



# The Java<sup>®</sup> Language Specification

Java SE 8 Edition

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



ORACLE<sup>®</sup>

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



The Java<sup>®</sup> Language  
Specification  
*Java SE 8 Edition*

*This page intentionally left blank*

# The Java<sup>®</sup> Language Specification *Java SE 8 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, 2014, 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](mailto:corpsales@pearsontechgroup.com). For sales outside the United States, please contact International Sales, [international@pearson.com](mailto:international@pearson.com).

Visit us on the Web: [informit.com/aw](http://informit.com/aw)

Library of Congress Control Number: 2014936248

ISBN-13: 978-0-13-390069-9

ISBN-10: 0-13-390069-X

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 Edwards Brothers Malloy in Ann Arbor, Michigan. First printing, May 2014.

To Maurizio, with deepest thanks.

*This page intentionally left blank*

# Table of Contents

---

## **Preface to the Java SE 8 Edition xxi**

## **1 Introduction 1**

- 1.1 Organization of the Specification 2
- 1.2 Example Programs 6
- 1.3 Notation 6
- 1.4 Relationship to Predefined Classes and Interfaces 7
- 1.5 Feedback 7
- 1.6 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 19
- 3.5 Input Elements and Tokens 19
- 3.6 White Space 20
- 3.7 Comments 21
- 3.8 Identifiers 22
- 3.9 Keywords 24
- 3.10 Literals 24
  - 3.10.1 Integer Literals 25
  - 3.10.2 Floating-Point Literals 31
  - 3.10.3 Boolean Literals 34
  - 3.10.4 Character Literals 34
  - 3.10.5 String Literals 35
  - 3.10.6 Escape Sequences for Character and String Literals 37
  - 3.10.7 The Null Literal 38
- 3.11 Separators 38
- 3.12 Operators 38

## 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 53
  - 4.3.2 The Class `Object` 55
  - 4.3.3 The Class `String` 56
  - 4.3.4 When Reference Types Are the Same 56
- 4.4 Type Variables 57
- 4.5 Parameterized Types 59
  - 4.5.1 Type Arguments of Parameterized Types 60
  - 4.5.2 Members and Constructors of Parameterized Types 63
- 4.6 Type Erasure 64
- 4.7 Reifiable Types 64
- 4.8 Raw Types 66
- 4.9 Intersection Types 70
- 4.10 Subtyping 71
  - 4.10.1 Subtyping among Primitive Types 71
  - 4.10.2 Subtyping among Class and Interface Types 71
  - 4.10.3 Subtyping among Array Types 73
  - 4.10.4 Least Upper Bound 73
- 4.11 Where Types Are Used 75
- 4.12 Variables 80
  - 4.12.1 Variables of Primitive Type 81
  - 4.12.2 Variables of Reference Type 81
  - 4.12.3 Kinds of Variables 83
  - 4.12.4 `final` Variables 85
  - 4.12.5 Initial Values of Variables 87
  - 4.12.6 Types, Classes, and Interfaces 88

## 5 Conversions and Contexts 91

- 5.1 Kinds of Conversion 94
  - 5.1.1 Identity Conversion 94
  - 5.1.2 Widening Primitive Conversion 94
  - 5.1.3 Narrowing Primitive Conversion 96
  - 5.1.4 Widening and Narrowing Primitive Conversion 99
  - 5.1.5 Widening Reference Conversion 99
  - 5.1.6 Narrowing Reference Conversion 99
  - 5.1.7 Boxing Conversion 100
  - 5.1.8 Unboxing Conversion 102
  - 5.1.9 Unchecked Conversion 103
  - 5.1.10 Capture Conversion 103

- 5.1.11 String Conversion 105
- 5.1.12 Forbidden Conversions 106
- 5.1.13 Value Set Conversion 106
- 5.2 Assignment Contexts 107
- 5.3 Invocation Contexts 112
- 5.4 String Contexts 114
- 5.5 Casting Contexts 114
  - 5.5.1 Reference Type Casting 117
  - 5.5.2 Checked Casts and Unchecked Casts 121
  - 5.5.3 Checked Casts at Run Time 122
- 5.6 Numeric Contexts 124
  - 5.6.1 Unary Numeric Promotion 124
  - 5.6.2 Binary Numeric Promotion 125

## 6 Names 129

- 6.1 Declarations 130
- 6.2 Names and Identifiers 137
- 6.3 Scope of a Declaration 139
- 6.4 Shadowing and Obscuring 142
  - 6.4.1 Shadowing 144
  - 6.4.2 Obscuring 147
- 6.5 Determining the Meaning of a Name 148
  - 6.5.1 Syntactic Classification of a Name According to Context 149
  - 6.5.2 Reclassification of Contextually Ambiguous Names 152
  - 6.5.3 Meaning of Package Names 154
    - 6.5.3.1 Simple Package Names 155
    - 6.5.3.2 Qualified Package Names 155
  - 6.5.4 Meaning of *PackageOrTypeNames* 155
    - 6.5.4.1 Simple *PackageOrTypeNames* 155
    - 6.5.4.2 Qualified *PackageOrTypeNames* 155
  - 6.5.5 Meaning of Type Names 155
    - 6.5.5.1 Simple Type Names 156
    - 6.5.5.2 Qualified Type Names 156
  - 6.5.6 Meaning of Expression Names 156
    - 6.5.6.1 Simple Expression Names 156
    - 6.5.6.2 Qualified Expression Names 157
  - 6.5.7 Meaning of Method Names 160
    - 6.5.7.1 Simple Method Names 160
- 6.6 Access Control 161
  - 6.6.1 Determining Accessibility 162
  - 6.6.2 Details on `protected` Access 166
    - 6.6.2.1 Access to a `protected` Member 167
    - 6.6.2.2 Qualified Access to a `protected` Constructor 167
- 6.7 Fully Qualified Names and Canonical Names 169

## 7 Packages 173

- 7.1 Package Members 173

- 7.2 Host Support for Packages 175
- 7.3 Compilation Units 177
- 7.4 Package Declarations 178
  - 7.4.1 Named Packages 178
  - 7.4.2 Unnamed Packages 179
  - 7.4.3 Observability of a Package 179
- 7.5 Import Declarations 180
  - 7.5.1 Single-Type-Import Declarations 180
  - 7.5.2 Type-Import-on-Demand Declarations 183
  - 7.5.3 Single-Static-Import Declarations 184
  - 7.5.4 Static-Import-on-Demand Declarations 184
- 7.6 Top Level Type Declarations 185

## 8 Classes 189

- 8.1 Class Declarations 191
  - 8.1.1 Class Modifiers 191
    - 8.1.1.1 `abstract` Classes 192
    - 8.1.1.2 `final` Classes 194
    - 8.1.1.3 `strictfp` Classes 194
  - 8.1.2 Generic Classes and Type Parameters 194
  - 8.1.3 Inner Classes and Enclosing Instances 197
  - 8.1.4 Superclasses and Subclasses 200
  - 8.1.5 Superinterfaces 202
  - 8.1.6 Class Body and Member Declarations 205
- 8.2 Class Members 206
- 8.3 Field Declarations 211
  - 8.3.1 Field Modifiers 215
    - 8.3.1.1 `static` Fields 216
    - 8.3.1.2 `final` Fields 219
    - 8.3.1.3 `transient` Fields 219
    - 8.3.1.4 `volatile` Fields 220
  - 8.3.2 Field Initialization 221
  - 8.3.3 Forward References During Field Initialization 222
- 8.4 Method Declarations 225
  - 8.4.1 Formal Parameters 226
  - 8.4.2 Method Signature 230
  - 8.4.3 Method Modifiers 231
    - 8.4.3.1 `abstract` Methods 232
    - 8.4.3.2 `static` Methods 233
    - 8.4.3.3 `final` Methods 234
    - 8.4.3.4 `native` Methods 235
    - 8.4.3.5 `strictfp` Methods 235
    - 8.4.3.6 `synchronized` Methods 235
  - 8.4.4 Generic Methods 237
  - 8.4.5 Method Result 237
  - 8.4.6 Method Throws 238
  - 8.4.7 Method Body 240

- 8.4.8 Inheritance, Overriding, and Hiding 240
  - 8.4.8.1 Overriding (by Instance Methods) 241
  - 8.4.8.2 Hiding (by Class Methods) 245
  - 8.4.8.3 Requirements in Overriding and Hiding 246
  - 8.4.8.4 Inheriting Methods with Override-Equivalent Signatures 250
- 8.4.9 Overloading 250
- 8.5 Member Type Declarations 254
  - 8.5.1 Static Member Type Declarations 254
- 8.6 Instance Initializers 255
- 8.7 Static Initializers 255
- 8.8 Constructor Declarations 256
  - 8.8.1 Formal Parameters 257
  - 8.8.2 Constructor Signature 258
  - 8.8.3 Constructor Modifiers 258
  - 8.8.4 Generic Constructors 259
  - 8.8.5 Constructor Throws 259
  - 8.8.6 The Type of a Constructor 259
  - 8.8.7 Constructor Body 259
    - 8.8.7.1 Explicit Constructor Invocations 260
  - 8.8.8 Constructor Overloading 264
  - 8.8.9 Default Constructor 265
  - 8.8.10 Preventing Instantiation of a Class 266
- 8.9 Enum Types 266
  - 8.9.1 Enum Constants 267
  - 8.9.2 Enum Body Declarations 268
  - 8.9.3 Enum Members 271

## 9 Interfaces 277

- 9.1 Interface Declarations 278
  - 9.1.1 Interface Modifiers 278
    - 9.1.1.1 `abstract` Interfaces 279
    - 9.1.1.2 `strictfp` Interfaces 279
  - 9.1.2 Generic Interfaces and Type Parameters 279
  - 9.1.3 Superinterfaces and Subinterfaces 280
  - 9.1.4 Interface Body and Member Declarations 282
- 9.2 Interface Members 282
- 9.3 Field (Constant) Declarations 283
  - 9.3.1 Initialization of Fields in Interfaces 285
- 9.4 Method Declarations 286
  - 9.4.1 Inheritance and Overriding 287
    - 9.4.1.1 Overriding (by Instance Methods) 288
    - 9.4.1.2 Requirements in Overriding 289
    - 9.4.1.3 Inheriting Methods with Override-Equivalent Signatures 289
  - 9.4.2 Overloading 290
  - 9.4.3 Interface Method Body 291

- 9.5 Member Type Declarations 291
- 9.6 Annotation Types 292
  - 9.6.1 Annotation Type Elements 293
  - 9.6.2 Defaults for Annotation Type Elements 297
  - 9.6.3 Repeatable Annotation Types 298
  - 9.6.4 Predefined Annotation Types 302
    - 9.6.4.1 @Target 302
    - 9.6.4.2 @Retention 303
    - 9.6.4.3 @Inherited 304
    - 9.6.4.4 @Override 304
    - 9.6.4.5 @SuppressWarnings 305
    - 9.6.4.6 @Deprecated 306
    - 9.6.4.7 @SafeVarargs 307
    - 9.6.4.8 @Repeatable 308
    - 9.6.4.9 @FunctionalInterface 308
- 9.7 Annotations 308
  - 9.7.1 Normal Annotations 309
  - 9.7.2 Marker Annotations 311
  - 9.7.3 Single-Element Annotations 312
  - 9.7.4 Where Annotations May Appear 313
  - 9.7.5 Multiple Annotations of the Same Type 318
- 9.8 Functional Interfaces 319
- 9.9 Function Types 323

## 10 Arrays 329

- 10.1 Array Types 330
- 10.2 Array Variables 330
- 10.3 Array Creation 332
- 10.4 Array Access 332
- 10.5 Array Store Exception 333
- 10.6 Array Initializers 335
- 10.7 Array Members 336
- 10.8 `Class` Objects for Arrays 338
- 10.9 An Array of Characters Is Not a `String` 339

## 11 Exceptions 341

- 11.1 The Kinds and Causes of Exceptions 342
  - 11.1.1 The Kinds of Exceptions 342
  - 11.1.2 The Causes of Exceptions 343
  - 11.1.3 Asynchronous Exceptions 343
- 11.2 Compile-Time Checking of Exceptions 344
  - 11.2.1 Exception Analysis of Expressions 346
  - 11.2.2 Exception Analysis of Statements 346
  - 11.2.3 Exception Checking 347
- 11.3 Run-Time Handling of an Exception 349

## 12 Execution 353

- 12.1 Java Virtual Machine Startup 353
  - 12.1.1 Load the Class `Test` 354
  - 12.1.2 Link `Test`: Verify, Prepare, (Optionally) Resolve 354
  - 12.1.3 Initialize `Test`: Execute Initializers 355
  - 12.1.4 Invoke `Test.main` 356
- 12.2 Loading of Classes and Interfaces 356
  - 12.2.1 The Loading Process 357
- 12.3 Linking of Classes and Interfaces 358
  - 12.3.1 Verification of the Binary Representation 358
  - 12.3.2 Preparation of a Class or Interface Type 359
  - 12.3.3 Resolution of Symbolic References 359
- 12.4 Initialization of Classes and Interfaces 360
  - 12.4.1 When Initialization Occurs 361
  - 12.4.2 Detailed Initialization Procedure 363
- 12.5 Creation of New Class Instances 365
- 12.6 Finalization of Class Instances 369
  - 12.6.1 Implementing Finalization 370
  - 12.6.2 Interaction with the Memory Model 372
- 12.7 Unloading of Classes and Interfaces 373
- 12.8 Program Exit 374

## 13 Binary Compatibility 375

- 13.1 The Form of a Binary 376
- 13.2 What Binary Compatibility Is and Is Not 382
- 13.3 Evolution of Packages 383
- 13.4 Evolution of Classes 383
  - 13.4.1 `abstract` Classes 383
  - 13.4.2 `final` Classes 383
  - 13.4.3 `public` Classes 384
  - 13.4.4 Superclasses and Superinterfaces 384
  - 13.4.5 Class Type Parameters 385
  - 13.4.6 Class Body and Member Declarations 386
  - 13.4.7 Access to Members and Constructors 387
  - 13.4.8 Field Declarations 388
  - 13.4.9 `final` Fields and `static` Constant Variables 391
  - 13.4.10 `static` Fields 393
  - 13.4.11 `transient` Fields 393
  - 13.4.12 Method and Constructor Declarations 394
  - 13.4.13 Method and Constructor Type Parameters 394
  - 13.4.14 Method and Constructor Formal Parameters 395
  - 13.4.15 Method Result Type 396
  - 13.4.16 `abstract` Methods 396
  - 13.4.17 `final` Methods 397
  - 13.4.18 `native` Methods 397
  - 13.4.19 `static` Methods 398
  - 13.4.20 `synchronized` Methods 398

