

INDEX

Special Characters

[](square brackets), 79
<> (angle brackets), 159, 389
* (asterisk), 27, 79, 93
+ (plus sign), 103
/ (slash), 27, 93
= (equal sign), 124–125
.(dot), 125

A

A. M. Turing Award, 521
abstract classes, 144–146
abstract keyword, 145
abstract methods, 144
Abstract Windowing Toolkit (AWT),
 22, 807
accept() method, 896–897
acceptance testing, 26, 229
access control, 922
accessors, 47
 object-oriented software
 development case study,
 185–186
activity diagrams, UML, 67
acyclic graphs, directed, 562
adapter classes, 845–846
adaptive programming, 59
add() method, 296–297,
 298–299, 633, 779–780, 783
 complexity, 516, 519
add(element) method, time
 complexity, 328
add(element, position)
 method, time complexity, 328
addAll() method, 618, 630
addFirst() method, 297
addFirst(element) method,
 time complexity, 328

addition operator (+), 103
adjacency list(s), 593, 595–599
adjacency list representation of
 graphs, 593, 595–599
 space needs, 597
 time complexities of inser-
 tions and deletions, 598
adjacency matrix representation of
 graphs, 593–595
 space needs, 597
 time complexities of inser-
 tions and deletions, 598
adjacent nodes, 552
agile programming, 59
Agora, 45
AI (artificial intelligence), 396–397
Airport Wireless Networking, 852
ALGOL 60 compiler, 357
algorithm(s), 243–278, 666–670
 asymptotic analysis, 246,
 249–266
 asymptotic equality, 266
 asymptotically superior,
 247–248
 big-O notation, 247
 big-omega notation, 266
 big-theta notation, 266
 complexity, classes, 248–249
 cubic complexity, 263
 derivation of word, 270
 Dijkstra's, 584–593
 efficiency, 246–249
 exponential, 265–266
 Java Collection Framework,
 666–670
 Kruskal's, 577–583
 lower order, 247
 oldest child/next sibling, 415
 polynomial, 265

rebalancing, 465
recursive, 267–277
run times on computer
 types, 262–263, 264
stable sorting, 277
streams, 705
algorithm and data structure selec-
 tion phase of software life cycle,
 14–17
aliases, 119
Al-Khwarizmi, Muhammad ibn
 Musa, 270
Allen, Paul, 727
Altair 8080, 727
ancestors, nodes, 393
angle brackets (<>)
 generics, 159
 nonterminal symbols, 389
Apache Web server, 193
Apple Computer Inc., 806, 852
Apple I computers, 852
Apple II computers, 852
Apple Macintosh, 808
applets, 18
Application layer of OSI model, 890
approximations, good, importance,
 491–492
Araine 5, 17
arguments, 102
ARPANET, 878, 879, 880
array(s), 15–16
array resizing, 307
array-based implementation
 binary trees, 432–446
 heaps, using one-dimensional
 arrays, 477–485
 lists, 305–314
 maps, 526–529
 queues, 363–367
 stacks, 352–353

- ArrayList** class, 657–660
- Arrays** class, 670
- The Art of Computer Programming* (Knuth), 490
- artificial intelligence (AI), 396–397
- ASCII strings, extracting from datagrams, 909–910
- assignment operation, 118–119
- assignment operator, 118
- associations, UML class diagrams, 69–72, 74
- asterisk (*)
 - iteration marker, 79
 - Javadoc comments, 27, 93
- asymptotic analysis of algorithms, 246, 249–266
 - average-case vs. worst-case analysis, 249–251
 - critical section, 251–252
 - examples, 252–266
- asymptotic equality of algorithms, 266
- asymptotically superior algorithms, 247–248
- @author tag, 27
 - Javadoc comments, 93
- automatic variables, 73
- average-case behavior of algorithms, 149–151
- AVL trees, 464, 465–466
- AWT (Abstract Windowing Toolkit), 22, 807

B

- back-of-the-envelope calculations, 491–492
- balanced binary search trees, 464–474
- banking system example
 - of class diagram, 76

- of object-oriented programming, 63–65
- base case, 268, 271–272
- base type, sets, 516
- BASIC, 727
- Beck, Kent, 59
- behaviors
 - linear data structures, 378
 - methods. *See* methods; *specific methods*
 - object-oriented software development case study, 184–189
 - objects, 46
- benchmarking, 246
- Berkeley Sockets, 891
- best-case behavior of algorithms, 149–151
- beta testing, 26
- BFSTraverse()** method, 571–572
- big-O notation, 247
- big-omega notation, 266
- big-theta notation, 266
- binary search algorithm, 254–256
 - analysis of recursive implementation, 272–274
- binary search tree(s)
 - AVL trees, 464, 465–466
 - B-trees, 464, 465
 - degenerate, 451
 - full, 452
 - linked list implementation, 453–459
 - minimum height, 459–460
 - red-black trees, 460, 464, 465–474
- binary search tree property, 446
- binary search trees (BSTs), 446–474
 - balanced, 464–474
 - definition, 446–448

- searching operations, 448–460
 - tree sort, 462–464
- binary trees, 397–464
 - array-based implementation, 432–446
 - complete. *See* heap(s)
 - heaps. *See* heap(s)
 - methods, 405–406
 - operations, 399–414
 - ordered nature, 398
 - reference-based implementation, 416–431
 - representation of general trees, 414–416
 - search. *See* binary search trees (BSTs)
- binarySearch()** method, 666–667, 668
- bioinformatics, 722–723
- bit vector, 516
- BITNET, 879
- black box testing, 26
- blocking, indefinite, 748, 793
- blogs, 918
- BlueGene/L system, 264, 737
- Bombe, 520–521
- Boo, 45
- Booch, Grady, 46
- BookShelf** class, 638–640
- border layout, 824–825
- bounded type parameters, 160–161
- breadth-first spanning trees, 576
- breadth-first traversal of graphs, 568, 569, 571–573
- Brooks, Fred, 30–31
- brute force solutions, 265–266
- BSTs. *See* binary search trees (BSTs)
- B-trees, 464, 465
- bubble sort algorithm, 256–259

