Java

The Java® Language Specification

Java SE 7 Edition

James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley

ORACLE

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

The Java® Language Specification

Java SE 7 Edition

This page intentionally left blank

The Java® Language Specification

Java SE 7 Edition

James Gosling
Bill Joy
Guy Steele
Gilad Bracha
Alex Buckley

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Copyright © 1997, 2013, Oracle and/or its affiliates. All rights reserved.
500 Oracle Parkway, Redwood City, California 94065, U.S.A.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document, except as specified in the Limited License Grant herein at Appendix A. This document is subject to the Limited License Grant included herein as Appendix A, and may otherwise not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact U.S. Corporate and Government Sales, (800) 382-3419, `corpsales@pearsontechgroup.com`. For sales outside the United States, please contact International Sales, `international@pearson.com`.

Visit us on the Web: `informit.com/aw`

Library of Congress Control Number: 2012953694
ISBN-13: 978-0-13-326022-9
ISBN-10: 0-13-326022-4

Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

The Specification provided herein is provided to you only under the Limited License Grant included herein as Appendix A. Please see Appendix A.

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana. First printing, January 2013.

Table of Contents

Preface to the Java SE 7 Edition xvii

Preface to the Third Edition xix

Preface to the Second Edition xxiii

Preface to the First Edition xxv

1 Introduction 1

1.1 Organization of the Specification 2
1.2 Example Programs 5
1.3 Notation 6
1.4 Relationship to Predefined Classes and Interfaces 6
1.5 References 7

2 Grammars 9

2.1 Context-Free Grammars 9
2.2 The Lexical Grammar 9
2.3 The Syntactic Grammar 10
2.4 Grammar Notation 10

3 Lexical Structure 15

3.1 Unicode 15
3.2 Lexical Translations 16
3.3 Unicode Escapes 17
3.4 Line Terminators 18
3.5 Input Elements and Tokens 19
3.6 White Space 21
3.7 Comments 21
3.8 Identifiers 23
3.9 Keywords 24
3.10 Literals 25
  3.10.1 Integer Literals 25
  3.10.2 Floating-Point Literals 32
  3.10.3 Boolean Literals 35
  3.10.4 Character Literals 35
  3.10.5 String Literals 36
  3.10.6 Escape Sequences for Character and String Literals 38

3.10.7 The Null Literal 39
3.11 Separators 40
3.12 Operators 40

4 Types, Values, and Variables 41

4.1 The Kinds of Types and Values 41
4.2 Primitive Types and Values 42
4.2.1 Integral Types and Values 43
4.2.2 Integer Operations 43
4.2.3 Floating-Point Types, Formats, and Values 45
4.2.4 Floating-Point Operations 48
4.2.5 The `boolean` Type and boolean Values 51
4.3 Reference Types and Values 52
4.3.1 Objects 54
4.3.2 The Class `Object` 56
4.3.3 The Class `String` 57
4.3.4 When Reference Types Are the Same 57
4.4 Type Variables 58
4.5 Parameterized Types 60
4.5.1 Type Arguments and Wildcards 61
4.5.2 Members and Constructors of Parameterized Types 64
4.6 Type Erasure 65
4.7 Reifiable Types 66
4.8 Raw Types 67
4.9 Intersection Types 71
4.10 Subtyping 72
4.10.1 Subtyping among Primitive Types 72
4.10.2 Subtyping among Class and Interface Types 73
4.10.3 Subtyping among Array Types 73
4.11 Where Types Are Used 74
4.12 Variables 75
4.12.1 Variables of Primitive Type 76
4.12.2 Variables of Reference Type 76
4.12.3 Kinds of Variables 78
4.12.4 `final` Variables 80
4.12.5 Initial Values of Variables 81
4.12.6 Types, Classes, and Interfaces 82

5 Conversions and Promotions 85

5.1 Kinds of Conversion 88
5.1.1 Identity Conversion 88
5.1.2 Widening Primitive Conversion 88
5.1.3 Narrowing Primitive Conversion 90
5.1.4 Widening and Narrowing Primitive Conversion 93
5.1.5 Widening Reference Conversion 93
5.1.6 Narrowing Reference Conversion 93
5.1.7 Boxing Conversion 94

5.1.8 Unboxing Conversion 95
5.1.9 Unchecked Conversion 97
5.1.10 Capture Conversion 97
5.1.11 String Conversion 99
5.1.12 Forbidden Conversions 100
5.1.13 Value Set Conversion 100
5.2 Assignment Conversion 101
5.3 Method Invocation Conversion 106
5.4 String Conversion 108
5.5 Casting Conversion 108
5.5.1 Reference Type Casting 111
5.5.2 Checked Casts and Unchecked Casts 115
5.5.3 Checked Casts at Run Time 116
5.6 Numeric Promotions 117
5.6.1 Unary Numeric Promotion 118
5.6.2 Binary Numeric Promotion 119

6 Names 121

6.1 Declarations 122
6.2 Names and Identifiers 127
6.3 Scope of a Declaration 130
6.4 Shadowing and Obscuring 133
6.4.1 Shadowing 135
6.4.2 Obscuring 138
6.5 Determining the Meaning of a Name 140
6.5.1 Syntactic Classification of a Name According to Context 141
6.5.2 Reclassification of Contextually Ambiguous Names 143
6.5.3 Meaning of Package Names 145
6.5.3.1 Simple Package Names 145
6.5.3.2 Qualified Package Names 146
6.5.4 Meaning of *PackageOrTypeNames* 146
6.5.4.1 Simple *PackageOrTypeNames* 146
6.5.4.2 Qualified *PackageOrTypeNames* 146
6.5.5 Meaning of Type Names 146
6.5.5.1 Simple Type Names 146
6.5.5.2 Qualified Type Names 146
6.5.6 Meaning of Expression Names 147
6.5.6.1 Simple Expression Names 147
6.5.6.2 Qualified Expression Names 148
6.5.7 Meaning of Method Names 151
6.5.7.1 Simple Method Names 151
6.5.7.2 Qualified Method Names 151
6.6 Access Control 152
6.6.1 Determining Accessibility 153
6.6.2 Details on protected Access 157
6.6.2.1 Access to a `protected` Member 157
6.6.2.2 Qualified Access to a `protected` Constructor 158

6.7 Fully Qualified Names and Canonical Names 159

7 Packages 163

7.1 Package Members 163
7.2 Host Support for Packages 165
7.3 Compilation Units 167
7.4 Package Declarations 168
7.4.1 Named Packages 168
7.4.2 Unnamed Packages 169
7.4.3 Observability of a Package 170
7.5 Import Declarations 170
7.5.1 Single-Type-Import Declarations 171
7.5.2 Type-Import-on-Demand Declarations 173
7.5.3 Single-Static-Import Declarations 174
7.5.4 Static-Import-on-Demand Declarations 175
7.6 Top Level Type Declarations 175

8 Classes 179

8.1 Class Declarations 181
8.1.1 Class Modifiers 181
8.1.1.1 `abstract` Classes 182
8.1.1.2 `final` Classes 184
8.1.1.3 `strictfp` Classes 184
8.1.2 Generic Classes and Type Parameters 185
8.1.3 Inner Classes and Enclosing Instances 187
8.1.4 Superclasses and Subclasses 190
8.1.5 Superinterfaces 192
8.1.6 Class Body and Member Declarations 195
8.2 Class Members 196
8.3 Field Declarations 201
8.3.1 Field Modifiers 205
8.3.1.1 `static` Fields 205
8.3.1.2 `final` Fields 209
8.3.1.3 `transient` Fields 209
8.3.1.4 `volatile` Fields 209
8.3.2 Initialization of Fields 211
8.3.2.1 Initializers for Class Variables 211
8.3.2.2 Initializers for Instance Variables 212
8.3.2.3 Restrictions on the use of Fields during Initialization 212
8.4 Method Declarations 215
8.4.1 Formal Parameters 216
8.4.2 Method Signature 219
8.4.3 Method Modifiers 220
8.4.3.1 `abstract` Methods 221
8.4.3.2 `static` Methods 222
8.4.3.3 `final` Methods 223

8.4.3.4 `native` Methods 224
8.4.3.5 `strictfp` Methods 224
8.4.3.6 `synchronized` Methods 224
8.4.4 Generic Methods 226
8.4.5 Method Return Type 226
8.4.6 Method Throws 227
8.4.7 Method Body 228
8.4.8 Inheritance, Overriding, and Hiding 229
8.4.8.1 Overriding (by Instance Methods) 229
8.4.8.2 Hiding (by Class Methods) 232
8.4.8.3 Requirements in Overriding and Hiding 233
8.4.8.4 Inheriting Methods with Override-Equivalent Signatures 237
8.4.9 Overloading 238
8.5 Member Type Declarations 242
8.5.1 Static Member Type Declarations 242
8.6 Instance Initializers 243
8.7 Static Initializers 243
8.8 Constructor Declarations 244
8.8.1 Formal Parameters and Type Parameters 245
8.8.2 Constructor Signature 245
8.8.3 Constructor Modifiers 245
8.8.4 Generic Constructors 246
8.8.5 Constructor Throws 247
8.8.6 The Type of a Constructor 247
8.8.7 Constructor Body 247
8.8.7.1 Explicit Constructor Invocations 248
8.8.8 Constructor Overloading 251
8.8.9 Default Constructor 251
8.8.10 Preventing Instantiation of a Class 253
8.9 Enums 253
8.9.1 Enum Constants 254
8.9.2 Enum Body Declarations 256

9 Interfaces 263

9.1 Interface Declarations 264
9.1.1 Interface Modifiers 264
9.1.1.1 `abstract` Interfaces 265
9.1.1.2 `strictfp` Interfaces 265
9.1.2 Generic Interfaces and Type Parameters 265
9.1.3 Superinterfaces and Subinterfaces 266
9.1.4 Interface Body and Member Declarations 267
9.2 Interface Members 268
9.3 Field (Constant) Declarations 269
9.3.1 Initialization of Fields in Interfaces 271
9.4 Abstract Method Declarations 271
9.4.1 Inheritance and Overriding 272

9.4.1.1 Overriding (by Instance Methods) 273
9.4.1.2 Requirements in Overriding 273
9.4.1.3 Inheriting Methods with Override-Equivalent Signatures 273
9.4.2 Overloading 274
9.5 Member Type Declarations 274
9.6 Annotation Types 275
9.6.1 Annotation Type Elements 276
9.6.2 Defaults for Annotation Type Elements 280
9.6.3 Predefined Annotation Types 280
9.6.3.1 `@Target` 280
9.6.3.2 `@Retention` 281
9.6.3.3 `@Inherited` 281
9.6.3.4 `@Override` 282
9.6.3.5 `@SuppressWarnings` 283
9.6.3.6 `@Deprecated` 283
9.6.3.7 `@SafeVarargs` 284
9.7 Annotations 285
9.7.1 Normal Annotations 286
9.7.2 Marker Annotations 288
9.7.3 Single-Element Annotations 289

10 Arrays 291

10.1 Array Types 292
10.2 Array Variables 292
10.3 Array Creation 294
10.4 Array Access 294
10.5 Array Store Exception 295
10.6 Array Initializers 297
10.7 Array Members 298
10.8 `Class` Objects for Arrays 300
10.9 An Array of Characters is Not a `String` 301

11 Exceptions 303

11.1 The Kinds and Causes of Exceptions 304
11.1.1 The Kinds of Exceptions 304
11.1.2 The Causes of Exceptions 305
11.1.3 Asynchronous Exceptions 305
11.2 Compile-Time Checking of Exceptions 306
11.2.1 Exception Analysis of Expressions 308
11.2.2 Exception Analysis of Statements 308
11.2.3 Exception Checking 309
11.3 Run-Time Handling of an Exception 311

12 Execution 315

12.1 Java Virtual Machine Startup 315

12.1.1 Load the Class `Test` 316
12.1.2 Link `Test`: Verify, Prepare, (Optionally) Resolve 316
12.1.3 Initialize Test: Execute Initializers 317
12.1.4 Invoke `Test.main` 318
12.2 Loading of Classes and Interfaces 318
12.2.1 The Loading Process 319
12.3 Linking of Classes and Interfaces 320
12.3.1 Verification of the Binary Representation 320
12.3.2 Preparation of a Class or Interface Type 321
12.3.3 Resolution of Symbolic References 321
12.4 Initialization of Classes and Interfaces 322
12.4.1 When Initialization Occurs 323
12.4.2 Detailed Initialization Procedure 325
12.5 Creation of New Class Instances 327
12.6 Finalization of Class Instances 331
12.6.1 Implementing Finalization 332
12.6.2 Interaction with the Memory Model 334
12.7 Unloading of Classes and Interfaces 335
12.8 Program Exit 336

13 Binary Compatibility 337

13.1 The Form of a Binary 338
13.2 What Binary Compatibility Is and Is Not 343
13.3 Evolution of Packages 344
13.4 Evolution of Classes 344
13.4.1 `abstract` Classes 344
13.4.2 `final` Classes 344
13.4.3 `public` Classes 345
13.4.4 Superclasses and Superinterfaces 345
13.4.5 Class Type Parameters 346
13.4.6 Class Body and Member Declarations 347
13.4.7 Access to Members and Constructors 348
13.4.8 Field Declarations 350
13.4.9 `final` Fields and Constants 352
13.4.10 `static` Fields 354
13.4.11 `transient` Fields 354
13.4.12 Method and Constructor Declarations 354
13.4.13 Method and Constructor Type Parameters 355
13.4.14 Method and Constructor Formal Parameters 356
13.4.15 Method Result Type 357
13.4.16 `abstract` Methods 357
13.4.17 `final` Methods 358
13.4.18 `native` Methods 358
13.4.19 `static` Methods 359
13.4.20 `synchronized` Methods 359
13.4.21 Method and Constructor Throws 359
13.4.22 Method and Constructor Body 359

13.4.23 Method and Constructor Overloading 359
13.4.24 Method Overriding 361
13.4.25 Static Initializers 361
13.4.26 Evolution of Enums 361
13.5 Evolution of Interfaces 361
13.5.1 `public` Interfaces 361
13.5.2 Superinterfaces 362
13.5.3 Interface Members 362
13.5.4 Interface Type Parameters 362
13.5.5 Field Declarations 363
13.5.6 `abstract` Methods 363
13.5.7 Evolution of Annotation Types 363