- 13.4.21 Method and Constructor Throws 398
- 13.4.22 Method and Constructor Body 398
- 13.4.23 Method and Constructor Overloading 399
- 13.4.24 Method Overriding 400
- 13.4.25 Static Initializers 400
- 13.4.26 Evolution of Enums 400
- 13.5 Evolution of Interfaces 400
  - 13.5.1 `public` Interfaces 400
  - 13.5.2 Superinterfaces 401
  - 13.5.3 Interface Members 401
  - 13.5.4 Interface Type Parameters 401
  - 13.5.5 Field Declarations 402
  - 13.5.6 Interface Method Declarations 402
  - 13.5.7 Evolution of Annotation Types 403

## 14 Blocks and Statements 405

- 14.1 Normal and Abrupt Completion of Statements 405
- 14.2 Blocks 407
- 14.3 Local Class Declarations 407
- 14.4 Local Variable Declaration Statements 408
  - 14.4.1 Local Variable Declarators and Types 409
  - 14.4.2 Execution of Local Variable Declarations 410
- 14.5 Statements 410
- 14.6 The Empty Statement 412
- 14.7 Labeled Statements 413
- 14.8 Expression Statements 414
- 14.9 The `if` Statement 415
  - 14.9.1 The `if-then` Statement 415
  - 14.9.2 The `if-then-else` Statement 416
- 14.10 The `assert` Statement 416
- 14.11 The `switch` Statement 419
- 14.12 The `while` Statement 423
  - 14.12.1 Abrupt Completion of `while` Statement 424
- 14.13 The `do` Statement 424
  - 14.13.1 Abrupt Completion of `do` Statement 425
- 14.14 The `for` Statement 426
  - 14.14.1 The basic `for` Statement 426
    - 14.14.1.1 Initialization of `for` Statement 427
    - 14.14.1.2 Iteration of `for` Statement 427
    - 14.14.1.3 Abrupt Completion of `for` Statement 428
  - 14.14.2 The enhanced `for` statement 429
- 14.15 The `break` Statement 432
- 14.16 The `continue` Statement 434
- 14.17 The `return` Statement 436
- 14.18 The `throw` Statement 437
- 14.19 The `synchronized` Statement 439
- 14.20 The `try` statement 440

- 14.20.1 Execution of `try-catch` 444
- 14.20.2 Execution of `try-finally` and `try-catch-finally` 445
- 14.20.3 `try-with-resources` 447
  - 14.20.3.1 Basic `try-with-resources` 448
  - 14.20.3.2 Extended `try-with-resources` 451
- 14.21 Unreachable Statements 452

## 15 Expressions 459

- 15.1 Evaluation, Denotation, and Result 459
- 15.2 Forms of Expressions 460
- 15.3 Type of an Expression 461
- 15.4 FP-strict Expressions 462
- 15.5 Expressions and Run-Time Checks 462
- 15.6 Normal and Abrupt Completion of Evaluation 464
- 15.7 Evaluation Order 466
  - 15.7.1 Evaluate Left-Hand Operand First 466
  - 15.7.2 Evaluate Operands before Operation 468
  - 15.7.3 Evaluation Respects Parentheses and Precedence 469
  - 15.7.4 Argument Lists are Evaluated Left-to-Right 470
  - 15.7.5 Evaluation Order for Other Expressions 471
- 15.8 Primary Expressions 471
  - 15.8.1 Lexical Literals 472
  - 15.8.2 Class Literals 473
  - 15.8.3 `this` 474
  - 15.8.4 Qualified `this` 475
  - 15.8.5 Parenthesized Expressions 475
- 15.9 Class Instance Creation Expressions 476
  - 15.9.1 Determining the Class being Instantiated 478
  - 15.9.2 Determining Enclosing Instances 480
  - 15.9.3 Choosing the Constructor and its Arguments 481
  - 15.9.4 Run-Time Evaluation of Class Instance Creation Expressions 484
  - 15.9.5 Anonymous Class Declarations 485
    - 15.9.5.1 Anonymous Constructors 485
- 15.10 Array Creation and Access Expressions 487
  - 15.10.1 Array Creation Expressions 487
  - 15.10.2 Run-Time Evaluation of Array Creation Expressions 488
  - 15.10.3 Array Access Expressions 491
  - 15.10.4 Run-Time Evaluation of Array Access Expressions 492
- 15.11 Field Access Expressions 494
  - 15.11.1 Field Access Using a Primary 494
  - 15.11.2 Accessing Superclass Members using `super` 497
- 15.12 Method Invocation Expressions 499
  - 15.12.1 Compile-Time Step 1: Determine Class or Interface to Search 500
  - 15.12.2 Compile-Time Step 2: Determine Method Signature 502
    - 15.12.2.1 Identify Potentially Applicable Methods 509

- 15.12.2.2 Phase 1: Identify Matching Arity Methods Applicable by Strict Invocation 511
- 15.12.2.3 Phase 2: Identify Matching Arity Methods Applicable by Loose Invocation 512
- 15.12.2.4 Phase 3: Identify Methods Applicable by Variable Arity Invocation 513
- 15.12.2.5 Choosing the Most Specific Method 514
- 15.12.2.6 Method Invocation Type 516
- 15.12.3 Compile-Time Step 3: Is the Chosen Method Appropriate? 517
- 15.12.4 Run-Time Evaluation of Method Invocation 520
  - 15.12.4.1 Compute Target Reference (If Necessary) 520
  - 15.12.4.2 Evaluate Arguments 522
  - 15.12.4.3 Check Accessibility of Type and Method 523
  - 15.12.4.4 Locate Method to Invoke 524
  - 15.12.4.5 Create Frame, Synchronize, Transfer Control 528
- 15.13 Method Reference Expressions 529
  - 15.13.1 Compile-Time Declaration of a Method Reference 532
  - 15.13.2 Type of a Method Reference 537
  - 15.13.3 Run-Time Evaluation of Method References 539
- 15.14 Postfix Expressions 542
  - 15.14.1 Expression Names 543
  - 15.14.2 Postfix Increment Operator ++ 543
  - 15.14.3 Postfix Decrement Operator -- 544
- 15.15 Unary Operators 544
  - 15.15.1 Prefix Increment Operator ++ 546
  - 15.15.2 Prefix Decrement Operator -- 546
  - 15.15.3 Unary Plus Operator + 547
  - 15.15.4 Unary Minus Operator - 547
  - 15.15.5 Bitwise Complement Operator ~ 548
  - 15.15.6 Logical Complement Operator ! 548
- 15.16 Cast Expressions 549
- 15.17 Multiplicative Operators 550
  - 15.17.1 Multiplication Operator \* 551
  - 15.17.2 Division Operator / 552
  - 15.17.3 Remainder Operator % 554
- 15.18 Additive Operators 556
  - 15.18.1 String Concatenation Operator + 557
  - 15.18.2 Additive Operators (+ and -) for Numeric Types 559
- 15.19 Shift Operators 561
- 15.20 Relational Operators 562
  - 15.20.1 Numerical Comparison Operators <, <=, >, and >= 563
  - 15.20.2 Type Comparison Operator instanceof 564
- 15.21 Equality Operators 565
  - 15.21.1 Numerical Equality Operators == and != 566
  - 15.21.2 Boolean Equality Operators == and != 567
  - 15.21.3 Reference Equality Operators == and != 567
- 15.22 Bitwise and Logical Operators 568
  - 15.22.1 Integer Bitwise Operators &, ^, and | 568

- 15.22.2 Boolean Logical Operators `&`, `^`, and `|` 569
- 15.23 Conditional-And Operator `&&` 570
- 15.24 Conditional-Or Operator `||` 570
- 15.25 Conditional Operator `?` : 571
  - 15.25.1 Boolean Conditional Expressions 579
  - 15.25.2 Numeric Conditional Expressions 579
  - 15.25.3 Reference Conditional Expressions 580
- 15.26 Assignment Operators 581
  - 15.26.1 Simple Assignment Operator `=` 582
  - 15.26.2 Compound Assignment Operators 588
- 15.27 Lambda Expressions 594
  - 15.27.1 Lambda Parameters 596
  - 15.27.2 Lambda Body 599
  - 15.27.3 Type of a Lambda Expression 602
  - 15.27.4 Run-Time Evaluation of Lambda Expressions 604
- 15.28 Constant Expressions 605

## 16 Definite Assignment 607

- 16.1 Definite Assignment and Expressions 613
  - 16.1.1 Boolean Constant Expressions 613
  - 16.1.2 Conditional-And Operator `&&` 613
  - 16.1.3 Conditional-Or Operator `||` 614
  - 16.1.4 Logical Complement Operator `!` 614
  - 16.1.5 Conditional Operator `?` : 614
  - 16.1.6 Conditional Operator `?` : 615
  - 16.1.7 Other Expressions of Type `boolean` 615
  - 16.1.8 Assignment Expressions 615
  - 16.1.9 Operators `++` and `--` 616
  - 16.1.10 Other Expressions 616
- 16.2 Definite Assignment and Statements 617
  - 16.2.1 Empty Statements 617
  - 16.2.2 Blocks 617
  - 16.2.3 Local Class Declaration Statements 619
  - 16.2.4 Local Variable Declaration Statements 619
  - 16.2.5 Labeled Statements 619
  - 16.2.6 Expression Statements 620
  - 16.2.7 `if` Statements 620
  - 16.2.8 `assert` Statements 620
  - 16.2.9 `switch` Statements 621
  - 16.2.10 `while` Statements 621
  - 16.2.11 `do` Statements 622
  - 16.2.12 `for` Statements 622
    - 16.2.12.1 Initialization Part of `for` Statement 623
    - 16.2.12.2 Incrementation Part of `for` Statement 623
  - 16.2.13 `break`, `continue`, `return`, and `throw` Statements 624
  - 16.2.14 `synchronized` Statements 624
  - 16.2.15 `try` Statements 624

- 16.3 Definite Assignment and Parameters 626
- 16.4 Definite Assignment and Array Initializers 626
- 16.5 Definite Assignment and Enum Constants 626
- 16.6 Definite Assignment and Anonymous Classes 627
- 16.7 Definite Assignment and Member Types 627
- 16.8 Definite Assignment and Static Initializers 628
- 16.9 Definite Assignment, Constructors, and Instance Initializers 628

## **17 Threads and Locks 631**

- 17.1 Synchronization 632
- 17.2 Wait Sets and Notification 632
  - 17.2.1 Wait 633
  - 17.2.2 Notification 634
  - 17.2.3 Interruptions 635
  - 17.2.4 Interactions of Waits, Notification, and Interruption 635
- 17.3 Sleep and Yield 636
- 17.4 Memory Model 637
  - 17.4.1 Shared Variables 640
  - 17.4.2 Actions 640
  - 17.4.3 Programs and Program Order 641
  - 17.4.4 Synchronization Order 642
  - 17.4.5 Happens-before Order 643
  - 17.4.6 Executions 646
  - 17.4.7 Well-Formed Executions 647
  - 17.4.8 Executions and Causality Requirements 647
  - 17.4.9 Observable Behavior and Nonterminating Executions 650
- 17.5 `final` Field Semantics 652
  - 17.5.1 Semantics of `final` Fields 654
  - 17.5.2 Reading `final` Fields During Construction 654
  - 17.5.3 Subsequent Modification of `final` Fields 655
  - 17.5.4 Write-Protected Fields 656
- 17.6 Word Tearing 657
- 17.7 Non-Atomic Treatment of `double` and `long` 658

## **18 Type Inference 659**

- 18.1 Concepts and Notation 660
  - 18.1.1 Inference Variables 660
  - 18.1.2 Constraint Formulas 661
  - 18.1.3 Bounds 661
- 18.2 Reduction 663
  - 18.2.1 Expression Compatibility Constraints 663
  - 18.2.2 Type Compatibility Constraints 667
  - 18.2.3 Subtyping Constraints 668
  - 18.2.4 Type Equality Constraints 670
  - 18.2.5 Checked Exception Constraints 671
- 18.3 Incorporation 673
  - 18.3.1 Complementary Pairs of Bounds 674

18.3.2	Bounds Involving Capture Conversion	674
18.4	Resolution	675
18.5	Uses of Inference	677
18.5.1	Invocation Applicability Inference	678
18.5.2	Invocation Type Inference	679
18.5.3	Functional Interface Parameterization Inference	685
18.5.4	More Specific Method Inference	686

## **19 Syntax 689**

### **Index 715**

### **A Limited License Grant 755**

*This page intentionally left blank*

# Preface to the Java SE 8 Edition

---

**I**N 1996, James Gosling, Bill Joy, and Guy Steele wrote for the First Edition of *The Java® Language Specification*:

*"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."*

Java SE 8 represents the single largest evolution of the Java language in its history. A relatively small number of features - lambda expressions, method references, and functional interfaces - combine to offer a programming model that fuses the object-oriented and functional styles. Under the leadership of Brian Goetz, this fusion has been accomplished in a way that encourages best practices - immutability, statelessness, compositionality - while preserving "the feel of Java" - readability, simplicity, universality.

Crucially, the libraries of the Java SE platform have co-evolved with the Java language. This means that using lambda expressions and method references to represent behavior - for example, an operation to be applied to each element in a list - is productive and performant "out of the box". In a similar fashion, the Java Virtual Machine has co-evolved with the Java language to ensure that default methods support library evolution as consistently as possible across compile time and run time, given the constraints of separate compilation.