- buckets, 653
- buffer(s), 712–715
 - implementing, 781–786
 - synchronized, 783
- Buffer** class, 779–783
- BufferedReader** class, 712–714
- bug, origin of term, 650
- bulk operations, 618
- busy wait loops, 751
- bytecode, 166
- bytecode verification, 922

C

- C language, 91
- C++ language, 18, 43, 91, 223
- call stacks, 693
- Callback** interface, 407–408
- callbacks, 407–408
- Cantor, Georg, 509
- cardinality of sets, 513
- Cashier** class, 786–787, 789–793
- catch blocks, 694, 696–697
- catch or declare policy, 688
- Cerf, Vinton G., 879–880
- chaining, 544–551
- checked exceptions, 686, 688, 689
- chess software, 396–397
- children, nodes, 393
- Church, Alonzo, 346
- Circle** class, 133
- circular lists, 342–345
- class(es), 14, 40–41, 49–50. *See also specific classes*
- abstract, 144–146
 - extending, 135–143
 - identifying in design phase, 60–65
 - implementation, 197–205

- implementation of interfaces, 147–149
- inheritance. *See* inheritance
- Java library, 19
- nested (inner), 91
- package scope, 92
- packages, 91
- subclasses. *See* subclasses
- superclasses (parent classes), 50, 134
 - as unit of sharing, 19
- Class** class, 775–777
- class definitions, 91–132
 - behavior, 99–113
 - identity, 114–128
 - Square** class example, 128–132
 - state, 93–99
- class diagrams, UML, 67
 - association relationships, 69–72, 74
 - dependency relationships, 72–73
 - examples, 74–77
 - generalization relationships, 73–74
 - multiplicity, 71–72
 - navigability information, 71–72
- class files, 166
- class identification, object-oriented
 - software development case study, 183–184
- class methods, 107–113
- class variables, 95–96
- clear box testing, alpha testing, 26
- clients, 815, 895
- climatic change, 244–245
- Clinton, Bill, 880
- Clock** class, implementation, 215–218

- clone()** method, 152
- closing a stream, 705
- COBOL programming language, 671
- code base, 923
- coding, 17
- coding and debugging phase of
 - software life cycle, 17–23, 58
- collaboration diagrams, UML, 67
- collection(s)
 - Java Collection Framework, 615
 - maintaining, 616–621
- Collection** interface, 616–624
 - iterators, 621–624
 - maintaining collections, 616–621
- collection views, 637, 638
- Collections** class, 666
- collisions, 531
 - chaining, 544–551
 - open addressing, 535–543
- combination form of inheritance, 53, 55
- comments
 - Javadoc, 27–29, 93
 - multiline, 93
- Comparable** interface, 641
- comparators, 646–647
- compareTo()** method, 641–644
- compiled languages, 165–166
- compilers, 165–166
- compiling programs, 165–168
- complexity classes, algorithms, 248–249
- component(s), 9
 - specifications, 9
- component diagrams, UML, 67
- composite data types, 288–292
- computationally intractable
 - problems, 265
- computer networks, 882

- concurrency. *See* thread(s)
- concurrent access, 760
- concurrent execution of threads, 761–762
- concurrent processing, 21
- concurrent programming, 735–736
- condition markets, UML sequence diagrams, 79
- connected components, 561
- connected graphs, 561
- connected subgraphs, 561
- connectionless transport protocol, 888
- connection-oriented transport protocol, 888
- constant time algorithm, 256
- constructors, 47, 106–107
 - default, 107
 - object-oriented software development case study, 185
- `contains()` method, 513
- `Collections`
 - interface, 617
 - complexity, 516, 519
- `containsAll()` method, 618
- `containsKey()` method, 637
- content equivalent variables, 48–49
- content pane, 812
- context switches, 736
- Control Data 6600, 264
- cooperative multitasking, 749–759
- copy constructors, 120–121
- copying files using streams, 715–716
- Corporation for National Research Initiatives, 880
- cost, of software errors, 17
- cost list, 584
- course registration system graphical user interface example, 853–857

- Cray X-MP, 264
- Crick, James, 722
- critical sections
 - algorithms, 251–252
 - synchronization program, 762–764
- CSNET, 879
- CSP programming model, 795
- cubic complexity, 263
- Cunningham, Ward, 59
- current node, 404
- current position indicator, 294, 404
- cursors, 294, 404
- `Customer` class, 787–789
- cycles, graphs, 560–561

D

- daemon threads, 742
- Dahl, Ole-Johan, 43, 224
- DARPA (Defense Advanced Research Projects Agency), 878, 880
- data entry screen graphical user interface example, 857–864
- `data` field, lists, 293
- Data Link layer of OSI model, 885
- data structures, 288–292
 - hierarchical, 289–290
 - linear, 293–378. *See also* list(s); queues; stack(s)
- data types, 288
 - composite, 288–292
 - primitive, 288
 - simple, 288
- `DatagramPacket` class, 905
- datagrams, 886
 - extracting ASCII strings, 909–910
 - extracting integers, 907–908
 - representing, 905–909

- `DatagramSocket` class, 909–916
- day/time clients, 899–900
- day/time servers, 900–905
- Deep Blue, 396
- deep compares, 407
- Deep Thought, 396
- default constructor, 107
- Defense Advanced Research Projects Agency (DARPA), 878, 880
- degenerate binary search trees, 451
- degree of a node, 392
- dense graphs, 594
- dependency, UML class diagrams, 72–73
- deployment diagrams, UML, 67
- depth-first spanning trees, 576
- depth-first traversal of graphs, 568, 569, 573–574
- deque, behavior, 378
- dequeue, 370
- `dequeue()` method, 488–489, 774
- descendants, nodes, 393
- design
 - object-oriented, 11–12, 57–80
 - top-down, 10–11
- design phase of software life cycle, 9–14, 58
 - identifying classes, 60–65
- destination, shortest path problem, 583
- destructors, 47, 107
- dialog boxes, pop-up, 814
- `difference()` method, 518
 - complexity, 516, 519
- difference operator, 512
- Dijkstra, Edsger, 357
- Dijkstra's algorithm, 584–593
- directed acyclic graphs, 562