14 Blocks and Statements 365

14.1 Normal and Abrupt Completion of Statements 365
14.2 Blocks 367
14.3 Local Class Declarations 367
14.4 Local Variable Declaration Statements 369
14.4.1 Local Variable Declarators and Types 370
14.4.2 Execution of Local Variable Declarations 370
14.5 Statements 371
14.6 The Empty Statement 373
14.7 Labeled Statements 373
14.8 Expression Statements 374
14.9 The `if` Statement 375
14.9.1 The `if-then` Statement 375
14.9.2 The `if-then-else` Statement 376
14.10 The `assert` Statement 376
14.11 The `switch` Statement 379
14.12 The `while` Statement 383
14.12.1 Abrupt Completion of `while` Statement 384
14.13 The `do` Statement 385
14.13.1 Abrupt Completion of `do` Statement 385
14.14 The `for` Statement 387
14.14.1 The basic `for` Statement 387
14.14.1.1 Initialization of `for` Statement 388
14.14.1.2 Iteration of `for` Statement 388
14.14.1.3 Abrupt Completion of `for` Statement 389
14.14.2 The enhanced `for` statement 390
14.15 The `break` Statement 392
14.16 The `continue` Statement 394
14.17 The `return` Statement 396
14.18 The `throw` Statement 397
14.19 The `synchronized` Statement 399
14.20 The `try` statement 400
14.20.1 Execution of `try-catch` 403
14.20.2 Execution of `try-finally` and `try-catch-finally` 404

14.20.3 `try`-with-resources 407
14.20.3.1 Basic `try`-with-resources 408
14.20.3.2 Extended `try`-with-resources 410
14.21 Unreachable Statements 411

15 Expressions 417

15.1 Evaluation, Denotation, and Result 417
15.2 Variables as Values 418
15.3 Type of an Expression 418
15.4 FP-strict Expressions 419
15.5 Expressions and Run-Time Checks 419
15.6 Normal and Abrupt Completion of Evaluation 421
15.7 Evaluation Order 423
15.7.1 Evaluate Left-Hand Operand First 423
15.7.2 Evaluate Operands before Operation 425
15.7.3 Evaluation Respects Parentheses and Precedence 425
15.7.4 Argument Lists are Evaluated Left-to-Right 427
15.7.5 Evaluation Order for Other Expressions 428
15.8 Primary Expressions 428
15.8.1 Lexical Literals 429
15.8.2 Class Literals 430
15.8.3 `this` 430
15.8.4 Qualified `this` 431
15.8.5 Parenthesized Expressions 432
15.9 Class Instance Creation Expressions 432
15.9.1 Determining the Class being Instantiated 434
15.9.2 Determining Enclosing Instances 435
15.9.3 Choosing the Constructor and its Arguments 437
15.9.4 Run-Time Evaluation of Class Instance Creation Expressions 439
15.9.5 Anonymous Class Declarations 440
15.9.5.1 Anonymous Constructors 441
15.10 Array Creation Expressions 442
15.10.1 Run-Time Evaluation of Array Creation Expressions 443
15.11 Field Access Expressions 446
15.11.1 Field Access Using a Primary 447
15.11.2 Accessing Superclass Members using `super` 450
15.12 Method Invocation Expressions 451
15.12.1 Compile-Time Step 1: Determine Class or Interface to Search 452
15.12.2 Compile-Time Step 2: Determine Method Signature 453
15.12.2.1 Identify Potentially Applicable Methods 459
15.12.2.2 Phase 1: Identify Matching Arity Methods Applicable by Subtyping 460
15.12.2.3 Phase 2: Identify Matching Arity Methods Applicable by Method Invocation Conversion 461

15.12.2.4 Phase 3: Identify Applicable Variable Arity Methods 462
15.12.2.5 Choosing the Most Specific Method 462
15.12.2.6 Method Result and Throws Types 465
15.12.2.7 Inferring Type Arguments Based on Actual Arguments 466
15.12.2.8 Inferring Unresolved Type Arguments 477
15.12.3 Compile-Time Step 3: Is the Chosen Method Appropriate? 478
15.12.4 Run-Time Evaluation of Method Invocation 481
15.12.4.1 Compute Target Reference (If Necessary) 481
15.12.4.2 Evaluate Arguments 483
15.12.4.3 Check Accessibility of Type and Method 484
15.12.4.4 Locate Method to Invoke 485
15.12.4.5 Create Frame, Synchronize, Transfer Control 488
15.13 Array Access Expressions 490
15.13.1 Run-Time Evaluation of Array Access 491
15.14 Postfix Expressions 493
15.14.1 Expression Names 493
15.14.2 Postfix Increment Operator `++` 494
15.14.3 Postfix Decrement Operator `--` 494
15.15 Unary Operators 495
15.15.1 Prefix Increment Operator `++` 496
15.15.2 Prefix Decrement Operator `--` 497
15.15.3 Unary Plus Operator `+` 497
15.15.4 Unary Minus Operator `-` 498
15.15.5 Bitwise Complement Operator `~` 498
15.15.6 Logical Complement Operator `!` 499
15.16 Cast Expressions 499
15.17 Multiplicative Operators 500
15.17.1 Multiplication Operator `*` 501
15.17.2 Division Operator `/` 502
15.17.3 Remainder Operator `%` 503
15.18 Additive Operators 506
15.18.1 String Concatenation Operator `+` 506
15.18.2 Additive Operators (`+` and `-`) for Numeric Types 509
15.19 Shift Operators 511
15.20 Relational Operators 512
15.20.1 Numerical Comparison Operators `<`, `<=`, `>`, and `>=` 512
15.20.2 Type Comparison Operator `instanceof` 513
15.21 Equality Operators 514
15.21.1 Numerical Equality Operators `==` and `!=` 515
15.21.2 Boolean Equality Operators `==` and `!=` 516
15.21.3 Reference Equality Operators `==` and `!=` 517
15.22 Bitwise and Logical Operators 517
15.22.1 Integer Bitwise Operators `&`, `^`, and `|` 518
15.22.2 Boolean Logical Operators `&`, `^`, and `|` 519
15.23 Conditional-And Operator `&&` 519
15.24 Conditional-Or Operator `||` 520

15.25 Conditional Operator `? :` 521
15.26 Assignment Operators 523
    15.26.1 Simple Assignment Operator = 524
    15.26.2 Compound Assignment Operators 529
15.27 Expression 535
15.28 Constant Expressions 536

16 Definite Assignment 539

16.1 Definite Assignment and Expressions 545
    16.1.1 Boolean Constant Expressions 545
    16.1.2 Conditional-And Operator `&&` 545
    16.1.3 Conditional-Or Operator `||` 546
    16.1.4 Logical Complement Operator `!` 546
    16.1.5 Conditional Operator `? :` 546
    16.1.6 Conditional Operator `? :` 547
    16.1.7 Other Expressions of Type `boolean` 547
    16.1.8 Assignment Expressions 547
    16.1.9 Operators `++` and `--` 548
    16.1.10 Other Expressions 548
16.2 Definite Assignment and Statements 549
    16.2.1 Empty Statements 549
    16.2.2 Blocks 549
    16.2.3 Local Class Declaration Statements 551
    16.2.4 Local Variable Declaration Statements 551
    16.2.5 Labeled Statements 551
    16.2.6 Expression Statements 552
    16.2.7 `if` Statements 552
    16.2.8 `assert` Statements 552
    16.2.9 `switch` Statements 553
    16.2.10 `while` Statements 553
    16.2.11 `do` Statements 554
    16.2.12 `for` Statements 554
        16.2.12.1 Initialization Part of `for` Statement 555
        16.2.12.2 Incrementation Part of `for` Statement 555
    16.2.13 `break`, `continue`, `return`, and `throw` Statements 556
    16.2.14 `synchronized` Statements 556
    16.2.15 `try` Statements 556
16.3 Definite Assignment and Parameters 558
16.4 Definite Assignment and Array Initializers 558
16.5 Definite Assignment and Enum Constants 559
16.6 Definite Assignment and Anonymous Classes 559
16.7 Definite Assignment and Member Types 559
16.8 Definite Assignment and Static Initializers 560
16.9 Definite Assignment, Constructors, and Instance Initializers 560

17 Threads and Locks 563

17.1 Synchronization 564

17.2 Wait Sets and Notification 564
17.2.1 Wait 565
17.2.2 Notification 566
17.2.3 Interruptions 567
17.2.4 Interactions of Waits, Notification, and Interruption 567
17.3 Sleep and Yield 568
17.4 Memory Model 569
17.4.1 Shared Variables 572
17.4.2 Actions 572
17.4.3 Programs and Program Order 573
17.4.4 Synchronization Order 574
17.4.5 Happens-before Order 575
17.4.6 Executions 578
17.4.7 Well-Formed Executions 579
17.4.8 Executions and Causality Requirements 579
17.4.9 Observable Behavior and Nonterminating Executions 582
17.5 `final` Field Semantics 584
17.5.1 Semantics of `final` Fields 586
17.5.2 Reading `final` Fields During Construction 586
17.5.3 Subsequent Modification of `final` Fields 587
17.5.4 Write-protected Fields 588
17.6 Word Tearing 589
17.7 Non-atomic Treatment of `double` and `long` 590

18 Syntax 591

Index 607

A Limited License Grant 641

Preface to the Java SE 7 Edition

THE Java® SE 7 Edition of *The Java Language Specification* describes all the features that have been added to the Java programming language in Java SE 7. It also integrates changes made to the Java programming language under maintenance since the Third Edition in 2005.

Readers may send feedback about errors and ambiguities in *The Java Language Specification* to `jls-comments_ww@oracle.com`.

The majority of new features in this edition were specified by JSR 334, *Small Enhancements to the Java Programming Language*, led by Joe Darcy with an Expert Group of Joshua Bloch, Bruce Chapman, Alexey Kudravtsev, Mark Mahieu, Tim Peierls, and Olivier Thomann. The origins of these features lie in *Project Coin*, an OpenJDK project started in 2009 with the goal of "Making things programmers do every day easier". The project solicited proposals from the Java community for broadly useful language features that were, in comparison with "large" features like generics, relatively "small" in their specification, implementation, and testing.

Thousands of emails and six dozen proposals later, proposals were accepted from Joshua Bloch (the `try`-with-resources statement), Derek Foster/Bruce Chapman (improvements to literals), Neal Gafter (multi-`catch` and precise rethrow), Bob Lee (simplified variable arity method invocation), and Jeremy Manson (improved type inference for instance creation, a.k.a. the "diamond" syntax). The popular "strings in switch" feature was also accepted. Special thanks are due to Tom Ball, Stephen Colebourne, Rémi Forax, Shams Mahmood Imam, James Lowden, and all those who submitted interesting proposals and thoughtful comments to Project Coin. Over the course of the project, there were essential contributions from Mandy Chung, Jon Gibbons, Brian Goetz, David Holmes, and Dan Smith in areas ranging from library support to language specification. Stuart Marks led a "coinification" effort to apply the features to the Oracle JDK codebase, both to validate their utility and to develop conventions for wider use.

The "diamond" syntax and precise rethrow give type inference a new visibility in the Java programming language. To a great extent, inference is worthwhile only if it produces types no less specific than those in a manifestly-typed program prior to Java SE 7. Otherwise, new code may find inference insufficient, and migration from manifest to inferred types in existing code will be risky. To mitigate the risk, Joe Darcy and Maurizio Cimadamore measured the effectiveness of different inference schemes on a large corpus of open source Java code. Such "quantitative language design" greatly improves confidence in the suitability and safety of the final feature. The challenge of growing a mature language with millions of developers is partially offset by the ability of language designers to learn from developers' actual code.

The Java SE 7 platform adds features that cater for non-Java languages, effectively expanding the computational model of the platform. Without changes, the Java programming language would be unable to access or even express some of these features. Its static type system comes under particular stress when invoking code written in dynamically typed languages. Consequently, method invocation in the Java programming language has been modified to support method handle invocation as defined by JSR 292, *Dynamically Typed Languages on the Java Platform*.

The Java Compatibility Kit (JCK) team whose work helps validate this specification are due an enormous vote of thanks: Leonid Arbouzov, Alexey Gavrilov, Yulia Novozhilova, Sergey Reznick, and Victor Rudometov. Many other colleagues at Oracle (past or present) have also given valuable support to this specification: Uday Dhanikonda, Janet Koenig, Adam Messinger, Mark Reinhold, Georges Saab, Bill Shannon, and Bernard Traversat.

The following individuals have all provided many valuable comments which improved this specification: J. Stephen Adamczyk, Peter Ahé, Davide Ancona, Michael Bailey, Dmitry Batrak, Joshua Bloch, Kevin Bourrillion, Richard Bosworth, Martin Bravenboer, Martin Buchholz, Didier Cruette, Glenn Colman, Neal Gafter, Jim Holmlund, Ric Holt, Philippe Mulet, Bill Pugh, Vladimir Reshetnikov, John Spicer, Robert Stroud, and Mattias Ulbrich.

This edition is the first to be written in the DocBook XML format. Metadata in the XML markup forms a kind of static type system, classifying each paragraph by its role, such as a definition or an error. The reward is much crisper conformance testing. Many thanks go to Robert Stayton for sharing his considerable DocBook expertise and for helping to render DocBook in the traditional look and feel of *The Java Language Specification*.

Alex Buckley
Santa Clara, California

June, 2011

Preface to the Third Edition

THE Java SE 5.0 platform represents the largest set of changes in the history of the Java programming language. Generics, annotations, autoboxing and unboxing, enum types, foreach loops, variable arity methods, and static imports are all new to the language as of Autumn 2004.

This Third Edition of *The Java Language Specification* reflects these developments. It integrates all the changes made to the Java programming language since the publication of the Second Edition in 2000, including asserts from J2SE 1.4.

The Java programming language has grown a great deal in these past four years. Unfortunately, it is unrealistic to shrink a commercially successful programming language - only to grow it more and more. The challenge of managing this growth under the constraints of compatibility and the conflicting demands of a wide variety of uses and users is non-trivial. I can only hope that we have met this challenge successfully with this specification; time will tell.

This specification builds on the efforts of many people, both at Sun Microsystems and outside it.

The most crucial contribution is that of the people who actually turn the specification into real software. Chief among these are the maintainers of `javac`, the reference compiler for the Java programming language.

Neal Gafter was "Mr. `javac`" during the crucial period in which the large changes described here were integrated and productized. Neal's dedication and productivity can honestly be described as heroic. We literally could not have completed the task without him. In addition, his insight and skill made a huge contribution to the design of the new language features across the board. No one deserves more credit for this version of the Java programming language than he - but any blame for its deficiencies should be directed at myself and the members of the many JSR Expert Groups!