Initiatives to add first-class functions to the Java language have been around since the 1990s. The *BGGA* and *CICE* proposals circa 2007 brought new energy to the topic, while the creation of *Project Lambda* in OpenJDK circa 2009 attracted unprecedented levels of interest. The addition of method handles to the JVM in Java SE 7 opened the door to new implementation techniques while retaining "write once, run anywhere." In time, language changes were overseen by JSR 335, *Lambda Expressions for the Java Programming Language*, whose Expert Group consisted of Joshua Bloch, Kevin Bourrillion, Andrey Breslav, Rémi Forax, Dan Heidinga, Doug Lea, Bob Lee, David Lloyd, Sam Pullara, Srikanth Sankaran, and Vladimir Zakharov.

Programming language design typically involves grappling with degrees of complexity utterly hidden from the language's users. (For this reason, it is often compared to an iceberg: 90% of it is invisible.) In JSR 335, the greatest complexity

lurked in the interaction of implicitly typed lambda expressions with overload resolution. In this and many other areas, Dan Smith at Oracle did an outstanding job of thoroughly specifying the desired behavior. His words are to be found throughout this specification, including an entirely new chapter on type inference.

Another initiative in Java SE 8 has been to enhance the utility of annotations, one of the most popular features of the Java language. First, the Java grammar has been extended to allow annotations on types in many language constructs, forming the basis for novel static analysis tools such as the *Checker Framework*. This feature was specified by JSR 308, *Annotations on Java Types*, led by Michael Ernst with an Expert Group of myself, Doug Lea, and Srikanth Sankaran. The changes involved in this specification were wide-ranging, and the unstinting efforts of Michael Ernst and Werner Dietl over many years are warmly recognized. Second, annotations may be "repeated" on a language construct, to the great benefit of APIs that model domain-specific configuration with annotation types. Michael Keith and Bill Shannon in Java EE initiated and guided this feature.

Many colleagues in the Java Platform Group at Oracle have provided valuable support to this specification: Leonid Arbousov, Mandy Chung, Joe Darcy, Robert Field, Joel Franck, Sonali Goel, Jon Gibbons, Jeannette Hung, Stuart Marks, Eric McCorkle, Matherey Nunez, Mark Reinhold, Vicente Romero, John Rose, Georges Saab, Steve Sides, Bernard Traversat, and Michel Trudeau.

Perhaps the greatest acknowledgement must go to the compiler engineers who turn the specification into real software. Maurizio Cimadamore at Oracle worked heroically from the earliest days on the design of lambda expressions and their implementation in `javac`. Support for Java SE 8 features in Eclipse was contributed by Jayaprakash Arthanareeswaran, Shankha Banerjee, Anirban Chakraborty, Andrew Clement, Stephan Herrmann, Markus Keller, Jesper Møller, Manoj Palat, Srikanth Sankaran, and Olivier Thomann; and in IntelliJ by Anna Kozlova, Alexey Kudravnitskiy, and Roman Shevchenko. They deserve the thanks of the entire Java community.

Java SE 8 is a renaissance for the Java language. While some search for the "next great language", we believe that programming in Java is more exciting and productive than ever. We hope that it continues to wear well for you.

Alex Buckley  
*Santa Clara, California*  
*March, 2014*

---

# Introduction

**T**HE 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 bytecode instruction set and binary format defined in *The Java Virtual Machine Specification, Java SE 8 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 describes a variety of type inference algorithms used to test applicability of generic methods and to infer types in a generic method invocation.

Chapter 19 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.

The type system of the Java programming language occasionally relies on the notion of a *substitution*. The notation  $[F_1:=T_1, \dots, F_n:=T_n]$  denotes substitution of  $F_i$  by  $T_i$  for  $1 \leq i \leq n$ .

## 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 Core Reflection API (`java.lang.reflect`) and the Language Model API (`javax.lang.model`), 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 Core Reflection API can accomplish this. Readers should be aware of these additional mechanisms even though they are not mentioned in the text.

## 1.5 Feedback

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

Questions concerning the behavior of `javac` (the reference compiler for the Java programming language), and in particular its conformance to this specification, may be sent to `compiler-dev@openjdk.java.net`.

## 1.6 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++ Programming 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.

## Symbols

**= operator**, 582

assignment contexts, 109

expressions and run-time checks, 463, 464

normal and abrupt completion of evaluation,  
465, 465

**@Deprecated**, 306

**@FunctionalInterface**, 308

**@Inherited**, 304

**@Override**, 304

**@Repeatable**, 308

repeatable annotation types, 298

**@Retention**, 303

repeatable annotation types, 298

**@SafeVarargs**, 307

formal parameters, 228

**@SuppressWarnings**, 305

checked casts and unchecked casts, 121

formal parameters, 228

requirements in overriding and hiding, 246

unchecked conversion, 103

**@Target**, 302

multiple annotations of the same type, 318

repeatable annotation types, 298

where annotations may appear, 313

## A

**abrupt completion of do statement**, 425

do statement, 425

**abrupt completion of for statement**, 428

iteration of for statement, 428

**abrupt completion of while statement**, 424

while statement, 423

**abstract classes**, 192, 383

abstract methods, 232

anonymous class declarations, 485

array creation expressions, 487

final classes, 194

superinterfaces, 204

**abstract interfaces**, 279

**abstract methods**, 232, 396

abstract classes, 192

method body, 240

method declarations, 286

**access control**, 161

accessing superclass members using super,  
498

class literals, 474

class modifiers, 191

constructor declarations, 256

default constructor, 265

enum body declarations, 269

explicit constructor invocations, 262

field access using a primary, 495

identify potentially applicable methods, 509

import declarations, 180

interface modifiers, 278

local class declarations, 407

member type declarations, 254

method declarations, 286

normal annotations, 309

objects, 54

qualified expression names, 158, 158, 158

- qualified type names, 156, 156
- reference types and values, 53
- requirements in overriding and hiding, 247
- single-static-import declarations, 184
- single-type-import declarations, 181
- static-import-on-demand declarations, 184
- superclasses and subclasses, 200
- superinterfaces, 202
- superinterfaces and subinterfaces, 280
- type-import-on-demand declarations, 183
- access to a protected member**, 167
- access to members and constructors**, 387
- accessing superclass members using super**, 497
  - declarations, 132
  - field declarations, 213
  - field initialization, 221, 222
  - initialization of fields in interfaces, 285
  - instance initializers, 255
  - static methods, 234
  - syntactic classification of a name according to context, 150
- actions**, 640
  - synchronization order, 642
- additive operators**, 556
  - constant expressions, 605
  - integer operations, 43
- additive operators (+ and -) for numeric types**, 559
  - binary numeric promotion, 126
  - floating-point operations, 48
- an array of characters is not a String**, 339
- annotation type elements**, 293
  - @Target, 303
  - annotation types, 293
  - declarations, 130, 131, 131
  - marker annotations, 311
  - normal annotations, 309
  - single-element annotations, 312
  - syntactic classification of a name according to context, 150
  - where types are used, 76
- annotation types**, 292
  - @Target, 302
  - annotations, 308
  - declarations, 130
  - interface declarations, 278
- annotations**, 308
  - declarations, 132
  - defaults for annotation type elements, 297
  - syntactic classification of a name according to context, 150
- anonymous class declarations**, 485
  - class instance creation expressions, 477, 477
  - class modifiers, 191
  - definite assignment and anonymous classes, 627
  - enum constants, 268
  - form of a binary, 377
  - initialization of fields in interfaces, 285
  - inner classes and enclosing instances, 197
  - syntactic classification of a name according to context, 151
  - where types are used, 76, 77
- anonymous constructors**, 485
  - choosing the constructor and its arguments, 484
  - default constructor, 265
  - form of a binary, 381
- argument lists are evaluated left-to-right**, 470
- array access**, 332
- array access expressions**, 491
  - array access, 332
  - assignment operators, 582
  - compound assignment operators, 589
  - simple assignment operator =, 582

- syntactic classification of a name according to context, 151
  - unary numeric promotion, 125
- array creation**, 332
- array creation and access expressions**, 487
  - method reference expressions, 529
- array creation expressions**, 487
  - array creation, 332
  - array initializers, 335
  - objects, 53
  - syntactic classification of a name according to context, 151
  - unary numeric promotion, 125
  - where types are used, 76, 78
- array initializers**, 335
  - array creation, 332
  - definite assignment and array initializers, 626
  - normal and abrupt completion of evaluation, 464
  - objects, 53
  - run-time evaluation of array creation expressions, 488
- array members**, 336
  - declarations, 130, 130
  - happens-before order, 644
  - qualified expression names, 159
- array store exception**, 333
  - array variables, 332
  - assignment contexts, 109
  - expressions and run-time checks, 463, 464
  - normal and abrupt completion of evaluation, 465
  - variables, 80
- array types**, 330
  - annotation type elements, 294
  - enhanced for statement, 430
  - raw types, 66
  - reference types and values, 52
  - reifiable types, 65
  - where types are used, 77
- array variables**, 330
  - enhanced for statement, 431
  - formal parameters, 228
  - lambda parameters, 598
- arrays**, 329
  - array creation expressions, 487
  - kinds of variables, 83
  - when reference types are the same, 57
- assert statement**, 416
  - assert statements, 620
  - detailed initialization procedure, 365
  - when initialization occurs, 361
- assert statements**, 620
- assignment contexts**, 107
  - array initializers, 335
  - array store exception, 333
  - array variables, 332
  - class instance creation expressions, 478
  - execution of try-catch, 444
  - expressions and run-time checks, 463, 463
  - kinds of types and values, 42
  - lambda expressions, 595
  - method invocation expressions, 500
  - method reference expressions, 531
  - normal annotations, 310
  - reference conditional expressions, 580
  - return statement, 436
  - run-time evaluation of method references, 540
  - simple assignment operator =, 582, 583
  - switch statement, 420
  - throw statement, 437, 438
  - type of an expression, 461
  - variables, 80
- assignment expressions**, 615

- assignment operators**, 581
  - assignment contexts, 107
  - assignment expressions, 615
  - evaluation order for other expressions, 471
  - field initialization, 221
  - final variables, 86
  - forms of expressions, 460
  - initial values of variables, 87
  - syntactic classification of a name according to context, 151
  - variables, 80
- asynchronous Exceptions**, 343
  - causes of Exceptions, 343
- B**
- basic for statement**, 426
  - @Target, 303
  - for statements, 622
  - scope of a declaration, 140
  - syntactic classification of a name according to context, 150
  - where types are used, 76
- basic try-with-resources**, 448
- binary compatibility**, 375
- binary numeric promotion**, 125
  - additive operators (+ and -) for numeric types, 559
  - division operator /, 552
  - integer bitwise operators &, ^, and |, 568
  - multiplicative operators, 550
  - numeric conditional expressions, 580
  - numeric contexts, 124
  - numerical comparison operators <, <=, >, and >=, 563
  - numerical equality operators == and !=, 566
  - postfix decrement operator --, 544
  - postfix increment operator ++, 543
  - prefix decrement operator --, 547
  - prefix increment operator ++, 546
  - remainder operator %, 554
  - shift operators, 561
- bitwise and logical operators**, 568
  - constant expressions, 605
- bitwise complement operator** ~, 548
  - constant expressions, 605
  - integer operations, 43
  - unary numeric promotion, 125
- blocks**, 407, 617
  - blocks, 617
  - kinds of variables, 84
  - lambda body, 599
  - local class declarations, 407
  - scope of a declaration, 140
- blocks and statements**, 405
- boolean conditional expressions**, 579
- boolean constant expressions**, 613
- boolean equality operators** == and !=, 567
  - boolean type and boolean values, 51
- boolean literals**, 34
  - boolean type and boolean values, 51
  - boxing conversion, 101
  - constant expressions, 605
  - identifiers, 23
  - lexical literals, 473
- boolean logical operators** &, ^, and |, 569
  - boolean type and boolean values, 51
  - conditional-and operator &&, 570
  - conditional-or operator ||, 570
- boolean type and boolean values**, 51
  - boolean literals, 34
  - lexical literals, 473
- bounds**, 661
  - bounds involving capture conversion, 675
  - invocation applicability inference, 678
  - more specific method inference, 686

reduction, 663

**bounds involving capture conversion, 674**

incorporation, 673

**boxing conversion, 100**

assignment contexts, 107

casting contexts, 115

class literals, 473

conditional operator `? :`, 579

creation of new class instances, 366

floating-point operations, 49

integer operations, 44

invocation contexts, 113

invocation type inference, 680

normal and abrupt completion of evaluation,  
465

numeric conditional expressions, 579

objects, 53

postfix decrement operator `--`, 544

postfix increment operator `++`, 543

prefix decrement operator `--`, 547

prefix increment operator `++`, 546

type compatibility constraints, 668, 668

**break statement, 432**

break, continue, return, and throw statements,  
624

labeled statements, 413

names and identifiers, 137

normal and abrupt completion of statements,  
406

**break, continue, return, and throw  
statements, 624**

## C

**capture conversion, 103**

array access expressions, 491

assignment operators, 582

bounds, 662

cast expressions, 549

compile-time declaration of a method  
reference, 533, 534

compile-time step 3: is the chosen method  
appropriate?, 520

enhanced for statement, 430

expression compatibility constraints, 667,  
667

field access using a primary, 495

function types, 325

intersection types, 70

least upper bound, 75

members and constructors of parameterized  
types, 63

parameterized types, 59

qualified expression names, 158, 158, 158,  
159, 159

reference conditional expressions, 580

resolution, 676

simple expression names, 157

subtyping among class and interface types,  
72

type arguments of parameterized types, 61

type of a method reference, 538

**cast expressions, 549**

array types, 330

casting contexts, 114

compile-time step 3: is the chosen method  
appropriate?, 519

constant expressions, 605

expressions and run-time checks, 463, 464

floating-point operations, 48

forms of expressions, 460

happens-before order, 644

integer operations, 43

intersection types, 70

normal and abrupt completion of evaluation,  
465

- objects, 54
- syntactic classification of a name according to context, 151
- type comparison operator `instanceof`, 564
- unary operators, 544
- where types are used, 76
- casting contexts**, 114
  - boolean type and boolean values, 51
  - cast expressions, 549, 550
  - expressions and run-time checks, 463, 464
  - happens-before order, 644
  - kinds of types and values, 42
  - lambda expressions, 595
  - method reference expressions, 531
  - objects, 54
  - reference equality operators `==` and `!=`, 567
- casting conversions to primitive types**, 116
- casting conversions to reference types**, 117
- causes of Exceptions**, 343
- character literals**, 34
  - boxing conversion, 101
  - comments, 22
  - constant expressions, 605
  - escape sequences for character and String literals, 37
  - lexical literals, 473
  - unicode, 16
- check accessibility of type and method**, 523
  - run-time evaluation of method references, 540, 540, 541
- checked casts and unchecked casts**, 121
  - variables of reference type, 81
- checked casts at run time**, 122
  - checked casts and unchecked casts, 122
- checked exception constraints**, 671
- choosing the constructor and its arguments**, 481
  - anonymous constructors, 486
  - class instance creation expressions, 478
  - compile-time declaration of a method reference, 533
  - default constructor, 265
  - expression compatibility constraints, 664
  - formal parameters, 257
- choosing the most specific method**, 514
  - compile-time declaration of a method reference, 534, 535, 535
  - compile-time step 2: determine method Signature, 504
  - conditional operator `? :`, 572, 573
  - method and constructor overloading, 399
  - more specific method inference, 686, 686
  - phase 1: identify matching arity methods applicable by strict invocation, 512
  - phase 2: identify matching arity methods applicable by loose invocation, 513
  - phase 3: identify methods applicable by variable arity invocation, 513
- class body and member declarations**, 205, 386
  - class members, 206
  - member type declarations, 254, 254
  - scope of a declaration, 140
  - what binary compatibility is and is not, 382
- class declarations**, 191
  - declarations, 130
  - reference types and values, 52
  - types, classes, and interfaces, 88
- class instance creation expressions**, 476
  - conditional operator `? :`, 572, 573
  - constructor declarations, 256, 256
  - constructor overloading, 264
  - creation of new class instances, 365
  - exception analysis of expressions, 346
  - form of a binary, 379
  - forms of expressions, 460
  - functional interfaces, 319