- directed graphs, 552–553
 - acyclic, 562
- discrete event simulators, 181
- divide-and-conquer operations, 9
- DNA, studying, 722–723
- DNS (Domain Name System), 886, 887
- documentation
 - technical, 27–29
 - user, 27
- documentation and support phase
 - of software life cycle, 26–29
- Dog** class, 644–647
- Domain Name System (DNS), 886, 887
- dot (.), invoking methods, 125
- doubly linked lists, 330–342
- Dylan, 45
- Dynabook, 82

E

- edge(s)
 - graphs, 552
 - trees, 391
- Edge** class, 553–560
- efficiency of algorithms, 246–249
- embedded systems, 18
- “The Emperor’s Old Clothes” (Hoare), 765
- empirical testing, 24
- empty lists, 321
- empty()** method, 348
- empty sets, 513
- encapsulation, 40
- enqueue()** method, 774
- entrySet()** method, 638
- Environment.java** class, implementation, 197–199
- equal sign (=), equality operator, 124–125

- equality, 48
- equality operator (==), 124–125
- equals()** method, 153–154, 405, 407, 412–413, 617, 644–646, 655
- errors
 - cost of, 17
 - exception handling, 20
 - off-by-one, 321
 - reasons for, 6–8
- event(s), 807, 835
 - mouse, 847–851
- event handlers, 836
- event-driven programming, 807, 836
- event-listener interfaces, 836–839
- exabytes, 461
- exception(s), 682–704
 - checked, 686, 688, 689
 - design guidelines and examples, 700–701
 - generating (throwing), 690–694
 - handling. *See* exception handling
 - ignoring, 702
 - representing, 686–690
 - rethrowing, 702–703
 - unchecked, 686–687, 689
- Exception** class, 688–690
- exception classes (types), 686
- exception handlers, 686
- exception handling, 20, 694–698
 - catch blocks, 694, 696–697
 - declaring exceptions in method headers, 698
 - design guidelines and examples, 701–704
 - finally blocks, 695–697
 - returning null reference, 683–684

- state variables, 684–685
- System.exit()**, 682–683
 - try blocks, 694–695, 697
- exception objects, 686
- exhaustive testing, 24–25
- exit()** method, 742
- expandHeap()** method, 480–482
- exponential algorithms, 265–266
- extending classes, 135–143
 - accessibility rules, 137
- extension form of inheritance, 53, 54–55
- external iterators, 622–624
- extracting ASCII strings from datagrams, 909–910
- extracting integers from datagrams, 907–908
- extreme programming (XP), 59

F

- fast-fail iterators, 624
- Federalist Papers* (Hamilton and Madison), 918
- File Transfer Protocol (FTP), 879
- FilePermission** class, 925
- fill()** method, 161–163
- final variables, 97–98
- finalize()** method, 152
- finally blocks, 695–697
- find()** method, 405, 413–414
- first element of linear data structure, 288
- first()** method, 295
 - time complexity, 328
- flow layout, 821–822
- flow path, 24
- for-each loops, 623
- formal parameters, 102

- formal specification languages, 8
- forward Javadoc comments,
 - Javadoc comments, 93
- forward slash (/), Javadoc
 - comments, 27
- four-function calculator graphical
 - user interface example, 864–868
- frameworks, 613
- `front()` method, 488–489
- FTP (File Transfer Protocol), 879
- full binary search trees, 452
- `full()` method, 348
- function(s), 100
- functional programming languages,
 - 346–347
- Furnace** class, implementation,
 - 199–203

G

- Gage, John, 881
- “A Galactic Network”
 - (Licklider), 878
- game trees, 396–397
- garbage, 107
- garbage collection, 107
- Garrison, William Lloyd, 918
- Gates, Bill, 726–727
- GenBank, 722–723
- general trees, 391–393
 - binary tree representation,
 - 414–416
- generalization. *See also* inheritance
 - UML class diagrams, 73–74
- general-purpose
 - implementations, 651
- generics, 154–164
 - inheritance, 163–164
- genetic data banks, 722–723
- `get()` method, 297, 299, 405,
 - 411–412
 - time complexity, 328

- `get(position)` method, time
 - complexity, 328
- `getInputStream()`
 - method, 897
- `getLength()` method, 912
- `getOutputStream()`
 - method, 898
- `getSmallest()` method,
 - 483–485, 487
- gigabytes, 461
- gigaflop computer, 264
- “Go to Statement Considered
 - Harmful” (Dijkstra), 357
- good approximations, importance,
 - 491–492
- Google, 461, 880
- Gosling, James, 17, 44, 90–91
- grammar, 390
- graph(s), 290–291, 551–599
 - acyclic, directed, 562
 - adjacent and incident
 - nodes, 552
 - connected, 561
 - cycles, 560–561
 - definition, 551–552
 - dense, 594
 - directed, 552, 561–562
 - Edge** class, 554–560
 - edges, 552, 553
 - heads, 552
 - implementation, 593–599
 - nodes, 552
 - operations, 563–591
 - paths, 560–561
 - space needs for representa-
 - tion techniques, 597
 - sparse, 594
 - tails, 552
 - Vertex** class, 554–560
- graphical user interfaces (GUIs),
 - 21–22, 805–868
 - components, 827–834

- containers, 811–816
- event listeners, 836–847
- events, 835
- examples, 853–868
- handling layouts yourself,
 - 817–819
- layout managers, 816–827
- mouse events, 847–851
- Swing, 807, 809–811

- graphics design, 820
- greedy techniques, 577
- grid layout, 822–824
- GUIs. *See* graphical user
 - interfaces (GUIs)