Neal has gone on in search of new challenges, and has been succeeded by Peter von der Ahé, who continues to improve and stengthen the implementation. Before Neal's involvement, Bill Maddox was in charge of `javac` when the previous edition was completed, and he nursed features such as generics and asserts through their early days.

Another individual who deserves to be singled out is Joshua Bloch. Josh participated in endless language design discussions, chaired several Expert Groups and was a key contributor to the Java platform. It is fair to say that Josh and Neal care more about this book than I do myself!

Many parts of the specification were developed by various Expert Groups in the framework of the Java Community Process.

The most pervasive set of language changes is the result of JSR 14, *Adding Generics to the Java Programming Language*. The members of the JSR 14 Expert Group were Norman Cohen, Christian Kemper, Martin Odersky, Kresten Krab Thorup, Philip Wadler, and myself. In the early stages, Sven-Eric Panitz and Steve Marx were members as well. All deserve thanks for their participation.

JSR 14 represents an unprecedented effort to fundamentally extend the type system of a widely used programming language under very stringent compatibility requirements. A prolonged and arduous process of design and implementation led us to the current language extension. Long before the JSR for generics was initiated, Martin Odersky and Philip Wadler had created an experimental language called Pizza to explore the ideas involved. In the spring of 1998, David Stoutamire and myself began a collaboration with Martin and Philip based on those ideas, that resulted in GJ. When the JSR 14 Expert Group was convened, GJ was chosen as the basis for extending the Java programming language. Martin Odersky implemented the GJ compiler, and his implementation became the basis for `javac` (starting with JDK 1.3, even though generics were disabled until 1.5).

The theoretical basis for the core of the generic type system owes a great debt to the expertise of Martin Odersky and Philip Wadler. Later, the system was extended with wildcards. These were based on the work of Atsushi Igarashi and Mirko Viroli, which itself built on earlier work by Kresten Thorup and Mads Torgersen. Wildcards were initially designed and implemented as part of a collaboration between Sun and Aarhus University. Neal Gafter and myself participated on Sun's behalf, and Erik Ernst and Mads Torgersen, together with Peter von der Ahé and Christian Plesner-Hansen, represented Aarhus. Thanks to Ole Lehrmann-Madsen for enabling and supporting that work.

Joe Darcy and Ken Russell implemented much of the specific support for reflection of generics. Neal Gafter, Josh Bloch and Mark Reinhold did a huge amount of work generifying the JDK libraries.

Honorable mention must go to individuals whose comments on the generics design made a significant difference. Alan Jeffrey made crucial contributions to JSR 14 by pointing out subtle flaws in the original type system. Bob Deen suggested the "? super T" syntax for lower bounded wildcards.

JSR 201 included a series of changes: autoboxing, enums, foreach loops, variable arity methods and static import. The members of the JSR 201 Expert Group were Cédric Beust, David Biesack, Joshua Bloch (co-chair), Corky Cartwright, Jim des Rivieres, David Flanagan, Christian Kemper, Doug Lea, Changshin Lee, Tim Peierls, Michel Trudeau, and myself (co-chair). Enums and the foreach loop were primarily designed by Josh Bloch and Neal Gafter. Variable arity methods would never have made it into the Java programming language without Neal's special efforts designing them (not to mention the small matter of implementing them).

Josh Bloch bravely took upon himself the responsibility for JSR 175, which added annotations to the Java programming language. The members of JSR 175 Expert Group were Cédric Beust, Joshua Bloch (chair), Ted Farrell, Mike French, Gregor Kiczales, Doug Lea, Deeptendu Majunder, Simon Nash, Ted Neward, Roly Perera, Manfred Schneider, Blake Stone, and Josh Street. Neal Gafter, as usual, was a major contributor on this front as well.

Another change in this edition is a complete revision of the Java memory model, undertaken by JSR 133. The members of the JSR 133 Expert Group were Hans Boehm, Doug Lea, Tim Lindholm (co-chair), Bill Pugh (co-chair), Martin Trotter, and Jerry Schwarz. The primary technical authors of the memory model are Sarita Adve, Jeremy Manson, and Bill Pugh. The Java memory model chapter in this book is in fact almost entirely their work, with only editorial revisions. Joseph Bowbeer, David Holmes, Victor Luchangco, and Jan-Willem Maessen made significant contributions as well. Key sections dealing with finalization in Chapter 12 owe much to this work as well, and especially to Doug Lea.

Many people have provided valuable comments on this edition.

I'd like to express my gratitude to Archibald Putt, who provided insight and encouragement. His writings are always an inspiration. Thanks once again to Joe Darcy for introducing us, as well as for many useful comments, and his specific contributions on numerical issues and the design of hexadecimal literals.

Many colleagues at Sun (past or present) have provided useful feedback and discussion, and helped produce this work in myriad ways: Andrew Bennett, Martin Buchholz, Jerry Driscoll, Robert Field, Jonathan Gibbons, Graham Hamilton, Mimi Hills, Jim Holmlund, Janet Koenig, Jeff Norton, Scott Seligman, Wei Tao, and David Ungar.

Special thanks to Laurie Tolson, my manager, for her support throughout the long process of deriving these specifications.

The following individuals all provided many valuable comments that have contributed to this specification: Scott Annanian, Martin Bravenboer, Bruce

Chapman, Lawrence Gonsalves, Tim Hanson, David Holmes, Angelika Langer, Pat Lavarre, Philippe Mulet, and Cal Varnson.

Ann Sellers, Greg Doench, and John Fuller at Addison-Wesley were exceedingly patient and ensured that the book materialized, despite the many missed deadlines for this text.

As always, I thank my wife Weihong and my son Teva for their support and cooperation.

Gilad Bracha
Los Altos, California

January, 2005

Preface to the Second Edition

OVER the past few years, the Java programming language has enjoyed unprecedented success. This success has brought a challenge: along with explosive growth in popularity, there has been explosive growth in the demands made on the language and its libraries. To meet this challenge, the language has grown as well (fortunately, not explosively) and so have the libraries.

This Second Edition of *The Java Language Specification* reflects these developments. It integrates all the changes made to the Java programming language since the publication of the First Edition in 1996. The bulk of these changes were made in the 1.1 release of the Java platform in 1997, and revolve around the addition of nested type declarations. Later modifications pertained to floating-point operations. In addition, this edition incorporates important clarifications and amendments involving method lookup and binary compatibility.

This specification defines the language as it exists today. The Java programming language is likely to continue to evolve. At this writing, there are ongoing initiatives through the Java Community Process to extend the language with generic types and assertions, refine the memory model, etc. However, it would be inappropriate to delay the publication of the Second Edition until these efforts are concluded.

The specifications of the libraries are now far too large to fit into this volume, and they continue to evolve. Consequently, API specifications have been removed from this book. The library specifications can be found on the Web; this specification now concentrates solely on the Java programming language proper.

Many people contributed to this book, directly and indirectly. Tim Lindholm brought extraordinary dedication to his role as technical editor of the Java Series. He also made invaluable technical contributions, especially on floating-point issues. The book would likely not see the light of day without him. Lisa Friendly, the Series editor, provided encouragement and advice for which I am very thankful.

David Bowen first suggested that I get involved in the specifications of the Java platform. I am grateful to him for introducing me to this uncommonly rich area.

John Rose, the father of nested types in the Java programming language, has been unfailingly gracious and supportive of my attempts to specify them accurately.

Many people have provided valuable comments on this edition. Special thanks go to Roly Perera at Ergnosis and to Leonid Arbouzov and his colleagues on the

Java platform conformance team in Novosibirsk: Konstantin Bobrovsky, Natalia Golovleva, Vladimir Ivanov, Alexei Kaigorodov, Serguei Katkov, Dmitri Khukhro, Eugene Latkin, Ilya Neverov, Pavel Ozhdikhin, Igor Pyankov, Viatcheslav Rybalov, Serguei Samoilidi, Maxim Sokolnikov, and Vitaly Tchaiko. Their thorough reading of earlier drafts has greatly improved the accuracy of this specification.

I am indebted to Martin Odersky and to Andrew Bennett and the members of the `javac` compiler team, past and present: Iris Garcia, Bill Maddox, David Stoutamire, and Todd Turnidge. They all worked hard to make sure the reference implementation conformed to the specification. For many enjoyable technical exchanges, I thank them and my other colleagues at Sun: Lars Bak, Joshua Bloch, Cliff Click, Robert Field, Mohammad Gharahgouzloo, Ben Gomes, Steffen Grarup, Robert Griesemer, Graham Hamilton, Gordon Hirsch, Peter Kessler, Sheng Liang, James McIlree, Philip Milne, Srdjan Mitrovic, Anand Palaniswamy, Mike Paleczny, Mark Reinhold, Kenneth Russell, Rene Schmidt, David Ungar, Chris Vick, and Hong Zhang.

Tricia Jordan, my manager, has been a model of patience, consideration and understanding. Thanks are also due to Larry Abrahams, director of Java 2 Standard Edition, for supporting this work.

The following individuals all provided useful comments that have contributed to this specification: Godmar Bak, Hans Boehm, Philippe Charles, David Chase, Joe Darcy, Jim des Rivieres, Sophia Drossopoulou, Susan Eisenbach, Paul Haahr, Urs Hoelzle, Bart Jacobs, Kent Johnson, Mark Lillibridge, Norbert Lindenberg, Phillipe Mulet, Kelly O'Hair, Bill Pugh, Cameron Purdy, Anthony Scian, Janice Shepherd, David Shields, John Spicer, Lee Worall, and David Wragg.

Suzette Pelouch provided invaluable assistance with the index and, together with Doug Kramer and Atul Dambalkar, assisted with FrameMaker expertise; Mike Hendrickson and Julie Dinicola at Addison-Wesley were gracious, helpful and ultimately made this book a reality.

On a personal note, I thank my wife Weihong for her love and support.

Finally, I'd like to thank my coauthors, James Gosling, Bill Joy, and Guy Steele for inviting me to participate in this work. It has been a pleasure and a privilege.

Gilad Bracha
Los Altos, California

April, 2000

Preface to the First Edition

THE Java programming language was originally called Oak, and was designed for use in embedded consumer-electronic applications by James Gosling. After several years of experience with the language, and significant contributions by Ed Frank, Patrick Naughton, Jonathan Payne, and Chris Warth it was retargeted to the Internet, renamed, and substantially revised to be the language specified here. The final form of the language was defined by James Gosling, Bill Joy, Guy Steele, Richard Tuck, Frank Yellin, and Arthur van Hoff, with help from Graham Hamilton, Tim Lindholm, and many other friends and colleagues.

The Java programming language is a general-purpose concurrent class-based object-oriented programming language, specifically designed to have as few implementation dependencies as possible. It allows application developers to write a program once and then be able to run it everywhere on the Internet.

This book attempts a complete specification of the syntax and semantics of the language. We intend that the behavior of every language construct is specified here, so that all implementations will accept the same programs. Except for timing dependencies or other non-determinisms and given sufficient time and sufficient memory space, a program written in the Java programming language should compute the same result on all machines and in all implementations.

We believe that the Java programming language is a mature language, ready for widespread use. Nevertheless, we expect some evolution of the language in the years to come. We intend to manage this evolution in a way that is completely compatible with existing applications. To do this, we intend to make relatively few new versions of the language. Compilers and systems will be able to support the several versions simultaneously, with complete compatibility.

Much research and experimentation with the Java platform is already underway. We encourage this work, and will continue to cooperate with external groups to explore improvements to the language and platform. For example, we have already received several interesting proposals for parameterized types. In technically difficult areas, near the state of the art, this kind of research collaboration is essential.

We acknowledge and thank the many people who have contributed to this book through their excellent feedback, assistance and encouragement:

Particularly thorough, careful, and thoughtful reviews of drafts were provided by Tom Cargill, Peter Deutsch, Paul Hilfinger, Masayuki Ida, David Moon, Steven Muchnick, Charles L. Perkins, Chris Van Wyk, Steve Vinoski, Philip Wadler, Daniel Weinreb, and Kenneth Zadeck. We are very grateful for their extraordinary volunteer efforts.

We are also grateful for reviews, questions, comments, and suggestions from Stephen Adams, Bowen Alpern, Glenn Ammons, Leonid Arbouzov, Kim Bruce, Edwin Chan, David Chase, Pavel Curtis, Drew Dean, William Dietz, David Dill, Patrick Dussud, Ed Felten, John Giannandrea, John Gilmore, Charles Gust, Warren Harris, Lee Hasiuk, Mike Hendrickson, Mark Hill, Urs Hoelzle, Roger Hoover, Susan Flynn Hummel, Christopher Jang, Mick Jordan, Mukesh Kacker, Peter Kessler, James Larus, Derek Lieber, Bill McKeeman, Steve Naroff, Evi Nemeth, Robert O'Callahan, Dave Papay, Craig Partridge, Scott Pfeffer, Eric Raymond, Jim Roskind, Jim Russell, William Scherlis, Edith Schonberg, Anthony Scian, Matthew Self, Janice Shepherd, Kathy Stark, Barbara Steele, Rob Strom, William Waite, Greg Weeks, and Bob Wilson. (This list was generated semi-automatically from our E-mail records. We apologize if we have omitted anyone.)

The feedback from all these reviewers was invaluable to us in improving the definition of the language as well as the form of the presentation in this book. We thank them for their diligence. Any remaining errors in this book - we hope they are few - are our responsibility and not theirs.

We thank Francesca Freedman and Doug Kramer for assistance with matters of typography and layout. We thank Dan Mills of Adobe Systems Incorporated for assistance in exploring possible choices of typefaces.

Many of our colleagues at Sun Microsystems have helped us in one way or another. Lisa Friendly, our series editor, managed our relationship with Addison-Wesley. Susan Stambaugh managed the distribution of many hundreds of copies of drafts to reviewers. We received valuable assistance and technical advice from Ben Adida, Ole Agesen, Ken Arnold, Rick Cattell, Asmus Freytag, Norm Hardy, Steve Heller, David Hough, Doug Kramer, Nancy Lee, Marianne Mueller, Akira Tanaka, Greg Tarsy, David Ungar, Jim Waldo, Ann Wollrath, Geoff Wyant, and Derek White. We thank Alan Baratz, David Bowen, Mike Clary, John Doerr, Jon Kannegaard, Eric Schmidt, Bob Sproull, Bert Sutherland, and Scott McNealy for leadership and encouragement.