- initial values of variables, 87, 87
- instance initializers, 255
- invocation contexts, 112
- invocation type inference, 681
- kinds of variables, 84
- method reference expressions, 529
- names and identifiers, 138
- objects, 53, 53
- return statement, 436
- run-time handling of an exception, 350
- String conversion, 106
- syntactic classification of a name according to context, 151, 151, 151
- types, classes, and interfaces, 88
- where types are used, 76, 76, 77, 77
- class literals**, 473
  - declarations, 132
  - normal annotations, 310
  - syntactic classification of a name according to context, 150
- class loading**, 356
  - causes of Exceptions, 343
  - class literals, 474
  - load the class test, 354
- class members**, 206
  - abstract classes, 192
  - declarations, 130
  - function types, 323
  - members and constructors of parameterized types, 63
  - method invocation type, 516
- class modifiers**, 191
  - @Target, 302
  - anonymous class declarations, 485
  - local class declarations, 407
  - reference type casting, 118, 118, 118
- class Object**, 55
  - checked casts at run time, 122, 123
  - method invocation type, 517
- class objects for arrays**, 338
  - types, classes, and interfaces, 88
- class String**, 56
  - lexical literals, 473
  - literals, 24
  - objects, 53
  - String literals, 35, 36
- class type parameters**, 385
- classes**, 189
  - local class declarations, 407
  - package members, 173
  - qualified expression names, 158
  - reclassification of contextually ambiguous names, 153
  - top level type declarations, 185
- comments**, 21
  - input elements and tokens, 20
  - lexical grammar, 9
  - lexical translations, 16
  - line terminators, 19
  - unicode, 16
- compilation units**, 177
  - determining accessibility, 162
  - host support for packages, 175
  - observability of a package, 179
  - package members, 173
  - reclassification of contextually ambiguous names, 153, 153
  - scope of a declaration, 139
  - shadowing, 146
  - syntactic grammar, 10
- compile-time checking of Exceptions**, 344
  - checked exception constraints, 671
- compile-time declaration of a method reference**, 532
  - checked exception constraints, 672
  - choosing the most specific method, 515

- compile-time step 2: determine method Signature, 503
- expression compatibility constraints, 666, 667
- identify potentially applicable methods, 510
- invocation type inference, 683, 683
- method reference expressions, 531
- phase 1: identify matching arity methods applicable by strict invocation, 511
- run-time evaluation of method references, 541
- compile-time step 1: determine class or interface to search**, 500
  - class Object, 56
  - compile-time declaration of a method reference, 532
  - identify potentially applicable methods, 509
  - method invocation type, 517
  - raw types, 68
- compile-time step 2: determine method Signature**, 502
  - choosing the constructor and its arguments, 483, 483
  - compile-time declaration of a method reference, 532
  - enum constants, 268
  - overloading, 251
  - what binary compatibility is and is not, 382
- compile-time step 3: is the chosen method appropriate?**, 517
  - check accessibility of type and method, 523
  - choosing the most specific method, 515
  - create frame, synchronize, transfer control, 528
  - form of a binary, 379
  - locate method to invoke, 524, 524
  - method invocation expressions, 499
  - run-time evaluation of method references, 540, 541, 542
- complementary pairs of bounds**, 674
  - incorporation, 673
- compound assignment operators**, 588
  - evaluate left-hand operand first, 466
- compute target reference (if necessary)**, 520
  - method reference expressions, 531
- concepts and notation**, 660
- conditional expression type (primitive 3rd operand, part i)**, 574
- conditional expression type (primitive 3rd operand, part ii)**, 575
- conditional expression type (reference 3rd operand, part i)**, 576
- conditional expression type (reference 3rd operand, part ii)**, 577
- conditional expression type (reference 3rd operand, part iii)**, 578
- conditional operator ? :**, 571, 614, 615
  - binary numeric promotion, 126
  - boolean type and boolean values, 51, 51
  - conditional operator ? :, 614, 615
  - constant expressions, 605
  - floating-point operations, 48
  - forms of expressions, 460, 461
  - identify potentially applicable methods, 511
  - integer operations, 43
  - objects, 55
  - phase 1: identify matching arity methods applicable by strict invocation, 511
- conditional-and operator &&**, 570, 613
  - boolean type and boolean values, 51
  - conditional-and operator &&, 613
  - constant expressions, 605
- conditional-or operator ||**, 570, 614
  - boolean type and boolean values, 51
  - conditional-or operator ||, 614
  - constant expressions, 605
  - forms of expressions, 460

- constant expressions**, 605
  - assignment contexts, 108
  - boolean constant expressions, 613
  - class `String`, 56
  - creation of new class instances, 366
  - final fields and static constant variables, 391
  - final variables, 85
  - forms of expressions, 461
  - fp-strict expressions, 462
  - initialization of fields in interfaces, 285
  - normal annotations, 310
  - numeric conditional expressions, 579
  - objects, 53
  - String concatenation operator `+`, 557
  - String literals, 36
  - subsequent modification of final fields, 655
  - unreachable statements, 454, 454, 455
- constraint formulas**, 661
  - reduction, 663
- constructor body**, 259
  - constructor declarations, 256
  - definite assignment, constructors, and instance initializers, 629
  - initial values of variables, 87
  - kinds of variables, 84
  - return statement, 436
  - `this`, 474
- constructor declarations**, 256
  - class body and member declarations, 206
  - creation of new class instances, 366
  - declarations, 131, 131
  - final fields, 219
  - raw types, 67
  - return statement, 436
  - run-time evaluation of class instance creation expressions, 484
  - simple expression names, 157
  - syntactic classification of a name according to context, 150
  - where types are used, 77
- constructor modifiers**, 258
  - `@Target`, 303
- constructor overloading**, 264
- constructor Signature**, 258
  - form of a binary, 380
- constructor throws**, 259
  - compile-time checking of Exceptions, 344
  - declarations, 132
  - syntactic classification of a name according to context, 150
  - throw statement, 438
  - where types are used, 76, 78
- context-free grammars**, 9
  - compilation units, 177
- continue statement**, 434
  - break, continue, return, and throw statements, 624
  - labeled statements, 413
  - names and identifiers, 137
  - normal and abrupt completion of statements, 406
- conversions and contexts**, 91
  - parenthesized expressions, 476
  - type of an expression, 461
- create frame, synchronize, transfer control**, 528
  - run-time evaluation of method references, 540, 540, 541
- creation of new class instances**, 365
  - constructor declarations, 256
  - field initialization, 222
  - form of a binary, 377
  - instance initializers, 255
  - run-time evaluation of class instance creation expressions, 484

static fields, 216  
 String concatenation operator +, 557  
 when initialization occurs, 361

## D

### declarations, 130

functional interfaces, 322  
 members and constructors of parameterized types, 64  
 names and identifiers, 137  
 syntactic classification of a name according to context, 150  
 where types are used, 77, 78

### default constructor, 265

form of a binary, 381  
 method and constructor declarations, 394

### defaults for annotation type elements, 297

marker annotations, 312  
 single-element annotations, 312  
 syntactic classification of a name according to context, 152

### definite assignment, 607

assignment operators, 582  
 final variables, 85, 85, 86  
 initial values of variables, 87  
 inner classes and enclosing instances, 199  
 kinds of variables, 84  
 lambda body, 600  
 parenthesized expressions, 476

### definite assignment and anonymous classes, 627

### definite assignment and array initializers, 626

### definite assignment and enum constants, 626

### definite assignment and expressions, 613

### definite assignment and member types, 627

### definite assignment and parameters, 626

### definite assignment and statements, 617

**definite assignment and static initializers, 628**  
 definite assignment and enum constants, 626  
 final fields, 219

**definite assignment, constructors, and instance initializers, 628**  
 final fields, 219

### detailed initialization procedure, 363

field initialization, 221, 221  
 final fields and static constant variables, 393  
 form of a binary, 377  
 initialization of fields in interfaces, 285  
 simple expression names, 157  
 static initializers, 255  
 throw statement, 439  
 when initialization occurs, 361

### details on protected access, 166

determining accessibility, 163

### determining accessibility, 162

top level type declarations, 185

### determining enclosing instances, 480

anonymous constructors, 486  
 choosing the constructor and its arguments, 482, 482  
 class instance creation expressions, 478  
 compile-time declaration of a method reference, 535  
 default constructor, 265  
 formal parameters, 257  
 inner classes and enclosing instances, 198  
 method reference expressions, 531  
 run-time evaluation of method references, 541

### determining the class being instantiated, 478

abstract classes, 192  
 class instance creation expressions, 478  
 enum types, 267

### determining the meaning of a name, 148

class members, 207

- declarations, 131
- interface members, 283
- names and identifiers, 137
- obscuring, 147
- where types are used, 77

**division operator /**, 552

- compound assignment operators, 590
- evaluate operands before operation, 468
- integer operations, 44
- normal and abrupt completion of evaluation, 465

**do statement**, 424

- boolean type and boolean values, 51
- do statements, 622

**do statements**, 622

## E

**empty statement**, 412

- empty statements, 617

**empty statements**, 617

**enhanced for statement**, 429

- @Target, 303
- array variables, 331
- for statements, 622
- scope of a declaration, 141
- syntactic classification of a name according to context, 150
- where types are used, 76

**enum body declarations**, 268

- constructor declarations, 256
- declarations, 130, 130, 131

**enum constants**, 267

- @Target, 303
- definite assignment and enum constants, 626
- definite assignment and static initializers, 628
- enum types, 267

- normal annotations, 310
- where types are used, 76

**enum members**, 271

- declarations, 130
- form of a binary, 381

**enum types**, 266

- @Target, 302
- abstract methods, 232
- class declarations, 191
- declarations, 130
- normal annotations, 310
- superclasses and subclasses, 200
- switch statement, 420

**equality operators**, 565

- constant expressions, 605

**erasure**, 64

- assignment contexts, 109
- cast expressions, 549
- checked casts and unchecked casts, 122
- checked casts at run time, 122
- choosing the most specific method, 516
- class type parameters, 385
- compile-time step 3: is the chosen method appropriate?, 519
- constructor Signature, 258
- create frame, synchronize, transfer control, 528
- declarations, 131
- evaluate arguments, 522
- field declarations, 390
- form of a binary, 378, 379, 380
- invocation contexts, 114
- method and constructor formal parameters, 396
- method and constructor type parameters, 395
- method result type, 396
- method Signature, 230
- raw types, 66

- requirements in overriding and hiding, 246
- type variables, 58
- escape sequences for character and String literals, 37**
  - character literals, 34
  - String literals, 35
- evaluate arguments, 522**
  - formal parameters, 229
  - variables of reference type, 81
- evaluate left-hand operand first, 466**
- evaluate operands before operation, 468**
- evaluation order, 466**
- evaluation order for other expressions, 471**
- evaluation respects parentheses and precedence, 469**
- evaluation, denotation, and result, 459**
  - assert statement, 417
- evolution of annotation types, 403**
- evolution of classes, 383**
- evolution of enums, 400**
- evolution of interfaces, 400**
- evolution of packages, 383**
- example programs, 6**
- exception analysis of expressions, 346**
  - class instance creation expressions, 477
  - compile-time checking of Exceptions, 345
  - method invocation expressions, 499
- exception analysis of statements, 346**
  - compile-time checking of Exceptions, 345
  - explicit constructor invocations, 262
  - method throws, 238
  - throw statement, 438
  - try statement, 443
- exception checking, 347**
  - compile-time checking of Exceptions, 344
  - field initialization, 222
  - instance initializers, 255
  - method throws, 239
  - static initializers, 256
  - throw statement, 439, 439
  - try statement, 443
  - type of a lambda expression, 603
- Exceptions, 341**
  - floating-point operations, 49
  - integer operations, 44
  - normal and abrupt completion of statements, 406
  - throw statement, 437
- execution, 353**
- execution of local variable declarations, 410**
- execution of try-catch, 444**
  - try statement, 443
- execution of try-finally and try-catch-finally, 445**
  - try statement, 443
- executions, 646**
  - well-formed executions, 647
- executions and causality requirements, 647**
- explicit constructor invocations, 260**
  - anonymous constructors, 486, 486
  - constructor body, 259
  - creation of new class instances, 366
  - definite assignment, constructors, and instance initializers, 629, 629
  - enum body declarations, 269
  - exception analysis of statements, 347
  - form of a binary, 379
  - inner classes and enclosing instances, 197, 198
  - instance initializers, 255
  - invocation contexts, 112
  - syntactic classification of a name according to context, 151, 151
  - where types are used, 76, 77
- expression compatibility constraints, 663**
- expression names, 543**

