Machine Translated by Google

# OCP

## Oracle® Certified Professional Java SE 17 Developer

# STUDY GUIDE

**EXAM 1Z0-829**

**Includes one year of FREE access after activation to the interactive online learning environment and study tools:**

**3 custom practice exams**

**More than 500 electronic flashcards**

**Searchable key term glossary**

**SCOTT SELIKOFF**
**JEANNE BOYARSKY**

SYBEX®
A Wiley Brand

Machine Translated by Google

Machine Translated by Google

# OCP

# Profesional certificado de Oracle®

# Desarrollador Java SE 17

## Guía de estudio
## Examen 1Z0-829

Scott Selikoff

Jeanne Boyarsky

SYBEX®
A Wiley Brand

Machine Translated by Google

Copyright © 2022 de John Wiley & Sons, Inc. Todos los derechos reservados.

Publicado por John Wiley & Sons, Inc., Hoboken, Nueva Jersey.
Publicado simultáneamente en Canadá y el Reino Unido.

978-1-119-86458-5

978-1-119-86460-8 (excluido)
978-1-119-86459-2 (excluido)

Ninguna parte de esta publicación puede reproducirse, almacenarse en un sistema de recuperación ni transmitirse de ninguna forma ni por ningún medio, ya sea electrónico, mecánico, fotocopia, grabación, escaneo o de otro modo, excepto según lo permitido en virtud de la Sección 107 o 108 de la Ley de las Naciones Unidas de 1976. Ley de derechos de autor de los Estados Unidos, sin el permiso previo por escrito del editor o la autorización mediante el pago de la tarifa correspondiente por copia a Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, o en la web en www.copyright.com. Las solicitudes de permiso al editor deben dirigirse al Departamento de Permisos, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, o en línea en www . .wiley.com/go/permission.

Límite de responsabilidad/exención de responsabilidad de garantía: el editor y el autor no hacen declaraciones ni ofrecen garantías con respecto a la exactitud o integridad del contenido de este trabajo y específicamente renuncian a todas las garantías, incluidas, entre otras, las garantías de idoneidad para un propósito particular. No se puede crear ni extender ninguna garantía mediante materiales promocionales o de ventas. Los consejos y estrategias contenidos en este documento pueden no ser adecuados para todas las situaciones.
Este trabajo se vende en el entendido de que el editor no se dedica a prestar servicios legales, contables ni otros servicios profesionales. Si se requiere asistencia profesional, se deben buscar los servicios de un profesional competente. Ni el editor ni el autor serán responsables de los daños que surjan de este documento. El hecho de que en este trabajo se haga referencia a una organización o sitio web como una cita y/o una fuente potencial de información adicional no significa que el autor o el editor apruebe la información que la organización o el sitio web pueda proporcionar o las recomendaciones que pueda hacer. Además, los lectores deben tener en cuenta que los sitios web de Internet enumerados en este trabajo pueden haber cambiado o desaparecido entre el momento en que se escribió este trabajo y el momento en que se leyó.

Para obtener información general sobre nuestros otros productos y servicios o para soporte técnico, comuníquese con nuestro Departamento de Atención al Cliente dentro de los Estados Unidos al (800) 762-2974, fuera de los Estados Unidos al (317) 572-3993 o por fax (317) 572- 4002.

Wiley también publica sus libros en una variedad de formatos electrónicos. Es posible que parte del contenido que aparece impreso no esté disponible en formatos electrónicos. Para obtener más información sobre los productos Wiley, visite nuestro sitio web en www.wiley.com.

Número de control de la Biblioteca del Congreso: 2022932106

Marcas comerciales: WILEY, el logotipo de Wiley, Sybex y el logotipo de Sybex son marcas comerciales o marcas comerciales registradas de John Wiley & Sons, Inc. y/o sus afiliados, en los Estados Unidos y otros países, y no pueden usarse sin permiso por escrito. Oracle y java son marcas comerciales registradas de Oracle, Inc. Todas las demás marcas comerciales son propiedad de sus respectivos dueños. John Wiley & Sons, Inc. no está asociado con ningún producto o proveedor mencionado en este libro.

Imagen de portada: © Jeremy Woodhouse/Getty Images

Diseño de portada: Wiley

Machine Translated by Google

Para mi mamá a quien amo muchísimo: recibirás un centavo por cada página de este libro que leas.

—Scott

Al escenario de las conferencias de Java y su recuperación post-COVID.

—Jeanne

Machine Translated by Google

# Expresiones de gratitud

Scott y Jeanne quisieran agradecer a numerosas personas por sus contribuciones a este libro. Gracias a Kezia Endsley y Archana Pragash por guiarnos a lo largo del proceso y mejorar el libro en muchos sentidos. Gracias a Janeice DelVecchio por ser nuestra editora técnica mientras escribíamos este libro. Janeice señaló muchos errores sutiles además de los grandes. Gracias a Elena Felder por ser nuestra correctora técnica y encontrar los errores que logramos colarle a Janeice. Y un agradecimiento especial a nuestra editora Tiffany Taylor, por encontrar errores sutiles que todos (¡incluidos nosotros!) pasamos por alto. Este libro tampoco sería posible sin muchas personas en Wiley, incluidos Kenyon Brown, Pete Gaughan, Christine O'Connor y muchos otros.

Scott no podría haber llegado a este punto sin su esposa, Patti, y su familia, cuyo amor y apoyo hacen posible este libro. Le gustaría agradecer a sus hijas gemelas, Olivia y Sophia, y a su hija menor, Elysia, por su paciencia y comprensión y por traerle una taza de café frío cuando llegó “la hora de que papá trabajara en su oficina”. Scott quisiera extender su gratitud a su maravillosamente paciente coautora, Jeanne, por este octavo libro. Él no sabe cómo ella lo soporta, pero está contento de que así sea y está encantado con la calidad de los libros que producimos. Finalmente, Scott quisiera agradecer a su madre, Barbara Selikoff (una maestra jubilada), por enseñarle el valor de la educación, y a su padre, Mark Selikoff, por inculcarle los beneficios de trabajar duro.

Jeanne quisiera agradecer personalmente a todos los que la mantuvieron cuerda durante la pandemia de COVID, especialmente a Dani, Elena, Janeice, Joslyn, Norm, Rodrigo, Scott y Wendy, junto con el equipo de liderazgo de NYJavaSIG. También le gustaría agradecer al equipo de liderazgo de la KCDC (Conferencia de Desarrollo de la Ciudad de Kansas) por celebrar la primera conferencia en persona a la que asistió, donde pudo probar el material del libro con asistentes desprevenidos. Scott fue un gran coautor y mejoró todo lo que Jeanne escribió mientras escribía sus propios capítulos. Muchas gracias a todos en CodeRanch.com quienes hicieron y respondieron preguntas y comentarios sobre nuestros libros. Finalmente, Jeanne quisiera agradecer a todos los nuevos programadores de CodeRanch.com . y los equipos de robótica FIRST FRC 694 y FTC 310/479/8365 por los recordatorios constantes de cómo piensan los nuevos programadores.

Tanto Scott como Jeanne desean agradecer enormemente a los lectores de nuestros libros. Es una gran sensación escuchar a todos aquellos que disfrutaron el libro y aprobaron el examen. También nos gustaría agradecer a quienes señalaron errores e hicieron sugerencias para mejorar el libro Java 17. En enero de 2022, los dos primeros eran Tomasz Kasprzyk y Jos Roseboom.

Machine Translated by Google

# Sobre los autores

Scott Selikoff es un autor y desarrollador de software profesional con más de 20 años de experiencia en el desarrollo de sistemas completos basados en bases de datos. Scott, experto en una gran cantidad de lenguajes y plataformas de software, actualmente trabaja como ingeniero de software en Google, especializándose en arquitectura y servicios en la nube.

Originario de Toms River, Nueva Jersey, Scott obtuvo su licenciatura en Matemáticas e Informática de la Universidad de Cornell en 2002, después de tres años de estudio.
En 2003, recibió su Maestría en Ingeniería en Ciencias de la Computación, también de la Universidad de Cornell. Como alguien con un profundo amor por la educación, Scott siempre ha disfrutado enseñar a otros conceptos nuevos. Scott es líder del grupo de usuarios de Java de Garden State y ayuda a facilitar debates e intercambio de ideas dentro de la comunidad. También ha impartido conferencias en múltiples universidades y congresos.

Scott vive en Nueva Jersey con su amada esposa, Patti; tres hijas increíbles, las gemelas Olivia y Sofía y la pequeña Elysia; una perra muy juguetona, Georgette; y tres gatos tontos, Snowball, Sugar y Minnie Mouse. En su tiempo libre, toca el violín en la Orquesta Multigeneracional de Toms River. Puede obtener más información sobre Scott en www.linkedin.com/en/selikoff o sígalo en Twitter @ScottSelikoff.

Jeanne Boyarsky fue seleccionada como campeona de Java en 2019 y es líder de NYJavaSIG. Ha trabajado como desarrolladora de Java durante más de 20 años en un banco de la ciudad de Nueva York, donde desarrolla, asesora y lleva a cabo capacitaciones. Además de ser moderador senior en CodeRanch.com En su tiempo libre, trabaja en la base del código del foro. Jeanne también es mentora de la división de programación de un equipo de robótica de FIRST, donde trabaja con estudiantes que recién comienzan con Java. También habla en varias conferencias cada año.

Jeanne obtuvo su licenciatura en 2002 y su maestría en información informática. Licenciada en tecnología en 2005. Disfrutó obtener su maestría en un programa en línea mientras trabajaba a tiempo completo. ¡Esto fue antes de que la educación en línea fuera genial! Jeanne también es una Toastmaster distinguida y una Scrum Master. Puede obtener más información sobre Jeanne en www.jeanneboyarsky.com o síguela en Twitter @JeanneBoyarsky.

Scott y Jeanne son moderadores en CodeRanch.com . foros y se le puede contactar allí para hacer preguntas y comentarios. También son coautores de un blog técnico llamado Down Home Country Coding en www.selikoff.net.

Además de este libro, Scott y Jeanne son autores de siete libros sobre Java más vendidos:

- OCA: Guía de estudio del programador I de Java 8 (Sybex, 2015)
- OCP: Guía de estudio del programador Java 8 II (Sybex, 2016)
- Pruebas de práctica de programador OCA/OCP Java 8 (Sybex, 2017)
- Guía de estudio del programador I de OCP Java 11 (Sybex, 2019)
- Guía de estudio del programador II de OCP Java 11 (Sybex, 2020)
- Guía de estudio completa para desarrolladores de OCP Java 11 (Sybex, 2020)
- Pruebas de práctica de OCP Java 11 (Sybex, 2021)

Actualmente están en el proceso de escribir pruebas de práctica para desarrolladores de OCP Java 17. El libro saldrá a finales de este año.

Machine Translated by Google

# Acerca del editor técnico

Janeice DelVecchio ha sido desarrolladora de software profesional durante 12 años y ha amado la programación y las computadoras durante toda su vida. Editar libros técnicos es una tarea divertida para ella porque le gusta encontrar y corregir defectos de todo tipo. En su trabajo diario, utiliza una gama muy amplia de habilidades con tecnologías que incluyen computación en la nube, automatización de procesos, pruebas unitarias avanzadas y devops. También es voluntaria en CodeRanch.com, donde dirige la clase de Java conocida como Cattle Drive. Es experta en el lenguaje de programación Java. Si le pregunta qué idioma es el mejor, le dirá que los idiomas son herramientas y que elija el que se adapte a su caso de uso. El primer idioma que aprendió fue BÁSICO y algún día espera aprender sobre desarrollo de juegos. En su tiempo libre, le gusta cocinar, resolver acertijos, jugar videojuegos y criar pollos. Le encanta comer sushi, beber cerveza artesanal y acariciar perros; su placer culpable es la música pop de los 80. Vive en el condado de Litchfield, Connecticut.

# Acerca del corrector técnico

Elena Felder se dedicó al desarrollo de Java cuando el lenguaje carecía incluso de genéricos, y está encantada de que el lenguaje, sus herramientas y su comunidad hayan seguido creciendo y adaptándose para mantenerse al día con éxito en un mundo en constante cambio. Ella revisó uno de los primeros capítulos de la Guía de estudio de certificación Java 8 de Jeanne y Scott por diversión y terminó haciéndolo profesionalmente desde entonces.

Machine Translated by Google

# Contenido de un vistazo


| | | |
|---|---|---|
| Introducción | | xxiii |
| Prueba de evaluación | | xlv |
| Capítulo 1 | Bloques de construcción | 1 |
| Capitulo 2 | Operadores | [illegible] |
| Capítulo 3 | Tomando decisiones | 101 |
| Capítulo 4 | API principales | 155 |
| Capítulo 5 | Métodos | 219 |
| Capítulo 6 | Diseño de clase | 275 |
| Capítulo 7 | Más allá de las clases | 345 |
| Capítulo 8 | Lambdas e interfaces funcionales | 419 |
| Capítulo 9 | Colecciones y genéricos | 463 |
| Capítulo 10 | Corrientes | 531 |
| Capítulo 11 | Excepciones y localización | 591 |
| Capítulo 12 | Módulos | 661 |
| Capítulo 13 | concurrencia | 721 |
| Capítulo 14 | E/S | 785 |
| Capítulo 15 | JDBC | 863 |
| Apéndice | Respuestas a las preguntas de revisión | 909 |
| Índice | | 963 |

Machine Translated by Google

Machine Translated by Google

# Contenido


Introducción xxiii

Prueba de evaluación xlv

Capítulo 1 Bloques de construcción 1

Aprendiendo sobre el medio ambiente 2
- Componentes principales de Java 2
- Descargando un JDK 3

Comprender la estructura de clases 4
- Campos y métodos 4
- Comentarios 5
- Clases y archivos fuente 7

Escribir un método main() 8
- Creando un método main() 8
- Pasar parámetros a un programa Java 9

Comprensión de las declaraciones e importaciones de paquetes 11
- Paquetes 12
- comodines 13
- Importaciones redundantes 13
- Conflictos de nombres 15
- Creando un nuevo paquete dieciséis
- Compilación y ejecución de código con paquetes dieciséis
- Compilando en otro directorio 18
- Compilando con archivos JAR 20
- Crear un archivo JAR 20
- Ordenar elementos en una clase 21

Creando objetos 23
- Llamando a constructores 23
- Campos de miembros de lectura y escritura 24
- Ejecutar bloques inicializadores de instancia 24
- Siguiendo el orden de inicialización 25

Comprender los tipos de datos 26
- Usando tipos primitivos 27
- Usar tipos de referencia 29
- Distinguir entre primitivos y tipos de referencia 30
- Crear clases contenedoras 31
- Definición de bloques de texto 32

Declaración de variables 34
- Identificación de identificadores 35
- Declaración de múltiples variables 36

Machine Translated by Google




Inicializando variables 38
Creando variables locales 38
Pasar parámetros de constructor y método 40
Definición de variables de instancia y clase 41
Inferir el tipo con var 41
Gestión del alcance variable 45
Limitación del alcance 45
Alcance de seguimiento 46
Aplicar alcance a las clases 47
Revisión del alcance 48
Destruyendo objetos 48
Comprender la recolección de basura 48
Seguimiento de elegibilidad 49
Resumen 51
Conceptos básicos del examen 52
Preguntas de revisión 54

Capitulo 2 Operadores sesenta y cinco

Comprender los operadores de Java 66
Tipos de operadores 66
Precedencia del operador 67
Aplicar operadores unarios 69
Operadores de complemento y negación 70
Operadores de incremento y decremento 71
Trabajar con operadores aritméticos binarios 72
Operadores aritméticos 72
Promoción numérica 75
Asignar valores 77
Operador de asignación 77
Valores de fundición 77
Operadores de asignación compuesta 81
Valor de retorno de los operadores de asignación 82
Comparando valores 83
Operadores de igualdad 83
Operadores relacionales 84
Operadores logicos 87
Operadores condicionales 88
Tomar decisiones con el operador ternario 90
Resumen 92
Conceptos básicos del examen 92
Preguntas de revisión 94

Capítulo 3 Tomando decisiones 101

Crear declaraciones para la toma de decisiones 102
Declaraciones y bloques 102
La declaración si 103

Machine Translated by Google



La declaración else 104
Código de acortamiento con coincidencia de patrones 106
Aplicar declaraciones de cambio 110
La declaración de cambio 110
La expresión del interruptor 115
Escribir bucles while 121
La declaración mientras 121
La declaración hacer/mientras 123
Bucles infinitos 123
Construyendo bucles _ 124
El bucle for 124
El bucle para cada uno 129
Controlar el flujo con ramificaciones 131
Bucles anidados 131
Agregar etiquetas opcionales 132
La declaración de ruptura 133
La declaración de continuación 135
La declaración de devolución 137
Código inalcanzable 138
Revisión de ramificaciones 139
Resumen 139
Conceptos básicos del examen 140
Preguntas de revisión 142
Capítulo 4 API principales 155
Crear y manipular cadenas 156
Concatenando 157
Métodos de cadena importantes 158
Encadenamiento de métodos 169
Usando la clase StringBuilder 170
Mutabilidad y encadenamiento 171
Creando un constructor de cadenas 172
Métodos importantes de StringBuilder 172
Entendiendo la igualdad 175
Comparando iguales() y == 175
El grupo de cuerdas 176
Comprender las matrices 178
Creando una matriz de primitivas 179
Crear una matriz con variables de referencia 180
Usando una matriz 182
Clasificación 183
buscando 184
Comparando 185
Usando métodos con Varargs 187
Trabajar con matrices multidimensionales 188

Machine Translated by Google



Calcular con API matemáticas 190
Encontrar el mínimo y el máximo 190
Redondear números 191
Determinar el techo y el piso 191
Calcular exponentes 192
Generando números aleatorios 192
Trabajar con fechas y horas Crear 192
fechas y horas Manipular 193
fechas y horas Trabajar con períodos 197
Trabajar con duraciones 199
Período versus duración 202
Trabajar con instantes 204
Contabilizar el horario de 205
verano Resumen Preguntas de repaso de 206
conceptos 208
básicos del examen 209
210

Capítulo 5 Métodos 219

Métodos de diseño 220
Modificadores de acceso 221
Especificadores opcionales 222
Tipo de devolución 224
Nombre del método 226
Lista de parámetros 226
Firma del método 227
Lista de excepciones 227
Cuerpo del método 228
Declaración de variables locales y de instancia 228
Modificadores de variables locales 229
Variables efectivamente finales 230
Modificadores de variables de instancia 231
Trabajando con Varargs 232
Creando métodos con Varargs 232
Llamar a métodos con Varargs 233
Accediendo a elementos de un Vararg 234
Usar Varargs con otros parámetros del método 234
Aplicar modificadores de acceso 235
Acceso Privado 235
Acceso al paquete 236
Acceso protegido 237
Acceso público 242
Revisión de modificadores de acceso 242

Machine Translated by Google



Acceso a datos estáticos 243
Diseño de métodos y variables estáticos 243
Acceso a una variable o método estático 244
Clase versus instancia Membresía 245
Modificadores de variables 248
estáticas 250
Inicializadores 251
estáticos Importaciones estáticas 253
Pasar datos entre 253
métodos Pasar 255
objetos Devolver objetos Autoboxing y 256
Unboxing Variables 258
Sobrecarga de 259
métodos 260
Tipos de 261
261
referencia 261
Primitivas Autoboxing 262
Arrays 263
Varargs Uniéndolo 264
todo Resumen 265

Capítulo 6 Conceptos básicos 275
del examen Preguntas de 276
repaso Diseño de 276
clases 278
Comprensión de la herencia 279
Declarar una 279
subclase 281
Modificadores de 281
clase Herencia única o múltiple 282
Heredar objetos Crear clases 283
Extender una clase Aplicar 284
modificadores de acceso 286
a clases Acceder a esta 286
referencia Llamar a la súper 287
referencia Declarar constructores Crear un constructor El constructor predeterminado Llamar a c 289
Llamar a constructores principales con super() 292
Inicializando objetos 297
Inicializando clases 297
Inicializando campos 298
finales Inicializando instancias 300

Machine Translated by Google



Miembros herederos 304
Anular un método 305
Redeclarar métodos privados 311
Ocultar métodos estáticos 311
Ocultar variables 313
Redacción de métodos finales 314
Crear clases abstractas 315
Presentación de clases abstractas 315
Declaración de métodos abstractos 317
Creando una clase concreta 318
Crear constructores en clases abstractas 320
Detectar declaraciones inválidas 321
Creando objetos inmutables 323
Declarar una clase inmutable 323
Realizar una copia defensiva 325
Resumen 326
Conceptos básicos del examen 327
Preguntas de revisión 330

Capítulo 7 Más allá de las clases 345

Implementación de interfaces 346
Declaración y uso de una interfaz 346
Ampliar una interfaz 348
Heredar una interfaz 349
Insertar modificadores implícitos 351
Declaración de métodos de interfaz concretos 353
Trabajar con enumeraciones 361
Crear enumeraciones simples 361
Uso de enumeraciones en declaraciones de cambio 363
Agregar constructores, campos y métodos 364
Clases de sellado 367
Declarar una clase sellada 367
Compilando clases selladas 368
Especificación del modificador de subclase 369
Omitiendo la Cláusula de permisos 370
Interfaces de sellado 372
Revisar las reglas de la clase sellada 372
Encapsular datos con registros 373
Comprender la encapsulación 374
Aplicar registros 375
Comprender la inmutabilidad de registros 377
Declaración de constructores 378
Personalización de registros 381
Crear clases anidadas 382

Machine Translated by Google



Declarar una clase interna 382
Crear una clase anidada estática 386
Escribir una clase local 387
Definir una clase anónima Revisar 389
clases anidadas Comprender el 391
objeto de polimorfismo frente a los 392
objetos de conversión de 393
referencia La 395
instancia de Operador 397
Polimorfismo y método Anular Anular vs. Ocultar 397
miembros Resumen Preguntas de 399
repaso de 401
conceptos básicos 402
del examen 404

Capítulo 8 Lambdas e interfaces funcionales 419

Escribir lambdas simples 420
Mirando un ejemplo de Lambda 420
Aprender la sintaxis de Lambda 422
Codificación de interfaces funcionales 426
Definición de una interfaz funcional 426
Agregar métodos de objetos 427
Uso de referencias de métodos 429
Llamar a métodos estáticos 430
Llamar a métodos de instancia en un objeto particular 430
Llamar a métodos de instancia en un parámetro 432
Llamando a constructores 433
Revisión de referencias de métodos 433
Trabajar con interfaces funcionales integradas 434
Proveedor implementador 435
Implementación de Consumidor y BiConsumidor 436
Implementación de predicado y bipredicado 438
Implementación de función y bifunción 439
Implementación de UnaryOperator y BinaryOperator 440
Comprobación de interfaces funcionales 441
Uso de métodos convenientes en interfaces funcionales 442
Aprender las interfaces funcionales para primitivos 443
Trabajar con variables en Lambdas 445
Parámetros de listado 446
Usar variables locales dentro de un cuerpo Lambda 448
Hacer referencia a variables del cuerpo Lambda 449
Resumen 450
Conceptos básicos del examen 451
Preguntas de revisión 452

Machine Translated by Google



Capítulo 9 Colecciones y genéricos 463

Uso de API de colección comunes 464
Usando el operador de diamante 465
Agregar datos 466
Eliminación de datos 466
Elementos de conteo 467
Limpiar la colección 467
Verificar contenido 468
Eliminar con condiciones 468
Iterando 469
Determinando la igualdad 470
Usando la interfaz de lista 471
Comparación de implementaciones de listas 472
Crear una lista con una fábrica 472
Crear una lista con un constructor 473
Trabajar con métodos de lista 474
Convertir de lista a matriz 476
Usando la interfaz de configuración 477
Comparación de implementaciones de conjuntos 477
Trabajar con métodos establecidos 478
Uso de las interfaces Queue y Deque 479
Comparación de implementaciones de Deque 480
Trabajar con métodos de cola y deque 480
Usando la interfaz del mapa 483
Comparación de implementaciones de mapas 484
Trabajar con métodos de mapas 484
Llamar a métodos básicos 486
Iterando a través de un mapa 487
Obtener valores de forma segura 487
Reemplazo de valores 488
Poner si está ausente 488
Fusionar datos 488
Comparación de tipos de colecciones 490
Ordenar datos 492
Crear una clase comparable 492
Comparar datos con un comparador 496
Comparando comparable y comparador 497
Comparación de varios campos 498
Ordenar y buscar 500
Ordenar una lista 503
Trabajar con genéricos 503
Crear clases genéricas 504
Comprender el borrado de tipos 506