H

- Hamilton, Alexander, 918
- Hamiltonian cycles, 561
- has-a relationship, 51
- hash functions, 529
- hash tables, 529
- hash values, 529
- `hashCode()` method, 152
- `hashCode()` method, 655
- hashing, 256, 529–551, 654–655
 - collisions
 - open addressing, 535–543
- hashing functions, 530–531
 - perfect, 531
- HashMap** implementation of
 - maps, 665
- HashSet** class, 652–656
- `hasLeftChild()` method,
 - 405, 409
- `hasNext()` method, 724–725
- `hasParent()` method, 405, 409
- `hasRightChild()` method,
 - 405, 409
- HDTV, 461
- heads, graphs, 552

- heap(s), 474–489
 - application, 485–489
 - definition, 474–477
 - implementation using one-dimensional arrays, 477–485
 - level order, 477
 - order property, 474
 - structure property, 475
 - heap sorts, 485–488
 - building phase, 485–486
 - removing phase, 485, 486–487
 - heapforms, 477
 - HeatingSimulation.java**
 - class, implementation, 218–223
 - heavyweight components, 809
 - height()** method, 405, 413
 - heuristics, 266
 - hierarchical data structures, 289–290, 387–492. *See also*
 - heap(s); tree(s)
 - importance of good approximations, 491–492
 - Hoare, Charles Antony Richard, 224, 765, 795
 - home heating system example of
 - class diagram, 77
 - Hopper, Grace Murray, 650, 671
 - URLConnection** class, 917
 - hypothesis testing, 533–534
- I**
- IBM, 30–31, 264
 - identity, 47, 114–128
 - ignoring exceptions, 702
 - imaging, processing demands, 461
 - implementation classes, 650–655
 - ArrayList**, 657–660
 - HashSet**, 652–656
 - LinkedList**, 660–664
 - maps, 665
 - TreeSet**, 656
 - implementation of interfaces, 147–149
 - implementation phase of software life cycle, 17–23, 58
 - implementations, general-purpose, 651
 - importance of good approximations, 491–492
 - incident nodes, 552
 - indefinite blocking, 748, 793
 - indexes, 294
 - InetAddress** class, 892–893
 - information field, lists, 293
 - inheritance, 50–56, 132, 133–154
 - changing hierarchy, 141
 - forms, 52–55. *See also* combination form of inheritance; extension form of inheritance; limitation form of inheritance; specialization form of inheritance; specification form of inheritance
 - generic types, 163–164
 - multiple, 55, 135
 - object-oriented software
 - development case study, 189–192
 - single, 55
 - specialization form, 145
 - threads, 740
 - inheritances, 807
 - initial capacity of has table, 653
 - initialization expressions, 96–97
 - initializing variables, 118
 - inner classes, 91
 - inOrder()** method, 406, 407, 414
 - inorder traversal, 401, 402–403
 - input/output (I/O), 45
 - input/output specifications, 5–6
 - InputStream**, 709–710
 - insertHeapNode()** method, 479–482, 487
 - insertLeft()** method, 406, 411
 - insertRight()** method, 406, 411
 - instance methods, 105
 - invoking, 105–106
 - instance variables, 93–94
 - private, 93–94, 315
 - public, 93–94
 - synchronization, 770–772
 - instanceof operator, 844
 - instantiating objects, 115
 - instantiation, objects, 49–50
 - integer(s), 50
 - extracting from datagrams, 907–908
 - integer priority, threads, 748
 - interfaces, 146–149. *See also*
 - graphical user interfaces (GUIs)
 - implementation, 147–149, 203–204
 - List** data structure, 299–304
 - lists, 337
 - object-oriented software
 - development case study, 189–192
 - specifying for components, 9
 - stacks, 348–350
 - internal iterators, 621–622
 - internal nodes
 - of hierarchical data structure, 289
 - trees, 391, 392
 - Internet, growth, 887
 - Internet Protocol (IP), 886
 - interpreted languages, 165, 166
 - interpreters, 165, 166

- interpreting programs, 165–168
- `intersection()` method, 518
 - complexity, 516, 519
- intersections, 512
- `IntPriorityQueue` class, 634–635
- inventory program example, 60–63
- invoking methods, 46, 115–116, 125–126
- I/O (input/output), 45
- IP (Internet Protocol), 886
- iPod, 852
- is-a relationship, 51
- `isAlive()` method, 739, 750
- `isEmpty()` method, 513
 - complexity, 516, 519
- `isOffList()` method, 405
- `isOnList()` method, 297
- `isValid()` method, 409–410
- iteration, 289
- iteration markers, UML sequence diagrams, 79
- iterator(s), 404
 - external, 622–624
 - fast-fail, 624
 - internal, 621–622
- `Iterator` interface, 621–624
- iTunes, 852

J

- JARs (Java Archive Files), 917
- `JarURLConnection` class, 917
- Java
 - development, 17–18
 - features, 20
 - library of classes, 19
 - rise in favor, 44
 - as true object-oriented language, 18
- Java Archive Files (JARs), 917

- Java Collection Framework, 32, 599, 611–670
 - algorithms, 666–670
 - `Collection` interface, 616–624
 - collections, 615
 - implementation classes, 650–655
 - `List` interface, 627–636
 - `Map` interface, 636–640
 - `Set` interface, 624–627
 - sorted interfaces, 641–649
- Java Development Kit (JDK), 167–168
- Java Foundation Classes (JFCs), 809
- Java Virtual Machine (JVM), 166–168
 - termination, 742
- Javadoc comments, 27–29, 93
- `java.io` package, 706–709
- `java.net`, 23
- `JButton` component, 828, 830, 837
- `JCheckBox` component, 829, 830, 837
- `JComboBox` component, 828, 830, 837
- `JDialog` class, 814
- JDK (Java Development Kit), 167–168
- Jeffries, Ron, 59
- JFCs (Java Foundation Classes), 809
- `JFrame` class, 812–813
- `JFrame` component, 837
- `JLabel` component, 829, 837
- `JList` component, 829
- `JMenuBar` component, 829
- `JMenuItem` component, 837
- Jobs, Steve, 808, 852
- `join()` method, 757–758