- declarations, 132
- expression statements**, 414, 620
  - evaluation, denotation, and result, 459
  - expression statements, 620
  - identify potentially applicable methods, 510
  - initialization of for statement, 427
- expressions**, 459
- expressions and run-time checks**, 462
- extended try-with-resources**, 451

## F

- feedback**, 7
- field (constant) declarations**, 283
  - @Target, 303
  - array initializers, 335
  - array variables, 330, 331
  - declarations, 130
  - final variables, 85
  - interface body and member declarations, 282
  - kinds of variables, 83
  - obscuring, 147
  - shadowing, 144
  - syntactic classification of a name according to context, 150
  - where types are used, 76
- field access expressions**, 494
  - assignment operators, 582
  - names and identifiers, 138
  - normal and abrupt completion of evaluation, 465
  - objects, 54
  - raw types, 68
  - simple assignment operator =, 582
  - static initializers, 256
- field access using a primary**, 494
- field declarations**, 211, 388, 402
  - array initializers, 335
  - array variables, 330, 331
  - class body and member declarations, 205
  - creation of new class instances, 366
  - declarations, 130
  - field declarations, 402
  - obscuring, 147
  - raw types, 67
  - reclassification of contextually ambiguous names, 152
  - shadowing, 144
  - simple expression names, 157
  - syntactic classification of a name according to context, 150
  - where annotations may appear, 314
  - where types are used, 76
- field initialization**, 221
  - definite assignment and static initializers, 628
  - definite assignment, constructors, and instance initializers, 628
  - detailed initialization procedure, 364
  - exception checking, 347, 348
  - final fields and static constant variables, 393
  - initialization of fields in interfaces, 285
  - simple expression names, 157
  - static initializers, 255
  - this, 474
- field modifiers**, 215
  - @Target, 303
  - field declarations, 212
- final classes**, 194, 383
  - anonymous class declarations, 485
  - final methods, 234
  - superclasses and subclasses, 200
  - verification of the binary representation, 359
- final field semantics**, 652
  - memory model, 639
- final fields**, 219

- final fields and static constant variables**, 391
  - field declarations, 402
  - final variables, 85
  - verification of the binary representation, 359
- final methods**, 234, 397
  - verification of the binary representation, 359
- final variables**, 85
  - constant expressions, 606, 606
  - detailed initialization procedure, 364
  - enum body declarations, 269
  - field initialization, 221
  - final fields, 219
  - final fields and static constant variables, 391
  - form of a binary, 377
  - initialization of fields in interfaces, 285
  - inner classes and enclosing instances, 197, 199, 199
  - lambda body, 600
  - local variable declarators and types, 410
  - narrowing reference conversion, 99
  - try statement, 442
  - try-with-resources, 448
  - when initialization occurs, 361
- finalization of class instances**, 369
  - class Object, 56
  - enum body declarations, 269
  - happens-before order, 643
  - kinds of variables, 83
  - unloading of classes and interfaces, 373
- floating-point literals**, 31
  - constant expressions, 605
  - lexical literals, 473
- floating-point operations**, 48
  - additive operators (+ and -) for numeric types, 561
  - division operator /, 553
  - multiplication operator \*, 552
  - narrowing primitive conversion, 96
  - widening primitive conversion, 95
- floating-point types, formats, and values**, 45
  - cast expressions, 550
  - field declarations, 213
  - floating-point literals, 33
  - formal parameters, 229
  - fp-strict expressions, 462
  - lambda parameters, 598
  - lexical literals, 473, 473
  - local variable declarators and types, 410
  - narrowing primitive conversion, 97, 97
  - parenthesized expressions, 476
  - return statement, 437
  - unary minus operator -, 548
- floating-point value set parameters**, 46
  - floating-point types, formats, and values, 46, 46, 46
- for statement**, 426
  - boolean type and boolean values, 51
  - declarations, 131
  - initial values of variables, 87
  - kinds of variables, 84
  - local variable declaration statements, 409
- for statements**, 622
- forbidden conversions**, 106
- form of a binary**, 376
  - check accessibility of type and method, 523
  - compile-time step 3: is the chosen method appropriate?, 519
  - final variables, 85
  - loading of classes and interfaces, 356
  - locate method to invoke, 524
  - resolution of symbolic references, 359
  - top level type declarations, 187
  - when reference types are the same, 56, 57
- formal parameters**, 226, 257
  - @Target, 303
  - array variables, 331

- choosing the constructor and its arguments, 482
- compile-time declaration of a method reference, 537
- compile-time step 3: is the chosen method appropriate?, 519
- declarations, 131, 131
- default constructor, 265
- definite assignment and parameters, 626, 626
- evaluate arguments, 522
- form of a binary, 381
- formal parameters, 257
- initial values of variables, 87, 87
- invoke test.main, 356
- kinds of variables, 83, 84
- lambda parameters, 597, 598
- method and constructor formal parameters, 395
- method declarations, 226
- reclassification of contextually ambiguous names, 152, 152
- scope of a declaration, 140, 140
- shadowing and obscuring, 142
- shared variables, 640
- syntactic classification of a name according to context, 150, 150, 150
- this, 474
- variables of reference type, 81
- where types are used, 76, 76, 76, 77
- forms of expressions, 460**
  - boolean conditional expressions, 579
  - choosing the most specific method, 515
  - class instance creation expressions, 478
  - conditional operator `? :`, 572, 572
  - expression compatibility constraints, 663, 663
  - identify potentially applicable methods, 511
  - lambda expressions, 594
  - method reference expressions, 531
  - more specific method inference, 687
  - numeric conditional expressions, 579
  - parenthesized expressions, 476
- forward references during field initialization, 222**
  - when initialization occurs, 361
- fp-strict expressions, 462**
  - additive operators (+ and -) for numeric types, 560
  - cast expressions, 550
  - constant expressions, 606
  - division operator /, 553
  - floating-point types, formats, and values, 45
  - formal parameters, 229
  - method declarations, 286
  - multiplication operator \*, 551
  - return statement, 437
  - strictfp classes, 194
  - strictfp interfaces, 279
  - strictfp methods, 235
  - value set conversion, 106, 107
  - widening primitive conversion, 95
- fully qualified names and canonical names, 169**
  - compilation units, 177
  - form of a binary, 377, 380
  - import declarations, 180, 180
  - local class declarations, 407
  - named packages, 178
  - notation, 6
  - package members, 174
  - single-static-import declarations, 184
  - single-type-import declarations, 181
  - static-import-on-demand declarations, 184
  - top level type declarations, 185
  - type-import-on-demand declarations, 183

**function types, 323**

- checked exception constraints, 672
- expression compatibility constraints, 666
- type of a lambda expression, 602
- type of a method reference, 538

**functional interface parameterization**

**inference, 685**

- expression compatibility constraints, 664
- type of a lambda expression, 602

**functional interfaces, 319**

- @FunctionalInterface, 308
- checked exception constraints, 671
- expression compatibility constraints, 664
- functional interface parameterization
- inference, 685
- identify potentially applicable methods, 510
- lambda expressions, 595
- method reference expressions, 531
- type of a lambda expression, 602
- type of a method reference, 537

**G**

**generic classes and type parameters, 194**

- @Target, 303
- capture conversion, 103
- class instance creation expressions, 477
- declarations, 130
- form of a binary, 377
- generic constructors, 259
- generic methods, 237
- parameterized types, 59
- scope of a declaration, 140
- syntactic classification of a name according to context, 150
- type variables, 57
- types, classes, and interfaces, 89
- where types are used, 76

**generic constructors, 259**

- @Target, 303
- class instance creation expressions, 477
- declarations, 130
- form of a binary, 377
- scope of a declaration, 140
- syntactic classification of a name according to context, 150
- type erasure, 64
- type variables, 57
- where types are used, 76

**generic interfaces and type parameters, 279**

- @Target, 303
- capture conversion, 103
- declarations, 130
- form of a binary, 377
- parameterized types, 59
- scope of a declaration, 140
- subtyping among class and interface types, 72
- superinterfaces, 203
- syntactic classification of a name according to context, 150
- type variables, 57
- types, classes, and interfaces, 89
- where types are used, 76

**generic methods, 237**

- @Target, 303
- compile-time declaration of a method reference, 537
- compile-time step 2: determine method Signature, 504
- declarations, 130
- form of a binary, 377
- function types, 323, 323
- method declarations, 226, 287
- method invocation expressions, 500
- method result, 238

- method Signature, 230
- scope of a declaration, 140
- syntactic classification of a name according to context, 150
- type erasure, 64
- type variables, 57
- where types are used, 76

**grammar notation**, 10

**grammars**, 9

## H

**happens-before order**, 643

- executions, 646
- executions and causality requirements, 649
- finalization of class instances, 370

**hiding (by class methods)**, 245

- obscuring, 147
- shadowing, 144

**host support for packages**, 175

- top level type declarations, 187

## I

**identifiers**, 22

- declarations, 130
- keywords, 24
- lexical grammar, 9

**identify potentially applicable methods**, 509

- compile-time declaration of a method reference, 533, 533, 534
- compile-time step 1: determine class or interface to search, 500
- compile-time step 2: determine method Signature, 503
- phase 1: identify matching arity methods applicable by strict invocation, 511

- phase 2: identify matching arity methods applicable by loose invocation, 512
- phase 3: identify methods applicable by variable arity invocation, 513

**identity conversion**, 94

- assignment contexts, 107
- boxing conversion, 101
- capture conversion, 104
- casting contexts, 115
- invocation contexts, 112, 113
- numeric contexts, 124
- unary numeric promotion, 124

**if statement**, 415

- boolean type and boolean values, 51

**if statements**, 620

**if-then statement**, 415

- if statements, 620

**if-then-else statement**, 416

- if statements, 620

**implementing finalization**, 370

**import declarations**, 180

- compilation units, 177

**incorporation**, 673

**incrementation part of for statement**, 623

**inference variables**, 660

**inheritance and overriding**, 287

- compile-time checking of Exceptions, 345
- compile-time declaration of a method reference, 535
- compile-time step 3: is the chosen method appropriate?, 518
- interface members, 283
- method declarations, 287

**inheritance, overriding, and hiding**, 240

- class Object, 56
- compile-time declaration of a method reference, 535
- enum constants, 268

- function types, 323
- inheriting methods with override-equivalent signatures**, 250, 289
  - variables of reference type, 81
- initial values of variables**, 87
  - array initializers, 335
  - creation of new class instances, 366
  - field initialization, 221
  - final fields and static constant variables, 393
  - initialization of fields in interfaces, 285
  - kinds of variables, 83, 83, 83
  - preparation of a class or interface type, 359
  - run-time evaluation of array creation expressions, 488
  - run-time evaluation of class instance creation expressions, 484
  - variables, 80
- initialization of classes and interfaces**, 360
  - causes of Exceptions, 343
  - initialize test: execute initializers, 356
  - objects, 53
  - preparation of a class or interface type, 359
  - run-time handling of an exception, 350
  - static fields, 216
- initialization of fields in interfaces**, 285
  - detailed initialization procedure, 364
  - field initialization, 221
  - final fields and static constant variables, 393
- initialization of for statement**, 427
- initialization part of for statement**, 623
- initialize test: execute initializers**, 355
- inner classes and enclosing instances**, 197
  - anonymous class declarations, 485
  - compile-time step 3: is the chosen method appropriate?, 517
  - determining enclosing instances, 480
  - explicit constructor invocations, 261
  - form of a binary, 381
  - local class declarations, 407
  - method reference expressions, 531
  - qualified this, 475
  - when initialization occurs, 361
- input elements and tokens**, 19
  - invocation type inference, 681
  - lexical grammar, 9, 9
  - lexical translations, 16
  - unicode, 16
- instance creation**, 365, 476
  - conditional operator ? :, 572, 573
  - constructor declarations, 256, 256, 256
  - constructor overloading, 264
  - creation of new class instances, 365
  - exception analysis of expressions, 346
  - field initialization, 222
  - form of a binary, 377, 379
  - forms of expressions, 460
  - functional interfaces, 319
  - initial values of variables, 87, 87
  - instance initializers, 255, 255
  - invocation contexts, 112
  - invocation type inference, 681
  - kinds of variables, 84
  - method reference expressions, 529
  - names and identifiers, 138
  - objects, 53, 53
  - return statement, 436
  - run-time evaluation of class instance creation expressions, 484
  - run-time handling of an exception, 350
  - static fields, 216
  - String concatenation operator +, 557
  - String conversion, 106
  - syntactic classification of a name according to context, 151, 151, 151
  - types, classes, and interfaces, 88
  - when initialization occurs, 361

- where types are used, 76, 76, 77, 77
- instance initializers**, 255
  - class body and member declarations, 206
  - definite assignment, constructors, and instance initializers, 629
  - exception checking, 348
  - return statement, 436
  - simple expression names, 157
  - this, 474
  - throw statement, 439
- instanceof operator**, 564
  - expressions and run-time checks, 463
  - objects, 54
  - syntactic classification of a name according to context, 151
  - where types are used, 76, 78
- integer bitwise operators &, ^, and |**, 568
  - binary numeric promotion, 126
  - integer operations, 43
  - shift operators, 562, 562
- integer literals**, 25
  - boxing conversion, 101
  - constant expressions, 605
  - lexical literals, 473
- integer operations**, 43
- integral types and values**, 43
  - character literals, 34
  - integer literals, 25
  - lexical literals, 473, 473, 473
- interaction with the memory model**, 372
- interactions of waits, notification, and interruption**, 635
- interface body and member declarations**, 282
  - interface members, 282
  - member type declarations, 254, 254
  - scope of a declaration, 140
- interface declarations**, 278
  - declarations, 130
  - reference types and values, 52
  - types, classes, and interfaces, 88
- interface members**, 282, 401
  - check accessibility of type and method, 523
  - declarations, 130
  - form of a binary, 378, 380
- interface method body**, 291
  - superinterfaces, 204
  - this, 474
- interface method declarations**, 402
- interface methods**, 286
  - @Target, 303, 303
  - array variables, 331
  - declarations, 131, 131, 132
  - inheritance, overriding, and hiding, 241
  - interface body and member declarations, 282
  - raw types, 67
  - syntactic classification of a name according to context, 150, 150, 150
  - where types are used, 76, 76, 76, 78
- interface modifiers**, 278
  - @Target, 302
  - static member type declarations, 255
- interface type parameters**, 401
- interfaces**, 277
  - package members, 173
  - qualified expression names, 158
  - reclassification of contextually ambiguous names, 153
  - top level type declarations, 185
- interruptions**, 635
- intersection types**, 70
  - form of a binary, 378, 379
  - functional interfaces, 323
  - type variables, 58
  - where types are used, 77
- introduction**, 1
- invocation applicability inference**, 678