Machine Translated by Google




Implementación de interfaces genéricas 509
Escribir métodos genéricos 510
Crear un registro genérico 512
Tipos genéricos delimitadores 512
Poniendolo todo junto 517
Resumen 519
Conceptos básicos del examen 520
Preguntas de revisión 521

Capítulo 10 Corrientes 531

Devolver un opcional 532
Creando un opcional 533
Lidiando con una opción vacía 534
Usando transmisiones 536
Comprender el flujo de la tubería 536
Crear fuentes de transmisión 539
Uso de operaciones de terminal comunes 541
Uso de operaciones intermedias comunes 549
Armando el oleoducto 553
Trabajar con corrientes primitivas 557
Creando corrientes primitivas 557
Mapeo de flujos 560
Uso de opcional con secuencias primitivas 562
Resumen de estadísticas 564
Trabajar con conceptos avanzados de canalización de flujos 565
Vincular transmisiones a los datos subyacentes 565
Encadenamiento de opciones 566
Usando un divisor 569
Recopilación de resultados 570
Resumen 578
Conceptos básicos del examen 579
Preguntas de revisión 581

Capítulo 11 Excepciones y localización 591

Comprender las excepciones 592
El papel de las excepciones 592
Comprender los tipos de excepciones 593
Lanzar una excepción 596
Llamar a métodos que generan excepciones 598
Anulación de métodos con excepciones 599
Imprimir una excepción 600
Reconocer clases de excepción 600
Clases de excepción en tiempo de ejecución 601
Clases de excepción marcadas 604
Clases de error 605

Machine Translated by Google



Manejo de excepciones 605
Usando declaraciones try y catch 606
Bloques de captura de encadenamiento 607
Aplicar un bloque de captura múltiple 609
Agregar un bloque finalmente 611
Automatización de la gestión de recursos 615
Presentamos la prueba con recursos 615
Conceptos básicos de prueba con recursos 616
Aplicar efectivamente final 620
Comprender las excepciones suprimidas 621
Formatear valores 624
Formatear números 624
Formato de fechas y horas 625
Personalización del formato de fecha/hora 626
Apoyando la internacionalización y la localización 629
Elegir una ubicación 630
Localización de números 632
Localización de fechas 637
Especificación de una categoría regional 638
Cargando propiedades con paquetes de recursos 639
Crear un paquete de recursos 640
Elegir un paquete de recursos 641
Seleccionar valores de paquete de recursos 643
Formatear mensajes 645
Usando la clase de propiedades 645
Resumen 646
Conceptos básicos del examen 647
Preguntas de revisión 648
Capítulo 12 Módulos 661
Introducción de módulos 662
Explorando un módulo 663
Beneficios de los módulos 664
Crear y ejecutar un programa modular 664
Creando los archivos 665
Compilando nuestro primer módulo 666
Ejecutando nuestro primer módulo 668
Empaquetando nuestro primer módulo 669
Actualización de nuestro ejemplo para múltiples módulos 669
Actualización del módulo de alimentación 670
Crear un módulo de atención 670
Creando el Módulo de Charlas 672
Creando el módulo de personal 674

Machine Translated by Google



Profundizando en la Declaración del Módulo 675
Exportar un paquete 676
Requerir un módulo transitivamente 677
Abrir un paquete 679
Creando un servicio 680
Declaración de la interfaz del proveedor de servicios 681
Crear un localizador de servicios 682
Invocar desde un consumidor 684
Agregar un proveedor de servicios 685
Revisión de directivas y servicios 686
Descubriendo módulos 687
Identificación de módulos integrados 688
Obteniendo detalles con java 690
Describiendo con frasco 693
Aprendiendo sobre dependencias con jdeps 693
Usando la bandera --jdk-internals 695
Usando archivos de módulo con jmod 696
Creando tiempos de ejecución de Java con jlink 696
Revisar las opciones de la línea de comandos 697
Comparación de tipos de módulos 700
Módulos con nombre 701
Módulos automáticos 701
Módulos sin nombre 704
Revisión de tipos de módulos 704
Migrar una aplicación 704
Determinando el orden 705
Explorando una estrategia de migración de abajo hacia arriba 706
Explorando una estrategia de migración de arriba hacia abajo 707
Dividir un gran proyecto en módulos 709
No poder compilar con una dependencia cíclica 709
Resumen 711
Conceptos básicos del examen 712
Preguntas de revisión 713

Capítulo 13 concurrencia 721

Introduciendo hilos 722
Comprender la simultaneidad de subprocesos 723
Creando un hilo 724
Distinguir tipos de hilos 725
Gestionar el ciclo de vida de un hilo 727
Sondeo con sueño 727
Interrumpir un hilo 729
Crear subprocesos con la API de concurrencia 730
Presentamos el ejecutor de un solo subproceso 730

Machine Translated by Google



Cerrar un ejecutor de subprocesos 731
Envío de tareas 732
Esperando resultados 733
Programación de tareas 737
Aumento de la simultaneidad con los pools 739
Escribir código seguro para subprocesos 740
Comprender la seguridad de los subprocesos 740
Accediendo a datos con volátil 741
Protección de datos con clases atómicas 742
Mejorando el acceso con bloques sincronizados 744
Sincronización de métodos 746
Comprender el marco de bloqueo 747
Orquestar tareas con CyclicBarrier 751
Usar colecciones simultáneas 754
Comprender los errores de coherencia de la memoria 754
Trabajar con clases simultáneas 755
Obtención de colecciones sincronizadas 757
Identificar problemas de subprocesos 758
Entendiendo la vivacidad 758
Gestión de las condiciones de carrera 761
Trabajar con transmisiones paralelas 761
Creando flujos paralelos 762
Realizar una descomposición paralela 762
Procesamiento de reducciones paralelas 764
Resumen 770
Conceptos básicos del examen 770
Preguntas de revisión 772

Capítulo 14 E/S 785

Hacer referencia a archivos y directorios 786
Conceptualizando el sistema de archivos 786
Crear un archivo o ruta 789
Operando en archivo y ruta 793
Usar métodos de manejo de 793
funcionalidad compartida que declaran IOException 797
Proporcionar parámetros opcionales de NIO.2 797
Interactuar con rutas de NIO.2 799
Crear, mover y eliminar archivos y directorios Comparar archivos 805
con isSameFile() y mismatch() 809
Introducción a los flujos de E/S 811
Comprensión de los fundamentos de los flujos de E/S 811
Aprendizaje de la nomenclatura de los flujos 812
de E/S Lectura y escritura de 817
archivos Uso de flujos de E/S 817

Machine Translated by Google




Mejorando con archivos 820
Combinando con newBufferedReader() y newBufferedWriter() 822
Revisión de métodos comunes de lectura y escritura 823
Serializar datos 824
Aplicar la interfaz serializable 825
Datos de marcado transitorios 827
Garantizar que una clase sea serializable 827
Almacenamiento de datos con ObjectOutputStream y flujo de entrada de objetos 828
Comprender el proceso de creación de deserialización 830
Interactuar con los usuarios 832
Impresión de datos para el usuario 832
Lectura de entrada como flujo de E/S 833
Cerrar flujos del sistema 833
Adquirir información con la consola 834
Trabajar con API avanzadas 837
Manipulación de flujos de entrada 838
Descubriendo atributos de archivos 840
Atravesando un árbol de directorios 843
Buscando un directorio 847
Revisión de API clave 848
Resumen 850
Conceptos básicos del examen 851
Preguntas de revisión 852

Capítulo 15 JDBC 863

Introducción a las bases de datos relacionales y SQL 864
Identificar la estructura de una base de datos relacional 866
Escribir declaraciones SQL básicas 867
Presentamos las interfaces de JDBC 868
Conexión a una base de datos 870
Construyendo una URL JDBC 870
Obtener una conexión a la base de datos 871
Trabajar con una declaración preparada 873
Obtener una declaración preparada 874
Ejecutar una declaración preparada 875
Trabajar con parámetros 878
Actualización de múltiples registros 881
Obtener datos de un conjunto de resultados 882
Leer un conjunto de resultados 882
Obtener datos para una columna 885
Usando variables de enlace 887
Llamar a una declaración invocable 887

Machine Translated by Google



Llamar a un procedimiento sin parámetros 888
Pasar un parámetro IN 889
Devolver un parámetro OUT 889
Trabajar con un parámetro INOUT 890
Comparación de parámetros de declaración invocables 891
Usar opciones adicionales 891
Control de datos con transacciones 892
Comprometerse y retroceder 892
Marcadores con puntos de guardado 894
Revisión de API de transacciones 895
Cerrar recursos de base de datos 895
Resumen 897
Conceptos básicos del examen 898
Preguntas de revisión 900
Apéndice Respuestas a las preguntas de revisión 909
Capítulo 1: Bloques de construcción 910
Capítulo 2: Operadores 913
Capítulo 3: Tomar decisiones 916
Capítulo 4: API principales 921
Capítulo 5: Métodos 924
Capítulo 6: Diseño de clases 927
Capítulo 7: Más allá de las clases 932
Capítulo 8: Lambdas e interfaces funcionales 936
Capítulo 9: Colecciones y genéricos 939
Capítulo 10: Corrientes 942
Capítulo 11: Excepciones y localización 945
Capítulo 12: Módulos 949
Capítulo 13: Concurrencia 951
Capítulo 14: E/S 955
Capítulo 15: JDBC 959
Índice 963

Machine Translated by Google

# Introducción

Este libro es para aquellos que buscan obtener un título de Oracle Certified Professional: Java SE 17 Developer o Java Foundations Certified Junior Associate. Este libro también está dirigido a aquellos que buscan obtener una comprensión y apreciación más profunda de Java. No sólo queremos que apruebes tus exámenes, sino que también queremos ayudarte a mejorar y convertirte en un mejor desarrollador de software profesional.

El libro proporciona preparación detallada para los siguientes exámenes de certificación de Oracle:

Examen 1Z0-829: Desarrollador Java SE 17 El examen de Desarrollador cubre una amplia variedad de temas básicos en Java 17, incluidas clases, interfaces, flujos, colecciones, concurrencia y módulos.

Examen 1Z0-811: Fundamentos de Java El examen Fundamentos es un examen de certificación de nivel junior que contiene una variedad de temas introductorios y básicos de Java 8.

En esta introducción, comenzamos cubriendo información importante sobre los distintos exámenes. Luego pasamos a información sobre cómo está estructurado este libro. Finalmente, concluimos con una prueba de evaluación para que puedas ver cuánto estudio te queda por delante.

# Entendiendo el examen

Al final del día, el examen es una lista de preguntas. Cuanto más sepa sobre la estructura del examen, mejores resultados obtendrá. Por ejemplo, saber cuántas preguntas contiene el examen le permite gestionar mejor su progreso y el tiempo restante.
En esta sección, analizamos los detalles del examen, junto con algunos antecedentes de exámenes de certificación anteriores.

## Elegir qué examen realizar

Java tiene ahora más de 25 años y celebra su “nacimiento” en 1995. Como ocurre con cualquier cosa que tenga 25 años, existe una buena cantidad de historia y variaciones entre las diferentes versiones de Java. A lo largo de los años, los exámenes de certificación han cambiado para cubrir diferentes temas. También ha cambiado el número de exámenes y los nombres de las certificaciones.

Para Java 17, Oracle ha simplificado las cosas. Para convertirse en un profesional certificado de Oracle ahora es necesario aprobar solo un examen, no dos, y no hay exámenes de actualización de Java 17. Independientemente de las certificaciones anteriores que posea, todos realizan el mismo examen único de Java 17 para convertirse en un profesional certificado de Oracle.

Esto significa que su única opción es entre el examen OCP de Java 17 y los Fundamentos de Java. examen. Nuestro consejo es que sólo realice el examen de Fundamentos de Java si su empleador se lo ha pedido específicamente. Si bien es un examen más sencillo, está dirigido a una versión muy antigua de Java y no está destinado a profesionales que trabajan con Java todos los días.

Machine Translated by Google



# Considerando los objetivos del examen

Oracle proporciona una lista de objetivos para guiarlo sobre qué estudiar para cada examen. Cada objetivo define una lista de subobjetivos que proporcionan detalles adicionales sobre el objetivo. Lamentablemente, los objetivos no abarcan la cantidad total de material necesario para aprobar el examen.

Entonces, ¿cómo sabes qué estudiar? ¡Leyendo esta guía de estudio, por supuesto! hemos gastado Llevamos años estudiando los exámenes de certificación en todas sus formas y hemos cultivado cuidadosamente temas, materiales y preguntas de práctica que estamos seguros pueden conducir a aprobar el examen con éxito. Más recientemente, hemos trabajado mano a mano con Oracle para ayudar a crear y perfeccionar los objetivos y el material para los exámenes Java 11 y Java 17.

Como punto de partida, debe revisar la lista de objetivos presentados en esta introducción y marcar aquellos que no le resulten familiares. Esta lista, junto con la prueba de evaluación al final de esta introducción, le darán una idea aproximada de cuánto necesitará estudiar para el examen.

## Cambios en los exámenes

La Tabla I.1 muestra la información de los exámenes al momento de su publicación.

TABLA I.1 Información del examen

| Examen | Longitud | # de preguntas | Puntaje de aprobación |
|---|---|---|---|
| 1Z0-829 Desarrollador Java SE 17 | 90 minutos | 50 | 68% |
| 1Z0-811 Fundamentos de Java | 150 minutos | 75 | sesenta y cinco% |

Oracle tiende a alterar la duración del examen y la puntuación aprobatoria una vez que se publica. A Oracle también le gusta "modificar" los objetivos del examen con el tiempo. No sería una sorpresa que Oracle hiciera cambios menores en los objetivos del examen, el número de preguntas o la puntuación aprobatoria después de que este libro se imprima.

Si hay algún cambio en el examen después de la publicación de este libro, lo publicaremos en el página del libro de nuestro blog:

www.selikoff.net/ocp17

## Alcance de los objetivos

En exámenes de certificación anteriores, la lista de objetivos del examen tendía a incluir temas, clases y API específicos que necesitaba conocer para el examen. Por ejemplo, observe un objetivo para el examen OCP 8 (1Z0-809):

Machine Translated by Google



- Utilice BufferedReader, BufferedWriter, File, FileReader, FileWriter, FileInputStream, FileOutputStream, ObjectOutputStream, ObjectInputStream y PrintWriter en el paquete java.io.

Ahora compárelo con el objetivo equivalente para el examen OCP 17 (1Z0-829):

- Leer y escribir datos de consola y archivos usando I/O Stream.

¿Notas la diferencia? La versión anterior es más detallada y describe clases específicas que usted Necesito entender. La versión más nueva es mucho más vaga. También les brinda a los redactores del examen mucha más libertad para insertar una nueva característica sin tener que actualizar la lista de objetivos.

## Elegir las respuestas correctas

Cada examen consta íntegramente de preguntas de opción múltiple. Hay entre cuatro y siete respuestas posibles. Si una pregunta tiene más de una respuesta, la pregunta indica específicamente exactamente cuántas respuestas correctas hay. Este libro no hace eso. Decimos Elija todo lo que corresponda para hacer las preguntas más difíciles. Esto significa que las preguntas de este libro son generalmente más difíciles que las del examen. La idea es brindarte más práctica para que puedas encontrar la respuesta correcta más fácilmente en el examen real.

## Leer el código del examen

Muchas de las preguntas de cada examen son fragmentos de código en lugar de clases completas. Ahorrar espacio al no incluir importaciones y/o definiciones de clases deja espacio para muchos otros códigos. Sólo debe centrarse en las declaraciones de importación cuando la pregunta se refiera específicamente a ellas.

Por ejemplo, es común encontrar clases en el examen con declaraciones de importación y partes omitidas, como esta:

```
clase pública Zoo implementa Serializable {
    Nombre de cadena;
    // Getters/Setters/Constructores omitidos
}
```

En este caso, puede asumir que java.io.Serializable se importa y que los métodos como getName() y setName(), así como constructores relacionados, existen. Por ejemplo, esperaríamos que este código se compilara:

```
var nombre = nuevo Zoológico("Java Zoológico").getName();
```

## Encontrar material fuera de alcance

Cuando realiza un examen, es posible que vea algunas preguntas que parecen estar fuera de alcance. ¡No entrar en pánico! A menudo, estas preguntas no requieren saber nada sobre el tema para responder.

Machine Translated by Google



pregunta. Por ejemplo, después de leer este libro, debería poder detectar que lo siguiente no se compila, incluso si nunca ha oído hablar de la clase java.util.logging.Logger .

```
registrador final myLogger = Logger.getAnonymousLogger();
myLogger = Logger.getLogger(String.class.getName());
```

Las clases y métodos utilizados en esta pregunta no están dentro del alcance del examen, pero la razón por la que no se compila sí lo está. En particular, debes saber que no puedes reasignar una variable marcada como final.

¿Ver? No da tanto miedo, ¿verdad? Espere ver al menos algunas estructuras en el examen con las que no está familiarizado. Si no forman parte del material de preparación del examen, no es necesario que los comprenda para responder la pregunta.

## Revisar tipos de preguntas

La siguiente lista de temas está destinada a darle una idea de los tipos de preguntas y rarezas que podría encontrar en el examen. Ser consciente de estas categorías de preguntas puede ayudarle a obtener una puntuación más alta en un examen.

Preguntas con información adicional proporcionada Imagine que la pregunta incluye una declaración de que XMLParseException es una excepción marcada. Está bien si no sabes qué es XMLParseException o qué es XML, de hecho. (Si se lo pregunta, es un formato para datos). Esta pregunta es un regalo. Sabes que la pregunta es sobre el manejo de excepciones.

Preguntas con preguntas integradas Para responder algunas preguntas del examen, es posible que deba responder dos o tres subpreguntas. Por ejemplo, la pregunta puede contener dos líneas en blanco y pedirle que elija las dos respuestas que completan cada espacio en blanco. En algunos casos, las dos opciones de respuesta no están relacionadas, lo que significa que en realidad estás respondiendo varias preguntas, ¡no solo una! Estas preguntas se encuentran entre las más difíciles y que requieren más tiempo del examen porque contienen múltiples preguntas, a menudo independientes, para responder. Desafortunadamente, el examen no otorga crédito parcial, así que tenga cuidado al responder preguntas como estas.

Preguntas con API desconocidas Si ve una clase o método que no se trató en este libro, asuma que funciona como es de esperar. Algunas de estas API con las que puede encontrarse, como SecurityManager, estaban en el examen de Java 11 y no forman parte de los exámenes de Java 17. Suponga que la parte del código que utiliza esa API es correcta y busque con atención otros errores.

Preguntas con conceptos inventados o incorrectos En el contexto de un problema escrito, el examen puede mostrar un término o concepto que no tiene ningún sentido, como decir que una interfaz hereda de una clase, lo cual no es una afirmación correcta. En otros casos, el examen puede utilizar una palabra clave que no existe en Java, como struct. Para estos, sólo tienes que leer atentamente y reconocer cuando el examen utiliza terminología no válida.

Machine Translated by Google



Preguntas que realmente están fuera de alcance Al introducir nuevas preguntas, Oracle las incluye al principio como preguntas sin puntuación. Esto permite a los creadores del examen ver cómo les va a los examinados reales sin afectar su puntuación. Aún recibirá la cantidad de preguntas que enumera el examen. Sin embargo, es posible que algunos de ellos no cuenten. Estas preguntas sin puntuación pueden contener material fuera de alcance o incluso errores. No se marcarán como sin puntuación, por lo que aún tendrás que hacer todo lo posible para responderlas. Sigue los consejos anteriores para asumir que todo lo que no hayas visto antes es correcto. ¡Eso lo cubrirá si se cuenta la pregunta!

Como todos los exámenes, la certificación Oracle Certified Professional: Java SE 17 Developer o Java Foundations Certified Junior Associate de Oracle se actualiza periódicamente y eventualmente puede retirarse o reemplazarse. En algún momento, después de que Oracle ya no ofrezca este examen, las ediciones antiguas de nuestros libros y herramientas en línea se retirarán. Si compró este libro después de que se retiró el examen o está intentando registrarse en el entorno de aprendizaje en línea de Sybex después de que se retiró el examen, sepa que no ofrecemos garantías de que las herramientas Sybex en línea de este examen estarán disponibles una vez que finalice el examen. Ya no está disponible.

# Leyendo este libro

Podría ser útil tener alguna idea de cómo se escribió este libro. Esta sección contiene detalles sobre algunas de las estructuras y características comunes que encontrará en este libro, dónde acudir para obtener ayuda adicional y cómo obtener material adicional para este libro.

## ¿Quién debería comprar este libro?

Si desea obtener la certificación de programador Java OCP 17, este libro es definitivamente para usted. Si desea adquirir una base sólida en Java y su objetivo es prepararse para el examen, este libro también es para usted. Encontrará explicaciones claras de los conceptos que necesita comprender y mucha ayuda para alcanzar el alto nivel de competencia profesional que necesita para tener éxito en el campo elegido.

Este libro pretende ser comprensible para cualquiera que tenga un mínimo de conocimientos de Java. Si nunca antes ha leído un libro sobre Java, le recomendamos comenzar con un libro que enseñe programación desde el principio y luego volver a esta guía de estudio.

Este libro es para cualquier persona, desde estudiantes de secundaria hasta aquellos que comienzan a programar. viaje a profesionales experimentados que necesitan una revisión para la certificación.

Machine Translated by Google



# Cómo está organizado este libro

Este libro está dividido en 15 capítulos, además de material complementario en línea: un glosario de términos importantes, más de 500 tarjetas didácticas y tres exámenes de práctica que simulan el examen real.

A diferencia de los objetivos del examen, organizamos nuestros capítulos de forma orgánica para que cada capítulo se base en el material de los capítulos anteriores. También queremos que las cosas sean más fáciles de aprender y recordar. Esto significa que algunos capítulos cubren múltiples objetivos.

Los capítulos están organizados de la siguiente manera:

- Capítulo 1: Bloques de construcción describe los conceptos básicos de Java, como por ejemplo cómo ejecutar un programa. Cubre variables como primitivas, tipos de datos de objetos y variables de alcance. También analiza la recolección de basura.
- Capítulo 2: Operadores explica las operaciones con variables. También habla de casting y la precedencia de los operadores.
- Capítulo 3: Tomar decisiones cubre construcciones lógicas centrales como declaraciones de decisión, coincidencia de patrones y bucles.
- Capítulo 4: Las API principales funcionan con String, StringBuilder, matrices y fechas.
- Capítulo 5: Métodos explica cómo diseñar y escribir métodos. También introduce el acceso modificadores, que se utilizan a lo largo del libro.
- Capítulo 6: Diseño de clases cubre la estructura de clases, los constructores, la herencia y la inicialización. También te enseña cómo crear clases abstractas y métodos de sobrecarga.
- Capítulo 7: Más allá de las clases presenta muchos tipos de nivel superior (además de las clases), incluidas interfaces, enumeraciones, clases selladas, registros y clases anidadas. También cubre el polimorfismo.
- Capítulo 8: Lambdas e interfaces funcionales muestra cómo utilizar lambdas, referencias de métodos e interfaces funcionales integradas.
- Capítulo 9: Colecciones y genéricos muestra referencias de métodos, genéricos con comodines y colecciones. La parte de Colecciones cubre muchas interfaces, clases y métodos comunes que son útiles para el examen y en el desarrollo de software cotidiano.
- Capítulo 10: Streams explica las canalizaciones de flujos en detalle. También cubre la clase Opcional . Si desea adquirir habilidades en la creación de transmisiones, lea este capítulo más de una vez.
- Capítulo 11: Excepciones y localización demuestra los diferentes tipos de clases de excepción y cómo aplicarlas para crear programas más resistentes. Concluye con la localización y el formateo, que permiten que su programa admita sin problemas varios países o idiomas.
- Capítulo 12: Módulos detalla los beneficios de la nueva característica del módulo. Muestra cómo compilar y ejecutar programas de módulos desde la línea de comandos. Además, describe los servicios y cómo migrar una aplicación a una infraestructura modular.
- Capítulo 13: Concurrencia presenta el concepto de ciclo de vida de subprocesos y seguridad de subprocesos. Le enseña cómo crear programas multiproceso utilizando la API de concurrencia y transmisiones paralelas.