- `JPanel` class, 813–814
- `JScrollPane` class, 815–816
- `JTextArea` component, 829, 837
- `TextField` component, 829, 830, 837
- JVM (Java Virtual Machine), 166–168

K

- Kahn, Robert E., 879–880
- Kasparov, Gary, 396
- Kay, Alan, 43, 82, 808
- key(s), maps, 523
- key field, 292, 521
- key-access tables, 636–640. *See also* map(s)
- `keySet()` method, 638
- Kleinrock, Leonard, 878, 879
- Knuth, Donald, 490
- Kruskal's algorithm, 577–583

L

- Lakeside Programmers Group, 727
- lambda calculus, 346
- Lampson, Butler, 808
- last element, of linear data structure, 288
- `last()` method, 295
 - time complexity, 328
- layout managers, 820–827
- layout problem, 817
- leaves
 - hierarchical data structure, 289
 - trees, 391
- levels, nodes, 393
- Levy, David, 396
- The Liberator* (Garrison), 918
- library system example of class diagram, 75–76

- Licklider, J. C. R., 878
- lightweight components, 809
- lightweight processes. *See* `thread(s)`
- limitation form of inheritance, 53, 55
- linear algorithm, 253–254
- linear data structures, 288–289, 293–378. *See also* `list(s)`; `queues`; `stack(s)`
 - behavior, 378
- linked `list(s)`, 16
- linked list-based implementation
 - binary search trees, 453–459
 - maps, 527–529
 - queues, 367–369
 - stacks, 354–356, 691–692
- `LinkedList` class, 660–664
- `LinkedListNode` class, 315–316
- Linux, 193
- LISP, 346–347
- `list(s)`, 293–347
 - adjacency, 593, 595–599
 - behavior, 378
 - circular, singly linked, 342–345
 - `data` field, 293
 - empty, 321
 - first node, 321
 - implementation, 305–345, 657–664
 - information field, 293
 - interfaces, 337
 - last node, 321
 - linked. *See* linked `list(s)`
 - neighbor, 595–596
 - `next` field, 293
 - nodes, 293, 595
 - operations, 294–305
 - removing duplicates, 633
 - singly linked, 330

- `List` interface, 627–636
- listener registration method, 839–840
- `ListIterator` class, 663–664
- `ListIterator` interface, 630–633
- load factor, 653
 - open addressing, 541
- local variables, 94
- locks, 765–778
 - implementing buffers, 781–783
 - mutually exclusive (mutex), 765–766
 - static, 772
- lower order algorithms, 247
- LucasFilm, 852

M

- Mac OS X, 852
- Macintosh computer, 852
- Macintosh operating system, 806
- Madison, James, 918
- `main()` method, 111–112
- maintenance phase of software life cycle, 29
- `map(s)`, 292, 521–551
 - definition, 521–522
 - hashing, 529–551
 - implementation, 526–529
 - Java Collection Framework, 665
 - keys, 523
 - operations, 522–526
 - sorted, 647–649
- `Map` interface, 636–640
- Mark I computer, 671
- Mark II computer, 650, 671
- Mark III computer, 671
- marking nodes in graphs, 569

- Matsumoto, Yukihiro, 45
- `max()` method, 667
- McCarthy, John, 346
- megaflop computer, 264
- memory, garbage collection, 107
- memory leaks, 107
- Mendel, Gregor, 722
- `menu(s)`, 833
- menu bar, 812, 833
- menu items, 833
- merge sort algorithm, analysis, 274–277
- Merholz, Peter, 918
- messages, 46
- METAFONT, 490
- methods, 12, 14, 46, 99–113. *See also* specific methods
 - abstract, 144
 - arguments, 102
 - class, 107–113
 - constructors, 106–107
 - destructors, 107
 - formal parameters, 102
 - functions, 100
 - instance. *See* instance methods
 - invoking, 46, 115–116, 125–126
 - mutators, 101–102, 103
 - overloaded, 103
 - procedures, 101
 - signature, 102, 112
 - stable, 667
 - static, 113, 668–669
 - subclass overriding of super-class methods, 140–143
 - synchronized, 774–778
 - time complexity, 328–330
 - types, 47
 - wrapper, 668
- Microsoft, 727, 808
- `min()` method, 667

- minimum height binary search trees, 459–460
- minimum spanning trees (MSTs), 576–583
- modal containers, 814
- modern software development, definition, 2
- mouse events, 847–851
- `MouseListener` class, 849–851
- Mozilla, size, 4
- MSTs (minimum spanning trees), 576–583
- multigraphs, 553
- multiline comments, 93
- multiple inheritance, 55, 135
- multiplicative congruency method, 534
- multiplicity, UML class diagrams, 71–72
- multitasking, 734. *See also* thread(s)
 - cooperative, 749–759
- multitasking operating systems, 735
- multithreaded servers, 902–905
- multithreading, 21
- mutators, 47, 101–102, 103
 - object-oriented software development case study, 185–186
- mutually exclusive (mutex) locks, 765–766
- The Mythical Man-Month* (Brooks), 31
- `MyWindowListener` class, 844–845

N

- naïve set theory, 509
- name equivalent variables, 48, 49, 119

- NASA, 614
- National Center for Atmospheric Research (NCAR), 244–245
- National Science Foundation (NSF), 880
- natural comparison method, 642
- natural language, 7–8
- natural ordering of objects, 641–647
 - sorted interfaces, 641–647
- navigability information, UML class diagrams, 71–72
- NCAR (National Center for Atmospheric Research), 244–245
- neighbor lists, 595–596
- neighbor nodes, 595
- neighbors of nodes in graphs, 569–571
- nested classes, 91
- Network Link layer of OSI model, 885–886
- networking, 22–23, 877–927
 - historical background, 878–882
 - Java, 891–892
 - security, 921–925
 - socket classes, 891, 892–916
 - TCP/IP, 882–890
 - URL classes, 891–892, 917–921
- NeXT Computer, 852
- `next` field, lists, 293
- `next()` method, 296, 631, 632, 663–664, 668, 724–725
 - time complexity, 328
- No Silver Bullet* (Brooks), 31
- node(s)
 - ancestors, 393
 - current, 404
 - degree, 392
 - descendants, 393