- invocation type inference, 679, 680, 680, 682
    - phase 1: identify matching arity methods applicable by strict invocation, 512
    - phase 2: identify matching arity methods applicable by loose invocation, 512
    - phase 3: identify methods applicable by variable arity invocation, 513
  - invocation contexts**, 112
    - class instance creation expressions, 478
    - constraint formulas, 661, 661
    - create frame, synchronize, transfer control, 528
    - expression compatibility constraints, 663
    - formal parameters, 229
    - kinds of types and values, 42
    - lambda expressions, 595
    - method invocation expressions, 500
    - method reference expressions, 531
    - reference conditional expressions, 580
    - type compatibility constraints, 668
  - invocation type inference**, 679
    - expression compatibility constraints, 664, 667
    - invocation type inference, 681
    - method invocation type, 516
  - invoke test.main**, 356
  - iteration of for statement**, 427
- J**
- Java Virtual Machine startup**, 353
- K**
- keywords**, 24
    - identifiers, 23
    - lexical grammar, 9
    - primitive types and values, 42
  - kinds and causes of Exceptions**, 342
  - kinds of conversion**, 94
  - kinds of Exceptions**, 342
    - compile-time checking of Exceptions, 345, 345
    - generic classes and type parameters, 195
    - method throws, 238, 239
    - narrowing primitive conversion, 97
    - throw statement, 438
    - widening primitive conversion, 95
  - kinds of types and values**, 41
    - capture conversion, 103
    - lexical literals, 473
    - literals, 24
    - null literal, 38
    - throw statement, 438
  - kinds of variables**, 83
    - formal parameters, 227
- L**
- labeled statements**, 413, 619
    - break statement, 432
    - continue statement, 434
    - labeled statements, 619
    - names and identifiers, 137
  - lambda body**, 599
    - choosing the most specific method, 515
    - identify potentially applicable methods, 510, 510
    - kinds of variables, 84
    - type of a lambda expression, 603
  - lambda expressions**, 594
    - exception analysis of expressions, 346
    - forms of expressions, 460, 461
    - functional interfaces, 319
    - identify potentially applicable methods, 510

- return statement, 436
- scope of a declaration, 140
- lambda parameters**, 596
  - array variables, 331
  - choosing the most specific method, 514, 514
  - compile-time step 2: determine method Signature, 503
  - declarations, 131
  - kinds of variables, 84
  - phase 1: identify matching arity methods applicable by strict invocation, 511
  - shadowing and obscuring, 142
  - syntactic classification of a name according to context, 150
  - where types are used, 76
- least upper bound**, 73
  - intersection types, 70
  - resolution, 676
- lexical grammar**, 9
- lexical literals**, 472
  - kinds of types and values, 42
- lexical structure**, 15
  - lexical grammar, 9
- lexical translations**, 16
- line terminators**, 19
  - character literals, 34
  - input elements and tokens, 19
  - lexical translations, 16
  - white space, 20
- link test: verify, prepare, (optionally) resolve**, 354
- linking of classes and interfaces**, 358
  - causes of Exceptions, 343
  - check accessibility of type and method, 523
  - link test: verify, prepare, (optionally) resolve, 354
  - resolution of symbolic references, 360
- literals**, 24
  - lexical grammar, 9
  - lexical literals, 472
- load the class test**, 354
- loading of classes and interfaces**, 356
  - causes of Exceptions, 343
  - class literals, 474
  - load the class test, 354
- loading process**, 357
  - superclasses and subclasses, 202
  - superinterfaces and subinterfaces, 281
- local class declaration statements**, 619
- local class declarations**, 407
  - class instance creation expressions, 477
  - class modifiers, 191
  - form of a binary, 377
  - inner classes and enclosing instances, 197
  - local class declaration statements, 619
  - reclassification of contextually ambiguous names, 153
  - shadowing and obscuring, 142
- local variable declaration statements**, 408, 619
  - @Target, 303
  - array initializers, 335
  - declarations, 131
  - initial values of variables, 87
  - initialization of for statement, 427
  - initialization part of for statement, 623
  - kinds of variables, 84
  - local variable declaration statements, 619
  - objects, 53
  - reclassification of contextually ambiguous names, 152
  - scope of a declaration, 140
  - shadowing and obscuring, 142
  - shared variables, 640
  - syntactic classification of a name according to context, 150

- where types are used, 76
- local variable declarators and types**, 409
  - array variables, 330, 331
- locate method to invoke**, 524
  - run-time evaluation of method references, 540, 540, 541
- logical complement operator !**, 548, 614
  - boolean type and boolean values, 51
  - constant expressions, 605
  - logical complement operator !, 614

## M

- marker annotations**, 311
  - annotations, 308
- meaning of expression names**, 156
  - expression names, 543
  - forms of expressions, 460
  - names and identifiers, 137
- meaning of method names**, 160
  - names and identifiers, 137
- meaning of package names**, 154
  - names and identifiers, 137
- meaning of packageortypenames**, 155
  - type-import-on-demand declarations, 183
- meaning of type names**, 155
  - names and identifiers, 137
- member type declarations**, 254, 291
  - @Target, 302, 302
  - class body and member declarations, 205
  - class instance creation expressions, 477, 477
  - class modifiers, 191, 191
  - declarations, 130, 130, 130, 130
  - definite assignment and member types, 627, 627
  - form of a binary, 377, 377, 381, 381
  - inner classes and enclosing instances, 197, 197
  - interface body and member declarations, 282
  - interface modifiers, 279
  - member type declarations, 291
  - obscuring, 147, 147
  - qualified type names, 156, 156
  - reclassification of contextually ambiguous names, 153, 153, 154, 154
  - reference types and values, 53, 53
  - shadowing, 144, 144
  - static-import-on-demand declarations, 185, 185
  - syntactic classification of a name according to context, 150, 150
  - type-import-on-demand declarations, 183, 183
  - where types are used, 76, 76, 76, 76
- members and constructors of parameterized types**, 63
- memory model**, 637
  - interaction with the memory model, 372
  - volatile fields, 220
- method and constructor body**, 398
- method and constructor declarations**, 394
  - method and constructor formal parameters, 395
  - method result type, 396
- method and constructor formal parameters**, 395
  - interface method declarations, 402
- method and constructor overloading**, 399
  - interface method declarations, 402
- method and constructor throws**, 398
  - interface method declarations, 402
- method and constructor type parameters**, 394
  - class type parameters, 385
- method body**, 240
  - abstract methods, 232
  - constructor body, 260

- method declarations, 226
- native methods, 235
- this, 474
- method declarations**, 225, 286
  - @Target, 303, 303
  - array variables, 331
  - class body and member declarations, 205
  - declarations, 130, 131, 131, 132
  - evaluation, denotation, and result, 459
  - inheritance, overriding, and hiding, 241
  - interface body and member declarations, 282
  - raw types, 67, 67
  - return statement, 436
  - simple expression names, 157
  - syntactic classification of a name according to context, 150, 150, 150, 150
  - where types are used, 76, 76, 76, 78
- method invocation expressions**, 499
  - anonymous constructors, 486, 486
  - compile-time declaration of a method reference, 532
  - conditional operator `? :`, 572, 573
  - constructor declarations, 257
  - declarations, 132
  - evaluation, denotation, and result, 459
  - exception analysis of expressions, 346
  - expressions and run-time checks, 463
  - field initialization, 221, 222
  - formal parameters, 229
  - forms of expressions, 461
  - happens-before order, 644
  - hiding (by class methods), 245
  - initial values of variables, 87
  - initialization of fields in interfaces, 285
  - instance initializers, 255
  - invocation contexts, 112
  - invocation type inference, 681
  - kinds of variables, 83, 84
  - lambda parameters, 598
  - method declarations, 226
  - names and identifiers, 138
  - normal and abrupt completion of evaluation, 465, 465
  - objects, 54
  - overloading, 251
  - overriding (by instance methods), 242
  - return statement, 436
  - run-time handling of an exception, 350
  - simple method names, 160
  - static initializers, 256
  - syntactic classification of a name according to context, 150, 151, 152
  - this, 474
  - where types are used, 76, 77
- method invocation type**, 516
  - choosing the constructor and its arguments, 483, 483
  - compile-time declaration of a method reference, 537
  - compile-time step 3: is the chosen method appropriate?, 518
  - exception analysis of expressions, 346, 346
  - exception analysis of statements, 347
  - expression compatibility constraints, 667
  - invocation type inference, 679
  - type compatibility constraints, 668
  - type of a method reference, 538
- method modifiers**, 231
  - @Target, 303
  - method declarations, 226
  - objects, 55
- method overriding**, 400
- method reference expressions**, 529
  - access to a protected member, 167
  - declarations, 132
  - expressions and run-time checks, 463, 464

- form of a binary, 379
- forms of expressions, 460, 461
- functional interfaces, 319
- identify potentially applicable methods, 510
- names and identifiers, 138
- syntactic classification of a name according to context, 150, 151, 152
- where types are used, 76, 77, 77
- method result**, 237
  - abstract classes, 192
  - abstract methods, 232
  - class literals, 473
  - constructor declarations, 256
  - function types, 323
  - functional interfaces, 319
  - interface method body, 291
  - method body, 240
  - method declarations, 226
  - requirements in overriding and hiding, 246
  - return statement, 436
  - type erasure, 64
  - where types are used, 76
- method result type**, 396
  - interface method declarations, 402
- method Signature**, 230
  - abstract classes, 192
  - abstract methods, 232
  - choosing the most specific method, 516
  - constructor Signature, 258
  - form of a binary, 379
  - functional interfaces, 319
  - hiding (by class methods), 245
  - inheritance and overriding, 287
  - inheritance, overriding, and hiding, 241, 241
  - inheriting methods with override-equivalent signatures, 250, 289
  - interface members, 282
  - method declarations, 226, 287
  - method result, 238
  - overloading, 290
  - overriding (by instance methods), 241, 242, 288
  - requirements in overriding and hiding, 246
  - type erasure, 64
- method throws**, 238
  - abstract methods, 232
  - compile-time checking of Exceptions, 344
  - constructor throws, 259
  - declarations, 132
  - method declarations, 226
  - syntactic classification of a name according to context, 150
  - throw statement, 438
  - where types are used, 76, 78
- more specific method inference**, 686
  - choosing the most specific method, 514
- multiple annotations of the same type**, 318
  - annotation types, 292
  - annotations, 308
  - class modifiers, 191
  - constructor modifiers, 258
  - enum constants, 268
  - field (constant) declarations, 283
  - field modifiers, 215
  - formal parameters, 228
  - generic classes and type parameters, 195
  - generic interfaces and type parameters, 280
  - interface modifiers, 278
  - lambda parameters, 598
  - local variable declaration statements, 409
  - method declarations, 286
  - method modifiers, 231
  - named packages, 178
- multiplication operator \***, 551
- multiplicative operators**, 550
  - binary numeric promotion, 126

- constant expressions, 605
- floating-point operations, 48
- forms of expressions, 460
- integer operations, 43

## N

### **name classification**, 149

- access control, 161
- declarations, 132
- reference types and values, 53

### **name reclassification**, 152

- method invocation expressions, 499

### **named packages**, 178

- @Target, 302

### **names**, 129

### **names and identifiers**, 137

- compile-time declaration of a method reference, 535
- import declarations, 180
- local class declarations, 407
- named packages, 178
- shadowing and obscuring, 142

### **narrowing primitive conversion**, 96

- casting contexts, 115
- floating-point operations, 49
- narrowing primitive conversion, 97
- postfix decrement operator --, 544
- postfix increment operator ++, 543
- prefix decrement operator --, 547
- prefix increment operator ++, 546
- widening and narrowing primitive conversion, 99

### **narrowing reference conversion**, 99

- casting contexts, 115

### **native methods**, 235, 397

- method body, 240

### **new keyword**, 476

- conditional operator ? :, 572, 573

- constructor declarations, 256, 256

- constructor overloading, 264

- creation of new class instances, 365

- exception analysis of expressions, 346

- form of a binary, 379

- forms of expressions, 460

- functional interfaces, 319

- initial values of variables, 87, 87

- instance initializers, 255

- invocation contexts, 112

- invocation type inference, 681

- kinds of variables, 84

- method reference expressions, 529

- names and identifiers, 138

- objects, 53, 53

- return statement, 436

- run-time handling of an exception, 350

- String conversion, 106

- syntactic classification of a name according to context, 151, 151, 151

- types, classes, and interfaces, 88

- where types are used, 76, 76, 77, 77

### **non-atomic treatment of double and long**, 658

### **normal and abrupt completion of evaluation**, 464

- causes of Exceptions, 343

- normal and abrupt completion of statements, 406, 406

- run-time handling of an exception, 350

### **normal and abrupt completion of statements**, 405

- interface method body, 291

- method body, 240

- normal and abrupt completion of evaluation, 466

- run-time handling of an exception, 350

### **normal annotations**, 309

- annotations, 308
    - defaults for annotation type elements, 297
    - names and identifiers, 138
    - syntactic classification of a name according to context, 152
  - notation**, 6
  - notification**, 634
  - null literal**, 38
    - compile-time step 3: is the chosen method appropriate?, 518
    - identifiers, 23
    - kinds of types and values, 42
    - lexical literals, 473
  - numeric conditional expressions**, 579
  - numeric contexts**, 124
    - floating-point operations, 48
    - integer operations, 44
  - numerical comparison operators** <, <=, >, and >=, 563
    - binary numeric promotion, 126
    - floating-point operations, 48
    - floating-point types, formats, and values, 47
    - integer operations, 43
  - numerical equality operators** == and !=, 566
    - binary numeric promotion, 126
    - floating-point operations, 48
    - floating-point types, formats, and values, 47
    - integer operations, 43
- O**
- object creation**, 365
    - constructor declarations, 256
    - field initialization, 222
    - form of a binary, 377
    - instance initializers, 255
    - run-time evaluation of class instance creation expressions, 484
    - static fields, 216
    - String concatenation operator +, 557
      - when initialization occurs, 361
  - objects**, 53, 55
    - checked casts at run time, 122, 123
    - method invocation type, 517
    - String literals, 36
  - obscuring**, 147
    - labeled statements, 413
    - shadowing, 144
  - observability of a package**, 179
    - qualified package names, 155
    - scope of a declaration, 139
  - observable behavior and nonterminating executions**, 650
    - actions, 640, 641
  - operators**, 38
    - input elements and tokens, 20
    - lexical grammar, 9
  - operators ++ and --**, 616
  - organization of the specification**, 2
  - other expressions**, 616
  - other expressions of type boolean**, 615
  - overload resolution**, 502, 509, 511, 512, 513, 514
    - choosing the constructor and its arguments, 483, 483, 483
    - choosing the most specific method, 514
    - compile-time declaration of a method reference, 532, 533, 533, 534, 534, 534, 534, 534, 534, 534, 535, 535, 535, 535
    - compile-time step 1: determine class or interface to search, 500
    - compile-time step 2: determine method Signature, 503, 503, 503, 503, 503, 503, 503, 504
    - conditional operator ? :, 572, 573