Machine Translated by Google



■ Capítulo 14: E/S le presenta la gestión de archivos y directorios utilizando E/S y API NIO.2. Cubre una serie de clases de flujo de E/S, le enseña cómo serializar datos y muestra cómo interactuar con un usuario. Además, incluye técnicas para utilizar secuencias para recorrer y buscar en el sistema de archivos.

■ Capítulo 15: JDBC proporciona los conceptos básicos para trabajar con bases de datos en Java, incluido trabajar con procedimientos almacenados y transacciones.

Al final de cada capítulo, encontrará algunos elementos que puede utilizar para prepararse para el examen:

Resumen Esta sección revisa los temas más importantes que se trataron en el capítulo y sirve como una buena revisión.

Conceptos básicos del examen Esta sección resume los aspectos más destacados que se trataron en el capítulo. Debería poder transmitir la información descrita.

Preguntas de repaso Cada capítulo concluye con al menos 20 preguntas de repaso. Debe responder estas preguntas y comparar sus respuestas con las proporcionadas en el Apéndice. Si no puede responder correctamente al menos el 80 por ciento de estas preguntas, regrese y revise el capítulo, o al menos aquellas secciones que parezcan tener dificultades.

Las preguntas de repaso, las pruebas de evaluación, los exámenes de práctica y otros ejemplos de código incluidos en este libro no se derivan de las preguntas reales del examen, ¡así que no los memorice! Aprender el tema subyacente no sólo le ayuda a aprobar el examen, sino que también le convierte en un programador de mayor calidad en el lugar de trabajo: el objetivo final de una certificación.

Para aprovechar al máximo este libro, debes leer cada capítulo de principio a fin antes yendo a los elementos del final del capítulo. Son muy útiles para comprobar y reforzar su comprensión. Incluso si ya estás familiarizado con un tema, deberías hojear el capítulo.
Hay una serie de sutilezas en Java que es posible que no encuentres incluso cuando trabajas con Java durante años. Por ejemplo, lo siguiente compila:

```
var $núm = (Entero)nulo;
```

Incluso un desarrollador de Java experimentado podría quedar desconcertado por este. El examen requiere que conozcas este tipo de sutilezas.

## Las convenciones usadas en este libro

Este libro utiliza ciertos estilos tipográficos para ayudarle a identificar rápidamente información importante y evitar confusiones sobre el significado de las palabras, como las indicaciones en pantalla. En particular, busque los siguientes estilos:

- El texto en cursiva indica términos clave que se describen detalladamente por primera vez en un capítulo. (Las cursivas también se utilizan para dar énfasis).

Machine Translated by Google



- Una fuente monoespaciada indica código o texto de línea de comandos. A menudo utilizamos negrita para resaltar palabras o métodos importantes dentro de un ejemplo de código.
- El texto en cursiva y monoespaciado indica una variable.

Además de estas convenciones de texto, que pueden aplicarse a palabras individuales o a párrafos completos, algunas convenciones resaltan segmentos de texto.

Un consejo es algo para llamar la atención sobre un aspecto del trabajo con una función de lenguaje o API.

Una nota indica información que es útil o interesante. A menudo es algo a lo que hay que prestar especial atención durante el examen.

Barras laterales

Una barra lateral es como una nota pero más larga. La información de la barra lateral es útil, pero no encaja en el flujo principal del texto.

**Real World Scenario**

Un escenario del mundo real es un tipo de barra lateral que describe una tarea o un ejemplo que está particularmente basado en el mundo real. Esto es algo que es útil en el mundo real pero que no aparecerá en el examen.

## Obteniendo ayuda

Ambos autores son moderadores en CodeRanch.com. Este sitio es un foro de programación bastante grande y activo, amigable para los principiantes de Java. Tiene un foro solo para este examen llamado Certificación de Programador. También tiene un foro llamado Beginning Java para preguntas no específicas de exámenes. Mientras lee el libro, no dude en hacer sus preguntas en cualquiera de esos foros. Podría ser que estés teniendo problemas para compilar una clase o simplemente estés confundido acerca de algo. Obtendrá una respuesta de un programador experto en Java. ¡Incluso podría ser uno de nosotros!

Recuerde consultar nuestra página de libros antes de realizar el examen. Contiene actualizaciones recientes. Oracle llega al examen.

www.selikoff.net/ocp17

Machine Translated by Google



## Entorno interactivo de aprendizaje en línea y banco de pruebas

Hemos reunido algunas herramientas en línea realmente excelentes para ayudarlo a aprobar los exámenes. El entorno interactivo de aprendizaje en línea que acompaña a esta guía de estudio proporciona un banco de pruebas y herramientas de estudio para ayudarle a prepararse para el examen. Al utilizar estas herramientas, puede aumentar drásticamente sus posibilidades de aprobar el examen en el primer intento.

Para registrarse y obtener acceso a este entorno interactivo de aprendizaje en línea, visite esta URL:

www.wiley.com/go/Sybextestprep

El banco de pruebas en línea incluye lo siguiente:

Tres exámenes de práctica Se proporcionan muchas preguntas de práctica a lo largo de este libro y en línea, incluida la prueba de evaluación, que encontrará al final de esta introducción, y las pruebas de los capítulos, que incluyen las preguntas de repaso al final de cada capítulo.
Además, hay tres exámenes de práctica adicionales. Utilice estas preguntas para evaluar su conocimiento del material de la guía de estudio. El banco de pruebas en línea se ejecuta en múltiples dispositivos.

Más de 500 tarjetas didácticas El banco de pruebas en línea incluye dos conjuntos de tarjetas didácticas escritas específicamente diez para impactarte con fuerza, ¡así que no te desanimes si no las dominas al principio! Están ahí para garantizar que esté realmente preparado para el examen. Y no te preocupes
armado con las preguntas de repaso, los exámenes de práctica y las tarjetas didácticas, ¡estará más que preparado cuando llegue el día del examen! Las preguntas se proporcionan en formato de tarjeta didáctica digital (una pregunta seguida de una única respuesta correcta). Puede utilizar las tarjetas didácticas para reforzar su aprendizaje y proporcionar preparación para el examen de último momento antes del examen.

Recursos adicionales Un glosario de términos clave de este libro y sus definiciones está disponible en formato PDF con capacidad de búsqueda completa.

# Estudiando para el examen

Esta sección incluye sugerencias y recomendaciones sobre cómo debe prepararse para el examen de certificación. En lugar de simplemente leer este libro, recomendamos escribir y ejecutar programas como parte del proceso de estudio. La forma en que estudias puede ser tan importante como lo que estudias.

## Crear un plan de estudio

Roma no se construyó en un día, por lo que no deberías intentar estudiar para el examen en un solo día.
Incluso si ha obtenido la certificación con una versión anterior de Java, la nueva prueba incluye características y componentes exclusivos de Java 12–17.

Machine Translated by Google



Una vez que haya decidido realizar el examen, debe elaborar un plan de estudio que se ajuste a su horario. Te recomendamos que dediques algo de tiempo cada día, aunque sean solo unos minutos durante el almuerzo, para leer o practicar para el examen. La idea es mantener el impulso durante todo el proceso de preparación del examen. Cuanto más constante seas en tu forma de estudiar, mejor preparado estarás para el examen. Trate de evitar tomarse unos días o semanas de descanso del estudio, o es probable que dedique mucho tiempo a volver a aprender el material existente en lugar de pasar a material nuevo.

## Crear y ejecutar el código

Aunque algunas personas pueden aprender Java simplemente leyendo un libro de texto, no es así como recomendamos que estudies para un examen de certificación. Queremos que escriba sus propias aplicaciones de muestra de Java a lo largo de este libro para que no sólo aprenda el material sino que también lo comprenda. Por ejemplo, puede que no sea obvio por qué la siguiente línea de código no se compila, pero si intenta compilarla usted mismo, el compilador de Java le indicará el problema:

```
valor flotante = 102,0; // NO COMPILA
```

Mucha gente publica la pregunta "¿Por qué no se compila este código?" en el foro CodeRanch.com . Si está atascado o simplemente tiene curiosidad acerca de un comportamiento en Java, le recomendamos que publique en el foro. Hay mucha gente agradable en la comunidad Java dispuesta a ayudarle.

### Clase de prueba de muestra

A lo largo de este libro, presentamos numerosos fragmentos de código y le preguntamos si se compilarán o no y cuál es su resultado. Puede colocar estos fragmentos dentro de una aplicación Java simple que inicie, ejecute el código y finalice. Puede lograr esto compilando y ejecutando una clase pública que contenga un método main() y agregando las declaraciones de importación necesarias, como las siguientes:

```
// Agregue aquí las declaraciones de importación necesarias
clase pública Clase de prueba {
    público estático vacío principal (String [] argumentos) {
        //Añadir código de prueba aquí

        // Agregue cualquier declaración impresa aquí
        System.out.println("¡Hola mundo!");
    }
}
```

Machine Translated by Google



Esta aplicación no es particularmente interesante: simplemente genera ¡Hola mundo! y sale.

Dicho esto, puedes insertar muchos de los fragmentos de código presentados en este libro en el archivo main() método para determinar si el código se compila, así como qué genera el código cuando se compila.

**Real World Scenario**

Software IDE

Mientras estudias para un examen, debes desarrollar código utilizando un editor de texto y un compilador Java de línea de comandos. Es posible que algunos de ustedes tengan experiencia previa con entornos de desarrollo integrados (IDE), como Eclipse, IntelliJ y Visual Studio Code. Un IDE es una aplicación de software que facilita el desarrollo de software para programadores de computadoras. Aunque estas herramientas son extremadamente valiosas en el desarrollo de software, pueden interferir con su capacidad para detectar problemas fácilmente en un examen.

## Identificar su eslabón más débil

Las preguntas de repaso de cada capítulo están diseñadas para ayudarle a centrarse en aquellas características del lenguaje Java en las que puede tener debilidades y que son conocimientos necesarios para el examen. Para cada capítulo, debe anotar qué preguntas se equivocó, comprender por qué se equivocó y estudiar esas áreas aún más. Una vez que haya releído el capítulo y haya escrito mucho código, podrá volver a realizar las preguntas de repaso. De hecho, puedes responder las preguntas de repaso una y otra vez para reforzar tu aprendizaje, siempre y cuando te expliques por qué cada respuesta es correcta.

“Estudiar demasiado” los exámenes de práctica en línea

Aunque recomendamos leer este libro y escribir sus propias aplicaciones de muestra varias veces, rehacer los exámenes de práctica en línea una y otra vez puede tener un impacto negativo a largo plazo. Por ejemplo, algunas personas estudian tanto los exámenes de práctica que acaban memorizando las respuestas. En este escenario, fácilmente pueden volverse demasiado confiados; es decir, pueden lograr puntuaciones perfectas en los exámenes de práctica pero pueden reprobar el examen real.

Machine Translated by Google



# Aplicar estrategias para tomar exámenes

Esta sección incluye sugerencias que puede utilizar cuando realice el examen. Si es un examinador experimentado o ya ha realizado un examen de certificación antes, la mayor parte de esto debería ser de conocimiento común. Para aquellos que realizan el examen por primera vez, ¡no se preocupen! Presentamos una serie de consejos y estrategias prácticas para ayudarle a prepararse para el examen.

## Uso del material de escritura proporcionado

Dependiendo de su centro de pruebas en particular, es posible que se le proporcione una hoja de papel en blanco o una pizarra para ayudarlo a responder preguntas. Según nuestra experiencia, lo más habitual es que se reparta una pizarra con un rotulador y una goma de borrar. Si se sienta y no le proporcionan nada, asegúrese de solicitar dichos materiales. Si no le dan un borrador, no dude en solicitar una segunda página de pizarra.

Después de comprobar primero si el código se compila, es hora de comprender qué es el programa. ¡hace! Una de las aplicaciones más útiles del material de escritura es el seguimiento del estado de las variables primitivas y de referencia. Por ejemplo, supongamos que encontró el siguiente fragmento de código en una pregunta sobre la recolección de basura:

```
Objeto o = nueva Tortuga();
Mamífero m = nuevo Mono();
Animal a = nuevo Conejo();
o = metro;
```

En una situación como esta, puede resultar útil dibujar un diagrama del estado actual de las referencias variables. A medida que cada variable de referencia cambia a qué objeto apunta, borra o tacha la flecha entre ellas y dibuja una nueva para un objeto diferente.

Usar el material de escritura para rastrear el estado también es útil para preguntas complejas que involucran un bucle, especialmente preguntas con bucles integrados. Por ejemplo, el valor de una variable puede cambiar cinco o más veces durante la ejecución de un bucle. Debe utilizar el material de escritura proporcionado para mejorar su puntuación.

Si bien no puede traer ningún material externo a un examen, puede anotar el material al comienzo del examen. Por ejemplo, si tiene problemas para recordar qué interfaces funcionales toman qué argumentos genéricos, podría resultar útil dibujar una tabla al comienzo del examen con el material de escritura proporcionado. Luego puede utilizar esta información para responder varias preguntas.

Machine Translated by Google



## Entendiendo la pregunta

La mayoría de las preguntas del examen contienen fragmentos de código y le piden que responda preguntas sobre ellos. Para aquellos elementos que contienen fragmentos de código, la pregunta número uno que recomendamos que responda antes de intentar resolver la pregunta es la siguiente:

¿Se compila el código?

Suena simple, pero muchas personas se lanzan a responder la pregunta sin verificar si el código realmente se compila. Si puede determinar si un conjunto particular de código se compila y qué línea o líneas hacen que no se compile, responder la pregunta a menudo resulta fácil.

Si todas las respuestas a una pregunta son valores impresos, es decir, no existe la opción No compila, considere esa pregunta como un regalo. Significa que cada línea se compila y es posible que pueda utilizar la información de esta pregunta para responder otras preguntas.

## Aplicar el proceso de eliminación

Aunque es posible que no sepas inmediatamente la respuesta correcta a una pregunta, si puedes reducir la pregunta de cinco respuestas a tres, tus probabilidades de adivinar la respuesta correcta mejoran notablemente. Además, si puedes reducir una pregunta de cuatro respuestas a dos, ¡duplicarás tus posibilidades de adivinar la respuesta correcta!

En algunos casos, es posible que puedas eliminar opciones de respuesta sin siquiera leer la pregunta. Si encuentra preguntas de este tipo en el examen, considérelo un regalo. ¿Puedes responder correctamente a la siguiente pregunta en la que se ha omitido el código de la aplicación?

1. ¿ Qué línea, cuando se inserta de forma independiente en la línea m1, permite que se compile el código?

   - Código omitido -

   A. resumen público final int swim();
   B. natación vacía abstracta pública ();
   C. natación abstracta pública ();
   D. natación nula abstracta pública () {}
   E. natación vacía pública () {}

Sin leer el código ni saber qué línea es m1 , podemos eliminar tres de las cinco opciones de respuesta. Las opciones A, C y D contienen declaraciones no válidas, lo que nos deja con las opciones B y E como las únicas respuestas correctas posibles.

Machine Translated by Google

En versiones anteriores del examen, el software de realización de exámenes permitía eliminar una opción haciendo clic derecho sobre ella. Luego, la opción se presentó con una línea tachada sobre ella. Desafortunadamente, Oracle ya no ofrece esta función, por lo que deberá utilizar el material de escritura proporcionado para realizar un seguimiento de las opciones. ¡Con suerte, Oracle recuperará esta función con una actualización!

## Saltarse preguntas difíciles

El software del examen incluye una opción para "calificar" una pregunta y revisar todas las preguntas marcadas al final del examen. Si tiene poco tiempo, responda una pregunta lo mejor que pueda y luego márquela para volver a consultarla más tarde.

Todas las preguntas tienen el mismo peso, por lo que dedicar 10 minutos a responder correctamente cinco preguntas es un uso mucho mejor del tiempo que dedicar 10 minutos a una sola pregunta. Si termina el examen antes de tiempo, tiene la opción de revisar las preguntas marcadas, así como todas las preguntas del examen, si así lo desea.

## Sospechar de las palabras fuertes

Muchas preguntas del examen incluyen opciones de respuesta con oraciones descriptivas en lugar de líneas de código. Cuando vea este tipo de preguntas, tenga cuidado con cualquier opción de respuesta que incluya palabras fuertes como "debe", "todos" o "no puedo". Si piensas en las complejidades de los lenguajes de programación, es raro que una regla no tenga excepciones o casos especiales. Por lo tanto, si está atrapado entre dos respuestas y una de ellas usa "debe" mientras que la otra usa "puede" o "puede", será mejor que elija la que tiene la palabra más débil, ya que es una afirmación más ambigua.

## Elegir la mejor respuesta

A veces lees una pregunta e inmediatamente detectas un error del compilador que te dice exactamente lo que plantea la pregunta. Otras veces, sin embargo, es posible que te quedes mirando la declaración de un método durante un par de minutos y no tengas idea de cuál es la respuesta. A diferencia de otras pruebas estandarizadas, no hay penalización por responder una pregunta incorrectamente o por dejarla en blanco. Si casi se te acaba el tiempo o simplemente no puedes decidir una respuesta, selecciona una opción aleatoria y continúa. Si ha podido eliminar incluso una opción de respuesta, entonces su suposición es mejor que la suerte.

**¡Responda todas las preguntas!**

Debe establecer una parada brusca cuando queden cinco minutos de tiempo restante del examen para asegurarse de haber respondido todas y cada una de las preguntas. Recuerde, si no responde una pregunta, definitivamente se equivocará y perderá puntos; pero si adivinas, al menos existe la posibilidad de que estés en lo cierto. ¡No hay nada de malo en adivinar!

Machine Translated by Google



En caso de duda, generalmente recomendamos elegir una respuesta aleatoria que incluya "No compila" si está disponible, aunque la opción que seleccione no es tan importante como asegurarse de no dejar ninguna pregunta sin respuesta en el examen.

## Descansar bien por la noche

Aunque muchas personas tienden a acumular todo el material que pueden en las horas previas a un examen, la mayoría de los estudios han demostrado que esta es una mala estrategia para realizar exámenes. ¡Lo mejor que podemos recomendarte que hagas antes de realizar un examen es descansar bien por la noche!

Dada la duración del examen y la cantidad de preguntas, el examen puede resultar bastante agotador, especialmente si es la primera vez que realiza un examen de certificación. Es posible que llegue esperando terminar 30 minutos antes y descubrir que solo ha completado una cuarta parte del examen y le queda la mitad del tiempo. En algún momento, puede comenzar a entrar en pánico, y es en esos momentos cuando estas habilidades para tomar exámenes son más importantes. Sólo recuerda respirar profundamente, mantener la calma, eliminar tantas respuestas incorrectas como puedas y asegurarte de responder todas las preguntas. ¡Es en momentos estresantes como estos cuando descansar bien y dormir bien por la noche es más beneficioso!

# tomando el examen

¿Entonces has decidido hacer el examen? ¡Eso esperamos, si has comprado este libro! En esta sección, analizamos el proceso de programación y realización del examen, junto con varias opciones para cada uno.

## Programar el examen

El examen lo administra Pearson VUE y se puede realizar en cualquier centro de pruebas de Pearson VUE. Para encontrar un centro de pruebas o registrarse para el examen, vaya a:

certview.oracle.com

A continuación, elija Administrar examen en Pearson | VUE. Si tiene algún problema para navegar por sitio web, vea nuestros consejos en

www.selikoff.net/exam-signup

Si no ha estado antes en el centro de pruebas, le recomendamos visitarlo con antelación. Alguno Los centros de pruebas son agradables y están administrados profesionalmente. Otros te meten en un pequeño armario con mucha gente hablando a tu alrededor. ¡No querrás realizar el examen con gente quejándose de sus portátiles rotos cerca!

Machine Translated by Google



En este momento, podrás reprogramar el examen sin penalización hasta 24 horas antes. Esto significa que puede registrarse en un horario conveniente con mucha antelación, sabiendo que puede retrasarse si no está listo para esa hora. La reprogramación es fácil y se puede realizar completamente en el sitio web de Pearson VUE. Esto puede cambiar, así que consulte las reglas antes de pagar.

## Realizar un examen supervisado en línea

Pearson VUE ofrece la posibilidad de realizar el examen en su hogar u oficina a través del servicio OnVUE. Agendas una fecha y hora específica para tomarlo de forma remota desde tu computadora personal o de trabajo. Esta opción es especialmente atractiva para quienes viven lejos de un centro de pruebas o pueden tener problemas de salud al realizar el examen en persona.

Antes de programar un examen supervisado en línea, le recomendamos encarecidamente que revise la lista de requisitos en el sitio web de Pearson VUE:

www.pearsonvue.com/oracle/onvue

Le recomendamos que realice el examen en cualquier lugar donde se sienta cómodo y seguro. Dicho esto, realizar un examen supervisado en línea es una experiencia muy diferente a realizar un examen en un centro de pruebas. A continuación se destacan algunos aspectos del proceso de examen supervisado en línea que consideramos importantes. Consulte el sitio web de Pearson VUE para obtener detalles adicionales, ya que están sujetos a cambios:

- Su computadora portátil o de escritorio debe cumplir con ciertos requisitos mínimos, debe incluir una cámara/micrófono y no debe tener monitores adicionales. No se permiten tabletas ni pantallas táctiles.
- Debe tener una conexión a Internet estable (se recomienda Ethernet por cable) y no estar detrás de un firewall o VPN corporativo.
- Un supervisor lo supervisará de cerca en vivo durante todo el examen, además de grabarlo. Está estrictamente prohibido moverse fuera de la vista de la cámara, mirar su teléfono celular o usar el baño.
- Su área de trabajo debe estar bien iluminada y su escritorio libre de todo material. Antes de comenzar el examen, el supervisor le pedirá que gire la cámara alrededor de su área para asegurarse de que no haya materiales inapropiados a su alcance o a la vista.
- El material de escritura durante el examen se proporciona en forma de un blanco digital en línea. tablero dentro del software de examen.
- Debe realizar la prueba en un lugar donde pueda garantizar la privacidad. Nadie más puede estar en la sala ni ver su examen. Si alguien entra sin darse cuenta, debe decirle que se vaya inmediatamente.
- El software de examen monitorea los movimientos de los ojos y la cabeza. Es posible que reciba un mensaje de advertencia mientras realiza el examen si parece que está apartando demasiado la vista de la pantalla.

La elección entre realizar el examen en un centro de pruebas o en casa es personal. Piensa detenidamente cuál se adapta mejor a tus necesidades.

Machine Translated by Google



## El día del examen

El examen requiere dos formas de identificación, incluida una emitida por el gobierno. Consulte la lista de identificaciones aceptables de Pearson aquí:

www.pearsonvue.com/policies/1S.pdf

Cuando realice el examen en persona, trate de no llevar demasiado consigo, ya que no se permitirá la entrada a la sala de examen. Si bien puedes revisar tus pertenencias, es mejor dejar artículos extra en casa o en el auto.