- graphs, 552
- internal, 391, 392
- levels, 393
- lists, 293
- neighbor, 595
- parents, children, and siblings, 393
- paths, 393

- node lists, 595
- noncooperative threads, 749
- nondeterministic programs, 744
- nonpreemptive schedulers, 747
- nonterminal symbols, 389–390
- `notify()` method, 783–786
- `notifyAll()` method, 791
- NSF (National Science Foundation), 880
- NSFNet, 880–881
- null hypothesis, 533
- null pointer, 13
- null references, exception handling, 683–684
- Nygaard, Kristen, 43, 224

O

- Oak, 44, 90–91
- object(s), 40, 45–49
 - behavior, 46
 - definition, 46
 - identity, 47
 - instantiating, 115
 - instantiation, 49–50
 - methods. *See* methods; *specific methods*
 - state, 46
- `Object` class, 151–154
- object diagrams, UML, 67
- object-oriented design, 11–12, 57–80
 - development, 18

- object-oriented language, UML.
 - See* Unified Modeling Language (UML)
- object-oriented programming, 45–56
 - classes, 49–50
 - inheritance, 50–56
 - objects, 45–49
- object-oriented software development case study, 175–230
 - implementation, 197–223
 - problem requirements, 176–178
 - problem specification document, 178–183
 - software design testing, 224–229
- object-oriented view, 41–42
- off-by-one errors, 321
- oldest child/next sibling algorithm, 415
- open addressing, 535–543
- Open Shortest Path First (OSPF), 593
- open source movement, 193
- Open Systems Interconnection (OSI) reference model, 884–890
- opening a stream, 705
- order property, heaps, 474
- OS/360 operating system, 30–31
- OSI (Open Systems Interconnection) reference model, 884–890
- OSPF (Open Shortest Path First), 593
- overloaded methods, 103

P

- package(s), 91
- package scope, 92

- packet switching, 878
- pair programming, 113–114
- parallel processing, 21
- @param tag, 27
 - Javadoc comments, 93
- parameter passing, 121–123
 - by value, 121
- parameter-passing mechanisms, 46–47
- parent, nodes, 393
- parent classes, 50
- parse trees, 388–391
- paths
 - graphs, 560–561
 - nodes, 393
 - simple, 393
- perfect hashing functions, 531
- permission(s), 924–925
- Permission class, 924
- petabytes, 461
- petaflop(s), 737
- petaflop computer, 264
- Physical layer of OSI model, 884
- Pixar, 852
- pixel numbering, 817–819
- plus sign (+), addition operator, 103
- policy configuration files, 923
- POLS (Principle of Least Surprise), 45
- polymorphism, 149–151
- polynomial algorithms, 265
- popping stacks, 347
- pop-up dialog boxes, 814
- ports, 888–889
 - well-known, 889
- position numbers, 294
- postconditions, specifying for components, 9
- Postel, John, 879
- postOrder() method, 406, 407, 414

- postorder traversal, 401, 402–403
- powerful computer systems, 737
- preconditions, specifying for components, 9
- predecessor, of linear data structure elements, 288
- preemptive schedulers, 747
- preOrder() method, 406, 407, 414
- preorder traversal, 399–401
- Presentation layer of OSI model, 890
- previous() method, 296, 320–321, 631, 632, 663–664
 - time complexity, 328
- primitive data type, 288
- Principle of Least Surprise (POLS), 45
- priorities, 370
 - threads, 747–749
- priority queues, 370, 634–635
 - behavior, 378
- private instance variables, 93–94, 315
- problem specification documents, 6, 7, 57
 - object-oriented software development case study, 178–183
- problem specification phase of software life cycle, 5–9, 57–58
- procedure-oriented view, 42
- procedures, 41, 101
- processes, 735
 - lightweight. *See* thread(s)
- processing streams, 708, 709
- program(s), size, 2–3
- program maintenance, 29
- program verification, 23–24
- properties, objects, 46
- protocol(s), 882–884

“A Protocol for Packet Network Interconnection” (Cerf and Kahn), 879
protocol identifier, 919
`prune()` method, 406–407, 412
public instance variables, 93–94
pushing values into stacks, 347–348
`putAll()` method, 638
Python, 45, 223

Q

quadratic algorithm, 259
quantum computing, 143–144
qubits, 143–144
queues, 358–378
 behavior, 378
 implementation, 363–369
 operations, 358–362
 priority, 370, 634–635
 ready, 735, 746
 thread-safe, 774
 wait, 746
 waiting, 735
Quicksort, 795

R

radiation overdoses, 764–765
rapid prototyping, 6–7
`read()` method, 709–710
`Reader`, 709, 710
reading character data from the
 keyboard using streams, 719–720
reading data from files, streams
 using, 709–715
read-only collections, 668–670
read-only wrappers, 668
ready queue, 735, 746
ready state, 745

real-time programs, 464
rebalancing algorithm, 465
`receive()` method, 912
recurrence relation, 271–272
recursive algorithms, 267–277
 analysis, 270–277
recursive case, 268, 271–272
recursive method calls, 267
red-black trees, 460, 464, 465–474
reference variables, 47–49, 117–118
reference-based implementation, binary trees, 416–431
reference-based implementation of lists, 314–330
registration, 839
`remove()` method, 296, 298, 322, 779–780, 783
 complexity, 516, 519
 Iterator interface, 623–624
 time complexity, 328
`remove(position)` method, time complexity, 328
`removeAll()` method, 618
`removeDuplicates()` method, 626–627
repeated substitutions technique, 272, 276
Request for Comments, 884
requirements phase of software life cycle, 57
resource identifier, 919
 reading from, 920–921
 representing, 919–920
`retainAll()` method, 618
rethrowing exceptions, 702–703
return statement, 351
`@return` tag, 27
 Javadoc comments, 93
reusability, 614–615

rings, 343
Roberts, Larry, 879
`Room` class, implementation, 204–210
roots
 hierarchical data structure, 289
 trees, 391
Ruby, 44–45
`run()` method, 739–741, 742
run state, 745
`Runnable` interface, 740–741
run-time binding, 149–151
run-time stacks, 351