We are thankful for the tools and services we had at our disposal in writing this book: telephones, overnight delivery, desktop workstations, laser printers, photocopiers, text formatting and page layout software, fonts, electronic mail, the World Wide Web, and, of course, the Internet. We live in three different states, scattered across a continent, but collaboration with each other and with our reviewers has seemed almost effortless. Kudos to the thousands of people who have worked over the years to make these excellent tools and services work quickly and reliably.

Mike Hendrickson, Katie Duffy, Simone Payment, and Rosa Aimée González of Addison-Wesley were very helpful, encouraging, and patient during the long process of bringing this book to print. We also thank the copy editors.

Rosemary Simpson worked hard, on a very tight schedule, to create the index. We got into the act at the last minute, however; blame us and not her for any jokes you may find hidden therein.

Finally, we are grateful to our families and friends for their love and support during this last, crazy, year.

In their book *The C Programming Language*, Brian Kernighan and Dennis Ritchie said that they felt that the C language "wears well as one's experience with it grows." If you like C, we think you will like the Java programming language. We hope that it, too, wears well for you.

James Gosling
Cupertino, California

Bill Joy
Aspen, Colorado

Guy Steele
Chelmsford, Massachusetts

July, 1996

This page intentionally left blank

CHAPTER 1

Introduction

THE Java® programming language is a general-purpose, concurrent, class-based, object-oriented language. It is designed to be simple enough that many programmers can achieve fluency in the language. The Java programming language is related to C and C++ but is organized rather differently, with a number of aspects of C and C++ omitted and a few ideas from other languages included. It is intended to be a production language, not a research language, and so, as C. A. R. Hoare suggested in his classic paper on language design, the design has avoided including new and untested features.

The Java programming language is strongly and statically typed. This specification clearly distinguishes between the *compile-time errors* that can and must be detected at compile time, and those that occur at run time. Compile time normally consists of translating programs into a machine-independent byte code representation. Run-time activities include loading and linking of the classes needed to execute a program, optional machine code generation and dynamic optimization of the program, and actual program execution.

The Java programming language is a relatively high-level language, in that details of the machine representation are not available through the language. It includes automatic storage management, typically using a garbage collector, to avoid the safety problems of explicit deallocation (as in C's `free` or C++'s `delete`). High-performance garbage-collected implementations can have bounded pauses to support systems programming and real-time applications. The language does not include any unsafe constructs, such as array accesses without index checking, since such unsafe constructs would cause a program to behave in an unspecified way.

The Java programming language is normally compiled to the bytecoded instruction set and binary format defined in *The Java Virtual Machine Specification, Java SE 7 Edition*.

1.1 Organization of the Specification

Chapter 2 describes grammars and the notation used to present the lexical and syntactic grammars for the language.

Chapter 3 describes the lexical structure of the Java programming language, which is based on C and C++. The language is written in the Unicode character set. It supports the writing of Unicode characters on systems that support only ASCII.

Chapter 4 describes types, values, and variables. Types are subdivided into primitive types and reference types.

The primitive types are defined to be the same on all machines and in all implementations, and are various sizes of two's-complement integers, single- and double-precision IEEE 754 standard floating-point numbers, a `boolean` type, and a Unicode character `char` type. Values of the primitive types do not share state.

Reference types are the class types, the interface types, and the array types. The reference types are implemented by dynamically created objects that are either instances of classes or arrays. Many references to each object can exist. All objects (including arrays) support the methods of the class `Object`, which is the (single) root of the class hierarchy. A predefined `String` class supports Unicode character strings. Classes exist for wrapping primitive values inside of objects. In many cases, wrapping and unwrapping is performed automatically by the compiler (in which case, wrapping is called boxing, and unwrapping is called unboxing). Class and interface declarations may be generic, that is, they may be parameterized by other reference types. Such declarations may then be invoked with specific type arguments.

Variables are typed storage locations. A variable of a primitive type holds a value of that exact primitive type. A variable of a class type can hold a null reference or a reference to an object whose type is that class type or any subclass of that class type. A variable of an interface type can hold a null reference or a reference to an instance of any class that implements the interface. A variable of an array type can hold a null reference or a reference to an array. A variable of class type `Object` can hold a null reference or a reference to any object, whether class instance or array.

Chapter 5 describes conversions and numeric promotions. Conversions change the compile-time type and, sometimes, the value of an expression. These conversions include the boxing and unboxing conversions between primitive types and reference types. Numeric promotions are used to convert the operands of a numeric operator to a common type where an operation can be performed. There are no loopholes in the language; casts on reference types are checked at run time to ensure type safety.

Chapter 6 describes declarations and names, and how to determine what names mean (denote). The language does not require types or their members to be declared before they are used. Declaration order is significant only for local variables, local classes, and the order of initializers of fields in a class or interface.

The Java programming language provides control over the scope of names and supports limitations on external access to members of packages, classes, and interfaces. This helps in writing large programs by distinguishing the implementation of a type from its users and those who extend it. Recommended naming conventions that make for more readable programs are described here.

Chapter 7 describes the structure of a program, which is organized into packages similar to the modules of Modula. The members of a package are classes, interfaces, and subpackages. Packages are divided into compilation units. Compilation units contain type declarations and can import types from other packages to give them short names. Packages have names in a hierarchical name space, and the Internet domain name system can usually be used to form unique package names.

Chapter 8 describes classes. The members of classes are classes, interfaces, fields (variables) and methods. Class variables exist once per class. Class methods operate without reference to a specific object. Instance variables are dynamically created in objects that are instances of classes. Instance methods are invoked on instances of classes; such instances become the current object `this` during their execution, supporting the object-oriented programming style.

Classes support single implementation inheritance, in which the implementation of each class is derived from that of a single superclass, and ultimately from the class `Object`. Variables of a class type can reference an instance of that class or of any subclass of that class, allowing new types to be used with existing methods, polymorphically.

Classes support concurrent programming with `synchronized` methods. Methods declare the checked exceptions that can arise from their execution, which allows compile-time checking to ensure that exceptional conditions are handled. Objects can declare a `finalize` method that will be invoked before the objects are discarded by the garbage collector, allowing the objects to clean up their state.

For simplicity, the language has neither declaration "headers" separate from the implementation of a class nor separate type and class hierarchies.

A special form of classes, enums, support the definition of small sets of values and their manipulation in a type safe manner. Unlike enumerations in other languages, enums are objects and may have their own methods.

Chapter 9 describes interface types, which declare a set of abstract methods, member types, and constants. Classes that are otherwise unrelated can implement the same interface type. A variable of an interface type can contain a reference to any object that implements the interface. Multiple interface inheritance is supported.

Annotation types are specialized interfaces used to annotate declarations. Such annotations are not permitted to affect the semantics of programs in the Java programming language in any way. However, they provide useful input to various tools.

Chapter 10 describes arrays. Array accesses include bounds checking. Arrays are dynamically created objects and may be assigned to variables of type `Object`. The language supports arrays of arrays, rather than multidimensional arrays.

Chapter 11 describes exceptions, which are nonresuming and fully integrated with the language semantics and concurrency mechanisms. There are three kinds of exceptions: checked exceptions, run-time exceptions, and errors. The compiler ensures that checked exceptions are properly handled by requiring that a method or constructor can result in a checked exception only if the method or constructor declares it. This provides compile-time checking that exception handlers exist, and aids programming in the large. Most user-defined exceptions should be checked exceptions. Invalid operations in the program detected by the Java Virtual Machine result in run-time exceptions, such as `NullPointerException`. Errors result from failures detected by the Java Virtual Machine, such as `OutOfMemoryError`. Most simple programs do not try to handle errors.

Chapter 12 describes activities that occur during execution of a program. A program is normally stored as binary files representing compiled classes and interfaces. These binary files can be loaded into a Java Virtual Machine, linked to other classes and interfaces, and initialized.

After initialization, class methods and class variables may be used. Some classes may be instantiated to create new objects of the class type. Objects that are class instances also contain an instance of each superclass of the class, and object creation involves recursive creation of these superclass instances.

When an object is no longer referenced, it may be reclaimed by the garbage collector. If an object declares a finalizer, the finalizer is executed before the object is reclaimed to give the object a last chance to clean up resources that would not otherwise be released. When a class is no longer needed, it may be unloaded.

Chapter 13 describes binary compatibility, specifying the impact of changes to types on other types that use the changed types but have not been recompiled. These considerations are of interest to developers of types that are to be widely distributed, in a continuing series of versions, often through the Internet. Good program development environments automatically recompile dependent code whenever a type is changed, so most programmers need not be concerned about these details.

Chapter 14 describes blocks and statements, which are based on C and C++. The language has no `goto` statement, but includes labeled `break` and `continue` statements. Unlike C, the Java programming language requires `boolean` (or `Boolean`) expressions in control-flow statements, and does not convert types to `boolean` implicitly (except through unboxing), in the hope of catching more errors at compile time. A `synchronized` statement provides basic object-level monitor locking. A `try` statement can include `catch` and `finally` clauses to protect against non-local control transfers.

Chapter 15 describes expressions. This document fully specifies the (apparent) order of evaluation of expressions, for increased determinism and portability. Overloaded methods and constructors are resolved at compile time by picking the most specific method or constructor from those which are applicable.

Chapter 16 describes the precise way in which the language ensures that local variables are definitely set before use. While all other variables are automatically initialized to a default value, the Java programming language does not automatically initialize local variables in order to avoid masking programming errors.

Chapter 17 describes the semantics of threads and locks, which are based on the monitor-based concurrency originally introduced with the Mesa programming language. The Java programming language specifies a memory model for shared-memory multiprocessors that supports high-performance implementations.

Chapter 18 presents a syntactic grammar for the language.

1.2 Example Programs

Most of the example programs given in the text are ready to be executed and are similar in form to:

```
    class Test {
        public static void main(String[] args) {
            for (int i = 0; i < args.length; i++)
                System.out.print(i == 0 ? args[i] : " " + args[i]);
            System.out.println();
        }
    }
```

On a machine with the Oracle JDK installed, this class, stored in the file `Test.java`, can be compiled and executed by giving the commands:

```
javac Test.java
java Test Hello, world.
```

producing the output:

```
Hello, world.
```

1.3 Notation

Throughout this specification we refer to classes and interfaces drawn from the Java SE platform API. Whenever we refer to a class or interface (other than those declared in an example) using a single identifier *N*, the intended reference is to the class or interface named *N* in the package `java.lang`. We use the canonical name (§6.7) for classes or interfaces from packages other than `java.lang`.

Non-normative information, designed to clarify the specification, is given in smaller, indented text.

This is non-normative information. It provides intuition, rationale, advice, examples, etc.

1.4 Relationship to Predefined Classes and Interfaces

As noted above, this specification often refers to classes of the Java SE platform API. In particular, some classes have a special relationship with the Java programming language. Examples include classes such as `Object`, `Class`, `ClassLoader`, `String`, `Thread`, and the classes and interfaces in package `java.lang.reflect`, among others. This specification constrains the behavior of such classes and interfaces, but does not provide a complete specification for them. The reader is referred to the Java SE platform API documentation.

Consequently, this specification does not describe reflection in any detail. Many linguistic constructs have analogs in the reflection API, but these are generally not discussed here. For example, when we list the ways in which an object can be created, we generally do not include the ways in which the reflection API can accomplish this. Readers should be aware of these additional mechanisms even though they are not mentioned in the text.

1.5 References

Apple Computer. *Dylan Reference Manual*. Apple Computer Inc., Cupertino, California. September 29, 1995.

Bobrow, Daniel G., Linda G. DeMichiel, Richard P. Gabriel, Sonya E. Keene, Gregor Kiczales, and David A. Moon. *Common Lisp Object System Specification*, X3J13 Document 88-002R, June 1988; appears as Chapter 28 of Steele, Guy. *Common Lisp: The Language*, 2nd ed. Digital Press, 1990, ISBN 1-55558-041-6, 770-864.

Ellis, Margaret A., and Bjarne Stroustrup. *The Annotated C++ Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1990, reprinted with corrections October 1992, ISBN 0-201-51459-1.

Goldberg, Adele and Robson, David. *Smalltalk-80: The Language*. Addison-Wesley, Reading, Massachusetts, 1989, ISBN 0-201-13688-0.

Harbison, Samuel. *Modula-3*. Prentice Hall, Englewood Cliffs, New Jersey, 1992, ISBN 0-13-596396.

Hoare, C. A. R. *Hints on Programming Language Design*. Stanford University Computer Science Department Technical Report No. CS-73-403, December 1973. Reprinted in SIGACT/SIGPLAN Symposium on Principles of Programming Languages. Association for Computing Machinery, New York, October 1973.

IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std. 754-1985. Available from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112-5704 USA; 800-854-7179.

Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*, 2nd ed. Prentice Hall, Englewood Cliffs, New Jersey, 1988, ISBN 0-13-110362-8.

Madsen, Ole Lehrmann, Birger Møller-Pedersen, and Kristen Nygaard. *Object-Oriented Programming in the Beta Programming Language*. Addison-Wesley, Reading, Massachusetts, 1993, ISBN 0-201-62430-3.

Mitchell, James G., William Maybury, and Richard Sweet. *The Mesa Programming Language, Version 5.0*. Xerox PARC, Palo Alto, California, CSL 79-3, April 1979.

Stroustrup, Bjarne. *The C++ Progamming Language*, 2nd ed. Addison-Wesley, Reading, Massachusetts, 1991, reprinted with corrections January 1994, ISBN 0-201-53992-6.

Unicode Consortium, The. *The Unicode Standard, Version 6.0.0*. Mountain View, CA, 2011, ISBN 978-1-936213-01-6.