No se le permite traer papel, su teléfono ni nada similar a la sala de examen. Algunos centros son más estrictos que otros. ¡En un centro nos quitaron hasta los pañuelos! La mayoría de centros te permiten conservar tu DNI y tu dinero. Sin embargo, te observan mientras haces el examen, así que ni se te ocurra escribir notas sobre dinero.

Como mencionamos anteriormente, el centro de pruebas le brindará materiales de escritura para usar durante el examen, ya sea papel borrador o una pizarra. Si no le entregan estos materiales, recuerde preguntar. Estos elementos se recogen al final del examen.

Si bien no puede traer ninguna pertenencia a la sala de pruebas, algunos centros de pruebas ruidosos ofrecen tapones para los oídos. Si su centro de pruebas tiene mucho ruido de fondo (como bocinas de automóviles o obras en construcción), no está de más pedirle al supervisor un par de tapones para los oídos antes de comenzar el examen.

## Averiguar su puntuación

Tan pronto como complete su examen, sabrá si lo aprobó. Para obtener su puntaje real, deberá esperar hasta poder verificarlo en línea. Muchos examinados verifican su puntuación desde un dispositivo móvil mientras salen del centro de exámenes.

CertView generalmente se actualiza poco después de terminar el examen, pero en algunos casos puede demorar hasta una hora. Además de tu puntuación, también verás los objetivos en los que te equivocaste en una pregunta. Una vez que haya aprobado el examen 1Z0-829 y haya cumplido los requisitos previos requeridos, el título OCP 17 se otorgará en unos pocos días.

Oracle se ha asociado con Credly Acclaim, que es una plataforma Open Badges. Al obtener una certificación de Oracle, también recibe una “insignia” que puede elegir compartir públicamente con empleadores actuales o potenciales.

# Mapa objetivo

Este libro ha sido escrito para cubrir todos los objetivos de los exámenes Developer y Foundation.

Machine Translated by Google



# Desarrollador Java SE 17 (1Z0-829)

La siguiente tabla proporciona un desglose de la cobertura del examen de este libro para el examen Java SE 17 Developer (1Z0-829), y muestra el capítulo donde se cubre cada objetivo o subobjetivo.

| Objetivo del examen | Capítulo |
|---|---|
| Manejo de valores de fecha, hora, texto, numéricos y booleanos. | |
| Utilice primitivas y clases contenedoras, incluida la API de matemáticas, paréntesis, promoción de tipos y conversión para evaluar expresiones aritméticas y booleanas. | 1, 2, 4 |
| Manipular texto, incluidos bloques de texto, utilizando String y Clases de StringBuilder | 4 |
| Manipule objetos de fecha, hora, duración, período, instante y zona horaria utilizando la API de fecha y hora | 4 |
| Controlar el flujo del programa | |
| Cree construcciones de control de flujo de programas que incluyan if/else, declaraciones y expresiones de cambio, bucles y declaraciones de interrupción y continuación. | 3 |
| Utilizando el enfoque orientado a objetos de Java | |
| Declare y cree instancias de objetos Java, incluidos objetos de clase anidados, y explique el ciclo de vida del objeto, incluida la creación, la reasignación de referencias y la recolección de basura. | 1, 7 |
| Cree clases y registros, y defina y utilice campos y métodos estáticos y de instancia, constructores e inicializadores estáticos y de instancia. | 5, 6, 7 |
| Implementar sobrecarga, incluidos los métodos var-arg. | 5 |
| Comprender los alcances de las variables, utilizar la inferencia de tipos de variables locales, aplicar la encapsulación y hacer que los objetos sean inmutables. | 1, 6, 7, 8 |
| Implemente polimorfismo y diferencie el tipo de objeto del tipo de referencia. Realizar conversión de tipos, identificar tipos de objetos utilizando el operador de instancia y la coincidencia de patrones. | 3, 6, 7 |
| Crear y utilizar interfaces, identificar interfaces funcionales y utilizar métodos de interfaz privados, estáticos y predeterminados. | 7, 8 |
| Crear y usar enumeraciones con campos, métodos y constructores. | 7 |
| Manejo de excepciones | |
| Manejar excepciones usando try/catch/finally, try-with-resources y bloques multi catch, incluidas excepciones personalizadas | 11 |

Machine Translated by Google



| Objetivo del examen | Capítulo |
|---|---|
| Trabajar con matrices y colecciones | |
| Cree matrices Java, listas, conjuntos, mapas y colecciones Deque, y agregue, elimine, actualice, recupere y ordene sus elementos. | 4, 9 |
| Trabajar con Streams y expresiones Lambda | |
| Utilice objetos Java y flujos primitivos, incluidas expresiones lambda que implementen interfaces funcionales, para suministrar, filtrar, mapear, consumir y ordenar datos. | 10 |
| Realizar descomposición, concatenación y reducción, y agrupación y partición en flujos secuenciales y paralelos. | 10, 13 |
| Empaquetar e implementar código Java y utilizar la plataforma Java Sistema de módulos | |
| Defina módulos y sus dependencias, exponga el contenido del módulo incluso para reflexión. Definir servicios, productores y consumidores. | 12 |
| Compile código Java, produzca archivos jar modulares y no modulares, imágenes en tiempo de ejecución e implemente la migración utilizando módulos automáticos y sin nombre. | 12 |
| Gestionar la ejecución simultánea de código | |
| Cree subprocesos de trabajo utilizando Runnable y Callable, administre el ciclo de vida del subproceso, incluidas las automatizaciones proporcionadas por diferentes servicios de Ejecutor y API concurrentes. | 13 |
| Desarrollar código seguro para subprocesos, utilizando diferentes mecanismos de bloqueo y API concurrentes. | 13 |
| Procesar colecciones de Java simultáneamente, incluido el uso de flujos paralelos 13 | |
| Usando la API de E/S de Java | |
| Leer y escribir datos de consola y archivos mediante flujos de E/S | 14 |
| Serializar y deserializar objetos Java | 14 |
| Cree, recorra, lea y escriba objetos Path y sus propiedades utilizando la API java.nio.file | 14 |
| Accediendo a bases de datos usando JDBC | |
| Cree conexiones, cree y ejecute declaraciones básicas, preparadas y ejecutables, procese resultados de consultas y controle transacciones utilizando la API JDBC. | 15 |
| Implementación de la localización | |
| Implemente la localización utilizando configuraciones regionales, paquetes de recursos, análisis y formato de mensajes, fechas, horas y números, incluidos valores de moneda y porcentaje. | 11 |

Machine Translated by Google



# Fundamentos de Java (1Z0-811)

La siguiente tabla proporciona un desglose de la cobertura del examen de este libro para el examen Java Foundations (1Z0-811), y muestra el capítulo donde se cubre cada objetivo o subobjetivo.

Algunos temas están en el examen de Fundamentos de Java, pero no en el 1Z0-829. Estos están cubiertos aquí:

www.selikoff.net/java-foundations

Además, los objetivos pueden actualizarse si Oracle actualiza el examen Java Foundations para una versión posterior de Java. Visite nuestro sitio web para conocer esas actualizaciones también.

| Objetivo del examen | Capítulo |
|---|---|
| ¿Que es Java? | |
| Describir las características de Java. | 1 |
| Describir las aplicaciones del mundo real de Java. | 1 + en línea |
| Conceptos básicos de Java | |
| Describir el kit de desarrollo de Java (JDK) y el tiempo de ejecución de Java. Medio Ambiente (JRE) | 1 |
| Describir los componentes de la programación orientada a objetos. | 1 |
| Describir los componentes de un programa Java básico. | 1 |
| Compilar y ejecutar un programa Java | 1 |
| Elementos básicos de Java | |
| Identificar las convenciones a seguir en un programa Java. | 1 |
| Utilice palabras reservadas de Java | 1 |
| Utilice comentarios de una y varias líneas en programas Java | 1 |
| Importe otros paquetes de Java para hacerlos accesibles en su código | 1 |
| Describir el paquete java.lang | 1 |
| Trabajar con tipos de datos Java | |
| Declare e inicialice variables, incluida una variable usando final | 1 |
| Transmitir un valor de un tipo de datos a otro, incluida la promoción automática y manual. | 2 |

Machine Translated by Google



| Objetivo del examen | Capítulo |
|---|---|
| Declarar e inicializar una variable String | 1 |
| Trabajar con operadores de Java | |
| Utilice operadores aritméticos básicos para manipular datos, incluidos +, -, *, / y %. | 2 |
| Utilice los operadores de incremento y decremento. | 2 |
| Utilice operadores relacionales que incluyen ==, !=, >, >=, < y <= | 2 |
| Utilice operadores de asignación aritmética | 2 |
| Utilice operadores condicionales, incluidos &&, \|\| y ? | 2 |
| Describir la precedencia de operadores y el uso de paréntesis. | 2 |
| Trabajar con la clase String | |
| Desarrollar código que utilice métodos de la clase String. | 4 |
| Formatee cadenas usando secuencias de escape que incluyen %d, %n y %s | 11 |
| Trabajar con clases aleatorias y de matemáticas | |
| Utilice la clase aleatoria | En línea |
| Usa la clase de Matemáticas | 4 |
| Usar declaraciones de decisión | |
| Utilice la declaración de toma de decisiones (si-entonces y si-entonces-si no) | 3 |
| Utilice la declaración de cambio | 3 |
| Compare cómo == difiere entre primitivas y objetos | 4 |
| Compare dos objetos String utilizando los métodos compareTo y equals | 4 |
| Usar declaraciones en bucle | |
| Describir declaraciones en bucle | 3 |
| Utilice un bucle for que incluya un bucle for mejorado | 3 |
| Usa un bucle while | 3 |
| Utilice un bucle do- while | 3 |
| Compara y contrasta los bucles for, while y do- while | 3 |
| Desarrollar código que utilice declaraciones de interrupción y continuación. | 3 |

Machine Translated by Google



| Objetivo del examen | Capítulo |
|---|---|
| Depuración y manejo de excepciones | |
| Identificar errores de sintaxis y lógica. | 1, 2, 3 |
| Usar manejo de excepciones | 11 |
| Manejar excepciones comunes lanzadas | 11 |
| Usa bloques para intentar atrapar | 11 |
| Matrices y listas de matrices | |
| Utilice una matriz unidimensional | 4 |
| Crear y manipular un ArrayList | 9 |
| Recorrer los elementos de un ArrayList mediante el uso de iteradores y bucles, incluido el bucle for mejorado | 9 |
| Comparar una matriz y una ArrayList | 4, 9 |
| Clases y constructores | |
| Crea una nueva clase que incluya un método principal. | 1 |
| Usa el modificador privado | 5 |
| Describir la relación entre un objeto y sus miembros. | 6 |
| Describir la diferencia entre una variable de clase, una variable de instancia y una variable local | dieciséis |
| Desarrollar código que cree el constructor predeterminado de un objeto y modifique los campos del objeto. | 6 |
| Utilice constructores con y sin parámetros. | 6 |
| Desarrollar código que sobrecargue a los constructores. | 6 |
| Métodos Java | |
| Describir y crear un método. | 5 |
| Crear y utilizar métodos de acceso y mutador. | 5 |
| Crear métodos sobrecargados | 5 |
| Describir un método estático y demostrar su uso dentro de un programa. | 5 |

Machine Translated by Google



# Prueba de evaluación

Utilice la siguiente prueba de evaluación para evaluar su nivel actual de habilidad en Java para el 1Z0-829. Esta prueba está diseñada para resaltar algunos temas de sus fortalezas y debilidades para que sepa qué capítulos quizás desee leer varias veces. Incluso si obtiene buenos resultados en la prueba de evaluación, debe leer el libro de principio a fin, ya que los exámenes reales son bastante desafiantes.

1. ¿Cuál es el resultado de ejecutar el siguiente fragmento de código?

```
puntuación int final1 = 8, puntuación2 = 3;
char miPuntuación = 7;
var objetivo = cambiar (myScore) {
    predeterminado -> {if(10>score1) produce "desconocido";}
    puntuación del caso1 -> "excelente";
    caso 2, 4, 6 -> "bueno";
    puntuación de caso2, 0 -> {"malo";}
};
System.out.println(objetivo);
```

A. desconocido

B. genial

C. bueno

D. malo

E. desconocidogenialbuenomalo

F. Es necesario cambiar exactamente una línea para que se compile el código.

G. Es necesario cambiar exactamente dos líneas para que se pueda compilar el código.

H. Ninguna de las anteriores

2. ¿Cuál es el resultado del siguiente fragmento de código?

```
int luna = 9, estrella = 2 + 2 * 3;
sol flotante = estrella>10? 1: 3;
doble júpiter = (sol + luna) - 1.0f;
int marte = --luna <= 8? 2: 3;
System.out.println(sol+", "+júpiter+", "+marte);
```

R.1 , 11, 2

B.3.0 , 11.0, 2

C.1.0 , 11.0, 3

D.3.0 , 13.0, 3

E. 3.0f, 12, 2

F. El código no se compila porque una de las asignaciones requiere un número explícito elenco.

Machine Translated by Google



3. ¿Qué cambios, cuando se realizan de forma independiente, garantizan que el siguiente fragmento de código se imprima 100 en tiempo de ejecución? (Elija todas las que correspondan).

```
Lista<Integer> datos = nueva ArrayList<>();
IntStream.range(0,100).parallel().forEach(s -> data.add(s));
System.out.println(data.size());
```

A. Cambie los datos a una variable de instancia y márquelos como volátiles.

B. Eliminar paralelo() en la operación de flujo.

C. Cambie forEach() a forEachOrdered() en la operación de flujo.

D. Cambie paralelo() a serial() en la operación de flujo.

E. Envolver el cuerpo lambda con un bloque sincronizado.

F. El fragmento de código siempre imprimirá 100 tal como está.

4. ¿Cuál es el resultado de este código?

```
Predicado<Cadena> vacío = Cadena::isEmpty;
Predicado<Cadena> notEmpty = vacío.negate();

resultado var = Stream.generate(() -> "")
        .filtro(no vacío)
        .collect(Collectors.groupingBy(k -> k))
        .entrySet()
        .arroyo()
        .map(Entrada::getValue)
        .flatMap(Colección::corriente)
        .collect(Collectors.partitioningBy(notEmpty));
System.out.println(resultado);
```

A. Genera {}.

B. Genera {false=[], true=[]}.

C. El código no se compila.

D. El código no termina.

5. ¿Cuál es el resultado del siguiente programa?

```
funciones matemáticas de clase pública {
    addToInt vacío estático público (int x, int cantidadAgregar) {
        x = x + cantidadAAgregar;
    }
    público estático vacío principal (String [] argumentos) {
        var a = 15;
        var b = 10;
        MathFunctions.addToInt(a, b);
        System.out.println(a); } }
```

Machine Translated by Google



R.10 _

B.15 _

C.25 _

D. Error del compilador en la línea

3 E. Error del compilador en la línea 8

F. Ninguna de las anteriores

6. Supongamos que tenemos los siguientes archivos de propiedades y código. ¿Qué valores están impresos en las líneas 8 y 9, respectivamente?

```
Penguin.properties
nombre=Billy
edad=1

Penguin_de.properties
nombre=Edad
fría=4

Penguin_es.properties
nombre=Willy

5: Configuración regional fr = nueva
configuración regional("fr"); 6: Locale.setDefault(new
Locale("en", "US"); 7: var b = ResourceBundle.getBundle("Penguin",
fr); 8: System.out.println(b.getString("nombre")); 9:
System.out.println(b.getString("edad"));
```

A. Billy y 1 B. Billy y

nulo C. Willy y 1 D. Willy y

nulo E. Chilly y nulo

F. El código no se compila.

7. ¿ Qué se garantiza que se imprimirá con el siguiente código? (Elija todas las que correspondan).

```
int[] matriz = {6,9,8};
System.out.println("B" + Arrays.binarySearch(matriz,9));
System.out.println("C" + Arrays.compare(array, new int[]
   {6, 9, 8}));
System.out.println("M" + Arrays.mismatch(array, new int[]
   {6, 9, 8}));
```

Machine Translated by Google



A B1

B B2

C.C -1

D.C0_ _

E.M -1

F.M0 _

G. El código no se compila.

8. ¿Qué interfaces funcionales completan el siguiente código, suponiendo que existe la variable r? (Elija todas las que correspondan).

```
6:  _______ x = r.negar();
7:  _______ y = () -> System.out.println();
8:  _______ z = (a, b) -> a - b;
```

A. Predicado binario<Entero, Entero>

B. Comparable <Entero>

C. Comparador <Entero>

D. Consumidor <Entero>

E. Predicado <Entero>

F. Ejecutable

G. Ejecutable <Entero>

9. Suponga que tiene un módulo llamado com.vet. ¿Dónde podría colocar el siguiente archivo module-info.java para crear un módulo válido?

```
módulo público com.vet {
    exportaciones com.vet;
}
```

A. Al mismo nivel que la carpeta com.

B. Al mismo nivel que la carpeta veterinaria

C. Dentro de la carpeta veterinaria

D. Ninguna de las anteriores

10. ¿Cuál es el resultado del siguiente programa? (Elija todas las que correspondan).

```
1: interfaz HasTail {privado int getTailLength(); }
2: la clase abstracta Puma implementa HasTail {
3:         Cadena getTailLength() { return "4"; }
4: }
5: la clase pública Cougar implementa HasTail {
6:         público estático vacío principal (String [] argumentos) {
7:             var puma = nuevo Puma() {};
```

Machine Translated by Google



```
8:          System.out.println(puma.getTailLength());
9:      }
10:     public int getTailLength(int longitud) { return 2; }
11: }
```

R.2 _

B.4 _

C. El código no se compilará debido a la línea 1.

D. El código no se compilará debido a la línea 3.

E. El código no se compilará debido a la línea 5.

F. El código no se compilará debido a la línea 7.

G. El código no se compilará debido a la línea 10.

H. El resultado no se puede determinar a partir del código proporcionado.

11. ¿Qué líneas en Tadpole.java dan un error de compilación? (Elija todas las que correspondan).

```
// rana.java
1: animal de paquete;
2: rana de clase pública {
3:      ribbit vacío protegido() { }
4:      salto vacío() { }
5: }

// renacuajo.java
1: paquete otro;
2: animal importado.*;
3: la clase pública Tadpole extiende Frog {
4:      público estático vacío principal (String [] argumentos) {
5:          Renacuajo t = nuevo Renacuajo();
6:          t.ribbit();
7:          t.jump();
8:          Rana f = nuevo Renacuajo();
9:          f.ribbit();
10:         f.saltar();
11:     } }
```

A. Línea 5

B. Línea 6

C. Línea 7

D. Línea 8

E. Línea 9

F. Línea 10

G. Todas las líneas se compilan.

Machine Translated by Google



12. ¿Cuál de los siguientes puede completar los espacios en blanco para compilar este código?

```
_____ un = _____.getConexión(
    URL, nombre de usuario, contraseña);
_____ b = a.prepareStatement(sql);
_____ c = b.executeQuery();
if (c.next()) System.out.println(c.getString(1));
```

A. Conexión, controlador, declaración preparada, conjunto de resultados

B. Conexión, DriverManager, PreparedStatement, ResultSet

C. Conexión, fuente de datos, declaración preparada, conjunto de resultados

D. Controlador, Conexión, Declaración preparada, Conjunto de resultados

E. DriverManager, Conexión, Declaración Preparada, Conjunto de Resultados

F. Fuente de datos, conexión, declaración preparada, conjunto de resultados

13. ¿ Cuál de las siguientes afirmaciones puede llenar el espacio en blanco para que el código se compile correctamente? ¿completamente? (Elija todas las que correspondan).

```
Establecer<? extiende RuntimeException> mySet = nuevo _____ ();
```

A. HashSet <? extiende RuntimeException>

B. HashSet <Excepción>

C. TreeSet <RuntimeException>

D. TreeSet<NullPointerException>

E. Ninguna de las anteriores

14. Supongamos que Birds.dat existe, es accesible y contiene datos para un objeto Bird. ¿Cuál es el resultado de ejecutar el siguiente código? (Elija todas las que correspondan).

```
1: importar java.io.*;
2: clase pública Pájaro {
3:         nombre de cadena privada;
4:         edad entera transitoria privada;
5:
6:         // Captadores/definidores omitidos
7:
8:         público estático vacío principal (String [] argumentos) {
9:             intentar (var es = nuevo ObjectInputStream (
10:                     nuevo BufferedInputStream(
11:                     nuevo FileInputStream("birds.dat")))) {
12:                 Pájaro b = is.readObject();
13:                 System.out.println(b.edad);
14:             } } }
```

Machine Translated by Google



A. Compila e imprime 0 en tiempo de ejecución.

B. Compila e imprime nulo en tiempo de ejecución.

C. Compila e imprime un número en tiempo de ejecución.

D. El código no se compilará debido a las líneas 9 a 11.

E. El código no se compilará debido a la línea 12.

F. Se compila pero arroja una excepción en tiempo de ejecución.

15. ¿Cuáles de los siguientes son miembros de instancia válidos de una clase? (Elija todas las que correspondan).

A. era era = 3; B. Caso

Var = nueva Var(); C. var vacía() {}

D.int Var() { var _ = 7; devolver _;}

E. Cadena nueva = "var"; F. var var()

{ return nulo; }

16. ¿Qué es cierto si la tabla está vacía antes de ejecutar este código? (Elija todas las que correspondan).

```
var sql = "INSERTAR EN VALORES de personas (?, ?, ?)";
conexión.setAutoCommit(falso);

intentar (var ps = conn.prepareStatement(sql,
    ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATABLE)) {

    ps.setInt(1, 1);
    ps.setString(2, "Joslyn");
    ps.setString(3, "Nueva
    York"); ps.executeUpdate();

    Punto de guardado sp = conn.setSavepoint();

    ps.setInt(1, 2);
    ps.setString(2, "Kara");
    ps.executeUpdate();

    conexión. _____ ;
}
```

Machine Translated by Google



R. Si la línea en blanco contiene rollback(), no hay filas en la tabla.

B. Si la línea en blanco contiene rollback(), hay una fila en la tabla.

C. Si la línea en blanco contiene rollback(sp), no hay filas en la tabla.

D. Si la línea en blanco contiene rollback(sp), hay una fila en la tabla.

E. El código no se compila.

F. El código genera una excepción porque la segunda actualización no establece todos los parámetros.

17. ¿Qué es cierto si el contenido de la ruta1 comienza con el texto Hola? (Escoge dos.)

```
System.out.println(Files.mismatch(ruta1,ruta2));
```

R. Si la ruta2 no existe, el código imprime -1.

B. Si la ruta2 no existe, el código imprime 0.

C. Si la ruta2 no existe, el código genera una excepción.

D. Si el contenido de la ruta2 comienza con Hola, el código imprime -1.

E. Si el contenido de la ruta2 comienza con Hola, el código imprime 0.

F. Si el contenido de la ruta2 comienza con Hola, el código imprime 1.

18. ¿ Cuál de los siguientes tipos se puede insertar en el espacio en blanco para permitir que el programa se compile correctamente? (Elija todas las que correspondan).

```
import java.util.*;
anfibio de clase final {}
la clase abstracta Tadpole extiende Anfibio {}
clase pública FindAllTadpoles {
        público estático vacío principal (cadena... argumentos) {
            var renacuajos = new ArrayList<Tadpole>();
            para (var anfibio: renacuajos) {
                ________________ renacuajo = anfibio;
} } }
```

A. Lista<Renacuajo>

B. booleano

C. anfibio

D. renacuajo

E. Objeto

F. Ninguna de las anteriores

19. ¿Cuál es el resultado de compilar y ejecutar el siguiente programa?

```
clase pública FeedingSchedule {
        público estático vacío principal (String [] argumentos) {
            var x = 5;
            var j = 0;
```

Machine Translated by Google



```
5:      EXTERIOR: para (var i = 0; i < 3;)
6:          INTERIOR: hacer {
7:              yo ++;
8:              x++;
9:              si (x>10) rompe INTERIOR;
10:             x += 4;
11:             j++;
12:         } mientras (j <= 2);
13:     System.out.println(x);
14: } }
```

R.10 _

B.11 _

C.12 _

D.17 _

E. El código no se compilará debido a la línea 5.

F. El código no se compilará debido a la línea 6.

20. Cuando se imprime, ¿qué cadena da el mismo valor que este bloque de texto?

```
var pooh = """
    "Oh, molesta". -Pooh
    """.sangría(1);
System.out.print(pooh);
```

A. "\n\"Oh, molesta.\" -Pooh\n"

B. "\n \"Oh, molesta.\" -Pooh\n"

C. " \"Oh, molesta.\" -Pooh\n"

D. "\n\"Oh, molesta.\" -Pooh"

E. "\n \"Oh, molesta.\" -Pooh"

F. " \"Oh, molesta.\" -Pooh"

G. Ninguno de los anteriores

21. Un(a) _____ módulo siempre contiene un archivo module-info.java, mientras que un(a) _____ El módulo siempre exporta todos sus paquetes a otros módulos.