S

sandbox, 922–923
Sanger, Fred, 722
`Scanner` class, 723–725
scatter functions, 532
schedulers, 735
 nonpreemptive, 747
 preemptive, 747
search trees, binary. *See* binary search trees (BSTs)
security, 23, 921–925
`SecurityManager` class, 923
selection sort algorithm, 259–260
Self, 45
`send()` method, 911–912
sequence diagrams, UML, 67
sequential search algorithm, 252–253
serial execution of threads, 760–761
servers, 895
 multithreaded, 902–905
`ServerSocket` class, 896–897
Session layer of OSI model, 890
set(s), 291–292, 508–519
 base type, 516

- cardinality, 513
- empty, 513
- implementation, 514–519, 652–656
- intersections, 512
- method complexity, 515–516, 519
- naïve set theory, 509
- operations, 509–513
- sorted, 647–649
- unions, 512
- Set** interface, 624–627
- set()** method, 297, 299, 406, 412
- set(element, position)** method, time complexity, 328
- set(element)** method, time complexity, 328
- set theory, 521
- setDaemon()** method, 742
- setDefaultCloseOperation()** method, 847
- shallow copies, 123
- shallow copy, 118–119
- Shape** class, 133
- Shell, Donald, 795
- Shell sorts, 795
- Shor, Peter, 144
- shortest path problem, 583–591
- siblings, nodes, 393
- signature, methods, 102, 112
- signer of a class, 924
- simple data type, 288
- Simple Mail Transfer Protocol (SMTP), 883
- simple path, 393
 - graphs, 560
- Simula, 43, 224
- simulation clocks, 184
- single inheritance, 55
- singly linked circular lists, 342–345
- singly linked lists, 330
- sinks, streams, 705
- size
 - programs, 2–3
 - software packages, 4
- size()** method, 298, 405, 413
 - complexity, 516, 519
 - time complexity, 328
- slash (/), Javadoc comments, 27, 93
- sleep()** method, 751–755
- Smalltalk, 43, 46, 82
- SMTP (Simple Mail Transfer Protocol), 883
- socket(s), 891
- Socket** class, 897
- socket class(es), 891
 - reliable communication, 895–905
 - representing addresses in Java, 892–895
 - representing diagrams, 905–909
 - unreliable communication, 909–916
- software design documents, 9–10
- software engineering, 30
 - father of, 30–31
- software errors. *See* errors
- software life cycle, 4–29
 - algorithm and data structure selection phase, 14–17
 - coding and debugging phase, 17–23, 58
 - documentation and support phase, 26–29
 - maintenance phase, 29
 - problem specification phase, 5–9, 57
 - program design phase, 9–14, 58, 60–65
 - testing and verification phase, 23–26
- software packages, size, 4
- software productivity, 19
- software requirements documents, 7, 57
- software reuse, 19
- sort()** method, 667, 668
- sorted interfaces, 641–649
 - natural ordering of objects, 641–647
 - sorted sets and sorted maps, 647–649
- sorted maps, 647–649
- sorted sets, 647–649
- SortedMap** interface, 649
- SortedSet** interface, 647–649
- sources
 - shortest path problem, 583
 - streams, 705
- source/sink streams, 707–708
- spanning trees, 574–583
 - breadth-first, 576
 - depth-first, 576
 - minimum, 576–583
- sparse graphs, 594
- special cases, 321–322
- specialization form of inheritance, 53, 54, 145
 - object-oriented software development case study, 189–190
- specification form of inheritance, 53–54
 - object-oriented software development case study, 192
- square brackets ([]), iteration market, 79
- Square** class, 128–132
- Square** subclass, 135–136
- Squeak, 45
- stable methods, 667

- stable sorting algorithm, 277
- stack(s), 347–356
 - behavior, 378
 - implementation, 352–356
 - interfaces, 348–350
 - linked list-based implementation, 691–692
 - operations, 347–351
 - popping, 347
 - pushing values into, 347–348
 - run-time, 351
 - top, 347
- Stack** class, 356
- stack traces, 693
- StackException** class, 689–690
- StackOverflow** class, 690
- StackUnderflow** class, 690
- standard error streams, 718
- standard input streams, 718
- standard output streams, 718
- Standard Template Library (STL), 613
- starvation, 748, 793
- state(s), 93–99
 - instance variables, 93–94
 - local variables, 94
 - object-oriented software development case study, 184–185
 - objects, 46
- state diagrams, UML, 67
- state variables, exception handling, 684–685
- static class methods, 666
- static locks, 772
- static methods, 113, 668–669
- static synchronized methods, 777–778
- static variables, 95–96

- status value, traversal of graphs, 569–571
- STL (Standard Template Library), 613
- Strassen's method, 263
- streams, 20–21, 704–725
 - algorithms for using, 705
 - buffered, 712–715
 - closing, 705
 - java.io** package, 706–709
 - opening, 705
 - processing, 708, 709
 - Scanner** class, 723–725
 - source/sink, 707–708
 - standard error, 718
 - standard input, 718
 - standard output, 718
 - using, 709–722
 - wrapping, 708
- Stroustrup, Bjarne, 43
- structured programming, 357
- stubs, 844
- subclasses, 50–51, 133, 134
 - general form, 135
 - overriding superclass methods, 140–143
- subgraphs, connected, 561
- subList()** method, 634
- subset()** method, 513, 518
 - complexity, 516, 519
- subtrees, 391
- successors, of linear data structure elements, 288
- sumInput()** method, 720
- Sun Microsystems, 43–44
- super** keyword, 137–139
- superclasses, 50, 133
 - subclass overriding of methods, 140–143
- Swing, 807, 809–811
- symbolic constants, 98–99

- synchronization
 - instance variables, 770–772
 - threads, 759–793
- synchronized blocks, 766–767
- synchronized buffers, 783
- synchronized methods, 774–778
- static, 777–778
- System.exit()**, exception handling, 682–683