This page intentionally left blank

Index

Symbols

= operator
  assignment conversion, 103
  expressions and run-time checks, 420, 421
  normal and abrupt completion of evaluation, 422, 422
@Deprecated, 283
@Inherited, 281
@Override, 282
@Retention, 281
@SafeVarargs, 284
  formal parameters, 218
@SuppressWarnings, 283
  checked casts and unchecked casts, 115
  formal parameters, 218
  inheriting methods with override-equivalent signatures, 238, 238
  requirements in overriding and hiding, 233
  unchecked conversion, 97
@Target, 280

A

abrupt completion of do statement, 385
  do statement, 385
abrupt completion of for statement, 389
  iteration of for statement, 389
abrupt completion of while statement, 384
  while statement, 384
abstract classes, 182, 344
  abstract methods, 221
  anonymous class declarations, 440
  array creation expressions, 442
  final classes, 184
  resolution of symbolic references, 322
  superinterfaces, 194
abstract interfaces, 265
abstract method declarations, 271
  abstract classes, 182
  annotations, 285
  declarations, 122, 122
  interface body and member declarations, 267
  raw types, 68
  syntactic classification of a name according to context, 141, 141, 141
abstract methods, 221, 357, 363
  abstract classes, 182
access control, 152
  abstract method declarations, 272
  class literals, 430
  class modifiers, 182, 182
  constructor declarations, 245
  default constructor, 251, 251
  determining the class being instantiated, 434, 434, 435, 435
  explicit constructor invocations, 249, 250
  field access using a primary, 447
  identify potentially applicable methods, 459
  import declarations, 170
  local class declarations, 367
  member type declarations, 242
  method modifiers, 220
  normal annotations, 286
  objects, 55

qualified expression names, 148, 149, 149
qualified type names, 147, 147
reference types and values, 53
requirements in overriding and hiding, 234
shadowing and obscuring, 133
single-static-import declarations, 174
single-type-import declarations, 171
static-import-on-demand declarations, 175
superclasses and subclasses, 190
superinterfaces, 192
superinterfaces and subinterfaces, 266
top level type declarations, 175
type-import-on-demand declarations, 173

access to a protected member, 157

access to members and constructors, 348
resolution of symbolic references, 322

accessing superclass members using super, 450
field declarations, 203
initialization of fields in interfaces, 271
initializers for class variables, 212
initializers for instance variables, 212
instance initializers, 243
static methods, 222
syntactic classification of a name according to context, 142

actions, 572
synchronization order, 574

additive operators, 506
constant expressions, 536
integer operations, 43

additive operators (+ and -) for numeric types, 509
binary numeric promotion, 119
floating-point operations, 48

an array of characters is not a String, 301

annotation type elements, 276
annotation types, 276

annotation types, 275
abstract method declarations, 272
annotations, 285, 285
class modifiers, 181
constructor modifiers, 245
enum constants, 254
field (constant) declarations, 269
field modifiers, 205
formal parameters, 217
interface declarations, 264
interface modifiers, 264
method modifiers, 220
named packages, 168
syntactic classification of a name according to context, 143

annotations, 285
abstract method declarations, 272
annotation types, 275
class modifiers, 181
constructor modifiers, 245
defaults for annotation type elements, 280, 280
enum constants, 254
evaluation, denotation, and result, 418
field (constant) declarations, 269
field modifiers, 205
formal parameters, 217
interface modifiers, 264
local variable declarators and types, 370
member type declarations, 242
method modifiers, 220
named packages, 168
syntactic classification of a name according to context, 141, 142, 143

anonymous class declarations, 440
class instance creation expressions, 433, 433
class modifiers, 182

definite assignment and anonymous classes, 559
enum constants, 255
form of a binary, 339
initialization of fields in interfaces, 271
inner classes and enclosing instances, 187
syntactic classification of a name according to context, 142

anonymous constructors, 441
choosing the constructor and its arguments, 439

argument lists are evaluated left-to-right, 427

array access, 294

array access expressions, 490
array access, 294
assignment operators, 523
compound assignment operators, 530
expressions and run-time checks, 420, 421
normal and abrupt completion of evaluation, 422, 422
simple assignment operator =, 524
syntactic classification of a name according to context, 142
unary numeric promotion, 118

array creation, 294

array creation expressions, 442
array creation, 294
array initializers, 297
initial values of variables, 81
kinds of variables, 78
normal and abrupt completion of evaluation, 421, 421
objects, 54
syntactic classification of a name according to context, 142
unary numeric promotion, 118

array initializers, 297
array creation, 294
definite assignment and array initializers, 558
normal and abrupt completion of evaluation, 421
normal annotations, 286
objects, 54
run-time evaluation of array creation expressions, 443

array members, 298
declarations, 122, 122
happens-before order, 576

array store exception, 295
array variables, 294
assignment conversion, 103
expressions and run-time checks, 420, 421
normal and abrupt completion of evaluation, 422
variables, 75

array types, 292
enhanced for statement, 390
raw types, 67
reifiable types, 66

array variables, 292
enhanced for statement, 391
formal parameters, 218

arrays, 291
annotation type elements, 277
array creation expressions, 442
kinds of variables, 78
reference types and values, 52
when reference types are the same, 58

assert statement, 376
assert statements, 552
detailed initialization procedure, 327
when initialization occurs, 323

assert statements, 552

assignment conversion, 101
array initializers, 297

array store exception, 295
array variables, 294
execution of try-catch, 404
expressions and run-time checks, 420, 420
inferring unresolved type arguments, 477
normal annotations, 287
return statement, 396
simple assignment operator =, 524, 525
switch statement, 380
throw statement, 397, 398
type of an expression, 418
variables, 75

assignment expressions, 547

assignment operators, 523
assignment conversion, 101
assignment expressions, 547
evaluation order for other expressions, 428
initial values of variables, 82
initialization of fields, 211
syntactic classification of a name according to context, 142
try statement, 402
variables, 75

asynchronous Exceptions, 305
causes of Exceptions, 305

B

basic for statement, 387
for statements, 554
scope of a declaration, 131

basic try-with-resources, 408
extended try-with-resources, 411

binary compatibility, 337

binary numeric promotion, 119
additive operators (+ and -) for numeric types, 509
conditional operator ? :, 522
division operator /, 502
integer bitwise operators &, ^, and |, 518
multiplicative operators, 500
numeric promotions, 118
numerical comparison operators <, <=, >, and >=, 512
numerical equality operators == and !=, 515
postfix decrement operator --, 495
postfix increment operator ++, 494
prefix decrement operator --, 497
prefix increment operator ++, 496
remainder operator %, 504
shift operators, 511

bitwise and logical operators, 517
constant expressions, 536

bitwise complement operator ~, 498
constant expressions, 536
integer operations, 43
unary numeric promotion, 118

blocks, 367, 549
blocks, 549
kinds of variables, 79
local class declarations, 367
scope of a declaration, 131

blocks and statements, 365

boolean constant expressions, 545

boolean equality operators == and !=, 516
boolean type and boolean values, 51

boolean literals, 35
boolean type and boolean values, 51
constant expressions, 536
identifiers, 24
lexical literals, 430

boolean logical operators &, ^, and |, 519
boolean type and boolean values, 51
conditional-and operator &&, 519
conditional-or operator ||, 520

boolean type and boolean values, 51

boolean literals, 35
lexical literals, 430
boxing conversion, 94
assignment conversion, 101
boolean type and boolean values, 51
casting conversion, 109
class literals, 430
conditional operator ? :, 521, 522
creation of new class instances, 328
floating-point operations, 49
inferring unresolved type arguments, 477
integer operations, 44
method invocation conversion, 107
normal and abrupt completion of evaluation, 422
objects, 54
postfix decrement operator --, 495
postfix increment operator ++, 494
prefix decrement operator --, 497
prefix increment operator ++, 496
break statement, 392
break, continue, return, and throw statements, 556
labeled statements, 373
names and identifiers, 128, 129
normal and abrupt completion of statements, 366
break, continue, return, and throw statements, 556

C

capture conversion, 97
array access expressions, 490
assignment operators, 523
cast expressions, 499
conditional operator ? :, 522
field access using a primary, 447
inferring type arguments based on actual arguments, 477
inferring unresolved type arguments, 478
intersection types, 71
method result and throws types, 465, 465
parameterized types, 60
qualified expression names, 149, 149, 149, 150, 150
simple expression names, 148
subtyping among class and interface types, 73
cast expressions, 499
array types, 292
casting conversion, 108
compile-time step 3: is the chosen method appropriate?, 480
constant expressions, 536
expressions and run-time checks, 420, 421
floating-point operations, 48
happens-before order, 576
integer operations, 44
normal and abrupt completion of evaluation, 422
objects, 55
syntactic classification of a name according to context, 142
type comparison operator instanceof, 514
unary operators, 495
casting conversion, 108
cast expressions, 500
expressions and run-time checks, 420, 421
happens-before order, 576
objects, 55
reference equality operators == and !=, 517
casting conversions to primitive types, 110
casting conversions to reference types, 111
causes of Exceptions, 305
character literals, 35

comments, 23
constant expressions, 536
escape sequences for character and String literals, 38
lexical literals, 430
unicode, 16
check accessibility of type and method, 484
checked casts and unchecked casts, 115
variables of reference type, 76
checked casts at run time, 116
checked casts and unchecked casts, 115
choosing the constructor and its arguments, 437
choosing the most specific method, 462
compile-time step 2: determine method Signature, 454
method and constructor overloading, 360
phase 1: identify matching arity methods applicable by subtyping, 461
phase 2: identify matching arity methods applicable by method invocation conversion, 461
phase 3: identify applicable variable arity methods, 462
class body and member declarations, 195, 347
class members, 196
scope of a declaration, 130
what binary compatibility is and is not, 343
class declarations, 181
declarations, 122
types, classes, and interfaces, 82
class instance creation expressions, 432
abstract classes, 182
constructor declarations, 244, 244
constructor overloading, 251
creation of new class instances, 327
exception analysis of expressions, 308
form of a binary, 341
initial values of variables, 81, 81
instance initializers, 243
kinds of variables, 79
method invocation conversion, 106
names and identifiers, 128
normal and abrupt completion of evaluation, 421
objects, 54, 54
return statement, 396
run-time handling of an exception, 312
String conversion, 99
syntactic classification of a name according to context, 142, 142, 142
types, classes, and interfaces, 82
class literals, 430
normal annotations, 287
syntactic classification of a name according to context, 142
class loading
causes of Exceptions, 305
class literals, 430
load the class test, 316
superclasses and subclasses, 192
superinterfaces and subinterfaces, 267
class members, 196
declarations, 122
members and constructors of parameterized types, 64
class modifiers, 181
annotations, 285
anonymous class declarations, 441
local class declarations, 367
reference type casting, 112, 112, 112
top level type declarations, 175
class Object, 56
checked casts at run time, 116, 116
class objects for arrays, 300
types, classes, and interfaces, 83

class String, 57
lexical literals, 430
literals, 25
objects, 54
String literals, 37, 37
class type parameters, 346
classes, 179
intersection types, 71
local class declarations, 367
package members, 163
qualified expression names, 148
reclassification of contextually ambiguous names, 143
reference types and values, 52
top level type declarations, 175
comments, 21
input elements and tokens, 20, 20
lexical grammar, 9
lexical translations, 16
line terminators, 19
unicode, 16
compilation units, 167
determining accessibility, 153
host support for packages, 165
identify potentially applicable methods, 459
observability of a package, 170
package members, 163
reclassification of contextually ambiguous names, 143, 144
scope of a declaration, 130
shadowing, 137
syntactic grammar, 10
compile-time checking of Exceptions, 306
exception checking, 310
compile-time step 1: determine class or interface to search, 452
class Object, 57
identify potentially applicable methods, 459
raw types, 69
compile-time step 2: determine method Signature, 453
choosing the constructor and its arguments, 438, 439
compile-time step 1: determine class or interface to search, 452
enum constants, 255
overloading, 239
what binary compatibility is and is not, 343
compile-time step 3: is the chosen method appropriate?, 478
check accessibility of type and method, 484
choosing the most specific method, 464
create frame, synchronize, transfer control, 489
form of a binary, 341
locate method to invoke, 485, 485
method invocation expressions, 452
compound assignment operators, 529
evaluate left-hand operand first, 423
compute target reference (if necessary), 481
conditional operator ? :, 521, 546, 547
binary numeric promotion, 120
boolean type and boolean values, 51, 51
conditional operator ? :, 546, 547
constant expressions, 536
floating-point operations, 48
integer operations, 43
normal annotations, 287
objects, 55
conditional-and operator &&, 519, 545
boolean type and boolean values, 51
conditional-and operator &&, 545
constant expressions, 536
conditional-or operator ||, 520, 546
boolean type and boolean values, 51
conditional-or operator ||, 546

constant expressions, 536

constant expressions, 536
- assignment conversion, 102
- boolean constant expressions, 545
- class String, 57
- conditional operator ? :, 521, 522
- creation of new class instances, 328
- detailed initialization procedure, 326
- final fields and constants, 352
- final variables, 81
- fp-strict expressions, 419
- initialization of fields in interfaces, 271, 271
- initializers for class variables, 212
- normal annotations, 287
- objects, 54
- String concatenation operator +, 506
- String literals, 37
- subsequent modification of final fields, 587
- switch statement, 380
- unreachable statements, 413, 413, 414

constructor body, 247
- constructor declarations, 244
- definite assignment, constructors, and instance initializers, 561
- initial values of variables, 81
- inner classes and enclosing instances, 188
- kinds of variables, 79
- this, 431

constructor declarations, 244
- class body and member declarations, 196
- creation of new class instances, 328
- declarations, 123
- final fields, 209
- raw types, 68
- return statement, 396, 396
- run-time evaluation of class instance creation expressions, 439
- simple expression names, 147

constructor modifiers, 245
- annotations, 285

constructor overloading, 251

constructor Signature, 245
- form of a binary, 341

constructor throws, 247
- compile-time checking of Exceptions, 306
- syntactic classification of a name according to context, 141
- throw statement, 398

context-free grammars, 9
- compilation units, 167

continue statement, 394
- break, continue, return, and throw statements, 556
- labeled statements, 373
- names and identifiers, 128, 129
- normal and abrupt completion of statements, 366

conversions and promotions, 85

create frame, synchronize, transfer control, 488

creation of new class instances, 327
- constructor declarations, 244
- initialization of fields, 211
- instance initializers, 243
- run-time evaluation of class instance creation expressions, 439
- static fields, 206
- String concatenation operator +, 506
- when initialization occurs, 323

D

declarations, 122
- names and identifiers, 128

default constructor, 251
- method and constructor declarations, 355

defaults for annotation type elements, 280
definite assignment, 539
  assignment operators, 523
  final variables, 80, 81
  initial values of variables, 82
  inner classes and enclosing instances, 189
  kinds of variables, 79
  parenthesized expressions, 432
definite assignment and anonymous classes, 559
definite assignment and array initializers, 558
definite assignment and enum constants, 559
definite assignment and expressions, 545
definite assignment and member types, 559
definite assignment and parameters, 558
definite assignment and statements, 549
definite assignment and static initializers, 560
  definite assignment and enum constants, 559
  final fields, 209
definite assignment, constructors, and instance initializers, 560
  final fields, 209
detailed initialization procedure, 325
  final fields and constants, 354
  form of a binary, 339
  initialization of fields, 211
  initialization of fields in interfaces, 271, 271
  initializers for class variables, 212
  simple expression names, 147
  static initializers, 243
  throw statement, 399
  when initialization occurs, 323
details on protected access, 157
  determining accessibility, 153
determining accessibility, 153
determining enclosing instances, 435
  inner classes and enclosing instances, 188
determining the class being instantiated, 434
  enums, 254
determining the meaning of a name, 140
  class members, 197
  interface members, 269
  names and identifiers, 128
  obscuring, 138
division operator /, 502
  compound assignment operators, 531
  evaluate operands before operation, 425
  integer operations, 44
  normal and abrupt completion of evaluation, 422
do statement, 385
  boolean type and boolean values, 51
  do statements, 554
do statements, 554