A. automático, llamado

B. automático, sin nombre

C. con nombre, automático

D. con nombre, sin nombre

E. sin nombre, automático

F. sin nombre, con nombre

G. Ninguno de los anteriores

Machine Translated by Google



22. ¿Cuál es el resultado del siguiente código?

```
22: var treeMap = new TreeMap<Carácter, Entero>(); 23: treeMap.put('k',
1); 24: treeMap.put('k', 2); 25:
árbolMap.put('m', 3); 26:
árbolMap.put('M', 4); 27:
treeMap.replaceAll((k, v) -> v +
v); 28: treeMap.keySet() .forEach(k ->

29: System.out.print(treeMap.get(k)));
```

A. 268

B.468 _

C. 2468

D. 826

E. 846

F. 8246

G. Ninguno de los anteriores

23. ¿Cuál de las siguientes líneas puede completar el espacio en blanco para imprimir verdadero? (Elija todas las que correspondan).

```
10: principal vacío estático público (String [] argumentos) {
11:     System.out.println(prueba(                    ));
12: }
13: prueba booleana estática privada (Función <Integer, Boolean> b) { return b.apply(5);
14:
15: }
```

A. i::es igual a(5)

B. yo -> {yo == 5;}

C. (i) -> i == 5 D. (int i) -> i

== 5

E. (int i) -> {return i == 5;}

F. (i) -> {regresar i == 5;}

24. ¿Cuántas veces se imprime la palabra verdadero?

```
var s1 = "Java"; var
s2 = "Java"; var s3 =
s1.indent(1).strip(); var s4 = s3.interno();
var sb1 = nuevo
StringBuilder();
sb1.append("Ja").append("va");
```

Machine Translated by Google



```
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));
System.out.println(s1 == s3);
System.out.println(s1 == s4);
System.out.println(sb1.toString() == s1);
System.out.println(sb1.toString().equals(s1));
```

R. Una vez

B. dos veces

C. Tres veces

D. Cuatro veces

E. Cinco veces

F. El código no se compila.

25. ¿Cuál es el resultado del siguiente programa?

```
1: clase Ciervo {
2:         ciervo público() {System.out.print("Ciervo");}
3:         Ciervo público (int edad) {System.out.print ("DeerAge");}
4:         hasHorns booleano protegido() { return false; }
5: }
6: clase pública Reno extiende Ciervo {
7:         Reno público (int edad) {System.out.print ("Reno");}
8:         public boolean hasHorns() { return true; }
9:         público estático vacío principal (String [] argumentos) {
10:            Ciervo venado = nuevo Reno(5);
11:            System.out.println("," + ciervo.hasHorns());
12: } }
```

A. RenoCiervo,falso

B. DeerAgeReindeer, cierto

C. CiervoReno,verdadero

D. CiervoReno,falso

E. RenoCiervo, cierto

F. CiervoEdadReno,falso

G. El código no se compilará debido a la línea 4.

H. El código no se compilará debido a la línea 12.

Machine Translated by Google



26. ¿Cuáles de las siguientes son ciertas? (Elija todas las que correspondan).

```
magia vacía estática privada (Stream<Integer> s) {
    Opcional o =
        s .filter(x -> x <

        5) .limit(3) .max((x, y) -> xy);
    System.out.println(o.get());
}
```

A. magia(Stream.empty()); corre infinitamente.

B. magia (Stream.empty()); lanza una excepción.

C. magic(Stream.iterate(1, x -> x++)); corre infinitamente.

D. magic(Stream.iterate(1, x -> x++)); lanza una excepción.

E. magic(Stream.of(5, 10)); corre infinitamente.

F. magic(Stream.of(5, 10)); lanza una excepción.

G. El método no se compila.

27. Suponiendo que las siguientes declaraciones son tipos de nivel superior declarados en el mismo archivo, ¿cuáles se compilan con éxito? (Elija todas las que correspondan).

```
grabar música()
    { puntuación int final = 10;

} grabar canción (letra de cadena)

    { Canción { this.lyrics = letra + "Hola mundo";
    }

} clase sellada Danza {}
registro de marzo() {
    @Override String toString() { return nulo; }

} clase Ballet extiende Danza {}
```

A. Música

B. Canción

C. Danza

D. marzo

mi ballet

F. Ninguno de ellos compila.

Machine Translated by Google



28. ¿Cuál de las siguientes expresiones se compila sin errores? (Elija todas las que correspondan).

A. int lunes = 3 + 2,0;

B. doble martes = 5_6L;

C. miércoles booleano = 1 > 2? !verdadero;

D. jueves corto = (corto)Integer.MAX_VALUE;

E. viernes largo = 8,0 L;

F. var sábado = 2_.0;

G. Ninguno de los anteriores

29. ¿Cuál es el resultado de ejecutar la siguiente aplicación?

```
final var cb = new CyclicBarrier(3, () ->
    System.out.println("¡Limpiar!")); // u1
Servicio ExecutorService = Executors.newSingleThreadExecutor();
intentar {
    IntStream.generar(() -> 1)
        .límite(12)
        .paralelo()
        .forEach(i -> service.submit(() -> cb.await())); // u2
} finalmente { service.shutdown(); }
```

A. ¡ El resultado es Limpio! al menos una vez.

B. ¡ El resultado es Limpio! exactamente cuatro veces.

C. El código no se compilará debido a la línea u1.

D. El código no se compilará debido a la línea u2.

E. Compila pero genera una excepción en tiempo de ejecución.

F. Se compila pero espera una eternidad en tiempo de ejecución.

30. ¿Qué afirmación sobre el siguiente método es verdadera?

```
principal vacío estático público (cadena... sin usar) {
    System.out.print("a");
    intentar (lector de StringBuilder = nuevo StringBuilder()) {
        Sistema.out.print("b");
        lanzar nueva IllegalArgumentException();
    } captura (Excepción e || RuntimeException e) {
        System.out.print("c");
        lanzar nueva FileNotFoundException();
    } finalmente {
        System.out.print("d");
} }
```

Machine Translated by Google



A. Compila e imprime abc.

B. Compila e imprime abd.

C. Compila e imprime abcd.

D. Una línea contiene un error del compilador.

E. Dos líneas contienen un error del compilador.

F. Tres líneas contienen un error del compilador.

G. Compila pero imprime una excepción en tiempo de ejecución.

Machine Translated by Google



# Respuestas a la prueba de evaluación

1. G. La pregunta no se compila porque las líneas 44 y 47 no siempre devuelven un valor en el bloque de casos, que se requiere cuando se usa una expresión de cambio en una operación de asignación. A la línea 44 le falta una declaración de rendimiento cuando la declaración if se evalúa como falsa, mientras que a la línea 47 le falta una declaración de rendimiento por completo. Como dos líneas no se compilan, la opción G es la respuesta. Para obtener más información, consulte el Capítulo 3.

2. B. Inicialmente, a la luna se le asigna un valor de 9, mientras que a la estrella se le asigna un valor de 8. El multi El operador de multiplicación (*) tiene un orden de precedencia más alto que el operador de suma (+), por lo que se evalúa primero. Dado que la estrella no es mayor que 10, al sol se le asigna un valor de 3, que se promueve a 3,0f como parte de la asignación. El valor de Júpiter es (3,0f + 9) - 1,0, que es 11,0f. Este valor se promueve implícitamente al doble cuando se asigna. En la última tarea, la luna disminuye de 9 a 8, y el valor de la expresión se devuelve como 8.
   Dado que 8 menor o igual a 8 es verdadero, Marte se establece en un valor de 2. El resultado final es 3,0, 11,0, 2, lo que hace que la opción B sea la respuesta correcta. Tenga en cuenta que, si bien Java genera el decimal para valores flotantes y dobles, no genera f para valores flotantes. Para obtener más información, consulte el Capítulo 2.

3. B, C, E. El código puede imprimir 100 sin ningún cambio, pero como la clase de datos no es segura para subprocesos, este comportamiento no está garantizado. Por este motivo, la opción F es incorrecta. La opción A también es incorrecta, ya que volátil no garantiza la seguridad de los subprocesos. Las opciones B y C son correctas, ya que ambas hacen que la secuencia aplique la operación add() de forma sincronizada. La opción D es incorrecta, ya que serial() no es un método de transmisión. La opción E es correcta. La sincronización hará que cada hilo espere para que la Lista se modifique un elemento a la vez. Para obtener más información, consulte el Capítulo 13.

4. D. Primero, este lío de código se compila. Sin embargo, la fuente es una corriente infinita. La operación de filtrado comprobará cada elemento por turno para ver si alguno no está vacío. Mientras nada pase el filtro, el código no termina. Por tanto, la opción D es correcta. Para obtener más información, consulte el Capítulo 10.

5. B. El código se compila correctamente, por lo que las opciones D y E son incorrectas. El valor de a no se puede cambiar mediante el método addToInt(), sin importar lo que haga el método, porque solo se pasa una copia de la variable al parámetro x. Por lo tanto, a no cambia y el resultado en la línea 9 es 15, que es la opción B. Para obtener más información, consulte el Capítulo 5.

6. C. Java utilizará Penguin_en.properties como paquete de recursos coincidente en la línea 7.
   Como no hay coincidencias para el francés, se utiliza la configuración regional predeterminada. La línea 8 encuentra una clave coincidente en el archivo Penguin_en.properties. La línea 9 no encuentra ninguna coincidencia en el archivo Penguin_en.properties; por lo tanto, tiene que buscar más arriba en la jerarquía, Penguin.properties. Esto hace que la opción C sea la respuesta. Para obtener más información, consulte el Capítulo 11.

7. D, E. La matriz puede utilizar un inicializador anónimo porque está en la misma línea que la declaración. Los resultados de la búsqueda binaria no están definidos ya que la matriz no está ordenada.
   Dado que la pregunta se refiere a la producción garantizada, las opciones A y B son incorrectas.

Machine Translated by Google



La opción D es correcta porque el método compare() devuelve 0 cuando las matrices tienen la misma longitud y los mismos elementos. La opción E es correcta porque el método mismatch() devuelve un -1 cuando las matrices son equivalentes. Para obtener más información, consulte el Capítulo 4.

8. C, E, F. Primero, tenga en cuenta que la opción A es incorrecta porque la interfaz debe ser BiPredicate y no BinaryPredicate. La línea 6 requiere que sepas que negate() es un método conveniente en Predicate. Esto hace que la opción E sea correcta. La línea 7 toma cero parámetros y no devuelve nada, lo que la convierte en Runnable. Recuerde que Runnable no usa genéricos. Esto hace que la opción F sea correcta. Finalmente, la línea 8 toma dos parámetros y devuelve un int. La opción C es correcta. Comparable está ahí para engañarlo, ya que solo toma un parámetro en su único método abstracto. Para obtener más información, consulte el Capítulo 8.

9. D. Si se tratara de un archivo module-info.java válido, debería colocarse en el directorio raíz del módulo, que es la opción A. Sin embargo, un módulo no puede utilizar el modificador de acceso público. La opción D es correcta porque el archivo proporcionado no se compila independientemente de su ubicación en el proyecto. Para obtener más información, consulte el Capítulo 12.

10. C. El método getTailLength() en la interfaz es privado; por lo tanto, debe incluir un cuerpo. Por esta razón la línea 1 es la única línea que no compila y la opción C es correcta.
La línea 3 utiliza un tipo de retorno diferente para el método, pero como es privado en la interfaz, no se considera una anulación. Tenga en cuenta que la línea 7 define una clase anónima utilizando la clase principal abstracta de Puma. Para obtener más información, consulte el Capítulo 7.

11. C, E, F. El método jump() tiene acceso al paquete, lo que significa que solo se puede acceder a él desde el mismo paquete. Tadpole no está en el mismo paquete que Frog, lo que provoca que las líneas 7 y 10 activen errores del compilador y nos brinden las opciones C y F. El método ribbit() tiene acceso protegido, lo que significa que solo se puede acceder a él desde una referencia de subclase o en el mismo paquete. La línea 6 está bien porque Tadpole es una subclase. La línea 9 no se compila y nuestra respuesta final es la opción E porque la referencia de la variable es a una Rana, que no otorga acceso al método protegido. Para obtener más información, consulte el Capítulo 5.

12. B. DataSource no está en el examen, por lo que cualquier pregunta que contenga uno es incorrecta. Las variables clave utilizadas al ejecutar una consulta son Conexión, Declaración preparada y Conjunto de resultados. Se obtiene una Conexión a través de un DriverManager, haciendo que la opción B sea correcta. Para obtener más información, consulte el Capítulo 15.

13. C, D. La declaración mySet define un límite superior de tipo RuntimeException.
Esto significa que las clases pueden especificar RuntimeException o cualquier subclase de RuntimeException como parámetro de tipo. La opción B es incorrecta porque Exception es una superclase, no una subclase, de RuntimeException. La opción A es incorrecta porque el comodín no puede aparecer en el lado derecho de la tarea. Las opciones C y D se compilan y son las respuestas. Para obtener más información, consulte el Capítulo 9.

14. D, E. La línea 10 incluye una IOException marcada no controlada, mientras que la línea 11 incluye una FileNotFoundException marcada no controlada, lo que hace que la opción D sea correcta. La línea 12 no se compila porque is.readObject() debe convertirse en un objeto Bird para asignarlo a b. Tampoco se compila porque incluye dos excepciones marcadas no controladas, IOException
y ClassNotFoundException, lo que hace que la opción E sea correcta. Si se agregó una operación de conversión en la línea 12 y el método main() se actualizó en la línea 8 para declarar los diversos marcados

Machine Translated by Google



excepciones, el código se compilaría pero arrojaría una excepción en tiempo de ejecución ya que Bird no implementa Serializable. Finalmente, si la clase implementara Serializable, el programa imprimiría nulo en tiempo de ejecución, ya que ese es el valor predeterminado para la edad del campo transitorio. Para obtener más información, consulte el Capítulo 14.

15. C. La opción A es incorrecta porque var solo se permite como tipo para variables locales, no para miembros de instancia. Las opciones B y E son incorrectas porque new y case son palabras reservadas y no se pueden utilizar como identificadores. La opción C es correcta, ya que var se puede utilizar como nombre de método.
La opción D es incorrecta porque no se puede utilizar un solo guión bajo (_) como identificador. Finalmente, la opción F es incorrecta porque no se puede especificar var como tipo de retorno de un método. Para obtener más información, consulte el Capítulo 1.

16. A, D. Este código es correcto y elimina las opciones E y F. JDBC utilizará el conjunto de parámetros existente si no lo reemplaza. Esto significa que la fila de Kara se configurará para usar NY como tercer parámetro. Regresar a un punto de guardado descarta cualquier cambio realizado desde entonces. Esto deja a Joslyn y elimina a Kara, haciendo que la opción D sea correcta. Revertir sin un punto de guardado nos devuelve al comienzo de la transacción, que es la opción A. Para obtener más información, consulte el Capítulo 15.

17. C, F. La opción C es correcta ya que falta de coincidencia() genera una excepción si los archivos no existen a menos que ambos hagan referencia al mismo archivo. Además, la opción F es correcta porque se devuelve el primer índice que difiere, que es el segundo carácter. Dado que Java utiliza índices de base cero, este es 1.
Para obtener más información, consulte el Capítulo 14.

18. F. La clase Amphibian está marcada como final, lo que significa que la línea 3 genera un error del compilador y la opción F es correcta. Para obtener más información, consulte el Capítulo 6.

19. C. El código se compila y ejecuta sin problemas; por lo tanto, las opciones E y F son incorrectas. Este El tipo de problema se examina mejor una iteración de bucle a la vez:

    - En la primera iteración del bucle exterior, i es 0, por lo que el bucle continúa.
    - En la primera iteración del bucle interno, i se actualiza a 1 y x a 6. La instrucción if la rama no se ejecuta y x aumenta a 10 y j a 1.
    - En la segunda iteración del bucle interno (dado que j = 1 y 1 <= 2), i se actualiza a 2 yx a 11. En este punto, la rama if se evaluará como verdadera durante el resto de la ejecución del programa, lo que hace que el flujo salga del bucle interno cada vez que se alcanza.
    - En la segunda iteración del bucle externo (dado que i = 2), i se actualiza a 3 y x a 12. Como antes, el bucle interno se rompe ya que x sigue siendo mayor que 10.
    - En la tercera iteración del bucle externo, el bucle externo se rompe, ya que i ya no es menor que 3. Se genera el valor más reciente de x, 12, por lo que la respuesta es la opción C.

    Para obtener más información, consulte el Capítulo 3.

20. C. Primero, observe que el bloque de texto tiene el cierre """ en una línea separada, lo que significa que hay hay una nueva línea al final y descarta las opciones D, E y F. Además, los bloques de texto no comienzan con una nueva línea, lo que descarta las opciones A y B. Por lo tanto, la opción C es correcta. Para obtener más información, consulte el Capítulo 1.

Machine Translated by Google



21. C. Sólo los módulos con nombre deben tener un archivo module-info.java, descartando las opciones A, B, E y F. Los módulos sin nombre no son legibles por ningún otro tipo de módulo, descartando la opción D. Módulos automáticos Siempre exporte todos los paquetes a otros módulos, haciendo que la opción de respuesta sea C. Para obtener más información, consulte el Capítulo 12.

22. E. Cuando se coloca la misma clave en un Mapa, anula el valor original. Esto significa que la línea 23 podría omitirse y el código sería el mismo, y solo hay tres pares clave/valor en el mapa. TreeMap ordena sus claves, haciendo el orden M seguido de k seguido de m.
Recuerde que el orden natural tiene mayúsculas antes que minúsculas. El reemplazarTodo()
El método se ejecuta contra cada elemento del mapa, duplicando el valor. Finalmente, repetimos cada clave, imprimiendo 846 y haciendo que la opción E sea correcta. Para obtener más información, consulte el Capítulo 9.

23. C, F. La opción A parece una referencia de método. Sin embargo, no llama a un método válido ni las referencias a métodos pueden tomar parámetros. La interfaz Predicate toma un único parámetro y devuelve un valor booleano. Las expresiones Lambda con un parámetro pueden omitir los paréntesis alrededor de la lista de parámetros, lo que hace que la opción C sea correcta. La declaración de devolución es opcional cuando hay una sola declaración en el cuerpo, lo que hace que la opción F sea correcta. La opción B es incorrecta porque se debe utilizar una declaración de devolución si se incluyen llaves alrededor del cuerpo. Las opciones D y E son incorrectas porque el tipo es Entero en el predicado e int en la lambda. El autoboxing funciona para colecciones, no para inferir predicados. Si estos dos se cambiaran a Integer, serían correctos. Para obtener más información, consulte el Capítulo 8.

24. D. Los literales de cadena se utilizan del grupo de cadenas. Esto significa que s1 y s2 se refieren a la mismo objeto y son iguales. Por lo tanto, las dos primeras declaraciones impresas se imprimen como verdaderas. Mientras que los métodos indent() y strip() crean nuevos objetos String y la tercera declaración imprime falso, el método intern() revierte el String al del grupo de cadenas. Por lo tanto, la cuarta declaración impresa es verdadera. La quinta declaración de impresión imprime falso porque toString() usa un método para calcular el valor y no es del grupo de cadenas. La declaración de impresión final nuevamente imprime verdadero porque igual () mira los valores de Cadena
objetos. Como cuatro son verdaderas, la opción D es la respuesta. Para obtener más información, consulte el Capítulo 4.

25. C. Se crea una instancia del objeto Reno utilizando el constructor que toma un valor int. Como no hay una llamada explícita al constructor principal, el compilador inserta super() como la primera línea del constructor en la línea 7. Se llama al constructor principal y Deer se imprime en la línea 2. El flujo regresa al constructor en la línea 7, con Reno impreso. A continuación, se llama al método hasHorns(). El tipo de referencia es Ciervo y el tipo de objeto subyacente es Reno. Dado que Reindeer anula correctamente el método hasHorns(), se llama a la versión en Reindeer, con la línea 11 impresa como verdadera. Por tanto, la opción C es correcta. Para obtener más información, consulte el Capítulo 6.

26. B, F. Llamar a get() en un Opcional vacío provoca que se lance una excepción, haciendo
La opción B es correcta. La opción F también es correcta porque filter() deja el Opcional vacío antes de llamar a get(). La opción C es incorrecta porque la operación límite intermedio() hace que el flujo infinito sea finito. Las opciones A y E son incorrectas porque los flujos fuente no son infinitos. Por lo tanto, la llamada a max() ve sólo tres elementos y finaliza. Para obtener más información, consulte el Capítulo 10.

Machine Translated by Google



27. C. La música no se compila porque los registros no pueden incluir variables de instancia que no figuran en la declaración del expediente, ya que podría romper la inmutabilidad. Song no se compila porque un constructor compacto no puede establecer una variable de instancia. El registro se compilaría si esto se eliminaron del constructor compacto, ya que los constructores compactos pueden modificar los parámetros de entrada. Marzo no se compila porque se trata de una anulación no válida; reduce la visibilidad del método toString() desde el acceso público al paquete. Ballet no se compila porque la subclase de una clase sellada debe marcarse como final, sellada o no sellada.
Como el único que compila es Dance, la opción C es la respuesta. Para obtener más información, consulte el Capítulo 7.

28. B, D. La opción A no compila, ya que la expresión 3 + 2.0 se evalúa como doble, y un doble requiere que se asigne una conversión explícita a un int. La opción B se compila sin problemas, ya que un valor largo se puede convertir implícitamente a doble. La opción C no se compila porque al operador ternario (? :) le faltan dos puntos (:), seguidos de una segunda expresión.
La opción D es correcta. Aunque el valor int es mayor que un short, se convierte explícitamente en un short, lo que significa que el valor se ajustará para caber en un short. La opción E es incorrecta, ya que no se puede utilizar un decimal (.) con el sufijo largo (L). Finalmente, la opción F es incorrecta, ya que no se puede utilizar un guión bajo junto a un punto decimal. Para obtener más información, consulte el Capítulo 2.

29. F. El código se compila sin problemas. La clave para entender este código es notar que nuestro ejecutor de subprocesos contiene solo un subproceso, pero nuestro límite de CyclicBarrier es 3. Aunque se envían con éxito 12 tareas al servicio, la primera tarea se bloqueará para siempre en la llamada a await() . Como nunca se alcanza la barrera, no se imprime nada y el programa se bloquea, lo que hace que la opción F sea correcta. Para obtener más información, consulte el Capítulo 13.

30. F. La línea 5 no se compila porque el método no maneja ni declara la excepción FileNotFoundException lanzada en la línea 12. La línea 7 no se compila porque StringBuilder no implementa AutoCloseable y, por lo tanto, no es compatible con una declaración de prueba con recurso. Finalmente, la línea 10 no se compila ya que RuntimeException es una subclase de Exception
en el bloque de capturas múltiples, haciéndolo redundante. Dado que este método contiene tres errores del compilador, la opción F es la respuesta correcta. Para obtener más información, consulte el Capítulo 11.