- enum constants, 268
- invocation applicability inference, 678, 678
- invocation type inference, 680
- method and constructor overloading, 399
- more specific method inference, 686, 686, 686, 686, 686
- overloading, 251
- phase 1: identify matching arity methods applicable by strict invocation, 511, 512, 512
- phase 2: identify matching arity methods applicable by loose invocation, 512, 512, 513, 513
- phase 3: identify methods applicable by variable arity invocation, 513, 513, 513, 513
- what binary compatibility is and is not, 382
- overloading**, 250, 290
  - constructor overloading, 264
- overriding (by instance methods)**, 241, 288
  - final classes, 194
  - locate method to invoke, 525
  - superinterfaces, 204

## P

- package declarations**, 178
  - compilation units, 177
  - declarations, 130
- package members**, 173
- packages**, 173
- parameterized types**, 59
  - annotation type elements, 294
  - capture conversion, 103, 104
  - checked casts and unchecked casts, 121
  - class literals, 474
  - declarations, 131
  - determining the class being instantiated, 478
  - field declarations, 390
  - functional interfaces, 323
  - generic classes and type parameters, 195
  - generic interfaces and type parameters, 280
  - method and constructor formal parameters, 396
  - method reference expressions, 530
  - method result type, 396
  - normal annotations, 310
  - raw types, 66
  - reference type casting, 117
  - reference types and values, 52
  - superclasses and subclasses, 200
  - superinterfaces, 202
  - superinterfaces and subinterfaces, 281
  - type erasure, 64
  - types, classes, and interfaces, 89
  - where types are used, 77
- parenthesized expressions**, 475
  - conditional operator `?:`, 572, 572
  - constant expressions, 606
  - forms of expressions, 460
  - identify potentially applicable methods, 511
  - phase 1: identify matching arity methods applicable by strict invocation, 511
- phase 1: identify matching arity methods applicable by strict invocation**, 511
  - compile-time declaration of a method reference, 534, 534, 534, 535, 535
  - compile-time step 2: determine method Signature, 503, 503
  - invocation applicability inference, 678
  - more specific method inference, 686
  - phase 2: identify matching arity methods applicable by loose invocation, 512
  - phase 3: identify methods applicable by variable arity invocation, 513, 513
- phase 2: identify matching arity methods applicable by loose invocation**, 512

- compile-time declaration of a method reference, 534, 534
- compile-time step 2: determine method Signature, 503, 503
- more specific method inference, 686
- phase 1: identify matching arity methods applicable by strict invocation, 512
- phase 3: identify methods applicable by variable arity invocation, 513**
  - choosing the constructor and its arguments, 483
  - choosing the most specific method, 514
  - compile-time declaration of a method reference, 534, 534
  - compile-time step 2: determine method Signature, 503, 503
  - invocation applicability inference, 678
  - invocation type inference, 680
  - more specific method inference, 686
  - phase 2: identify matching arity methods applicable by loose invocation, 513
- poly expressions, 460**
  - boolean conditional expressions, 579
  - choosing the most specific method, 515
  - class instance creation expressions, 478
  - conditional operator `? :`, 572, 572
  - expression compatibility constraints, 663, 663
  - identify potentially applicable methods, 511
  - lambda expressions, 594
  - method reference expressions, 531
  - more specific method inference, 687
  - numeric conditional expressions, 579
  - parenthesized expressions, 476
- postfix decrement operator --, 544**
  - floating-point operations, 48, 49
  - integer operations, 43, 44
- normal and abrupt completion of evaluation, 465
- operators `++` and `--`, 616
- variables, 80
- postfix expressions, 542**
  - final variables, 86
  - forms of expressions, 460
  - syntactic classification of a name according to context, 151
- postfix increment operator ++, 543**
  - floating-point operations, 48, 49
  - integer operations, 43, 44
  - normal and abrupt completion of evaluation, 465
  - operators `++` and `--`, 616
  - variables, 80
- potentially applicable methods, 509**
  - compile-time declaration of a method reference, 533, 533, 534
  - compile-time step 1: determine class or interface to search, 500
  - compile-time step 2: determine method Signature, 503
  - phase 1: identify matching arity methods applicable by strict invocation, 511
  - phase 2: identify matching arity methods applicable by loose invocation, 512
  - phase 3: identify methods applicable by variable arity invocation, 513
- predefined annotation types, 302**
- prefix decrement operator --, 546**
  - floating-point operations, 48, 49
  - integer operations, 43, 44
  - normal and abrupt completion of evaluation, 465
  - operators `++` and `--`, 616
  - variables, 80
- prefix increment operator ++, 546**
  - floating-point operations, 48, 49

integer operations, 43, 44  
normal and abrupt completion of evaluation, 465  
operators ++ and --, 616  
variables, 80

**preparation of a class or interface type, 359**

kinds of variables, 83  
link test: verify, prepare, (optionally) resolve, 354

**preventing instantiation of a class, 266**

constructor declarations, 256

**primary expressions, 471**

access to a protected member, 167  
forms of expressions, 460  
postfix expressions, 542

**primitive types and values, 42**

class literals, 473  
conditional operator ? :, 572  
evaluation, denotation, and result, 459  
kinds of types and values, 41  
literals, 24  
reifiable types, 65  
unboxing conversion, 103  
variables, 80  
where types are used, 77

**program exit, 374**

**programs and program order, 641**

happens-before order, 644  
synchronization order, 642

**public classes, 384**

**public interfaces, 400**

## Q

**qualified access to a protected constructor, 167**

**qualified expression names, 157**

access control, 161

constant expressions, 606  
field access expressions, 494  
field declarations, 213  
members and constructors of parameterized types, 64

**qualified package names, 155**

**qualified packageortypenames, 155**

**qualified this, 475**

declarations, 132  
syntactic classification of a name according to context, 150

**qualified type names, 156**

access control, 161  
members and constructors of parameterized types, 64  
single-static-import declarations, 184  
single-type-import declarations, 181  
static-import-on-demand declarations, 184  
type-import-on-demand declarations, 183

## R

**raw types, 66**

assignment contexts, 108  
functional interfaces, 323  
invocation contexts, 113  
method reference expressions, 531  
reifiable types, 65  
subtyping among class and interface types, 72  
type arguments of parameterized types, 61  
unchecked conversion, 103  
variables of reference type, 81  
where types are used, 77

**reading final fields during construction, 654**

**reclassification of contextually ambiguous names, 152**

method invocation expressions, 499

- reduction**, 663
  - invocation applicability inference, 678
- reference conditional expressions**, 580
- reference equality operators == and !=**, 567
  - objects, 55
- reference type casting**, 117
  - casting contexts, 115
- reference types and values**, 52
  - class literals, 473
  - evaluation, denotation, and result, 459
  - initial values of variables, 87
  - kinds of types and values, 41
  - variables, 80
- references**, 7
- reifiable types**, 64
  - @SafeVarargs, 307
  - array creation expressions, 487
  - array initializers, 335
  - expressions and run-time checks, 463, 464
  - formal parameters, 228
  - method reference expressions, 531
  - type comparison operator instanceof, 564
- relational operators**, 562
  - constant expressions, 605
- relationship to predefined classes and interfaces**, 7
- remainder operator %**, 554
  - evaluate operands before operation, 468
  - floating-point operations, 49
  - integer operations, 44
  - normal and abrupt completion of evaluation, 465
- repeatable annotation types**, 298
  - @Repeatable, 308
  - evolution of annotation types, 403
  - multiple annotations of the same type, 318
- requirements in overriding**, 289
  - variables of reference type, 81
- requirements in overriding and hiding**, 246
  - compile-time checking of Exceptions, 345
  - method throws, 239
  - requirements in overriding, 289, 289, 289
  - type of a lambda expression, 603
  - type of a method reference, 538
  - variables of reference type, 81
- resolution**, 675
  - invocation applicability inference, 679
  - invocation type inference, 681
- resolution of symbolic references**, 359
  - link test: verify, prepare, (optionally) resolve, 355
- return statement**, 436
  - break, continue, return, and throw statements, 624
  - constructor body, 260, 260
  - instance initializers, 255
  - interface method body, 291
  - method body, 240
  - normal and abrupt completion of statements, 406
  - static initializers, 255
- run-time evaluation of array access expressions**, 492
  - array access, 333
  - evaluation order for other expressions, 471
  - normal and abrupt completion of evaluation, 465, 465
- run-time evaluation of array creation expressions**, 488
  - evaluation order for other expressions, 471
  - initial values of variables, 87
  - kinds of variables, 83
  - normal and abrupt completion of evaluation, 464, 465
- run-time evaluation of class instance creation expressions**, 484

- evaluation order for other expressions, 471
- normal and abrupt completion of evaluation, 464
- throw statement, 439

**run-time evaluation of lambda expressions,** 604

- creation of new class instances, 366
- evaluation order for other expressions, 471
- normal and abrupt completion of evaluation, 464

**run-time evaluation of method invocation,** 520

- evaluation order for other expressions, 471
- overloading, 251

**run-time evaluation of method references,** 539

- creation of new class instances, 366
- evaluation order for other expressions, 471
- normal and abrupt completion of evaluation, 464

**run-time handling of an exception,** 349

- expressions and run-time checks, 464
- initial values of variables, 87
- kinds of variables, 84
- throw statement, 437
- try statement, 443

## S

**scope of a declaration,** 139

- basic for statement, 427
- class body and member declarations, 206
- class declarations, 191
- class literals, 474
- compile-time step 1: determine class or interface to search, 500
- enhanced for statement, 430, 430
- enum constants, 268

- field declarations, 213
- formal parameters, 228
- forward references during field initialization, 222

- generic classes and type parameters, 195
- generic constructors, 259
- generic interfaces and type parameters, 280
- generic methods, 237
- import declarations, 180
- interface body and member declarations, 282
- interface declarations, 278
- lambda parameters, 598
- local class declarations, 407
- local variable declarators and types, 410
- member type declarations, 254
- method declarations, 226
- named packages, 178
- reclassification of contextually ambiguous names, 152, 153
- simple package names, 155
- top level type declarations, 185
- try statement, 442
- try-with-resources, 448
- type variables, 57

**semantics of final fields,** 654

**separators,** 38

- lexical grammar, 9

**shadowing,** 144

- compile-time step 1: determine class or interface to search, 500
- obscuring, 147
- scope of a declaration, 139
- simple expression names, 156

**shadowing and obscuring,** 142

- basic for statement, 427
- class body and member declarations, 206
- class declarations, 191
- enhanced for statement, 430

- enum constants, 268
- field declarations, 213
- formal parameters, 228
- generic classes and type parameters, 195
- generic constructors, 259
- import declarations, 180
- interface declarations, 278
- lambda parameters, 598
- local class declarations, 407
- local variable declarators and types, 410
- member type declarations, 254
- method declarations, 226
- named packages, 178
- top level type declarations, 185
- try statement, 442
- try-with-resources, 448
- shared variables**, 640
  - happens-before order, 644
- shift operators**, 561
  - constant expressions, 605
  - integer operations, 43
  - unary numeric promotion, 125
- simple assignment operator** =, 582
  - assignment contexts, 109
  - expressions and run-time checks, 463, 464
  - normal and abrupt completion of evaluation, 465, 465
- simple expression names**, 156
  - constant expressions, 606
  - field access expressions, 494
- simple method names**, 160
  - method declarations, 226
- simple package names**, 155
- simple packageortypenames**, 155
- simple type names**, 156
- single-element annotations**, 312
  - annotation type elements, 295
  - annotations, 308
- single-static-import declarations**, 184
  - declarations, 130, 131
  - identify potentially applicable methods, 509
  - import declarations, 180
  - reclassification of contextually ambiguous names, 153, 153
  - scope of a declaration, 139
  - simple method names, 160
  - single-type-import declarations, 181
  - syntactic classification of a name according to context, 150
- single-type-import declarations**, 180
  - declarations, 130, 131
  - import declarations, 180
  - reclassification of contextually ambiguous names, 153
  - scope of a declaration, 139
  - single-static-import declarations, 184
  - syntactic classification of a name according to context, 150
- sleep and yield**, 636
- standalone expressions**, 460
  - boolean conditional expressions, 579
  - choosing the most specific method, 515
  - class instance creation expressions, 478
  - conditional operator ? :, 572, 572
  - expression compatibility constraints, 663, 663
  - identify potentially applicable methods, 511
  - lambda expressions, 594
  - method reference expressions, 531
  - more specific method inference, 687
  - numeric conditional expressions, 579
  - parenthesized expressions, 476
- statements**, 410
- static fields**, 216, 393
  - generic classes and type parameters, 195
  - kinds of variables, 83, 83