E

empty statement, 373
  empty statements, 549
empty statements, 549
enhanced for statement, 390
  for statements, 554
  scope of a declaration, 131
enum body declarations, 256
  default constructor, 252
enum constants, 254
  definite assignment and enum constants, 559
  definite assignment and static initializers, 560
  switch statement, 380
enums, 253
  abstract classes, 182
  abstract methods, 221
  annotation type elements, 277
  annotations, 285
  class declarations, 181

constructor body, 247
constructor modifiers, 245
declarations, 122
determining the class being instantiated, 434, 435
enums, 253
switch statement, 380

equality operators, 514
constant expressions, 536

erasure
assignment conversion, 103
checked casts and unchecked casts, 115
checked casts at run time, 116
choosing the most specific method, 464
class type parameters, 347
compile-time step 3: is the chosen method appropriate?, 480
constructor Signature, 245
create frame, synchronize, transfer control, 489
evaluate arguments, 483
field declarations, 352
form of a binary, 340, 341, 341, 342
inferring type arguments based on actual arguments, 475
method and constructor formal parameters, 356
method and constructor type parameters, 356
method invocation conversion, 107
method result and throws types, 465, 465
method result type, 357
method Signature, 219
phase 3: identify applicable variable arity methods, 462
raw types, 67
requirements in overriding and hiding, 234
type variables, 59

escape sequences for character and String literals, 38
character literals, 35
String literals, 36

evaluate arguments, 483
formal parameters, 218
variables of reference type, 76

evaluate left-hand operand first, 423

evaluate operands before operation, 425

evaluation order, 423

evaluation order for other expressions, 428

evaluation respects parentheses and precedence, 425

evaluation, denotation, and result, 417
assert statement, 376

evolution of annotation types, 363

evolution of classes, 344

evolution of enums, 361

evolution of interfaces, 361

evolution of packages, 344

example programs, 5

exception analysis of expressions, 308
class instance creation expressions, 433
compile-time checking of Exceptions, 307
method result and throws types, 465

exception analysis of statements, 308
compile-time checking of Exceptions, 307
explicit constructor invocations, 249
method throws, 227
throw statement, 398
try statement, 402

exception checking, 309
compile-time checking of Exceptions, 306
initialization of fields, 211
instance initializers, 243
method throws, 228
static initializers, 244
throw statement, 399, 399

try statement, 402
Exceptions, 303
floating-point operations, 49
integer operations, 44
normal and abrupt completion of statements, 366
throw statement, 397
execution, 315
execution of local variable declarations, 370
execution of try-catch, 403
try statement, 403
execution of try-finally and try-catch-finally, 404
try statement, 403
executions, 578
well-formed executions, 579
executions and causality requirements, 579
explicit constructor invocations, 248
anonymous constructors, 441, 441
constructor body, 247
creation of new class instances, 328
definite assignment, constructors, and instance initializers, 561, 561
exception analysis of statements, 309
form of a binary, 341
instance initializers, 243
method invocation conversion, 106
syntactic classification of a name according to context, 142
expression, 535
expression names, 493
expression statements, 374, 552
compile-time step 3: is the chosen method appropriate?, 479
evaluation, denotation, and result, 417
expression statements, 552
initialization of for statement, 388
expressions, 417
expressions and run-time checks, 419
extended try-with-resources, 410

F

field (constant) declarations, 269
annotations, 285
array initializers, 297
array variables, 292, 293
declarations, 122
interface body and member declarations, 267
kinds of variables, 78
obscuring, 139
shadowing, 136
syntactic classification of a name according to context, 141
field access expressions, 446
assignment operators, 523
names and identifiers, 128
normal and abrupt completion of evaluation, 422
objects, 55
simple assignment operator =, 524
static initializers, 244
field access using a primary, 447
field declarations, 201, 350, 363
array initializers, 297
array variables, 292, 293
class body and member declarations, 195
creation of new class instances, 328
declarations, 122
field declarations, 363
obscuring, 139
raw types, 68
reclassification of contextually ambiguous names, 143
resolution of symbolic references, 322
shadowing, 136

simple expression names, 147
syntactic classification of a name according to context, 141
field modifiers, 205
annotations, 285
final classes, 184, 344
anonymous class declarations, 441
final methods, 223
superclasses and subclasses, 190
verification of the binary representation, 321
final field semantics, 584
memory model, 571
final fields, 209
final fields and constants, 352
detailed initialization procedure, 326
field declarations, 363
final variables, 81
form of a binary, 339, 339
initialization of fields in interfaces, 271
initializers for class variables, 212
verification of the binary representation, 321
final methods, 223, 358
verification of the binary representation, 321
final variables, 80
constant expressions, 536, 536
enum body declarations, 257
final fields, 209, 209
final fields and constants, 352
form of a binary, 339
inner classes and enclosing instances, 187, 189
local variable declarators and types, 370
narrowing reference conversion, 93
try-with-resources, 407
when initialization occurs, 323
finalization of class instances, 331
class Object, 56
happens-before order, 575
kinds of variables, 78
unloading of classes and interfaces, 335
floating-point literals, 32
constant expressions, 536
lexical literals, 430
floating-point operations, 48
additive operators (+ and -) for numeric types, 510
division operator /, 503
multiplication operator *, 502
narrowing primitive conversion, 90
widening primitive conversion, 89
floating-point types, formats, and values, 45
cast expressions, 499
field declarations, 203
floating-point literals, 34
formal parameters, 218
fp-strict expressions, 419
lexical literals, 430, 430
local variable declarators and types, 370
narrowing primitive conversion, 90, 91
parenthesized expressions, 432
return statement, 397
unary minus operator -, 498
floating-point value set parameters, 46
floating-point types, formats, and values, 46, 46, 46
for statement, 387
array variables, 293
boolean type and boolean values, 51
compile-time step 3: is the chosen method appropriate?, 479
declarations, 123
initial values of variables, 82
kinds of variables, 79
local variable declaration statements, 369
for statements, 554
forbidden conversions, 100

form of a binary, 338
check accessibility of type and method, 484
compile-time step 3: is the chosen method appropriate?, 480
final variables, 81
loading of classes and interfaces, 318
locate method to invoke, 485
resolution of symbolic references, 321
top level type declarations, 177
when reference types are the same, 57, 58
formal parameters, 216
annotations, 285
array variables, 293
compile-time step 3: is the chosen method appropriate?, 479
declarations, 122
definite assignment and parameters, 558
evaluate arguments, 483
formal parameters and type parameters, 245
initial values of variables, 81
invoke test.main, 318
kinds of variables, 79
method and constructor formal parameters, 356
method declarations, 215
reclassification of contextually ambiguous names, 143
scope of a declaration, 131
shadowing and obscuring, 133
shared variables, 572
syntactic classification of a name according to context, 141
variables of reference type, 76
formal parameters and type parameters, 245
declarations, 122, 122
definite assignment and parameters, 558
initial values of variables, 81
kinds of variables, 79
reclassification of contextually ambiguous names, 143
scope of a declaration, 131
syntactic classification of a name according to context, 141
fp-strict expressions, 419
additive operators (+ and -) for numeric types, 510
cast expressions, 499
constant expressions, 536
division operator /, 503
floating-point types, formats, and values, 45
formal parameters, 218
multiplication operator *, 501
normal annotations, 287
return statement, 397
strictfp classes, 184
strictfp interfaces, 265
strictfp methods, 224
value set conversion, 100, 101
widening primitive conversion, 89
fully qualified names and canonical names, 159
compilation units, 167
form of a binary, 338, 342
import declarations, 170, 170
local class declarations, 367
named packages, 168
notation, 6
package members, 164
single-static-import declarations, 174
single-type-import declarations, 171
static-import-on-demand declarations, 175
top level type declarations, 177
type-import-on-demand declarations, 173

G

generic classes and type parameters, 185
capture conversion, 97
class instance creation expressions, 433
declarations, 122
form of a binary, 339
generic constructors, 246
generic methods, 226
parameterized types, 60
scope of a declaration, 131
syntactic classification of a name according to context, 141
type variables, 58
types, classes, and interfaces, 83
generic constructors, 246
class instance creation expressions, 433
form of a binary, 339
scope of a declaration, 131
type variables, 58
generic interfaces and type parameters, 265
capture conversion, 97
declarations, 122
form of a binary, 339
parameterized types, 60
scope of a declaration, 131
superinterfaces, 195
type variables, 58
types, classes, and interfaces, 83
generic methods, 226
abstract method declarations, 272
compile-time step 2: determine method Signature, 454
declarations, 122
form of a binary, 339
method declarations, 215
scope of a declaration, 131
type erasure, 65
type variables, 58
grammar notation, 10
grammars, 9

H

happens-before order, 575
executions, 578
executions and causality requirements, 581
finalization of class instances, 332
hiding (by class methods), 232
inheritance, overriding, and hiding, 229
obscuring, 139
shadowing, 136
host support for packages, 165
top level type declarations, 177

I

identifiers, 23
declarations, 122
keywords, 24
lexical grammar, 9
identify potentially applicable methods, 459
compile-time step 2: determine method Signature, 454
phase 1: identify matching arity methods applicable by subtyping, 460
phase 2: identify matching arity methods applicable by method invocation conversion, 461
phase 3: identify applicable variable arity methods, 462
identity conversion, 88
assignment conversion, 101
boolean type and boolean values, 51
boxing conversion, 95
capture conversion, 98

casting conversion, 108
method invocation conversion, 106
numeric promotions, 117
unary numeric promotion, 118
if statement, 375
boolean type and boolean values, 51
if statements, 552
if-then statement, 375
if statements, 552
if-then-else statement, 376
if statements, 552
implementing finalization, 332
import declarations, 170
compilation units, 167
incrementation part of for statement, 555
inferring type arguments based on actual arguments, 466
choosing the most specific method, 463, 463, 464
compile-time step 2: determine method Signature, 454
conditional operator ? :, 522
generic constructors, 246
generic methods, 226
inferring unresolved type arguments, 477, 477, 478
intersection types, 71
phase 1: identify matching arity methods applicable by subtyping, 460
phase 2: identify matching arity methods applicable by method invocation conversion, 461
phase 3: identify applicable variable arity methods, 462
try statement, 402
inferring unresolved type arguments, 477
inheritance and overriding, 272
abstract method declarations, 272
compile-time checking of Exceptions, 307
interface members, 268
inheritance, overriding, and hiding, 229
class Object, 56
inheriting methods with override-equivalent signatures, 237, 273
variables of reference type, 76
initial values of variables, 81
array initializers, 297
creation of new class instances, 328
final fields and constants, 354
initialization of fields in interfaces, 271
initializers for class variables, 212
kinds of variables, 78, 78, 78
preparation of a class or interface type, 321
run-time evaluation of array creation expressions, 443
run-time evaluation of class instance creation expressions, 439
variables, 75
initialization of classes and interfaces, 322
causes of Exceptions, 305
initialize test: execute initializers, 318
objects, 54
preparation of a class or interface type, 321
run-time handling of an exception, 312
static fields, 205
initialization of fields, 211
exception checking, 309
static initializers, 243
initialization of fields in interfaces, 271
detailed initialization procedure, 326
final fields and constants, 354
initializers for class variables, 212
initialization of for statement, 388
initialization part of for statement, 555
initialize test: execute initializers, 317
initializers for class variables, 211

definite assignment and static initializers, 560
detailed initialization procedure, 326
final fields and constants, 354
initialization of fields in interfaces, 271
simple expression names, 147
initializers for instance variables, 212
definite assignment, constructors, and instance initializers, 561
simple expression names, 147
inner classes and enclosing instances, 187
anonymous class declarations, 441
compile-time step 3: is the chosen method appropriate?, 478
compute target reference (if necessary), 481, 481
determining enclosing instances, 435, 435, 436
determining the class being instantiated, 434, 435
explicit constructor invocations, 248
form of a binary, 342
local class declarations, 367
qualified this, 431
when initialization occurs, 323
input elements and tokens, 19
lexical grammar, 9, 9
lexical translations, 16
unicode, 16
instance creation
abstract classes, 182
constructor declarations, 244, 244, 244
constructor overloading, 251
creation of new class instances, 327
exception analysis of expressions, 308
form of a binary, 341
initial values of variables, 81, 81
initialization of fields, 211
instance initializers, 243, 243
kinds of variables, 79
method invocation conversion, 106
names and identifiers, 128
normal and abrupt completion of evaluation, 421
objects, 54, 54
return statement, 396
run-time evaluation of class instance creation expressions, 439
run-time handling of an exception, 312
static fields, 206
String concatenation operator +, 506
String conversion, 99
syntactic classification of a name according to context, 142, 142, 142
types, classes, and interfaces, 82
when initialization occurs, 323
instance initializers, 243
class body and member declarations, 196
definite assignment, constructors, and instance initializers, 561
return statement, 396
simple expression names, 147
throw statement, 399
integer bitwise operators &, ^, and |, 518
binary numeric promotion, 120
integer operations, 43
shift operators, 511, 511
integer literals, 25
constant expressions, 536
lexical literals, 429
integer operations, 43
integral types and values, 43
character literals, 36
integer literals, 26
lexical literals, 429, 429, 430
interaction with the memory model, 334

interactions of waits, notification, and interruption, 567
interface body and member declarations, 267
 scope of a declaration, 130
interface declarations, 264
 declarations, 122
 types, classes, and interfaces, 82
interface members, 268, 362
 check accessibility of type and method, 484
 declarations, 122
 form of a binary, 340, 342
interface methods
 abstract classes, 182
 annotations, 285
 declarations, 122, 122
 interface body and member declarations, 267
 raw types, 68
 syntactic classification of a name according to context, 141, 141, 141
interface modifiers, 264
 annotations, 285
 static member type declarations, 243
 top level type declarations, 175
interface type parameters, 362
interfaces, 263
 array creation expressions, 442
 package members, 163
 qualified expression names, 149
 reclassification of contextually ambiguous names, 143
 reference types and values, 52
 top level type declarations, 175
interruptions, 567
intersection types, 71
 form of a binary, 340, 340
 type variables, 59
introduction, 1
invoke test.main, 318
iteration of for statement, 388