T

- tables, 292
- tags
 - Javadoc comments, 27, 93
 - traversal of graphs, 569–571
- tails, graphs, 552
- taxonomies, 288
- TCP (Transmission Control Protocol), 888–889
- TCP/IP, 878, 882–890
 - OSI model, 884–890
 - protocols, 882–884
- technical documentation, 27–29
- techniques, inheritance as, 56
- Technorati, 918
- terabytes, 461
- teraflop(s), 737
- teraflop computer, 264
- terminal symbols, 389
- testing
 - acceptance, 229
 - black box (acceptance; beta), 26
 - clear box (alpha), 26
 - empirical, 24
 - exhaustive, 24–25
 - integration, 25–26
 - object-oriented software development case study, 224–229
 - unit, 24, 224–229

- testing and verification phase of
 - software life cycle, 23–26
 - `TeX`, 490
 - Therac-25 accidents, 764–765
 - `Thermostat` class, implementa-
 - tion, 211–215
 - 32 bits, 886
 - this object, 447
 - `this` keyword, 125–128
 - `this` variable, 773–774
 - `thread(s)`, 21, 733–794
 - associating names with
 - threads, 742–743
 - cooperative multitasking,
 - 750–759
 - crating, 738–744
 - daemon, 742
 - inheritance, 740
 - multiple, 743
 - noncooperative, 749
 - `Runnable` interface,
 - 740–741
 - scheduling and thread prior-
 - ities, 745–750
 - synchronization, 759–793
 - `Thread` class, 738
 - thread-safe queues, 774
 - throwing exceptions, 690–694
 - throws clauses, 698
 - time complexity, methods,
 - 328–330
 - time quantum, 747
 - time-sharing operating
 - systems, 735
 - time-space trade-off, 549
 - Tiobe Programming Community
 - Index, 91
 - `toArray()` method, complexity,
 - 516, 519
 - `ToLeftChild()` method,
 - 405, 410
 - top of a stack, 347
 - `toParent()` method, 405, 410
 - top-down design, 10–11
 - `toRightChild()` method, 405,
 - 410–411
 - `toRoot()` method, 405
 - `toString()` method, 152–153
 - object-oriented software
 - development case
 - study, 186
 - transforming data using streams,
 - 717–718
 - Transmission Control Protocol
 - (TCP), 888–889
 - Transport layer of OSI model,
 - 887–889
 - traveling salesman problem, 265
 - traversal
 - graphs, 568–574
 - lists, 664
 - `tree(s)`, 290, 387–474
 - binary. *See* binary trees
 - edges, 391
 - general, 391–393
 - height, 393
 - internal nodes, 391, 392
 - leaves, 391
 - nonterminal symbols,
 - 389–390
 - parse, 388–391
 - roots, 391
 - spanning, 574–583
 - subtrees, 391
 - terminal symbols, 389
 - tree traversal, 399–404
 - infix representation, 402
 - inorder, 401, 402–403
 - postfix representation, 402
 - postorder, 401, 402–403
 - prefix representation,
 - 401–402
 - preorder, 399–401
 - `TreeMap` implementation of
 - maps, 665
 - `TreeSet` class, 656
 - try blocks, 694–695, 697
 - tuples, 521
 - Turing, Alan M., 520–521
 - Turing machines, 520–521
 - 2-tuples, 521
 - type-safe languages, 157
- ## U
- UDP (User Datagram Protocol),
 - 888–889, 909–910
 - UDP day/time client, 914–916
 - UDP day/time server, 912–914
 - ULTRA, 264
 - UML. *See* Unified Modeling
 - Language (UML)
 - unchecked exceptions,
 - 686–687, 689
 - undirected graphs, 552–553
 - Unicon, 45
 - Unified Modeling Language
 - (UML), 8, 65–80
 - class diagrams, 68–77
 - diagram types, 66–67
 - sequence diagrams, 77–80
 - views, 66
 - `union()` method, 518
 - complexity, 516, 519
 - unions, 512
 - unit testing, 24, 224–229
 - UNIVAC I, 671
 - Universal Resource Locators
 - (URLs), 919–921
 - unmodifiable wrappers, 668
 - unreliable communication,
 - 909–916
 - unweighted edges, graphs, 553
 - URL(s) (Universal Resource
 - Locators), 919–921

- URL classes, 891–892, 917–921
 - reading from URLs, 920–921
 - representing URLs, 919–920
- `URLClassLoader` class, 917
- use-case diagrams, UML, 67
- User Datagram Protocol (UDP), 888–889, 909–910
- user documentation, 27

V

- value(s)
 - objects, 46
 - parameters passed by, 121
- value field, 292, 521
- variables
 - automatic, 73
 - class, 95–96
 - content equivalent, 48–49
 - final, 97–98
 - initializing, 118
 - name equivalent, 48, 49, 119
 - reference, 47–49, 117–118
 - static, 95–96

- Venn diagrams, 512
- verification, 23–24
- `@version` tag, 27
 - Javadoc comments, 93
- `Vertex` class, 553–560
- vertices, graphs, 552
- veterinary system example of class diagram, 75
- viewports, 815
- views, UML, 66
- `visit()` method, 407
- von Neumann, John, 143
- von Neumann architecture, 41, 143

W

- wait loops, busy, 751
- `wait()` method, 783, 784–786, 791
- wait queue, 746
- wait state, 745
- waiting queue, 735
- Watson, Thomas, 722
- weighted edges, graphs, 553
- well-known ports, 889

- Wikipedia, 193
- WORA (Write Once, Run Anywhere), 90–91
- worst-case behavior of algorithms, 149–151
- Wozniak, Steve, 852
- wrapper(s)
 - read-only, 668
 - unmodifiable, 668
- wrapper methods, 668
- wrapping streams, 708
- Write Once, Run Anywhere (WORA), 90–91

X

- Xerox Alto, 808, 852
- Xerox PARC, 808
- Xerox STAR, 808
- XP (extreme programming), 59

Y

- `yield()` method, 756–757