

Contents

List of Figures	xvii
List of Tables	xxi
Listings	xxiii

Part I. Getting Started

1. Expressions, Variables and Assignments	3
1.1 Introduction	3
1.2 My first Java programs	3
1.2.1 A minimalist program	3
1.2.2 Hello World	4
1.3 Expressions and programs as calculators	5
1.3.1 Arithmetic operations and priority order	6
1.3.2 Mathematical functions	8
1.3.3 Declaring constants	10
1.4 Commenting Java programs	10
1.5 Indenting programs	11
1.6 Variables, assignments and type checking	11
1.6.1 Variables for storing intermediate values	12
1.6.2 Type checking for assignments and casting	15
1.6.3 The inner mechanisms of assignments	17

1.7	Incrementing/decrementing variables	17
1.7.1	General mechanism for incrementation	17
1.7.2	Pre-incrementation and post-incrementation	18
1.7.3	A calculator for solving quadratic equations	19
1.8	Basics of Java input/output (I/O)	20
1.8.1	Computing does not mean displaying	20
1.8.2	Keyboard input	21
1.8.3	File redirections	23
1.9	Bugs and the art of debugging	24
1.10	Integrated development environments (IDEs)	26
1.11	Exercises	27
1.11.1	Note to instructors	27
1.11.2	First set of exercises	28
2.	Conditional Structures and Loops	31
2.1	Instruction workflow	31
2.2	Conditional structures: Simple and multiple choices	32
2.2.1	Branching conditions: <code>if ... else ...</code>	32
2.2.2	Ternary operator for branching instructions: <code>Predicate</code> <code>? A : B</code>	34
2.2.3	Nested conditionals	35
2.2.4	Relational and logical operators for comparisons	36
2.2.5	Multiple choices: <code>switch case</code>	39
2.3	Blocks and scopes of variables	40
2.3.1	Blocks of instructions	40
2.3.2	Nested blocks and variable scopes	41
2.4	Looping structures	41
2.4.1	Loop statement: <code>while</code>	42
2.4.2	Loop statement: <code>do-while</code>	43
2.4.3	Loop statement: <code>for</code>	45
2.4.4	Boolean arithmetic expressions	46
2.5	Unfolding loops and program termination	47
2.5.1	Unfolding loops	47
2.5.2	Never ending programs	47
2.5.3	Loop equivalence to universal <code>while</code> structures	48
2.5.4	Breaking loops at any time with <code>break</code>	48
2.5.5	Loops and program termination	48
2.6	Certifying programs: Syntax, compilation and numerical bugs . .	49
2.7	Parsing program arguments from the command line	51
2.8	Exercises	53

3. Functions and Recursive Functions	57
3.1 Advantages of programming functions	57
3.2 Declaring and calling functions	58
3.2.1 Prototyping functions	58
3.2.2 Examples of basic functions	59
3.2.3 A more elaborate example: The iterative factorial function	60
3.2.4 Functions with conditional statements	61
3.3 Static (class) variables	62
3.4 Pass-by-value of function arguments	64
3.4.1 Basic argument passing mechanism	64
3.4.2 Local memory and function call stack	64
3.4.3 Side-effects of functions: Changing the calling environment	67
3.4.4 Function signatures and function overloading	68
3.5 Recursion	70
3.5.1 Revisiting the factorial function: A recursive function ...	71
3.5.2 Fibonacci sequences	72
3.5.3 Logarithmic mean	73
3.6 Terminal recursion for program efficiency **	74
3.7 Recursion and graphics **	76
3.8 Halting problem: An undecidable task	77
3.9 Exercises	79
4. Arrays	83
4.1 Why do programmers need arrays?	83
4.2 Declaring and initializing arrays	83
4.2.1 Declaring arrays	83
4.2.2 Creating and initializing arrays	84
4.2.3 Retrieving the size of arrays: length	85
4.2.4 Index range of arrays and out-of-range exceptions	86
4.2.5 Releasing memory and garbage collector	87
4.3 The fundamental concept of array references	87
4.4 Arrays as function arguments	90
4.5 Multi-dimensional arrays: Arrays of arrays	93
4.5.1 Multi-dimensional regular arrays	93
4.5.2 Multi-dimensional ragged arrays **	95
4.6 Arrays of strings and main function	97
4.7 A basic application of arrays: Searching **	99
4.8 Exercises	101