J

Java Virtual Machine startup, 315

K

keywords, 24
 identifiers, 24
 lexical grammar, 9
 primitive types and values, 42
kinds and causes of Exceptions, 304
kinds of conversion, 88
kinds of Exceptions, 304
 compile-time checking of Exceptions, 307, 307
 generic classes and type parameters, 186
 method throws, 227, 228
 narrowing primitive conversion, 92
 throw statement, 398
 widening primitive conversion, 89
kinds of types and values, 41
 capture conversion, 97
 lexical literals, 430
 literals, 25
 null literal, 39
 throw statement, 398
kinds of variables, 78

L

labeled statements, 373, 551
 break statement, 392
 continue statement, 394
 labeled statements, 374, 551
 names and identifiers, 128, 129
lexical grammar, 9

lexical literals, 429
kinds of types and values, 42
lexical structure, 15
lexical grammar, 9
lexical translations, 16
line terminators, 18
character literals, 36
input elements and tokens, 19
lexical translations, 16
white space, 21
link test: verify, prepare, (optionally) resolve, 316
linking of classes and interfaces, 320
causes of Exceptions, 305
check accessibility of type and method, 484
link test: verify, prepare, (optionally) resolve, 316
resolution of symbolic references, 322
literals, 25
lexical grammar, 9
lexical literals, 429
load the class test, 316
loading of classes and interfaces, 318
causes of Exceptions, 305
class literals, 430
load the class test, 316
superclasses and subclasses, 192
superinterfaces and subinterfaces, 267
loading process, 319
local class declaration statements, 551
local class declarations, 367
class instance creation expressions, 433
class modifiers, 182
determining enclosing instances, 436, 436
explicit constructor invocations, 250
form of a binary, 339
inner classes and enclosing instances, 187
local class declaration statements, 551
reclassification of contextually ambiguous names, 143
shadowing and obscuring, 133
local variable declaration statements, 369, 551
array initializers, 297
declarations, 123
initial values of variables, 82
initialization of fields, 211
initialization of for statement, 388
initialization part of for statement, 555
kinds of variables, 79
local variable declaration statements, 551
objects, 54
reclassification of contextually ambiguous names, 143
scope of a declaration, 131
shadowing and obscuring, 133
shared variables, 572
syntactic classification of a name according to context, 142
local variable declarators and types, 370
annotations, 285
array variables, 292
locate method to invoke, 485
overriding (by instance methods), 230
logical complement operator !, 499, 546
boolean type and boolean values, 51
constant expressions, 536
logical complement operator !, 546

M

marker annotations, 288
meaning of expression names, 147
expression names, 493
names and identifiers, 128
meaning of method names, 151

names and identifiers, 128
meaning of package names, 145
names and identifiers, 128
meaning of packageortypenames, 146
meaning of type names, 146
names and identifiers, 128
reference types and values, 53
member type declarations, 242, 274
class body and member declarations, 195
class instance creation expressions, 433, 433
class modifiers, 182, 182
declarations, 122, 122, 122, 122
definite assignment and member types, 559, 559
determining enclosing instances, 436, 437
determining the class being instantiated, 434, 435
explicit constructor invocations, 250
form of a binary, 339, 339
inner classes and enclosing instances, 187, 187
interface body and member declarations, 267
interface modifiers, 265
member type declarations, 274
obscuring, 139, 139
qualified type names, 147, 147
reclassification of contextually ambiguous names, 143, 143, 144, 144
reference types and values, 53, 53
shadowing, 136, 136
static-import-on-demand declarations, 175, 175
type-import-on-demand declarations, 174, 174
members and constructors of parameterized types, 64
memory model, 569
interaction with the memory model, 334
volatile fields, 209
method and constructor body, 359
method and constructor declarations, 354
method and constructor formal parameters, 356
method result type, 357
resolution of symbolic references, 322
method and constructor formal parameters, 356
abstract methods, 363
method and constructor overloading, 359
abstract methods, 363
method and constructor throws, 359
abstract methods, 363
method and constructor type parameters, 355
class type parameters, 347
method body, 228
constructor body, 247
method declarations, 215
method declarations, 215
class body and member declarations, 195
declarations, 122
evaluation, denotation, and result, 417
raw types, 68
return statement, 396, 396, 396
simple expression names, 147
syntactic classification of a name according to context, 141
method invocation conversion, 106
create frame, synchronize, transfer control, 489
formal parameters, 218
inferring type arguments based on actual arguments, 466
phase 2: identify matching arity methods applicable by method invocation conversion, 461
method invocation expressions, 451

anonymous constructors, 441, 441
compute target reference (if necessary), 481
constructor declarations, 245
evaluation, denotation, and result, 417
exception analysis of expressions, 308
expressions and run-time checks, 420
formal parameters, 218
happens-before order, 576
hiding (by class methods), 233
initial values of variables, 81
initialization of fields in interfaces, 271
initializers for class variables, 212
initializers for instance variables, 212
instance initializers, 243
kinds of variables, 79
method invocation conversion, 106
names and identifiers, 128
normal and abrupt completion of evaluation, 422, 422
objects, 55
overloading, 239
overriding (by instance methods), 230
return statement, 396
run-time handling of an exception, 312
static initializers, 244
syntactic classification of a name according to context, 142, 142
this, 431
method modifiers, 220
abstract classes, 182
annotations, 285
method declarations, 215
objects, 56
method overriding, 361
method result and throws types, 465
choosing the constructor and its arguments, 438
exception analysis of expressions, 308, 308
exception analysis of statements, 309
method result type, 357
abstract methods, 363
method return type, 226
abstract classes, 183
class literals, 430
inheriting methods with override-equivalent signatures, 238
method declarations, 215
requirements in overriding and hiding, 233
type erasure, 65
method Signature, 219
abstract classes, 183
abstract method declarations, 272
abstract methods, 221
choosing the most specific method, 464
constructor Signature, 245
form of a binary, 341
hiding (by class methods), 232
inheriting methods with override-equivalent signatures, 237, 273
interface members, 268
method declarations, 215
overloading, 274
overriding (by instance methods), 230, 273
requirements in overriding and hiding, 234
type erasure, 65
method throws, 227
compile-time checking of Exceptions, 306
constructor throws, 247
inheriting methods with override-equivalent signatures, 238
method declarations, 215
syntactic classification of a name according to context, 141
throw statement, 398
multiplication operator *, 501
multiplicative operators, 500

binary numeric promotion, 119
constant expressions, 536
floating-point operations, 48
integer operations, 43

N

name classification
access control, 152
named packages, 168
annotations, 285
names, 121
names and identifiers, 127
determining the class being instantiated, 434, 435
import declarations, 170
local class declarations, 367
named packages, 168
shadowing and obscuring, 133
narrowing primitive conversion, 90
casting conversion, 108
floating-point operations, 49
narrowing primitive conversion, 91
postfix decrement operator --, 495
postfix increment operator ++, 494
prefix decrement operator --, 497
prefix increment operator ++, 496
widening and narrowing primitive conversion, 93
narrowing reference conversion, 93
casting conversion, 109
native methods, 224, 358
new keyword
abstract classes, 182
constructor declarations, 244, 244
constructor overloading, 251
creation of new class instances, 327
exception analysis of expressions, 308
form of a binary, 341
initial values of variables, 81, 81
instance initializers, 243
kinds of variables, 79
method invocation conversion, 106
names and identifiers, 128
normal and abrupt completion of evaluation, 421
objects, 54, 54
return statement, 396
run-time handling of an exception, 312
String conversion, 99
syntactic classification of a name according to context, 142, 142, 142
types, classes, and interfaces, 82
non-atomic treatment of double and long, 590
normal and abrupt completion of evaluation, 421
causes of Exceptions, 305
normal and abrupt completion of statements, 366, 366
run-time handling of an exception, 312
normal and abrupt completion of statements, 365
method body, 229
normal and abrupt completion of evaluation, 423
run-time handling of an exception, 312
normal annotations, 286
notation, 6
notification, 566
null literal, 39
compile-time step 3: is the chosen method appropriate?, 480
identifiers, 24
kinds of types and values, 42
lexical literals, 430
numeric promotions, 117

floating-point operations, 48
integer operations, 44
numerical comparison operators <, <=, >, and >=, 512
binary numeric promotion, 119
floating-point operations, 48
floating-point types, formats, and values, 47
integer operations, 43
numerical equality operators == and !=, 515
binary numeric promotion, 119
floating-point operations, 48
floating-point types, formats, and values, 47
integer operations, 43

O

object creation
constructor declarations, 244
initialization of fields, 211
instance initializers, 243
run-time evaluation of class instance creation expressions, 439
static fields, 206
String concatenation operator +, 506
when initialization occurs, 323
objects, 54
checked casts at run time, 116, 116
String literals, 37
obscuring, 138
labeled statements, 374
shadowing, 136
observability of a package, 170
qualified package names, 146
scope of a declaration, 130
observable behavior and nonterminating executions, 582
actions, 572, 573
operators, 40
input elements and tokens, 20
lexical grammar, 9
operators ++ and --, 548
organization of the specification, 2
other expressions, 548
other expressions of type boolean, 547
overload resolution
choosing the constructor and its arguments, 438, 439
compile-time step 1: determine class or interface to search, 452
compile-time step 2: determine method Signature, 453, 453, 453, 454, 454, 454, 454, 454
enum constants, 255
inferring type arguments based on actual arguments, 466, 466, 466
method and constructor overloading, 360
overloading, 239
phase 1: identify matching arity methods applicable by subtyping, 460, 461, 461
phase 2: identify matching arity methods applicable by method invocation conversion, 461, 461, 461
phase 3: identify applicable variable arity methods, 462, 462
what binary compatibility is and is not, 343
overloading, 238, 274
constructor overloading, 251
overriding (by instance methods), 229, 273
abstract classes, 182
final classes, 184
inheritance, overriding, and hiding, 229
locate method to invoke, 486

P

package declarations, 168

compilation units, 167
declarations, 122
syntactic classification of a name according to context, 141
package members, 163
packages, 163
parameterized types, 60
capture conversion, 97, 98
checked casts and unchecked casts, 115
class literals, 430
determining the class being instantiated, 434, 434, 435, 435
field declarations, 352
generic classes and type parameters, 185
generic interfaces and type parameters, 266
method and constructor formal parameters, 356
method result type, 357
raw types, 67
reference type casting, 111
reference types and values, 53
syntactic classification of a name according to context, 141
type erasure, 65
types, classes, and interfaces, 83
parenthesized expressions, 432
constant expressions, 536
phase 1: identify matching arity methods applicable by subtyping, 460
compile-time step 2: determine method Signature, 453, 454
inferring type arguments based on actual arguments, 466
phase 2: identify matching arity methods applicable by method invocation conversion, 461
compile-time step 2: determine method Signature, 453, 454
inferring type arguments based on actual arguments, 466
phase 1: identify matching arity methods applicable by subtyping, 461
phase 3: identify applicable variable arity methods, 462
compile-time step 2: determine method Signature, 453, 454
inferring type arguments based on actual arguments, 466
phase 2: identify matching arity methods applicable by method invocation conversion, 461
postfix decrement operator --, 494
floating-point operations, 48, 49
integer operations, 43, 44
normal and abrupt completion of evaluation, 422
operators ++ and --, 548
variables, 75
postfix expressions, 493
syntactic classification of a name according to context, 142
postfix increment operator ++, 494
floating-point operations, 48, 49
integer operations, 43, 44
normal and abrupt completion of evaluation, 422
operators ++ and --, 548
variables, 75
potentially applicable methods
compile-time step 2: determine method Signature, 454
phase 1: identify matching arity methods applicable by subtyping, 460
phase 2: identify matching arity methods applicable by method invocation conversion, 461

phase 3: identify applicable variable arity methods, 462
predefined annotation types, 280
prefix decrement operator --, 497
floating-point operations, 48, 49
integer operations, 43, 44
normal and abrupt completion of evaluation, 422
operators ++ and --, 548
variables, 75
prefix increment operator ++, 496
floating-point operations, 48, 49
integer operations, 43, 44
normal and abrupt completion of evaluation, 422
operators ++ and --, 548
variables, 75
preparation of a class or interface type, 321
kinds of variables, 78
link test: verify, prepare, (optionally) resolve, 316
preventing instantiation of a class, 253
primary expressions, 428
postfix expressions, 493
primitive types and values, 42
class literals, 430
evaluation, denotation, and result, 417
kinds of types and values, 41
literals, 25
reifiable types, 66
unboxing conversion, 96
variables, 75
program exit, 336
programs and program order, 573
happens-before order, 576
synchronization order, 574
public classes, 345
public interfaces, 361

Q

qualified access to a protected constructor, 158
qualified expression names, 148
access control, 152
constant expressions, 536
field access expressions, 447
field declarations, 203
qualified method names, 151
qualified package names, 146
qualified packageortypenames, 146
qualified this, 431
syntactic classification of a name according to context, 142
qualified type names, 146
access control, 152

R

raw types, 67
assignment conversion, 102
method invocation conversion, 107
reifiable types, 66
unchecked conversion, 97
variables of reference type, 76
reading final fields during construction, 586
reclassification of contextually ambiguous names, 143
reference equality operators == and !=, 517
objects, 55
reference type casting, 111
casting conversion, 109
reference types and values, 52
class literals, 430
determining the class being instantiated, 434
evaluation, denotation, and result, 417
initial values of variables, 81

kinds of types and values, 41
variables, 75
references, 7
reifiable types, 66
@SafeVarargs, 284
array creation expressions, 442
array initializers, 297
expressions and run-time checks, 420, 421
formal parameters, 218
type comparison operator instanceof, 514
relational operators, 512
constant expressions, 536
relationship to predefined classes and interfaces, 6
remainder operator %, 503
evaluate operands before operation, 425
floating-point operations, 49
integer operations, 44
normal and abrupt completion of evaluation, 422
requirements in overriding, 273
variables of reference type, 76
requirements in overriding and hiding, 233
compile-time checking of Exceptions, 307
method throws, 228
requirements in overriding, 273, 273, 273
variables of reference type, 76
resolution of symbolic references, 321
link test: verify, prepare, (optionally) resolve, 317
restrictions on the use of fields during initialization, 212
when initialization occurs, 323, 323
return statement, 396
break, continue, return, and throw statements, 556
constructor body, 247
instance initializers, 243
method body, 229, 229
normal and abrupt completion of statements, 366
static initializers, 243
run-time evaluation of array access, 491
evaluation order for other expressions, 428
run-time evaluation of array creation expressions, 443
evaluation order for other expressions, 428
run-time evaluation of class instance creation expressions, 439
evaluation order for other expressions, 428
throw statement, 398
run-time evaluation of method invocation, 481
evaluation order for other expressions, 428
overloading, 239
run-time handling of an exception, 311
expressions and run-time checks, 421
initial values of variables, 82
kinds of variables, 79
throw statement, 397
try statement, 402