Machine Translated by Google

Machine Translated by Google

Capítulo

1

# Bloques de construcción

## OBJETIVOS DEL EXAMEN OCP CUBIERTOS EN ESTE CAPÍTULO:

### Manejo de valores de fecha, hora, texto, numéricos y booleanos.

- Utilice primitivas y clases contenedoras, incluida la API de matemáticas, paréntesis, promoción de tipos y conversión para evaluar expresiones aritméticas y booleanas.

### Utilizando el enfoque orientado a objetos de Java

- Declarar y crear instancias de objetos Java, incluidos objetos de clase anidados, y explicar el ciclo de vida del objeto, incluida la creación, la reasignación de referencias y la recolección de basura.
- Comprender los alcances de las variables, utilizar la inferencia de tipos de variables locales, aplicar encapsulación y hacer que los objetos sean inmutables

Machine Translated by Google

Bienvenido al comienzo de su viaje para lograr una certificación Java 17. Suponemos que este no es el primer libro de programación Java que lee. Aunque hablamos de los conceptos básicos, lo hacemos sólo porque queremos asegurarnos de que tenga toda la terminología y los detalles que necesita para el examen. Si nunca antes ha escrito un programa Java, le recomendamos que consulte un libro introductorio sobre Java 8 o superior. Los ejemplos incluyen Head First Java, tercera edición (O'Reilly Media, 2022) y Cómo comenzar a programar con Java para principiantes (For Dummies, 2021). Luego regrese a esta guía de estudio de certificación.

Como dice el viejo refrán, antes de poder correr hay que aprender a caminar. Del mismo modo, debe aprender los conceptos básicos de Java antes de poder crear programas complejos. En este capítulo, presentamos los conceptos básicos de los paquetes, clases, variables y tipos de datos de Java, junto con los aspectos de cada uno que necesita conocer para el examen. Por ejemplo, podrías usar Java todos los días pero no saber que no puedes crear una variable llamada 3dMap o this. El examen espera que usted conozca y comprenda las reglas detrás de estos principios. Si bien se debe repasar la mayor parte de este capítulo, es posible que haya aspectos del lenguaje Java que sean nuevos para usted, ya que no aparecen con frecuencia en la práctica.

# Aprendiendo sobre el medio ambiente

El entorno Java consiste en comprender una serie de tecnologías. En las siguientes secciones, repasamos los términos y acrónimos clave que necesita conocer y luego analizamos qué software necesita estudiar para el examen.

## Componentes principales de Java

El kit de desarrollo de Java (JDK) contiene el software mínimo que necesita para realizar el desarrollo de Java. Los comandos clave incluyen:

- javac: convierte archivos fuente .java en código de bytes .class
- java: ejecuta el programa
- jar: empaqueta archivos juntos
- javadoc: genera documentación

El programa javac genera instrucciones en un formato especial llamado código de bytes que el comando java puede ejecutar. Luego, Java inicia la máquina virtual Java (JVM) antes

Machine Translated by Google



ejecutando el código. La JVM sabe cómo ejecutar el código de bytes en la máquina real en la que se encuentra. Puede pensar en la JVM como una caja mágica especial en su máquina que sabe cómo ejecutar su archivo .class dentro de su sistema operativo y hardware particulares.

¿A dónde fue el JRE?

En Java 8 y versiones anteriores, podía descargar Java Runtime Environment (JRE) en lugar del JDK completo. El JRE era un subconjunto del JDK que se usaba para ejecutar un programa pero no podía compilarlo. Ahora, la gente puede utilizar el JDK completo cuando ejecuta un programa Java. Alternativamente, los desarrolladores pueden proporcionar un ejecutable que contenga las piezas necesarias que habrían estado en el JRE.

Al escribir un programa, existen funciones y algoritmos comunes que los desarrolladores necesitan. Afortunadamente, no tenemos que escribir cada uno de ellos nosotros mismos. Java viene con un amplio conjunto de interfaces de programación de aplicaciones (API) que puede utilizar. Por ejemplo, hay una clase StringBuilder para crear una cadena grande y un método en Colecciones. para ordenar una lista. Al escribir un programa, resulta útil determinar qué partes de su tarea pueden realizarse mediante las API existentes.

Quizás hayas notado que dijimos que el JDK contiene el software mínimo que necesitas. Muchos desarrolladores utilizan un entorno de desarrollo integrado (IDE) para facilitar la escritura y ejecución de código. Si bien no recomendamos usar uno mientras estudias para el examen, es bueno saber que existen. Los IDE de Java comunes incluyen Eclipse, IntelliJ IDEA y Visual Studio Code.

## Descargando un JDK

Cada seis meses, Oracle lanza una nueva versión de Java. Java 17 salió en septiembre de 2021. Esto significa que Java 17 no será la última versión cuando descargue el JDK para estudiar para el examen. Sin embargo, aún debes usar Java 17 para estudiar, ya que se trata de un examen de Java 17. Las reglas y el comportamiento pueden cambiar con versiones posteriores de Java. ¡No querrás equivocarte en una pregunta porque estudiaste con una versión diferente de Java!

Puede descargar el JDK de Oracle en el sitio web de Oracle, utilizando la misma cuenta que utiliza para registrarse para el examen. Hay muchos JDK disponibles, el más popular de los cuales, además del JDK de Oracle, es OpenJDK.

Muchas versiones de Java incluyen funciones de vista previa que están desactivadas de forma predeterminada pero que usted puede habilitar. Las funciones de vista previa no están en el examen. Para evitar confusión sobre cuándo se agregó una característica al idioma, diremos "se introdujo oficialmente en" para indicar cuándo salió de la vista previa.

Machine Translated by Google



Verifique su versión de Java

Antes de continuar, aproveche esta oportunidad para asegurarse de tener la versión correcta de Java en su camino.

```
versión javac
versión java
```

Ambos comandos deben incluir la versión número 17.

# Comprender la estructura de clases

En los programas Java, las clases son los componentes básicos. Al definir una clase, se describen todas las partes y características de uno de esos bloques de construcción. En capítulos posteriores, verá otros bloques de construcción, como interfaces, registros y enumeraciones.

Para utilizar la mayoría de las clases, debes crear objetos. Un objeto es una instancia de ejecución de una clase en la memoria. A menudo se hace referencia a un objeto como instancia , ya que representa una representación única de la clase. Todos los diversos objetos de todas las diferentes clases representan el estado de su programa. Una referencia es una variable que apunta a un objeto.

En las siguientes secciones, analizamos campos, métodos y comentarios. También exploramos el relación entre clases y archivos.

## Campos y métodos

Las clases Java tienen dos elementos principales: métodos, a menudo llamados funciones o procedimientos en otros lenguajes, y campos, más generalmente conocidos como variables. En conjunto, estos se denominan miembros de la clase. Las variables mantienen el estado del programa y los métodos operan en ese estado. Si es importante recordar el cambio, una variable almacena ese cambio. Eso es todo lo que realmente hacen las clases. Es trabajo del programador crear y organizar estos elementos de tal manera que el código resultante sea útil e, idealmente, fácil de entender para otros programadores.

La clase Java más simple que puedes escribir se ve así:

```
1: clase pública Animal {
2: }
```

Java llama palabra clave a una palabra con significado especial, que hemos marcado en negrita en el fragmento anterior. A lo largo del libro, a menudo ponemos en negrita partes de fragmentos de código para llamar la atención sobre ellos. La línea 1 incluye la palabra clave pública , que permite que otras clases la utilicen. La palabra clave class indica que estás definiendo una clase. Animal da el nombre de la clase.
Por supuesto, esta no es una clase interesante, así que agreguemos su primer campo.

```
1: clase pública Animal {
2:     Nombre de cadena;
3: }
```

Machine Translated by Google

Los números de línea no forman parte del programa; solo están ahí para hacer que sea más fácil hablar del código.

En la línea 2, definimos una variable llamada nombre. También declaramos que el tipo de esa variable es String. Una Cadena es un valor en el que podemos poner texto, como "esto es una cadena". String también es una clase proporcionada con Java. A continuación podemos agregar métodos.

```
1: clase pública Animal {
2:     Nombre de cadena;
3:     cadena pública getName() {
4:         nombre de retorno;
5:     }
6:     setName público vacío (String nuevoNombre) {
7:         nombre = nuevoNombre;
8:     }
9: }
```

En las líneas 3 a 5, definimos un método. Un método es una operación que se puede llamar. Nuevamente, public se usa para indicar que este método puede ser llamado desde otras clases. Luego viene el tipo de retorno; en este caso, el método devuelve una Cadena. En las líneas 6 a 8 hay otro método.
Este tiene un tipo de devolución especial llamado void. La palabra clave void significa que no se devuelve ningún valor. Este método requiere que se le proporcione información desde el método de llamada; esta información se llama parámetro. El método setName() tiene un parámetro llamado newName y es de tipo String. Esto significa que la persona que llama debe pasar un parámetro String y esperar que no se devuelva nada.

El nombre del método y los tipos de parámetros se denominan firma del método. En este ejemplo, ¿puede identificar el nombre del método y los parámetros?

```
número int públicoVisitantes (mes int) {
    devolver 10;
}
```

El nombre del método es numberVisitors. Hay un parámetro llamado mes, que es de tipo int, que es de tipo numérico. Por lo tanto, la firma del método es numberVisitors(int).

## Comentarios

Otra parte común del código se llama comentario. Como los comentarios no son código ejecutable, puedes colocarlos en muchos lugares. Los comentarios pueden hacer que su código sea más fácil de leer.
Si bien los creadores del examen intentan hacer que el código sea más difícil de leer, todavía usan comentarios para llamar la atención sobre los números de línea. Esperamos que utilice comentarios en su propio código.
Hay tres tipos de comentarios en Java. El primero se llama comentario de una sola línea:

```
// comenta hasta el final de la línea
```

Machine Translated by Google



Un comentario de una sola línea comienza con dos barras. El compilador ignora todo lo que escriba después en la misma línea. Luego viene el comentario de varias líneas:

```
/* Múltiple
 * comentario de línea
 */
```

Un comentario de varias líneas (también conocido como comentario de varias líneas) incluye cualquier cosa desde el símbolo /* hasta el símbolo */. La gente suele escribir un asterisco (*) al principio de cada línea de un comentario de varias líneas para que sea más fácil de leer, pero no es necesario. Finalmente, tenemos un comentario de Javadoc:

```
/**
 * Comentario de varias líneas de Javadoc
 * @autor Jeanne y Scott
 */
```

Este comentario es similar a un comentario de varias líneas, excepto que comienza con /**. Esta sintaxis especial le indica a la herramienta Javadoc que preste atención al comentario. Los comentarios de Javadoc tienen una estructura específica que la herramienta Javadoc sabe leer. Probablemente no verá ningún comentario de Javadoc en el examen. Sólo recuerde que existe para que pueda leerlo en línea cuando comience a escribir programas para que otros los utilicen.

Como un poco de práctica, ¿puedes identificar qué tipo de comentario tiene cada uno de los seis siguientes? palabras está en? ¿Es un comentario de una sola línea o de varias líneas?

```
/*
 * // oso hormiguero
 */

// oso

// // gato

// /* perro */

/* elefante */
/*
 * /* hurón */
 */
```

¿Miraste de cerca? Algunos de estos son complicados. Aunque los comentarios técnicamente no lo son En el examen, es bueno practicar la observación del código con atención.

Machine Translated by Google



Bien, pasemos a las respuestas. El comentario que contiene el oso hormiguero está en un comentario de varias líneas. Todo lo que esté entre /* y */ es parte de un comentario de varias líneas, ¡incluso si incluye un comentario de una sola línea! El comentario que contiene oso es su comentario básico de una sola línea. Los comentarios que contienen perros y gatos también son comentarios de una sola línea. Todo desde // hasta el final de la línea es parte del comentario, incluso si es otro tipo de comentario. El comentario que contiene elefante es su comentario básico de varias líneas, aunque solo ocupe una línea.

La línea con hurón es interesante porque no se compila. Todo, desde el primer /* hasta el primer */ es parte del comentario, lo que significa que el compilador ve algo como esto:

```
/* */ */
```

Tenemos un problema. Hay un */. Esa no es una sintaxis válida, un hecho que el compilador está encantado de informarle.

## Clases y archivos fuente

La mayoría de las veces, cada clase Java se define en su propio archivo .java . En este capítulo, el único tipo de nivel superior es una clase. Un tipo de nivel superior es una estructura de datos que se puede definir de forma independiente dentro de un archivo fuente. Durante la mayor parte del libro, trabajamos con clases como el tipo de nivel superior, pero en el Capítulo 7, “Más allá de las clases”, presentamos otros tipos de nivel superior, así como tipos anidados.

Una clase de nivel superior suele ser pública, lo que significa que cualquier código puede llamarla. Curiosamente, Java no requiere que el tipo sea público. Por ejemplo, esta clase está bien:

```
1: clase Animal {
2:     Nombre de cadena;
3: }
```

Incluso puedes poner dos tipos en el mismo archivo. Cuando lo haga, como máximo uno de los tipos de nivel superior del archivo podrá ser público. Eso significa que un archivo que contenga lo siguiente también está bien:

```
1: clase pública Animal {
2:     nombre de cadena privada;
3: }
4: clase Animal2 {}
```

Si tiene un tipo público , debe coincidir con el nombre del archivo. La declaración de clase pública Animal2 no se compilaría en un archivo llamado Animal.java. En el Capítulo 5, “Métodos”, analizamos qué opciones de acceso están disponibles además del público.

¿Ya has notado un patrón? Este capítulo incluye numerosas referencias a temas que profundizaremos en capítulos posteriores. Si es un desarrollador de Java experimentado, notará que en este capítulo mantenemos muchos de los ejemplos y reglas simples. No te preocupes; ¡Tenemos el resto del libro para presentar más reglas y casos extremos complicados!

Machine Translated by Google



# Escribir un método main()

Un programa Java comienza su ejecución con su método main() . En esta sección, aprenderá cómo crear uno, pasar un parámetro y ejecutar un programa. El método main() a menudo se denomina punto de entrada al programa, porque es el punto de partida que busca la JVM cuando comienza a ejecutar un nuevo programa.

## Creando un método main()

El método main() permite que la JVM llame a nuestro código. La clase más simple posible con main() El método se ve así:

```
1: zoológico de clase pública {
2:     público estático vacío principal (String [] argumentos) {
3:         System.out.println("Hola mundo");
4:     }
5: }
```

Este código imprime Hola mundo. Para compilar y ejecutar este código, escríbalo en un archivo llamado Zoo.java y ejecute lo siguiente:

```
javac Zoo.java
zoológico de java
```

Si imprime Hola mundo, tuvo éxito. Si recibe mensajes de error, verifique que haya instalado el JDK de Java 17, que lo haya agregado a la RUTA y que no haya cometido ningún error tipográfico en el ejemplo. Si tiene alguno de estos problemas y no sabe qué hacer, publique una pregunta con el mensaje de error que recibió en el foro Beginning Java en CodeRanch:

www.coderanch.com/forums/f-33/java

Para compilar código Java con el comando javac , el archivo debe tener la extensión .java. El nombre del archivo debe coincidir con el nombre de la clase pública . El resultado es un archivo de código de bytes con el mismo nombre pero con una extensión de nombre de archivo .class . Recuerde que el código de bytes consta de instrucciones que la JVM sabe cómo ejecutar. Note que debemos omitir el .class extensión para ejecutar Zoo.class.

Las reglas sobre lo que contiene un archivo Java y en qué orden son más detalladas que lo que hemos explicado hasta ahora (habrá más sobre este tema más adelante en este capítulo). Para simplificar las cosas por ahora, seguimos este subconjunto de reglas:

- Cada archivo puede contener sólo una clase pública .
- El nombre del archivo debe coincidir con el nombre de la clase, incluido el caso, y tener una extensión .java .
- Si la clase Java es un punto de entrada para el programa, debe contener un método main() válido.

Primero revisemos las palabras en la firma del método main() , una a la vez. La palabra clave public es lo que se llama un modificador de acceso. Declara el nivel de exposición de este método a posibles llamantes en el programa. Naturalmente, público significa acceso completo desde cualquier parte del programa. Aprenderá más sobre los modificadores de acceso en el Capítulo 5.

Machine Translated by Google



La palabra clave static vincula un método a su clase para que pueda ser llamado solo por el nombre de la clase, como en, por ejemplo, Zoo.main(). Java no necesita crear un objeto para llamar a main() método, lo cual es bueno ya que aún no has aprendido a crear objetos. De hecho, la JVM hace esto, más o menos, cuando carga el nombre de clase que se le asigna. Si un método main() no tiene las palabras clave correctas, obtendrá un error al intentar ejecutarlo. Verá estática nuevamente en el Capítulo 6, "Diseño de clases".

La palabra clave void representa el tipo de devolución. Un método que no devuelve datos devuelve el control a la persona que llama de forma silenciosa. En general, es una buena práctica utilizar void para métodos que cambian el estado de un objeto. En ese sentido, el método main() cambia el estado del programa de iniciado a finalizado. También exploramos los tipos de devolución en el Capítulo 5. (¿Estás emocionado por el Capítulo 5 todavía?)

Finalmente, llegamos a la lista de parámetros del método main() , representada como una matriz de objetos java.lang.String . Puede utilizar cualquier nombre de variable válido junto con cualquiera de estos tres formatos:

```
argumentos de cadena []
Opciones de cadena[]
Cuerda... amigos
```

El compilador acepta cualquiera de estos. El nombre de la variable args es común porque sugiere que esta lista contiene valores que se leyeron (argumentos) cuando se inició la JVM. Los caracteres [] son corchetes y representan una matriz. Una matriz es una lista de tamaño fijo de elementos que son todos del mismo tipo. Los caracteres ... se llaman varargs (listas de argumentos variables). Aprenderá sobre String en este capítulo. Las matrices se encuentran en el Capítulo 4, “API principales”, y los varargs, en el Capítulo 5.

**Modificadores opcionales en los métodos main()**

Si bien la mayoría de los modificadores, como public y static, son necesarios para los métodos main() , se permiten algunos modificadores opcionales.

```
público final estático vacío principal ( cadena final [] argumentos) {}
```

En este ejemplo, ambos modificadores finales son opcionales y el método main() es un punto de entrada válido con o sin ellos. Cubrimos el significado de los métodos y parámetros finales en el Capítulo 6.

## Pasar parámetros a un programa Java

Veamos cómo enviar datos al método main() de nuestro programa . Primero, modificamos el zoológico. programa para imprimir los dos primeros argumentos pasados:

```
Zoológico de clase pública {
    público estático vacío principal (String [] argumentos) {
```

Machine Translated by Google



```
        System.out.println(args[0]);
        System.out.println(args[1]);
    }
}
```

El código args[0] accede al primer elemento de la matriz. Así es: los índices de matriz comienzan con 0 en Java. Para ejecutarlo, escriba esto:

```
javac Zoo.java
java Zoológico Zoológico del Bronx
```

El resultado es el que cabría esperar:

```
Bronx
Lindo
```

El programa identifica correctamente las dos primeras “palabras” como argumentos. Se utilizan espacios para separar los argumentos. Si desea espacios dentro de un argumento, debe utilizar comillas como en este ejemplo:

```
javac Zoo.java
java Zoo "San Diego" Zoo
```

Ahora tenemos un espacio en la salida:

```
San Diego
Lindo
```

Finalmente, ¿qué sucede si no pasa suficientes argumentos?

```
javac Zoo.java
zoológico de java zoológico
```

La lectura de argumentos [0] va bien y se imprime Zoo . Entonces Java entra en pánico. no hay segundo ¡argumento! ¿Qué hacer? Java imprime una excepción que le indica que no tiene idea de qué hacer con este argumento en la posición 1. (Aprenderá sobre las excepciones en el Capítulo 11, “Excepciones y localización”).

```
Lindo
Excepción en el hilo "principal" java.lang.ArrayIndexOutOfBoundsException:
    Índice 1 fuera de límites para la longitud 1
    en Zoo.main(Zoo.java:4)
```

Para revisar, el JDK contiene un compilador. Los archivos de clase Java se ejecutan en la JVM y, por lo tanto, se ejecutan en cualquier máquina con Java en lugar de solo en la máquina o sistema operativo en el que se compilaron.

Machine Translated by Google



Código fuente de archivo único

Si te cansas de escribir javac y java cada vez que quieres probar un ejemplo de código, existe un atajo. En su lugar puedes ejecutar

```
java Zoo.java Zoológico del Bronx
```

Aquí hay una diferencia clave. Al compilar primero, omitiste la extensión .java al ejecutar java. Al omitir el paso de compilación explícito, incluye esta extensión. Esta característica se llama iniciar programas de código fuente de un solo archivo y es útil para pruebas o para programas pequeños. El nombre le indica ingeniosamente que está diseñado para cuando su programa es un archivo.

# Comprensión de las declaraciones e importaciones de paquetes

Java viene con miles de clases integradas y hay muchas más de desarrolladores como usted. Con todas esas clases, Java necesita una forma de organizarlas. Maneja esto de manera similar a un archivador. Pones todos tus papeles en carpetas. Java pone clases en paquetes. Estas son agrupaciones lógicas para clases.

No lo pondríamos frente a un archivador y le diríamos que busque un documento específico. En lugar de eso, le diremos en qué carpeta buscar. Java funciona de la misma manera. Necesita que le digas en qué paquetes buscar para encontrar el código.

Supongamos que intentas compilar este código:

```
clase pública Selector de números {
    público estático vacío principal (String [] argumentos) {
        Aleatorio r = nuevo Aleatorio(); // NO COMPILA
        System.out.println(r.nextInt(10));
    }
}
```

El compilador de Java le muestra un error similar a este:

```
error: no se puede encontrar el símbolo
```

Este error podría significar que cometió un error tipográfico en el nombre de la clase. Lo vuelves a comprobar y descubres que no lo hiciste. La otra causa de este error es omitir una declaración de importación necesaria . Una declaración es una instrucción, y las declaraciones de importación le dicen a Java qué paquetes buscar clases. Como no le dijiste a Java dónde buscar Random, no tiene idea.

Intentar esto nuevamente con la importación permite que el código se compile.

Machine Translated by Google



```
importar java.util.Random; // la importación nos dice dónde encontrar Random
clase pública Selector de números {
    público estático vacío principal (String [] argumentos) {
        Aleatorio r = nuevo Aleatorio();
        System.out.println(r.nextInt(10)); // un número 0-9
    }
}
```

Ahora el código se ejecuta; imprime un número aleatorio entre 0 y 9. Al igual que las matrices, Java Le gusta empezar a contar con 0.

En el Capítulo 5, cubrimos otro tipo de importación denominada estática. importar. Le permite dar a conocer miembros estáticos de una clase, a menudo para que pueda usar variables y nombres de métodos sin tener que seguir especificando el nombre de la clase.

## Paquetes

Como vio en el ejemplo anterior, las clases de Java se agrupan en paquetes. la importacion La declaración le dice al compilador en qué paquete buscar para encontrar una clase. Esto es similar a cómo funciona el envío de una carta. Imagine que está enviando una carta al 123 Main Street, apartamento 9. El cartero primero lleva la carta al 123 de Main Street. Luego el transportista busca el buzón del apartamento número 9. La dirección es como el nombre del paquete en Java.
El número de apartamento es como el nombre de la clase en Java. Así como el cartero solo busca los números de apartamentos en el edificio, Java solo busca nombres de clases en el paquete.

Los nombres de los paquetes también son jerárquicos como el correo. El servicio postal comienza con el nivel superior, mirando primero a su país. También empiezas a leer el nombre de un paquete al principio. Por ejemplo, si comienza con java, significa que viene con el JDK. Si comienza con algo más, probablemente muestre de dónde viene usando el nombre del sitio web al revés. Por ejemplo, com.wiley.javabook nos dice que el código está asociado con wiley.com sitio web u organización. Después del nombre del sitio web, puedes agregar lo que quieras. Por ejemplo, com.wiley.java.mi.nombre también vino de wiley.com. Java llama paquetes secundarios a paquetes más detallados. El paquete com.wiley.javabook es un paquete secundario de com.wiley. Se nota porque es más largo y, por tanto, más específico.