Part II. Data-Structures & Algorithms

5. Objects and Strings	107
5.1 Why do programmers need objects?	107
5.2 Declaring classes and creating objects	108
5.2.1 Constructor and object creation	109
5.2.2 The common <code>null</code> object	110
5.2.3 Static (class) functions with objects as arguments	111
5.3 Objects and references	113
5.3.1 Copying objects: Cloning	114
5.3.2 Testing for object equality	114
5.4 Array of objects	115
5.5 Objects with array members	117
5.6 The standardized <code>String</code> objects	117
5.6.1 Declaring and assigning <code>String</code> variables	117
5.6.2 Length of a string: <code>length()</code>	118
5.6.3 Equality test for strings: <code>equals(String str)</code>	118
5.6.4 Comparing strings: Lexicographic order	119
5.7 Revisiting a basic program skeleton	122
5.8 Exercises	123
6. Searching and Sorting	127
6.1 Overview	127
6.2 Searching information	128
6.3 Sequential search	129
6.3.1 Complexity of sequential search	131
6.3.2 Dynamically adding objects	131
6.3.3 Dichotomy/bisection search	133
6.4 Sorting arrays	134
6.4.1 Sorting by selection: <code>SelectionSort</code>	135
6.4.2 Extending selection sort to objects	136
6.4.3 Complexity of selection sorting	138
6.5 QuickSort: Recursive sorting	139
6.5.1 Complexity analysis of QuickSort	140
6.6 Searching by hashing	140
6.7 Exercises	142
7. Linked Lists	145
7.1 Introduction	145
7.2 Cells and lists	145
7.2.1 Illustrating the concepts of cells and lists	145

7.2.2	List as an abstract data-structure	146
7.2.3	Programming linked lists in Java	146
7.2.4	Traversing linked lists	147
7.2.5	Linked lists storing String elements	148
7.2.6	Length of a linked list	149
7.2.7	Dynamic insertion: Adding an element to the list	150
7.2.8	Pretty printer for linked lists	151
7.2.9	Removing an element from a linked list	151
7.2.10	Common mistakes when programming lists	153
7.3	Recursion on linked lists	153
7.4	Copying linked lists	155
7.5	Creating linked lists from arrays	156
7.6	Sorting linked lists	156
7.6.1	Merging ordered lists	157
7.6.2	Recursive sorting of lists	158
7.7	Summary on linked lists	160
7.8	Application of linked lists: Hashing	160
7.8.1	Open address hashing	162
7.8.2	Solving collisions with linked lists	164
7.9	Comparisons of core data-structures	165
7.10	Exercises	165
8.	Object-Oriented Data-Structures	169
8.1	Introduction	169
8.2	Queues: First in first out (FIFO)	169
8.2.1	Queues as abstract data-structures: Interfaces	169
8.2.2	Basic queue implementation: Static functions	170
8.2.3	An application of queues: Set enumeration	172
8.3	Priority queues and heaps	173
8.3.1	Retrieving the maximal element	175
8.3.2	Adding an element	175
8.3.3	Removing the topmost element	177
8.4	Object-oriented data-structures: Methods	178
8.5	Revisiting object-oriented style data-structures	182
8.5.1	Object oriented priority queues	182
8.5.2	Object-oriented lists	183
8.6	Stacks: Last in first out (LIFO) abstract data-structures	185
8.6.1	Stack interface and an array implementation	186
8.6.2	Implementing generic stacks with linked lists	187
8.7	Exercises	189

9. Paradigms for Optimization Problems	191
9.1 Introduction	191
9.2 Exhaustive search	192
9.2.1 Filling a knapsack	192
9.2.2 Backtracking illustrated: The eight queens puzzle	198
9.3 Greedy algorithms: Heuristics for guaranteed approximations	201
9.3.1 An approximate solution to the 0-1 knapsack problem	201
9.3.2 A greedy algorithm for solving set cover problems	205
9.4 Dynamic programming: Optimal solution for the 0-1 knapsack problem	211
9.5 Optimization paradigms: Overview of complexity analysis	214
9.6 Exercices	216
10. The Science of Computing	219
10.1 The digital world	219
10.2 Nature of computing?	221
10.3 The digital equation	222
10.4 Birth of algorithms and computers	222
10.5 Computer science in the 21st century	223
<hr/>	
Part III. Exam Review	
<hr/>	
11. Exam & Solution	227
Bibliography	247
Index	249



<http://www.springer.com/978-1-84882-338-9>

A Concise and Practical Introduction to Programming
Algorithms in Java

Nielsen, F.

2009, XXVIII, 252 p., Softcover

ISBN: 978-1-84882-338-9