvoření
deklarace	veřejné prohlášení; část programu vymezující vlastnosti objektů; tím, že v programu něco deklaruji, oznamuji, jaké to bude, až to bude „živé"
ekvivalent	obdoba
entita	základní objekt zkoumání
explicitně	výslovně, zřetelně
hyperodkaz	zkratka z „*hypertextový odkaz*", což je aktivní odkaz do jiné části dokumentu nebo do jiného dokumentu, kam se dá přejít klepnutím na tento odkaz
hypotetický	založený na předpokladu, podmíněný, nejistý
implicitní	skrytý, předem daný (použije se, nebude-li řečeno jinak)
implementace	způsob provedení
interní	vnitřní, často také pro vnitřní potřebu, neukazovaný navenek

kompilace	překlad (programu)
metodika	1. pracovní postup; 2. nauka o metodě vyučování
mnemonický	např. název – vytvořený tak, aby se snadno zapamatoval
priorita	přednost
netriviální	opak triviální
realizace	provedení, uskutečnění
reference	dobrozdání, doporučení, posudek – např. v životopise se vás ptají na firmy, které by jim mohly na vás dát reference. Řada autorů však tento termín používá (špatně) i jako překlad anglického termínu *reference*, který se ale správně překládá jako odkaz.
robustní	důkladný (robustní program není citlivý na vadná vstupní data, přerušení síťového připojení ani další vlivy okolí)
seance	zasedání, sezení
separátní	oddělený, zvláštní
specifikace	bližší určení, vymezení něčeho s uvedením přesných rozlišujících údajů
specifikovat	určit, zadat
synonyma	slova, která mají stejný, nebo velmi podobný význam (např. běžet – utíkat)
syntaxe	pravidla pro vytváření přípustných programů; syntaxe se nezabývá tím, jestli bude program pracovat, ale pouze tím, jestli jej smíme takto napsat
triviální	jednoduché na pochopení nebo na realizaci (uskutečnění)
virtuální	zdánlivý

Příloha C
Česko-americký slovník

Možná byste na tomto místě čekali spíše česko-anglický slovník, ale snažil jsem se hned v nadpisu upozornit na to, že liší-li se u některých slov britský a americký překlad (nebo jen pravopis – např. colour × color), používám ten americký, protože počítačová angličtina je prakticky výhradně americká. Pro jistotu uvádím u anglických slovíček v hranatých závorkách i jejich výslovnost. V tomto přepisu označuje písmeno **ə** zvuk, který vydáváte po vyslovení samotné souhlásky – např. zkratku JDK můžete vyslovit buď dlouze [jédéká], anebo krátce [jədəkə].

balíček	package [pekidž]
barva	color [kalər]
budovatel	builder [bildər]
dědictví	inheritance [inhəritəns]
dědit	inherit [inhərit]
halda	heap [híp]
identifikátor	identifier [ajdentifaiər]
instance	instance [instəns]
jedináček	singleton [singltən]
jednoduchý název	simple name [simpl nejm]
mapa	map [mep]
metoda	method [methəd]
množina	set [set]
objekt	object [obdžikt]
odkaz	reference [refrəns]
parametr	parameter, argument [paramitr, ágjumənt]
pole	array [ərei]
proměnná	variable [variəbl]

přebít	override [ouvəraid]
přepravka	crate [kreit]
rozhraní	interface [intəfeis]
seznam	list [list]
služebník	servant [sévənt]
továrna, tovární	factory [fektəry]
třída	class [klás]
vlastnost	property [propəty]
zásobník	stack [stek]
zásobník návratových adres	return stack [ritəən stek]
závislost	dependence [dipendəns]
zpráva	message [mesidž]

Výslovnost některých slov v programu

boolean	[búlín]
break	[breik]
byte	[bait]
case	[keis]
class	[klás]
double	[dabl]
else	[els]
final	[fainl]
float	[flout]
for	[fór]
implements	[impliments]
instanceof	[instəns of]
interface	[intəfeis]
NetBeans	[netbíns]
new	[njú]
package	[pekidž]
private	[praivit]
public	[pablik]
return	[ritəən]
short	[šót]
static	[stetik]
super	[súpə]
switch	[swič]
while	[wail]

Část G
Seznamy

Seznam S1
Výpisy programů

Seznam S2
Seznam obrázků

Seznam S3
Seznam tabulek

Seznam S4 Seznam odboček – podšeděných bloků

Nové vydání knihy o programování v jazyce Java si klade ambiciózní cíl: naučit každého, včetně začátečníků, skutečně programovat moderním, objektově orientovaným stylem, jímž se v dnešní době vyvíjí drtivá většina klíčových aplikací. Nespokojí se jen s tím, že naučí čtenáře napsat nějaký program. Mnohem důležitější je zvládnout správný návrh a pochopit základní principy, na nichž je objektově orientované programování postaveno. Nedržíte tedy v ruce jen příručku jazyka Java, ale učebnici objektově orientovaného programování v Javě. Je možné, že práci „pouhých“ kodérů v Javě bude časem vykonávat umělá inteligence, ale schopný architekt a návrhář aplikací je zatím stále nenahraditelný. A právě tím se můžete stát s pomocí této knihy.

Knihu lektorovali vedoucí různých programátorských týmů, aby posoudili, nakolik odráží potřeby a zvyky jejich každodenní praxe. Někteří z nich se rozhodli, že ji ve svých týmech budou používat jako povinnou literaturu pro nováčky. Lepší doporučení snad ani nemohla dostat.

GRADA®

Grada Publishing, a. s.,
U Průhonu 22, 170 00 Praha 7
tel.: +420 234 264 401
e-mail: obchod@grada.cz, www.grada.cz

CZ 549 Kč / SK 22,95 €

9 788027 157518