S

scope of a declaration, 130
basic for statement, 387
class body and member declarations, 196
class declarations, 181
class literals, 430
compile-time step 1: determine class or interface to search, 452
enhanced for statement, 390, 391
enum constants, 254
field declarations, 202
formal parameters, 217
generic classes and type parameters, 185

generic constructors, 246
generic interfaces and type parameters, 266
generic methods, 226
import declarations, 171
interface body and member declarations, 268
interface declarations, 264
local class declarations, 368
local variable declarators and types, 370
member type declarations, 242
method declarations, 215
named packages, 168
reclassification of contextually ambiguous names, 143, 143
simple package names, 145
top level type declarations, 177
try statement, 401
try-with-resources, 407
type variables, 58
semantics of final fields, 586
separators, 40
lexical grammar, 9
shadowing, 135
identify potentially applicable methods, 459
obscuring, 139
scope of a declaration, 130
simple expression names, 147
simple method names, 151
shadowing and obscuring, 133
basic for statement, 387
class body and member declarations, 196
class declarations, 181
enhanced for statement, 390
enum constants, 254
field declarations, 202
formal parameters, 217
generic classes and type parameters, 185
generic constructors, 246
import declarations, 171
interface declarations, 264
local class declarations, 368
local variable declarators and types, 370
member type declarations, 242
method declarations, 215
named packages, 168
top level type declarations, 177
try statement, 401
try-with-resources, 407
shared variables, 572
happens-before order, 576
shift operators, 511
constant expressions, 536
integer operations, 43
unary numeric promotion, 118
simple assignment operator =, 524
assignment conversion, 103
expressions and run-time checks, 420, 421
normal and abrupt completion of evaluation, 422, 422
simple expression names, 147
constant expressions, 536
field access expressions, 447
simple method names, 151
simple package names, 145
simple packageortypenames, 146
simple type names, 146
single-element annotations, 289
annotation type elements, 278
single-static-import declarations, 174
compile-time step 1: determine class or interface to search, 452
identify potentially applicable methods, 459
import declarations, 170
reclassification of contextually ambiguous names, 143, 144
scope of a declaration, 130
simple method names, 151

single-type-import declarations, 172
syntactic classification of a name according to context, 141
single-type-import declarations, 171
declarations, 122
import declarations, 170
reclassification of contextually ambiguous names, 144
scope of a declaration, 130
single-static-import declarations, 174
syntactic classification of a name according to context, 141
top level type declarations, 176
sleep and yield, 568
statements, 371
static fields, 205, 354
generic classes and type parameters, 186
kinds of variables, 78, 78
when initialization occurs, 323
static initializers, 243, 361
class body and member declarations, 196
definite assignment and static initializers, 560
exception checking, 309
final fields, 209
generic classes and type parameters, 186
inner classes and enclosing instances, 187
return statement, 396
simple expression names, 147
static initializers, 361
throw statement, 399
static member type declarations, 242
anonymous class declarations, 441
class modifiers, 182
generic classes and type parameters, 186
interface modifiers, 265, 265
qualified type names, 147
static methods, 222, 359
generic classes and type parameters, 186
simple expression names, 147
static-import-on-demand declarations, 175
compile-time step 1: determine class or interface to search, 452
identify potentially applicable methods, 459
import declarations, 170
reclassification of contextually ambiguous names, 143, 144
scope of a declaration, 130
simple method names, 151
syntactic classification of a name according to context, 141
type-import-on-demand declarations, 174
strictfp classes, 184
fp-strict expressions, 419
strictfp interfaces, 265
fp-strict expressions, 419
strictfp methods, 224
fp-strict expressions, 419
String concatenation operator +, 506
boolean type and boolean values, 51
class String, 57
constructor declarations, 244
creation of new class instances, 328
floating-point operations, 48
integer operations, 44
normal and abrupt completion of evaluation, 421
objects, 54, 55
String conversion, 108
types, classes, and interfaces, 82
String conversion, 99, 108
boolean type and boolean values, 51
String concatenation operator +, 506
String conversion, 108
String literals, 36
class String, 57

comments, 23
constant expressions, 536
creation of new class instances, 328
escape sequences for character and String literals, 38
lexical literals, 430
reference equality operators == and !=, 517
unicode, 16

strings
lexical literals, 430
literals, 25
objects, 54
String literals, 37, 37

subsequent modification of final fields, 587

subtyping, 72
checked casts and unchecked casts, 115
method throws, 227
narrowing reference conversion, 93
parameterized types, 60
phase 1: identify matching arity methods applicable by subtyping, 460
type arguments and wildcards, 63
widening reference conversion, 93

subtyping among array types, 73
array types, 292

subtyping among class and interface types, 73
try statement, 402

subtyping among primitive types, 72

superclasses and subclasses, 190
class members, 196
class Object, 56
enums, 254
final classes, 184
kinds of variables, 78
loading process, 319
syntactic classification of a name according to context, 141

superclasses and superinterfaces, 345
loading process, 319
superinterfaces, 362
verification of the binary representation, 321

superinterfaces, 192, 362
abstract classes, 182
checked casts at run time, 116
class members, 196
superinterfaces and subinterfaces, 267
syntactic classification of a name according to context, 141
types, classes, and interfaces, 83

superinterfaces and subinterfaces, 266
loading process, 319
superclasses and subclasses, 192
superinterfaces, 193
syntactic classification of a name according to context, 141

switch statement, 379
scope of a declaration, 131
switch statements, 553

switch statements, 553

synchronization, 564
objects, 56
synchronized methods, 224
synchronized statement, 399
volatile fields, 209

synchronization order, 574
actions, 572
interaction with the memory model, 334

synchronized methods, 224, 359
class Object, 57
synchronization, 564
synchronized statement, 400

synchronized statement, 399
create frame, synchronize, transfer control, 489
objects, 56
synchronization, 564

synchronized statements, 556
synchronized statements, 556
syntactic classification of a name according to context, 141
access control, 152
syntactic grammar, 10
compilation units, 167
input elements and tokens, 20
lexical translations, 16
syntax, 591

T

this, 430
initialization of fields in interfaces, 271
initializers for class variables, 212
initializers for instance variables, 212
instance initializers, 243
static initializers, 244
static methods, 222
threads and locks, 563
objects, 56
throw statement, 397
throw statement, 397
break, continue, return, and throw statements, 556
causes of Exceptions, 305
exception analysis of statements, 308
initial values of variables, 82
kinds of variables, 79
normal and abrupt completion of statements, 366, 366
run-time handling of an exception, 311
top level type declarations, 175
class instance creation expressions, 433
class modifiers, 182
compilation units, 167
determining accessibility, 154
form of a binary, 338
host support for packages, 165
interface modifiers, 265
package members, 163, 164
scope of a declaration, 130, 130, 130
shadowing, 137
single-static-import declarations, 174
single-type-import declarations, 172
when initialization occurs, 323
transient fields, 209, 354
try statement, 400
array variables, 293
compile-time checking of Exceptions, 307, 308
declarations, 123
definite assignment and parameters, 558
exception analysis of statements, 309
expressions and run-time checks, 420, 421
final variables, 81, 81
initial values of variables, 82
kinds of variables, 79
labeled statements, 374
reclassification of contextually ambiguous names, 143
run-time handling of an exception, 311
scope of a declaration, 131
shadowing and obscuring, 133
shared variables, 572
syntactic classification of a name according to context, 142
throw statement, 397, 398, 398
try statements, 556
try statements, 556
try-catch statement
try statement, 403
try-catch-finally statement
try statement, 403
try-finally statement

try statement, 403
try-with-resources, 407
final variables, 81
scope of a declaration, 131
shadowing and obscuring, 134
try statement, 403
try-with-resources (basic)
extended try-with-resources, 411
type arguments and wildcards, 61
capture conversion, 97
checked casts and unchecked casts, 115
class instance creation expressions, 433, 433
reference types and values, 53
reifiable types, 66
subtyping among class and interface types, 73
syntactic classification of a name according to context, 141, 141, 141
types, classes, and interfaces, 83
unchecked conversion, 97
type comparison operator instanceof, 513
expressions and run-time checks, 420
objects, 55
syntactic classification of a name according to context, 142
type erasure, 65
assignment conversion, 103
checked casts and unchecked casts, 115
checked casts at run time, 116
choosing the most specific method, 464
class type parameters, 347
compile-time step 3: is the chosen method appropriate?, 480
constructor Signature, 245
create frame, synchronize, transfer control, 489
evaluate arguments, 483
field declarations, 352
form of a binary, 340, 341, 341, 342
inferring type arguments based on actual arguments, 475
method and constructor formal parameters, 356
method and constructor type parameters, 356
method invocation conversion, 107
method result and throws types, 465, 465
method result type, 357
method Signature, 219
phase 3: identify applicable variable arity methods, 462
raw types, 67
requirements in overriding and hiding, 234
type variables, 59
type of a constructor, 247
members and constructors of parameterized types, 64
type of an expression, 418
type variables, 58
class literals, 430
declarations, 122
field declarations, 352
generic classes and type parameters, 185
generic constructors, 246
generic interfaces and type parameters, 265
generic methods, 226
method and constructor formal parameters, 356
method result type, 357
reference types and values, 52
type erasure, 65
types, classes, and interfaces, 83
type-import-on-demand declarations, 173
declarations, 122
import declarations, 170
reclassification of contextually ambiguous names, 144

scope of a declaration, 130
shadowing, 137
static-import-on-demand declarations, 175
syntactic classification of a name according to context, 143
types
capture conversion, 97
lexical literals, 430
literals, 25
null literal, 39
throw statement, 398
types, classes, and interfaces, 82
types, values, and variables, 41

U

unary minus operator -, 498
constant expressions, 536
floating-point operations, 48
integer literals, 31, 31
integer operations, 43
unary numeric promotion, 118
unary numeric promotion, 118
array access, 295
array access expressions, 490
array creation expressions, 443
bitwise complement operator ~, 499
numeric promotions, 118
shift operators, 511
unary minus operator -, 498
unary plus operator +, 497
unary operators, 495
unary plus operator +, 497
constant expressions, 536
floating-point operations, 48
integer operations, 43
unary numeric promotion, 118
unboxing conversion, 95
additive operators, 506, 506
array creation expressions, 443
assert statement, 378
assignment conversion, 101
binary numeric promotion, 119
bitwise complement operator ~, 498
boolean equality operators == and !=, 516
boolean logical operators &, ^, and |, 519
casting conversion, 108, 109, 109
conditional operator ? :, 521, 522
conditional-and operator &&, 519, 520
conditional-or operator ||, 520, 520
do statement, 385
equality operators, 515
floating-point operations, 49
if-then statement, 375
if-then-else statement, 376
integer bitwise operators &, ^, and |, 518
integer operations, 44
iteration of for statement, 388
logical complement operator !, 499
method invocation conversion, 107
multiplicative operators, 500
numeric promotions, 117
numerical comparison operators <, <=, >, and >=, 512
numerical equality operators == and !=, 515
postfix decrement operator --, 494
postfix increment operator ++, 494
prefix decrement operator --, 497
prefix increment operator ++, 496
switch statement, 381
unary minus operator -, 498
unary numeric promotion, 118, 118
unary plus operator +, 497
while statement, 384
unchecked conversion, 97
@SafeVarargs, 284

assignment conversion, 102
casting conversion, 108, 109
members and constructors of parameterized types, 64
method invocation conversion, 107
method return type, 227
phase 1: identify matching arity methods applicable by subtyping, 460
variables of reference type, 76
unicode, 15
character literals, 35
lexical grammar, 9
primitive types and values, 42
unicode escapes, 17
unicode escapes, 17
escape sequences for character and String literals, 39
input elements and tokens, 19
lexical translations, 16
unicode, 16
unloading of classes and interfaces, 335
kinds of variables, 78
unnamed packages, 169
compilation units, 167
unreachable statements, 411
final fields and constants, 352
instance initializers, 243
static initializers, 243

V

value set conversion, 100
assignment conversion, 103
binary numeric promotion, 119
casting conversion, 109
compound assignment operators, 529, 531
create frame, synchronize, transfer control, 489
evaluation, denotation, and result, 418
floating-point types, formats, and values, 45
fp-strict expressions, 419
method invocation conversion, 107
simple assignment operator =, 525, 526
unary minus operator -, 498
unary numeric promotion, 118
variables as values, 418
variables, 75
evaluation, denotation, and result, 417
variables as values, 418
variables of primitive type, 76
variables of reference type, 76
@SafeVarargs, 284
type of an expression, 418
variables, 75
verification of the binary representation, 320
link test: verify, prepare, (optionally) resolve, 316
volatile fields, 209
happens-before order, 576
synchronization order, 574

W

wait, 565
happens-before order, 575
wait sets and notification, 564
class Object, 57
well-formed executions, 579
executions, 578
what binary compatibility is and is not, 343
when initialization occurs, 323
final variables, 81
initialize test: execute initializers, 317
when reference types are the same, 57
checked casts at run time, 116
where types are used, 74

while statement, 383
  boolean type and boolean values, 51
  while statements, 553
while statements, 553
white space, 21
  input elements and tokens, 20, 20
  lexical grammar, 9
  lexical translations, 16
widening and narrowing primitive conversion, 93
  casting conversion, 108
widening primitive conversion, 88
  assignment conversion, 101
  binary numeric promotion, 119
  casting conversion, 108, 109
  method invocation conversion, 106
  numeric promotions, 117
  unary numeric promotion, 118, 118
  widening and narrowing primitive conversion, 93
widening reference conversion, 93
  assignment conversion, 101
  casting conversion, 108, 109
  floating-point operations, 48
  integer operations, 44
  method invocation conversion, 107
word tearing, 589
write-protected fields, 588