- when initialization occurs, 361
- static initializers**, 255, 400
  - class body and member declarations, 206
  - definite assignment and static initializers, 628
  - exception checking, 347
  - final fields, 219
  - generic classes and type parameters, 195
  - inner classes and enclosing instances, 197
  - return statement, 436
  - simple expression names, 157
  - static initializers, 400
  - throw statement, 439
- static member type declarations**, 254
  - anonymous class declarations, 485
  - class modifiers, 191
  - generic classes and type parameters, 195
  - interface modifiers, 278, 279
- static methods**, 233, 398
  - generic classes and type parameters, 195
  - interface method declarations, 402
  - simple expression names, 157
- static-import-on-demand declarations**, 184
  - declarations, 130, 131
  - identify potentially applicable methods, 509
  - import declarations, 180
  - reclassification of contextually ambiguous names, 153, 153
  - scope of a declaration, 139
  - simple method names, 160
  - syntactic classification of a name according to context, 150
  - type-import-on-demand declarations, 183
- strictfp classes**, 194
  - fp-strict expressions, 462
- strictfp interfaces**, 279
  - fp-strict expressions, 462
- strictfp methods**, 235
  - fp-strict expressions, 462
- String concatenation operator +**, 557
  - boolean type and boolean values, 51
  - class String, 56
  - constructor declarations, 256
  - creation of new class instances, 366
  - floating-point operations, 48
  - integer operations, 44
  - normal and abrupt completion of evaluation, 464
  - objects, 53, 54
  - String contexts, 114
  - types, classes, and interfaces, 88
- String contexts**, 114
  - boolean type and boolean values, 51
- String conversion**, 105
  - String concatenation operator +, 557
  - String contexts, 114
- String literals**, 35
  - class String, 56
  - comments, 22
  - constant expressions, 605
  - creation of new class instances, 366
  - escape sequences for character and String literals, 37
  - lexical literals, 473
  - reference equality operators == and !=, 567
  - unicode, 16
- strings**, 56
  - lexical literals, 473
  - literals, 24
  - objects, 53
  - String literals, 35, 36
- subsequent modification of final fields**, 655
- subtyping**, 71
  - assignment contexts, 108
  - checked casts and unchecked casts, 121
  - choosing the most specific method, 514

- constraint formulas, 661
- invocation contexts, 113
- method throws, 238
- narrowing reference conversion, 99
- parameterized types, 59
- subtyping constraints, 668
- type arguments of parameterized types, 61, 61
- widening reference conversion, 99
- subtyping among array types, 73**
  - array types, 330
- subtyping among class and interface types, 71**
  - try statement, 442
- subtyping among primitive types, 71**
- subtyping constraints, 668**
- superclasses and subclasses, 200**
  - class members, 206
  - class Object, 55
  - enum types, 267
  - final classes, 194
  - kinds of variables, 83
  - loading process, 357
  - subtyping among class and interface types, 72
  - syntactic classification of a name according to context, 150
  - where types are used, 76, 77
- superclasses and superinterfaces, 384**
  - loading process, 357
  - superinterfaces, 401
  - verification of the binary representation, 359
- superinterfaces, 202, 401**
  - checked casts at run time, 122
  - class members, 206
  - subtyping among class and interface types, 72
  - superinterfaces and subinterfaces, 281
  - syntactic classification of a name according to context, 150
  - types, classes, and interfaces, 88
  - where types are used, 76, 77
- superinterfaces and subinterfaces, 280**
  - interface members, 282
  - loading process, 357
  - subtyping among class and interface types, 72
  - superclasses and subclasses, 202
  - superinterfaces, 203
  - syntactic classification of a name according to context, 150
  - where types are used, 76, 77
- switch statement, 419**
  - scope of a declaration, 140, 140
  - switch statements, 621
- switch statements, 621**
- synchronization, 632**
  - objects, 55
  - synchronized methods, 235
  - synchronized statement, 439
  - volatile fields, 220
- synchronization order, 642**
  - actions, 640
  - interaction with the memory model, 372
- synchronized methods, 235, 398**
  - class Object, 56
  - synchronization, 632
  - synchronized statement, 440
- synchronized statement, 439**
  - create frame, synchronize, transfer control, 528
  - objects, 55
  - synchronization, 632
  - synchronized statements, 624
- synchronized statements, 624**

**syntactic classification of a name according to context**, 149

- access control, 161
- declarations, 132
- reference types and values, 53

**syntactic grammar**, 10

- compilation units, 177
- input elements and tokens, 20
- lexical translations, 16

**syntax**, 689

## T

**this**, 474

- field initialization, 221, 222
- initialization of fields in interfaces, 285
- instance initializers, 255
- static initializers, 256
- static methods, 234

**threads and locks**, 631

- objects, 55
- throw statement, 437

**throw statement**, 437

- break, continue, return, and throw statements, 624
- causes of Exceptions, 343
- exception analysis of statements, 346
- initial values of variables, 87
- kinds of variables, 84
- normal and abrupt completion of statements, 406, 406
- run-time handling of an exception, 349

**top level type declarations**, 185

- class instance creation expressions, 477
- class modifiers, 191
- compilation units, 177
- determining accessibility, 163
- form of a binary, 377

- host support for packages, 175
- interface modifiers, 279
- package members, 173, 174
- scope of a declaration, 139, 139, 139
- shadowing, 146
- single-static-import declarations, 184
- single-type-import declarations, 181
- when initialization occurs, 361

**transient fields**, 219, 393

**try statement**, 440

- @Target, 303
- compile-time checking of Exceptions, 345
- declarations, 131, 132
- definite assignment and parameters, 626
- exception analysis of statements, 347
- expressions and run-time checks, 463, 464
- final variables, 85
- initial values of variables, 87
- kinds of variables, 84
- labeled statements, 413
- method throws, 239
- reclassification of contextually ambiguous names, 152
- run-time handling of an exception, 349
- scope of a declaration, 141
- shadowing and obscuring, 142
- shared variables, 640
- syntactic classification of a name according to context, 151
- throw statement, 437, 438, 438
- try statements, 624
- where types are used, 76, 78

**try statements**, 624

**try-catch statement**, 444

- try statement, 443

**try-catch-finally statement**, 445

- try statement, 443

**try-finally statement**, 445

- try statement, 443
- try-with-resources**, 447
  - @Target, 303
  - final variables, 85
  - local variable declaration statements, 409
  - scope of a declaration, 141
  - syntactic classification of a name according to context, 150
  - try statement, 444
  - where types are used, 76
- try-with-resources (basic)**, 448
- try-with-resources (extended)**, 451
- type arguments of parameterized types**, 60
  - capture conversion, 103
  - checked casts and unchecked casts, 121
  - class instance creation expressions, 477, 477
  - constraint formulas, 661
  - explicit constructor invocations, 261
  - method invocation expressions, 500
  - method reference expressions, 530
  - reference types and values, 52
  - reifiable types, 65
  - subtyping among class and interface types, 72
  - subtyping constraints, 669
  - type equality constraints, 670
  - types, classes, and interfaces, 89
  - unchecked conversion, 103
  - where types are used, 77
- type comparison operator instanceof**, 564
  - expressions and run-time checks, 463
  - objects, 54
  - syntactic classification of a name according to context, 151
  - where types are used, 76, 78
- type compatibility constraints**, 667
- type equality constraints**, 670
- type erasure**, 64
  - assignment contexts, 109
  - cast expressions, 549
  - checked casts and unchecked casts, 122
  - checked casts at run time, 122
  - choosing the most specific method, 516
  - class type parameters, 385
  - compile-time step 3: is the chosen method appropriate?, 519
  - constructor Signature, 258
  - create frame, synchronize, transfer control, 528
  - declarations, 131
  - evaluate arguments, 522
  - field declarations, 390
  - form of a binary, 378, 379, 380
  - invocation contexts, 114
  - method and constructor formal parameters, 396
  - method and constructor type parameters, 395
  - method result type, 396
  - method Signature, 230
  - raw types, 66
  - requirements in overriding and hiding, 246
  - type variables, 58
- type inference**, 659
  - compile-time step 2: determine method Signature, 504
  - generic constructors, 259
  - generic methods, 237
- type of a constructor**, 259
  - members and constructors of parameterized types, 63
- type of a lambda expression**, 602
  - checked exception constraints, 671
  - expression compatibility constraints, 664
  - invocation type inference, 682, 683
  - lambda parameters, 597
- type of a method reference**, 537

checked exception constraints, 672

**type of an expression**, 461

**type variables**, 57

class literals, 474

field declarations, 390

generic classes and type parameters, 194

generic constructors, 259

generic interfaces and type parameters, 279

generic methods, 237

intersection types, 70

method and constructor formal parameters, 396

method result type, 396

reference types and values, 52

type erasure, 64

types, classes, and interfaces, 89

where types are used, 77

**type-import-on-demand declarations**, 183

declarations, 130

import declarations, 180

reclassification of contextually ambiguous names, 153

scope of a declaration, 139

shadowing, 146

static-import-on-demand declarations, 185

syntactic classification of a name according to context, 152

**types**, 41

capture conversion, 103

lexical literals, 473

literals, 24

null literal, 38

throw statement, 438

**types, classes, and interfaces**, 88

**types, values, and variables**, 41

## U

**unary minus operator** -, 547

constant expressions, 605

floating-point operations, 48

integer literals, 30, 30

integer operations, 43

unary numeric promotion, 125

**unary numeric promotion**, 124

array access, 333

array access expressions, 491

array creation expressions, 487

bitwise complement operator ~, 548

numeric contexts, 124

shift operators, 561

unary minus operator -, 547

unary plus operator +, 547

**unary operators**, 544

final variables, 86

**unary plus operator** +, 547

constant expressions, 605

floating-point operations, 48

integer operations, 43

unary numeric promotion, 125

**unboxing conversion**, 102

additive operators, 556, 556

array creation expressions, 487

assert statement, 418

assignment contexts, 108

binary numeric promotion, 125

bitwise complement operator ~, 548

boolean equality operators == and !=, 567

boolean logical operators &, ^, and |, 569

casting contexts, 115, 115, 115

conditional operator ? :, 572, 579

conditional-and operator &&, 570, 570

conditional-or operator ||, 571, 571

do statement, 424

- equality operators, 565
  - floating-point operations, 49
  - if-then statement, 415
  - if-then-else statement, 416
  - integer bitwise operators `&`, `^`, and `|`, 568
  - integer operations, 44
  - invocation contexts, 113
  - iteration of for statement, 427
  - logical complement operator `!`, 548
  - multiplicative operators, 550
  - numeric contexts, 124
  - numerical comparison operators `<`, `<=`, `>`, and `>=`, 563
  - numerical equality operators `==` and `!=`, 566
  - postfix decrement operator `--`, 544
  - postfix increment operator `++`, 543
  - prefix decrement operator `--`, 546
  - prefix increment operator `++`, 546
  - switch statement, 421
  - unary minus operator `-`, 547
  - unary numeric promotion, 124, 124
  - unary plus operator `+`, 547
  - while statement, 423
  - unchecked conversion**, 103
    - `@SafeVarargs`, 307
    - assignment contexts, 108
    - casting contexts, 115, 115
    - invocation contexts, 113
    - method result, 238
    - type compatibility constraints, 668
    - variables of reference type, 81
  - unicode**, 15
    - character literals, 34
    - lexical grammar, 9
    - primitive types and values, 42
    - unicode escapes, 17
  - unicode escapes**, 17
  - escape sequences for character and String literals, 38
  - input elements and tokens, 19
  - lexical translations, 16
  - unicode, 16
  - unloading of classes and interfaces**, 373
    - kinds of variables, 83
  - unnamed packages**, 179
    - compilation units, 177
  - unreachable statements**, 452
    - final fields and static constant variables, 391
    - instance initializers, 255
    - lambda body, 599
    - static initializers, 255
  - uses of inference**, 677
- V**
- value set conversion**, 106
    - assignment contexts, 109
    - binary numeric promotion, 126
    - casting contexts, 115
    - compound assignment operators, 588, 590
    - create frame, synchronize, transfer control, 528
    - evaluation, denotation, and result, 459
    - floating-point types, formats, and values, 45
    - fp-strict expressions, 462
    - invocation contexts, 113
    - simple assignment operator `=`, 583, 584
    - unary minus operator `-`, 547
    - unary numeric promotion, 124
  - variables**, 80
    - evaluation, denotation, and result, 459
  - variables of primitive type**, 81
  - variables of reference type**, 81
    - `@SafeVarargs`, 307
    - type of an expression, 461

variables, 80

**verification of the binary representation**, 358

link test: verify, prepare, (optionally) resolve, 354

**volatile fields**, 220

happens-before order, 644

synchronization order, 642

## W

**wait**, 633

happens-before order, 643

**wait sets and notification**, 632

class Object, 56

**well-formed executions**, 647

executions, 646

**what binary compatibility is and is not**, 382

**when initialization occurs**, 361

final variables, 85

initialize test: execute initializers, 355

**when reference types are the same**, 56

checked casts at run time, 122

constraint formulas, 661

type equality constraints, 670

**where annotations may appear**, 313

annotation types, 292

annotations, 308

class modifiers, 191

constructor modifiers, 258

enum constants, 268

field (constant) declarations, 283

field modifiers, 215

formal parameters, 228

generic classes and type parameters, 195

generic interfaces and type parameters, 280

interface modifiers, 278

lambda parameters, 598

local variable declaration statements, 409

method declarations, 286

method modifiers, 231

named packages, 178

**where types are used**, 75

@Target, 303

lexical translations, 17

syntactic classification of a name according to context, 150

where annotations may appear, 313

**while statement**, 423

boolean type and boolean values, 51

while statements, 621

**while statements**, 621

**white space**, 20

input elements and tokens, 20

lexical grammar, 9

lexical translations, 16

**widening and narrowing primitive conversion**, 99

casting contexts, 115

**widening primitive conversion**, 94

assignment contexts, 107

binary numeric promotion, 126

casting contexts, 115, 115

invocation contexts, 112, 113

numeric contexts, 124

unary numeric promotion, 124, 124

widening and narrowing primitive conversion, 99

**widening reference conversion**, 99

assignment contexts, 107

casting contexts, 115, 115

floating-point operations, 48

integer operations, 44

invocation contexts, 113, 113

**word tearing**, 657

**write-protected fields**, 656