Verá nombres de paquetes en el examen que no siguen esta convención. No se sorprenda al ver nombres de paquetes como abc. La regla para los nombres de paquetes es que en su mayoría sean letras o números separados por puntos (.). Técnicamente, se te permiten un par de caracteres más entre los puntos (.). Incluso puedes usar nombres de paquetes de sitios web que no te pertenecen si así lo deseas, como com.wiley, ¡ aunque las personas que lean tu código podrían confundirse! Las reglas son las mismas que para los nombres de variables, que verá más adelante en este capítulo. Es posible que el examen intente engañarlo con nombres de variables no válidos. Afortunadamente, no intenta engañarte dando nombres de paquetes no válidos.

Machine Translated by Google



En las siguientes secciones, analizamos las importaciones con comodines y los conflictos de nombres con importaciones, cómo crear un paquete propio y cómo el examen formatea el código.

## comodines

Las clases del mismo paquete suelen importarse juntas. Puede utilizar un acceso directo para importar todas las clases de un paquete.

```
importar java.util.*; clase          // importa java.util.Random entre otras cosas
pública Selector de números {
    público estático vacío principal (String [] argumentos) {
        Aleatorio r = nuevo Aleatorio();
        System.out.println(r.nextInt(10));
    }
}
```

En este ejemplo, importamos java.util.Random y muchas otras clases. El * es un comodín que coincide con todas las clases del paquete. Cada clase del paquete java.util está disponible para este programa cuando Java lo compila. La declaración de importación no incluye paquetes, campos o métodos secundarios; importa solo clases directamente bajo el paquete. Digamos que desea utilizar la clase AtomicInteger (aprenderá sobre ella en el Capítulo 13, "Concurrencia") en el paquete java.util.concurrent.atomic . ¿Qué importación o importaciones respaldan esto?

```
import java.util.*;
importar java.util.concurrent.*;
importar java.util.concurrent.atomic.*;
```

Sólo la última importación permite que la clase sea reconocida porque los paquetes secundarios no están incluido con los dos primeros.

Podrías pensar que incluir tantas clases ralentiza la ejecución de tu programa, pero no es así. El compilador descubre lo que realmente se necesita. El enfoque que elija será una preferencia personal o una preferencia de equipo, si trabaja con otras personas en un equipo. Enumerar las clases utilizadas hace que el código sea más fácil de leer, especialmente para los nuevos programadores. El uso del comodín puede acortar la lista de importación. Verá ambos enfoques en el examen.

## Importaciones redundantes

¡Espera un minuto! Nos hemos estado refiriendo al Sistema sin importación cada vez que imprimimos texto, y Java lo encontró muy bien. Hay un paquete especial en el mundo Java llamado java.lang. Este paquete es especial porque se importa automáticamente. Puede escribir este paquete en una declaración de importación , pero no es necesario. En el siguiente código, ¿cuántas de las importaciones crees que son redundantes?

```
1: importar java.lang.System;
2: importar java.lang.*;
```

Machine Translated by Google



```
3: importar java.util.Random;
4: import java.util.*;
5: clase pública Selector de números {
6:         público estático vacío principal (String [] argumentos) {
7:             Aleatorio r = nuevo Aleatorio();
8:             System.out.println(r.nextInt(10));
9:         }
10: }
```

La respuesta es que tres de las importaciones son redundantes. Las líneas 1 y 2 son redundantes porque todo lo que hay en java.lang se importa automáticamente. La línea 4 también es redundante en este ejemplo porque Random ya se importó desde java.util.Random. Sin embargo, si la línea 3 no estuviera presente, java.util.* no sería redundante, ya que cubriría la importación de Random.

Otro caso de redundancia implica importar una clase que está en el mismo paquete que el clase importándolo. Java busca automáticamente otras clases en el paquete actual.

Echemos un vistazo a un ejemplo más para asegurarnos de que comprende los casos extremos de las importaciones. Para este ejemplo, los archivos y las rutas están en el paquete java.nio.file. El examen puede utilizar paquetes que quizás nunca antes haya visto. La pregunta le permitirá saber en qué paquete se encuentra la clase si necesita saberlo para responder la pregunta.

¿ Qué declaraciones de importación crees que funcionarían para compilar este código?

```
Importaciones de entrada de clase pública {
    lectura pública vacía ( archivos de archivos) {
        Rutas.get("nombre");
    }
}
```

Hay dos respuestas posibles. La más corta es utilizar un comodín para importar ambos al mismo tiempo.

```
importar java.nio.file.*;
```

La otra respuesta es importar ambas clases explícitamente.

```
importar java.nio.file.Files;
importar java.nio.file.Paths;
```

Ahora consideremos algunas importaciones que no funcionan.

```
import java.nio.*;                    // NO BUENO - un comodín sólo coincide
                                      // nombres de clases, no "archivo.Archivos"

import java.nio.*.*;                  // NO BUENO - solo puedes tener un comodín
                                      // y debe estar al final

importar java.nio.file.Paths.*; // NO BUENO - no puedes importar métodos
                                      // solo nombres de clases
```

Machine Translated by Google



## Conflictos de nombres

Una de las razones para usar paquetes es que los nombres de clases no tienen que ser únicos en todo Java. Esto significa que a veces querrás importar una clase que se pueda encontrar en varios lugares. Un ejemplo común de esto es la clase Fecha . Java proporciona implementaciones de java.util.Date y java.sql.Date. ¿ Qué declaración de importación podemos usar si queremos la versión java.util.Date ?

```
Conflictos de clase pública {
    Fecha fecha;
    // algo más de código
}
```

La respuesta ya debería ser fácil. Puede escribir import java.util.*; o importar java.util.Date;. Los casos difíciles surgen cuando hay otras importaciones presentes.

```
import java.util.*;
importar java.sql.*; // hace que la declaración de fecha no se compile
```

Cuando el nombre de la clase se encuentra en varios paquetes, Java genera un error de compilación. En nuestro ejemplo, la solución es fácil: eliminar el import java.sql.* que no necesitamos. Pero, ¿qué hacemos si necesitamos un montón de otras clases en el paquete java.sql ?

```
importar java.util.Fecha;
importar java.sql.*;
```

¡Ah, ahora funciona! Si importa explícitamente un nombre de clase, tiene prioridad sobre los comodines presentes. Java piensa: "El programador realmente quiere que asuma el uso de la clase java.util.Date ".

Un ejemplo más. ¿Qué hace Java con los "vínculos" de precedencia?

```
importar java.util.Fecha;
importar java.sql.Fecha;
```

Java es lo suficientemente inteligente como para detectar que este código no sirve. Como programador, has afirmado desear explícitamente que el valor predeterminado sean las implementaciones java.util.Date y java.sql.Date . Como no puede haber dos valores predeterminados, el compilador le dice que las importaciones son ambiguas.

### Si realmente necesita utilizar dos clases con el mismo nombre

A veces realmente deseas utilizar Date desde dos paquetes diferentes. Cuando esto sucede, puede elegir uno para usarlo en la declaración de importación y usar el nombre de clase completo del otro. O puede eliminar ambas declaraciones de importación y utilizar siempre el nombre de clase completo.

```
Conflictos de clase pública {
    java.util.Fecha fecha;
    java.sql.Date sqlDate;
}
```

Machine Translated by Google



## Creando un nuevo paquete

Hasta ahora, todo el código que hemos escrito en este capítulo ha estado en el paquete predeterminado. Este es un paquete especial sin nombre que debe usar sólo para código desechable. Puede saber que el código está en el paquete predeterminado porque no hay ningún nombre de paquete. En el examen, verá que el paquete predeterminado se usa mucho para ahorrar espacio en las listas de códigos. En la vida real, siempre nombre sus paquetes para evitar conflictos de nombres y permitir que otros reutilicen su código.

Ahora es el momento de crear un nuevo paquete. La estructura de directorios de su computadora está relacionada con el nombre del paquete. En esta sección, simplemente siga leyendo. Cubrimos cómo compilar y ejecutar el código en la siguiente sección.

Supongamos que tenemos estas dos clases:

```
paquete paquetea;
clase pública ClaseA {}

paquete paqueteb;
importar paquetea.ClassA;
clase pública ClaseB {
    público estático vacío principal (String [] argumentos) {
        Clase A;
        System.out.println("Entendido");
    }
}
```

Cuando ejecuta un programa Java, Java sabe dónde buscar esos nombres de paquetes. En este caso, ejecutar desde C:\temp funciona porque tanto el paquetea como el paqueteb están debajo.

## Compilación y ejecución de código con paquetes

Aprenderá Java mucho más fácilmente utilizando la línea de comandos para compilar y probar sus ejemplos. Una vez que conozca bien la sintaxis de Java, podrá cambiar a un IDE. Pero para el examen, su objetivo es conocer los detalles sobre el idioma y no que el IDE los oculte.

Siga este ejemplo para asegurarse de saber cómo utilizar la línea de comando. Si usted tiene Si tiene algún problema al seguir este procedimiento, publique una pregunta en el foro Beginning Java en CodeRanch. Describe lo que intentaste y lo que decía el error.

www.coderanch.com/forums/f-33/java

El primer paso es crear los dos archivos de la sección anterior. La tabla 1.1 muestra la nombres de archivos completos esperados y el comando para ingresar al directorio para los siguientes pasos.

Machine Translated by Google



TABLA 1.1 Procedimiento de configuración por sistema operativo

| Paso | ventanas | Mac/Linux |
|---|---|---|
| 1. Crea la primera clase. | C:\temp\paquetea\ ClaseA.java | /tmp/packagea/ClassA.java |
| 2. Crea una segunda clase. | C:\temp\paqueteb\ ClaseB.java | /tmp/paqueteb/ClassB.java |
| 3. Ir al directorio. | cd C:\temperatura | disco compacto/tmp |

Ahora es el momento de compilar el código. Por suerte, esto es igual independientemente del sistema operativo. Para compilar, escriba el siguiente comando:

```
paquete javaca/ClassA.java paqueteb/ClassB.java
```

Si este comando no funciona, recibirá un mensaje de error. Revise sus archivos cuidadosamente para errores tipográficos en los archivos proporcionados. Si el comando funciona, se crearán dos archivos nuevos: paquetea/ClassA.class y paqueteb/ClassB.class.

### Compilando con comodines

Puede utilizar un asterisco para especificar que desea incluir todos los archivos Java en un directorio. Esto es útil cuando tienes muchos archivos en un paquete. Podemos reescribir el javac anterior. comando como este:

```
paquete javaca/*.java paqueteb/*.java
```

Sin embargo, no puede utilizar un comodín para incluir subdirectorios. Si escribiera javac *.java, el código de los paquetes no se recogería.

Ahora que su código se ha compilado, puede ejecutarlo escribiendo el siguiente comando:

```
paquete javab.ClaseB
```

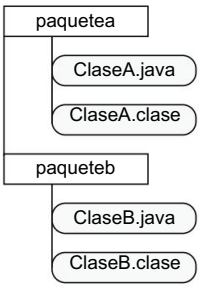
Si funciona, verás Lo tengo impreso. Es posible que hayas notado que escribimos ClassB en lugar de ClassB.class. Como se mencionó anteriormente, no se pasa la extensión cuando se ejecuta un programa.

La Figura 1.1 muestra dónde se crearon los archivos .class en la estructura del directorio.

Machine Translated by Google



FIGURA 1.1 Compilación con paquetes




# Compilando en otro directorio

De forma predeterminada, el comando javac coloca las clases compiladas en el mismo directorio que el código fuente. También proporciona una opción para colocar los archivos de clase en un directorio diferente. La opción -d especifica este directorio de destino.

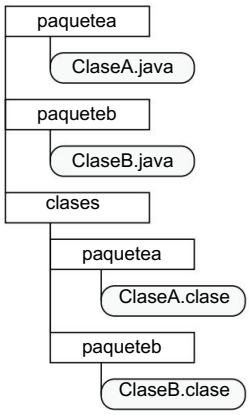
Las opciones de Java distinguen entre mayúsculas y minúsculas. Esto significa que no puedes pasar -D en lugar de -d.

Si sigue adelante, elimine los archivos ClassA.class y ClassB.class que se crearon en la sección anterior. ¿Dónde crees que este comando creará el archivo ClassA.class?

javac -d clases paquetea/ClassA.java paqueteb/ClassB.java

La respuesta correcta está en clases/paquetea/ClassA.class. La estructura del paquete es conservado en el directorio de destino solicitado. La figura 1.2 muestra esta nueva estructura.

FIGURA 1.2 Compilación con paquetes y directorios



Machine Translated by Google



Para ejecutar el programa, especifica la ruta de clase para que Java sepa dónde encontrar las clases. Hay tres opciones que puedes utilizar. Los tres hacen lo mismo:

```
paquete de clases java -cpb.ClassB
paquete de clases java -classpathb.ClassB
java --class-path clases paqueteb.ClassB
```

Observe que el último requiere dos guiones (--), mientras que los dos primeros requieren un guión (-). Si tiene una cantidad incorrecta de guiones, el programa no se ejecutará.

### Tres opciones de ruta de clase

Quizás se pregunte por qué hay tres opciones para el classpath. La opción -cp es la forma corta. Los desarrolladores frecuentemente eligen la forma abreviada porque somos mecanógrafos perezosos. Las versiones -classpath y --class-path pueden ser más claras de leer pero requieren más escritura.

La Tabla 1.2 y la Tabla 1.3 revisan las opciones que necesita conocer para el examen. ¡ Hay muchas otras opciones disponibles! Y en el Capítulo 12, “Módulos”, aprenderá opciones adicionales específicas de los módulos.

TABLA 1.2 Opciones importantes de javac

| Opción | Descripción |
|---|---|
| -cp <ruta de clase><br>-classpath <claseruta><br>--class-path <ruta de clase> | Ubicación de las clases necesarias para compilar el programa. |
| -d <dir> | Directorio en el que colocar los archivos de clase generados |

TABLA 1.3 Opciones importantes de Java

| Opción | Descripción |
|---|---|
| -cp <ruta de clase><br>-classpath <claseruta><br>--class-path <ruta de clase> | Ubicación de las clases necesarias para ejecutar el programa. |

Machine Translated by Google



## Compilando con archivos JAR

Al igual que el directorio de clases en el ejemplo anterior, también puedes especificar la ubicación de los otros archivos explícitamente usando una ruta de clase. Esta técnica es útil cuando los archivos de clase se encuentran en otro lugar o en archivos JAR especiales. Un archivo Java (JAR) es como un archivo ZIP compuesto principalmente por archivos de clase Java.

En Windows, escribe lo siguiente:

```
java -cp ".;C:\temp\someOtherLocation;c:\temp\myJar.jar" miPaquete.MiClase
```

Y en macOS/Linux, escribe esto:

```
java -cp ".:/tmp/someOtherLocation:/tmp/myJar.jar" myPackage.MyClass
```

El punto (.) indica que desea incluir el directorio actual en el classpath. El
El resto del comando dice buscar archivos de clases (o paquetes) sueltos en alguna otra ubicación .
y dentro de myJar.jar. Windows usa punto y coma (;) para separar partes del classpath; otros sistemas operativos utilizan dos puntos.

Al igual que cuando estás compilando, puedes usar un comodín (*) para hacer coincidir todos los JAR en un directorio. He aquí un ejemplo:

```
java -cp "C:\temp\directoryWithJars\*" miPaquete.MiClase
```

Este comando agregará al classpath todos los JAR que están en el directorio WithJars. No incluirá ningún JAR en el classpath que esté en un subdirectorio de directorioWithJars.

## Crear un archivo JAR

Algunos JAR son creados por otros, como los descargados de Internet o los creados por un compañero de equipo. Alternativamente, puedes crear un archivo JAR tú mismo. Para hacerlo, usas el frasco.
dominio. Los comandos más simples crean un jar que contiene los archivos en el directorio actual.
Puede utilizar el formulario corto o largo para cada opción.

```
jar -cvf miNuevoArchivo.jar .
jar --create --verbose --file myNewFile.jar .
```

Alternativamente, puede especificar un directorio en lugar de utilizar el directorio actual.

```
jar -cvf miNuevoArchivo.jar -C dir .
```

No existe una forma larga de la opción -C . La Tabla 1.4 enumera las opciones que necesita para usar el frasco . comando para crear un archivo JAR. En el Capítulo 12, verá nuevamente jar para los módulos.

Machine Translated by Google



TABLA 1.4 Opciones importantes de frascos

| Opción | Descripción |
|---|---|
| -C<br>--crear | Crea un nuevo archivo JAR |
| -en<br>--verboso | Imprime detalles al trabajar con archivos JAR |
| -f <nombre de archivo><br>--file <nombre de archivo> | nombre de archivo JAR |
| -C <directorio> | Directorio que contiene los archivos que se utilizarán para crear el JAR. |

## Ordenar elementos en una clase

Ahora que has visto las partes más comunes de una clase, echemos un vistazo al orden correcto para escribirlas en un archivo. Los comentarios pueden ir a cualquier parte del código. Más allá de eso, es necesario memorizar las reglas de la Tabla 1.5.

TABLA 1.5 Orden de declaración de una clase

| Elemento | Ejemplo | ¿Requerido? | ¿A dónde va? |
|---|---|---|---|
| Declaración de paquete | paquete abc; | No | Primera línea del archivo (excluyendo comentarios o líneas en blanco) |
| declaraciones de importación | import java.util.*; | No | Inmediatamente después del paquete (si está presente) |
| Declaración de tipo de nivel superior | clase pública C | Sí | Inmediatamente después de la importación (si corresponde) |
| Declaraciones de campo | valor entero; | No | Cualquier elemento de nivel superior dentro de una clase. |
| Declaraciones de métodos | método nulo() | No | Cualquier elemento de nivel superior dentro de una clase. |

Machine Translated by Google



Veamos algunos ejemplos para ayudarle a recordar esto. El primer ejemplo contiene uno de cada elemento:

```
estructura del paquete ; // el paquete debe ser el primero sin comentarios
importar java.util.*; // la importación debe venir después del paquete
public class Suricata { // luego viene la clase
    doble peso; // los campos y métodos pueden ir en cualquier orden
    público doble getPeso() {
        peso de retorno; }
    doble altura; // otro campo - no necesitan estar juntos
}
```

Hasta ahora, todo bien. Este es un patrón común con el que deberías estar familiarizado. ¿Que tal este?

```
/* encabezado */

estructura del paquete ;


// clase suricata
clase pública suricata { }
```

Sigue bien. Podemos poner comentarios en cualquier lugar, las líneas en blanco se ignoran y las importaciones son opcionales. En el siguiente ejemplo, tenemos un problema:

```
import java.util.*;
estructura del paquete ; // NO COMPILA
Nombre de cadena; // NO COMPILA
public class Suricata { } // NO COMPILA
```

Hay dos problemas aquí. Una es que las declaraciones de paquete e importación están invertidas. Aunque ambos son opcionales, el paquete debe venir antes de la importación, si está presente. El otro problema es que un campo intenta realizar una declaración fuera de una clase. Esto no esta permitido. Los campos y métodos deben estar dentro de una clase.

¿Tienes todo eso? Piense en el acrónimo PIC (imagen): paquete, importación y clase. Campos y Los métodos son más fáciles de recordar porque simplemente tienen que estar dentro de una clase.

A lo largo de este libro, si ve dos clases públicas en un fragmento de código o pregunta, puede asumir que están en archivos diferentes, a menos que diga específicamente que están en el mismo archivo .java .

Ahora ya sabes cómo crear y organizar una clase. Los capítulos posteriores le muestran cómo crear Clases con operaciones más potentes.

Machine Translated by Google



# Creando objetos

Nuestros programas no podrían hacer nada útil si no tuviéramos la capacidad de crear nuevos objetos. Recuerde que un objeto es una instancia de una clase. En las siguientes secciones, analizamos los constructores, los campos de objetos, los inicializadores de instancias y el orden en que se inicializan los valores.

## Llamando a constructores

Para crear una instancia de una clase, todo lo que tiene que hacer es escribir nuevo antes del nombre de la clase y agregar paréntesis después. He aquí un ejemplo:

```
Parque p = nuevo Parque();
```

Primero declaras el tipo que crearás (Park) y le das un nombre a la variable (p). Esto le da a Java un lugar para almacenar una referencia al objeto. Luego escribes new Park() para crear realmente el objeto.

Park() parece un método ya que va seguido de paréntesis. Se llama constructor que es un tipo especial de método que crea un nuevo objeto. Ahora es el momento de definir un constructor propio:

```
polluelo de clase pública {
    polluelo público () {
        System.out.println("en constructor");
    }
}
```

Hay dos puntos clave a tener en cuenta sobre el constructor: el nombre del constructor coincide con el nombre de la clase y no hay ningún tipo de retorno. Es posible que vea un método como este en el examen:

```
polluelo de clase pública {
    public void Chick() { } // NO ES UN CONSTRUCTOR
}
```

Cuando vea el nombre de un método que comienza con una letra mayúscula y tiene un tipo de retorno, preste especial atención. No es un constructor ya que hay un tipo de retorno. Es un método normal que se compila pero no se llamará cuando escribas new Chick().

El propósito de un constructor es inicializar campos, aunque puedes poner cualquier código allí. Otra forma de inicializar campos es hacerlo directamente en la línea en la que están declarados. Este ejemplo muestra ambos enfoques:

```
pollo de clase pública {
    int numHuevos = 12; // inicializar en línea
    Nombre de cadena;
```

Machine Translated by Google



```
    Pollo público() {
        nombre = "Duque"; // inicializar en el constructor
    }
}
```

Para la mayoría de las clases, no es necesario codificar un constructor; el compilador le proporcionará un constructor predeterminado que “no hace nada”. Hay algunos escenarios que requieren que declares un constructor. Aprenderá todo sobre ellos en el Capítulo 6.

## Campos de miembros de lectura y escritura

Es posible leer y escribir variables de instancia directamente desde la persona que llama. En este ejemplo, una madre cisne pone huevos:

```
cisne de clase pública {
    int númeroHuevos;                                   // Instancia variable
    público estático vacío principal (String [] argumentos) {
        Madre cisne = nuevo cisne();
        madre.númeroHuevos = 1;                         // establecer variable
        System.out.println(madre.númeroHuevos); // leer variable
    }
}
```

El “llamador” en este caso es el método main() , que podría estar en la misma clase o en otra clase. Esta clase establece numberEggs en 1 y luego lee numberEggs directamente para imprimirlo. En el Capítulo 5, aprenderá cómo utilizar la encapsulación para proteger a la clase Swan de que alguien coloque un número negativo de huevos.

Incluso puedes leer valores de campos ya inicializados en una línea que inicializa un nuevo campo:

```
1: nombre de clase pública {
2:     Cadena primero = "Theodore";
3:     Cadena último = "Alce";
4:     Cadena completa = primero + último;
5: }
```

Las líneas 2 y 3 escriben en campos. La línea 4 lee y escribe datos. Lee los campos primero y último. Luego escribe el campo completo.

## Ejecutar bloques inicializadores de instancia

Cuando aprendiste sobre los métodos, viste llaves ({}). El código entre llaves (a veces llamado "dentro de las llaves") se llama bloque de código. Cualquier lugar donde vea llaves es un bloque de código.

Machine Translated by Google



A veces, los bloques de código están dentro de un método. Estos se ejecutan cuando se llama al método. Otras veces, los bloques de código aparecen fuera de un método. Estos se denominan inicializadores de instancia. En el Capítulo 6, aprenderá a utilizar un inicializador estático .

¿Cuántos bloques ves en el siguiente ejemplo? ¿Cuántos inicializadores de instancia ves?

```
1: clase pública Pájaro {
2:         público estático vacío principal (String [] argumentos) {
3:             { System.out.println("Plumas"); }
4:         }
5:         { System.out.println("Nevado"); }
6: }
```

Hay cuatro bloques de código en este ejemplo: una definición de clase, una declaración de método, una bloque interno y un inicializador de instancia. Contar bloques de código es fácil: simplemente cuenta el número de pares de llaves. Si no hay la misma cantidad de llaves de apertura ({) y de cierre (}) o no están definidas en el orden correcto, el código no se compila. Por ejemplo, no puede utilizar una llave cerrada (}) si no hay una llave abierta correspondiente ({) que coincida con lo escrito anteriormente en el código. En programación, esto se conoce como el problema del paréntesis equilibrado y, a menudo, surge en las preguntas de las entrevistas de trabajo.

Cuando cuente los inicializadores de instancias, tenga en cuenta que no pueden existir dentro de un método. La línea 5 es un inicializador de instancia, con sus llaves fuera de un método. Por otro lado, la línea 3 no es un inicializador de instancia, ya que solo se llama cuando se ejecuta el método main() .
Hay un conjunto adicional de llaves en las líneas 1 y 6 que constituyen la declaración de clase.

## Siguiendo el orden de inicialización

Al escribir código que inicializa campos en varios lugares, debe realizar un seguimiento del orden de inicialización. Este es simplemente el orden en el que se llaman los diferentes métodos, constructores o bloques cuando se crea una instancia de la clase. Agregamos algunas reglas más al orden de inicialización en el Capítulo 6. Mientras tanto, debes recordar:

- Los campos y los bloques inicializadores de instancias se ejecutan en el orden en que aparecen en el archivo.
- El constructor se ejecuta después de que se hayan ejecutado todos los campos y bloques inicializadores de instancias.

Veamos un ejemplo:

```
1: polluelo de clase pública {
2:         nombre de cadena privada = "Fluffy";
3:         { System.out.println("campo de configuración"); }
4:         polluelo público() {
5:             nombre = "Pequeño";
6:             System.out.println("constructor de configuración");
7:         }
```

Machine Translated by Google



```
8:      público estático vacío principal (String [] argumentos) {
9:          Pollito pollito = nuevo pollito();
10:         System.out.println(pollo.nombre); } }
```

Al ejecutar este ejemplo se imprime esto:

```
campo de configuración
constructor de configuración
Diminuto
```

Veamos lo que está pasando aquí. Comenzamos con el método main() porque ahí es donde Java comienza la ejecución. En la línea 9, llamamos al constructor de Chick. Java crea un nuevo objeto. Primero inicializa el nombre a "Fluffy" en la línea 2. Luego ejecuta la declaración println() en el inicializador de instancia en la línea 3. Una vez que se han ejecutado todos los campos y los inicializadores de instancia, Java regresa al constructor. La línea 5 cambia el valor del nombre a "Tiny" y la línea 6 imprime otra declaración. En este punto, el constructor finaliza y luego la ejecución vuelve a la instrucción println() en la línea 10.

El orden es importante para los campos y bloques de código. No puede hacer referencia a una variable antes de que se haya definido:

```
{ System.out.println(nombre); } // NO COMPILA
nombre de cadena privada = "Fluffy";
```

Debería esperar ver una pregunta sobre la inicialización en el examen. Probemos uno más. ¿Qué crees que imprime este código?

```
huevo de clase pública {
    huevo público() {
        número = 5;
    }
    público estático vacío principal (String [] argumentos) {
        Huevo huevo = nuevo Huevo();
        System.out.println(número.de.huevo);
    }
    número entero privado = 3;
    { número = 4; } }
```

Si respondiste 5, acertaste. Los campos y bloques se ejecutan primero en orden, estableciendo el número a 3 y luego a 4. Luego, el constructor se ejecuta, estableciendo el número en 5. Verá muchas más reglas y ejemplos que cubren el orden de inicialización en el Capítulo 6. Aquí solo cubrimos los conceptos básicos para que pueda seguir el orden de inicialización de programas simples.

# Comprender los tipos de datos

Las aplicaciones Java contienen dos tipos de datos: tipos primitivos y tipos de referencia. En esta sección, analizamos las diferencias entre un tipo primitivo y un tipo de referencia.

Machine Translated by Google



# Usando tipos primitivos

Java tiene ocho tipos de datos integrados, denominados tipos primitivos de Java. Estos ocho tipos de datos representan los componentes básicos de los objetos Java, porque todos los objetos Java son sólo una colección compleja de estos tipos de datos primitivos. Dicho esto, una primitiva no es un objeto en Java ni representa un objeto. Una primitiva es solo un valor único en la memoria, como un número o un carácter.

## Los tipos primitivos

El examen supone que usted conoce bien los ocho tipos de datos primitivos, sus tamaños relativos y lo que se puede almacenar en ellos. La Tabla 1.6 muestra los tipos primitivos de Java junto con su tamaño en bits y el rango de valores que contiene cada uno.

TABLA 1.6 Tipos primitivos

| Tipo de palabra clave | | Valor mínimo | Valor máximo | Valor predeterminado | Ejemplo |
|---|---|---|---|---|---|
| booleano | verdadero o falso | n / A | n / A | FALSO | verdadero |
| byte | valor integral de 8 bits | -128 | 127 | 0 | 123 |
| valor integral corto de 16 bits | | -32,768 | 32.767 | 0 | 123 |
| En t | valor integral de 32 bits | -2.147.483.648 | 2.147.483.647 | 0 | 123 |
| largo | valor integral de 64 bits | -263 | 263 – 1 | 0L | 123L |
| valor flotante de punto flotante de 32 bits | | n / A | n / A | 0.0f | 123.45f |
| doble valor de punto flotante de 64 bits | | n / A | n / A | 0,0 | 123,456 |
| carbonizarse | Valor Unicode de 16 bits | 0 | 65.535 | \u0000 | 'a' |

### ¿Es la cuerda un primitivo?

No, no es. Dicho esto, String a menudo se confunde con una novena primitiva porque Java incluye soporte integrado para operadores y literales de String . Aprenderá más sobre String en el Capítulo 4, pero por ahora, recuerde que es un objeto, no una primitiva.

Machine Translated by Google



Hay mucha información en la Tabla 1.6. Veamos algunos puntos clave:

- Los tipos byte, short, int y long se utilizan para valores enteros sin dec. puntos imales.
- Cada tipo numérico utiliza el doble de bits que el tipo similar más pequeño. Por ejemplo, short usa el doble de bits que byte .
- Todos los tipos numéricos están firmados y reservan uno de sus bits para cubrir un rango negativo. Por ejemplo, en lugar de que el byte cubra de 0 a 255 (o incluso de 1 a 256) , en realidad cubre de -128 a 127.
- Un flotante requiere la letra f o F después del número para que Java sepa que es un flotante. Sin f o F, Java interpreta un valor decimal como doble.
- Un largo requiere la letra l o L después del número para que Java sepa que es largo. Sin l o L, Java interpreta un número sin punto decimal como un int en la mayoría de los escenarios.

No se le preguntará sobre los tamaños exactos de estos tipos, aunque debería tener una idea general del tamaño de los tipos más pequeños como byte y short. Una pregunta común entre los nuevos desarrolladores de Java es: ¿cuál es el tamaño en bits de un valor booleano? La respuesta es que no está especificado y depende de la JVM donde se ejecuta el código.

### Firmado y sin firmar: breve y char

Para el examen, debes tener en cuenta que short y char están estrechamente relacionados, ya que ambos se almacenan como tipos integrales con la misma longitud de 16 bits. La principal diferencia es que corta tiene signo, lo que significa que divide su rango entre los números enteros positivos y negativos. Alternativamente, char no está firmado, lo que significa que su rango es estrictamente positivo, incluido 0.

A menudo, los valores short y char se pueden convertir entre sí porque el tamaño de los datos subyacentes es el mismo. Aprenderá más sobre el casting en el Capítulo 2, "Operadores".

## Escribir literales

Hay algunas cosas más que debes saber sobre las primitivas numéricas. Cuando un número está presente en el código, se llama literal. De forma predeterminada, Java supone que está definiendo un valor int con un literal numérico. En el siguiente ejemplo, el número enumerado es mayor que lo que cabe en un int. Recuerde, no se espera que memorice el valor máximo de un int. El examen lo incluirá en la pregunta si surge.

```
largo máximo = 3123456789; // NO COMPILA
```

Machine Translated by Google



Java se queja de que el número está fuera de rango. Y lo es, durante un int. Sin embargo, no tenemos un int. La solución es añadir el carácter L al número:

```
largo máximo = 3123456789L; // Ahora Java sabe que es un largo
```

Alternativamente, puede agregar una l minúscula al número. Pero por favor use la L mayúscula. La l minúscula se parece al número 1.

Otra forma de especificar números es cambiar la "base". Cuando aprendiste a contar, Estudiaste los dígitos del 0 al 9. Este sistema de numeración se llama base 10 ya que hay 10 valores posibles para cada dígito. También se le conoce como sistema numérico decimal. Java le permite especificar dígitos en varios otros formatos:

- Octal (dígitos del 0 al 7), que utiliza el número 0 como prefijo; por ejemplo, 017.
- Hexadecimal (dígitos del 0 al 9 y letras A–F/a–f), que utiliza 0x o 0X como prefijo; por ejemplo, 0xFF, 0xff, 0XFf. El hexadecimal no distingue entre mayúsculas y minúsculas, por lo que todos estos ejemplos significan el mismo valor.
- Binario (dígitos 0 a 1), que utiliza el número 0 seguido de b o B como prefijo; por ejemplo, 0b10, 0B10.

No necesitarás realizar conversiones entre sistemas numéricos en el examen. tendrás que reconocer nizar valores literales válidos que se pueden asignar a números.

### Literales y el carácter de subrayado

Lo último que necesitas saber sobre los literales numéricos es que puedes tener guiones bajos en los números para que sean más fáciles de leer:

```
int millón1 = 1000000;
int millón2 = 1_000_000;
```

Preferiríamos leer el último porque los ceros no van juntos. Puede agregar guiones bajos en cualquier lugar excepto al principio de un literal, al final de un literal, justo antes de un punto decimal o justo después de un punto decimal. Incluso puedes colocar varios caracteres de subrayado uno al lado del otro, aunque no lo recomendamos.

Veamos algunos ejemplos:

```
doble notAtStart = _1000.00; doble          // NO COMPILA
notAtEnd = 1000.00_; doble                  // NO COMPILA
notByDecimal = 1000_.00; doble              // NO COMPILA
molestoPeroLegal = 1_00_0.0_0; // Feo, pero compila
doble realmente feo = 1__________2;         // También compila
```

## Usar tipos de referencia

Un tipo de referencia se refiere a un objeto (una instancia de una clase). A diferencia de los tipos primitivos que mantienen sus valores en la memoria donde se asigna la variable, las referencias no contienen el valor del objeto al que hacen referencia. En cambio, una referencia "apunta" a un objeto almacenando la memoria

Machine Translated by Google



dirección donde se encuentra el objeto, concepto denominado puntero . A diferencia de otros lenguajes, Java no permite saber cuál es la dirección de memoria física. Solo puedes usar la referencia para referirte al objeto.

Echemos un vistazo a algunos ejemplos que declaran e inicializan tipos de referencia. Supongamos que declaramos una referencia de tipo String:

```
Saludo de cadena ;
```

La variable de saludo es una referencia que solo puede apuntar a un objeto String . un valor es asignado a una referencia de una de dos maneras:

- Se puede asignar una referencia a otro objeto del mismo tipo o compatible.
- Se puede asignar una referencia a un nuevo objeto utilizando la nueva palabra clave.

Por ejemplo, la siguiente declaración asigna esta referencia a un nuevo objeto:

```
saludo = new String("¿Cómo estás?");
```

La referencia del saludo apunta a un nuevo objeto String , "¿Cómo estás?". La cuerda
El objeto no tiene nombre y sólo se puede acceder a él a través de la referencia correspondiente.

## Distinguir entre primitivos y tipos de referencia

Hay algunas diferencias importantes que debes conocer entre primitivos y tipos de referencia. Primero, observe que todos los tipos primitivos tienen nombres de tipo en minúsculas. Todas las clases que vienen con Java comienzan con mayúsculas. Aunque no es obligatorio, es una práctica estándar y también debes seguir esta convención para las clases que crees.

A continuación, los tipos de referencia se pueden utilizar para llamar a métodos, suponiendo que la referencia no sea nula. Los primitivos no tienen métodos declarados. En este ejemplo, podemos llamar a un método por referencia ya que es de tipo referencia. Se puede decir que la longitud es un método porque tiene () después. Vea si puede entender por qué el siguiente fragmento no se compila:

```
4: Referencia de cadena = "hola";
5: int len = referencia.longitud();
6: int malo = len.length(); // NO COMPILA
```

La línea 6 es un galimatías. No existen métodos en len porque es una primitiva int . Los primitivos no tienen métodos. Recuerde, una Cadena no es una primitiva, por lo que puede llamar a métodos como length() en una referencia de Cadena , como hicimos en la línea 5.

Finalmente, a los tipos de referencia se les puede asignar null, lo que significa que actualmente no hacen referencia a un objeto. Los tipos primitivos le darán un error de compilación si intenta asignarles un valor nulo. En este ejemplo, el valor no puede apuntar a nulo porque es de tipo int:

```
valor int = nulo;          // NO COMPILA
Nombre de cadena = nulo;
```

Pero, ¿qué pasa si no conoce el valor de un int y desea asignarlo a nulo? En eso
En este caso, debería utilizar una clase contenedora numérica, como Integer, en lugar de int.

Machine Translated by Google



# Crear clases contenedoras

Cada tipo primitivo tiene una clase contenedora, que es un tipo de objeto que corresponde al primitivo. La Tabla 1.7 enumera todas las clases contenedoras junto con cómo crearlas.

TABLA 1.7 Clases contenedoras

| Clase contenedora de tipo primitivo | | ¿ La clase contenedora hereda el número? | Ejemplo de creación |
|---|---|---|---|
| booleano | Booleano | No | Valor booleano de (verdadero) |
| byte | Byte | Sí | Byte.valorDe((byte) 1) |
| corto | Corto | Sí | Valor.cortoDe((corto) 1) |
| En t | Entero | Sí | Valor entero de (1) |
| largo | Largo | Sí | Valor largo de (1) |
| flotar | Flotar | Sí | Flotador.valorDe((flotante) 1.0) |
| doble | Doble | Sí | Doble.valorDe(1.0) |
| carbonizarse | Número de carácter | | Carácter.valorDe('c') |

También hay una variante valueOf() que convierte un String en la clase contenedora. Por ejemplo:

```
int primitivo = Integer.parseInt("123");
Envoltorio entero = Integer.valueOf("123");
```

La primera línea convierte una cadena en una primitiva int . El segundo convierte una cadena en una clase contenedora de enteros .

Todas las clases numéricas en la Tabla 1.7 extienden la clase Número , lo que significa que todas vienen con algunos métodos auxiliares útiles: byteValue(), shortValue(), intValue(), longValue(), floatValue() y doubleValue(). El booleano y el carácter Las clases contenedoras incluyen booleanValue() y charValue(), respectivamente.

Como probablemente habrás adivinado, estos métodos devuelven el valor primitivo de una instancia contenedora, en el tipo solicitado.

```
Manzana doble = Double.valueOf("200.99");
System.out.println(apple.byteValue());          // -56
System.out.println(apple.intValue());           // 200
System.out.println(apple.doubleValue()); // 200,99
```

Machine Translated by Google



Estos métodos auxiliares hacen todo lo posible para convertir valores, pero pueden provocar una pérdida de precisión. En el primer ejemplo, no hay 200 en bytes, por lo que pasa a -56. En el segundo ejemplo, el valor se trunca, lo que significa que se eliminan todos los números después del decimal. En el Capítulo 5, aplicamos autoboxing y unboxing para mostrar cuán fácil hace Java trabajar con valores primitivos y contenedores.

Algunas de las clases contenedoras contienen métodos auxiliares adicionales para trabajar con números. No es necesario que los memorice; puede asumir que cualquiera que le proporcionen es válido. Por ejemplo, Entero tiene:

- max(int num1, int num2), que devuelve el mayor de los dos números
- min(int num1, int num2), que devuelve el menor de los dos números
- suma(int num1, int num2), que suma los dos números

## Definición de bloques de texto

Anteriormente vimos una cadena simple con el valor "hola". ¿Qué pasa si queremos tener una cadena ? ¿Con algo más complicado? Por ejemplo, descubramos cómo crear una cadena.
con este valor:

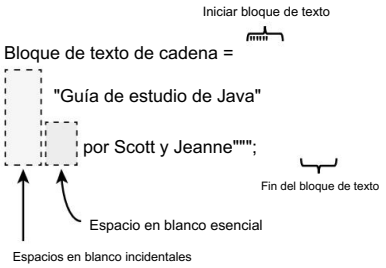
```
"Guía de estudio de Java"
    por Scott y Jeanne
```

Construir esto como String requiere dos cosas que aún no has aprendido. La sintaxis \" permite " en lugar de finalizar la denominan caracteres de cadena, y \n dice que desea una nueva línea. Tanto si dices que quieres uno de estos se escape porque la barra invertida proporciona un significado especial. Con estas dos nuevas habilidades, podemos escribir

```
String eyeTest = "\"Guía de estudio de Java\"\n por Scott & Jeanne";
```

Si bien esto funciona, es difícil de leer. Por suerte, Java tiene bloques de texto, también conocidos como cadenas multilínea. Consulte la Figura 1.3 para conocer el equivalente del bloque de texto.

FIGURA 1.3 Bloque de texto




Un bloque de texto comienza y termina con tres comillas dobles ("""") y el contenido no necesita para escapar. Esto es mucho más fácil de leer. Observe cómo el tipo sigue siendo String. Esto significa que los métodos que aprenderá en el Capítulo 4 para String funcionan tanto para un String normal como para un bloque de texto.

Machine Translated by Google



Es posible que hayas notado las palabras espacio en blanco incidental y esencial en la figura. ¿Qué es eso? Los espacios en blanco esenciales son parte de su cadena y son importantes para usted. Resulta que hay espacios en blanco incidentales para que el código sea más fácil de leer. Puede reformatear su código y cambiar la cantidad de espacios en blanco incidentales sin ningún impacto en su valor de cadena .

Imagine una línea vertical dibujada en el carácter que no es un espacio en blanco más a la izquierda en su bloque de texto. Todo lo que está a la izquierda es espacio en blanco incidental y todo lo que está a la derecha es espacio en blanco esencial. Probemos un ejemplo. ¿Cuántas líneas genera esto y cuántos espacios en blanco incidentales y esenciales comienzan cada línea?

```
14: Pirámide de cuerdas = """
15:       *
16: * *
17: * * *
18: """;
19: Sistema.out.print(pirámide);
```

Hay cuatro líneas de salida. Las líneas 15 a 17 tienen estrellas. La línea 18 es una línea sin ningún carácter. El triple de cierre " habría tenido que estar en la línea 17 si no quisiéramos esa línea en blanco. Aquí no hay espacios en blanco incidentales. Los "" " de cierre en la línea 18 son los caracteres más a la izquierda, por lo que la línea se dibuja en la posición más a la izquierda. La línea 15 tiene dos espacios en blanco esenciales para comenzar la línea y la línea 16 tiene uno. Ese espacio en blanco completa la línea dibujada para que coincida con la línea 18.

La Tabla 1.8 muestra algunas secuencias de formato especiales y compara cómo funcionan en una cadena normal y en un bloque de texto.

TABLA 1.8 Formato de bloque de texto

| Formato | Significado en cadena regular | Significado en bloque de texto |
|---|---|---|
| \" | " | " |
| \"""" | n/a – Inválido | """" |
| \"\"\" | """" | """" |
| Espacio (al final de la línea) | Espacio | ignorado |
| \s | Dos espacios (\s es un espacio y conserva el espacio inicial en la línea) | dos espacios |
| \ (al final de la línea) | n/a – Inválido | Omite nueva línea en esa línea |

Machine Translated by Google



Probemos algunos ejemplos. Primero, ¿ves por qué esto no se compila?

```
Bloque de cadena = """doe"""; // NO COMPILA
```

Los bloques de texto requieren un salto de línea después de la apertura """, lo que hace que éste no sea válido. Ahora vamos a Pruebe uno válido. ¿Cuántas líneas crees que hay en este bloque de texto?

```
Bloque de cuerda = """
    gama \
    ciervo""";
```

Solo uno. La salida es ciervo ya que \ le dice a Java que no agregue una nueva línea antes de ciervo. Intentemos determinar el número de líneas en otro bloque de texto:

```
Bloque de cadena = """
    = gama \n
    ciervo
    """;
```

Esta vez tenemos cuatro líneas. Dado que el bloque de texto tiene el cierre """ en una línea separada, tenemos tres líneas para las líneas en el bloque de texto más el \n explícito. Probemos uno más. ¿Qué crees que esto produce?

```
Bloque de cadena = """
    = "cierva\"\"\"
    \"ciervo\"""
    """;
System.out.print("*"+ bloque + "*");
```

La respuesta es

```
* "cierva"""
  "ciervo"""
*
```

Todos los \" escapan del ". Hay un espacio de espacio en blanco esencial en la cierva y el ciervo. líneas. Todos los demás espacios en blanco iniciales son espacios en blanco incidentales.

# Declaración de variables

Ya has visto algunas variables. Una variable es el nombre de una parte de la memoria que almacena datos. Cuando declara una variable, debe indicar el tipo de variable y darle un nombre. Darle un valor a una variable se llama inicializar una variable. Para inicializar una variable, simplemente escriba el nombre de la variable seguido de un signo igual, seguido del valor deseado. Este ejemplo muestra cómo declarar e inicializar una variable en una línea:

```
String zooName = "El mejor zoológico";
```

Machine Translated by Google



En las siguientes secciones, veremos cómo definir correctamente variables en una o varias líneas.

## Identificación de identificadores

Probablemente no le sorprenda que Java tenga reglas precisas sobre los nombres de los identificadores. Un identificador es el nombre de una variable, método, clase, interfaz o paquete. Afortunadamente, las reglas para los identificadores de variables se aplican a todos los demás tipos que puedes nombrar libremente.

Sólo hay cuatro reglas que recordar para los identificadores legales:

- Los identificadores deben comenzar con una letra, un símbolo de moneda o un símbolo _ . Los símbolos de moneda incluyen dólar ($), yuan (¥), euro (€), etc.
- Los identificadores pueden incluir números pero no comenzar con ellos.
- Un guión bajo único. _ no está permitido como identificador.
- No puede utilizar el mismo nombre que una palabra reservada de Java. Una palabra reservada es una palabra especial. palabra que Java ha mantenido a un lado para que no puedas utilizarla. Recuerde que Java distingue entre mayúsculas y minúsculas, por lo que puede utilizar versiones de las palabras clave que solo difieran en mayúsculas y minúsculas. Por favor, no lo hagas.

No se preocupe: no necesitará memorizar la lista completa de palabras reservadas. El examen solo le preguntará sobre los que se usan comúnmente, como clase y para. La Tabla 1.9 enumera todas las palabras reservadas en Java.

TABLA 1.9 Palabras reservadas

| | | | | |
|---|---|---|---|---|
| abstracto | afirmar | booleano | romper | byte |
| caso | atrapar | carbonizarse | clase | constante* |
| continuar | por defecto | hacer | doble | demás |
| enumeración | se extiende | final | finalmente | flotar |
| para | ir a* | si | implementos | importar |
| en vez de | En t | interfaz | largo | nativo |
| nuevo | paquete | privado | protegido | público |
| devolver | corto | estático | estrictofp | súper |
| cambiar | sincronizado | este | tirar | lanza |
| transitorio | intentar | vacío | volátil | mientras |

* Las palabras reservadas const y goto en realidad no se utilizan en Java. Están reservados para que personas que vienen de otros lenguajes de programación no los usen por accidente y, en teoría, en caso de que Java quiera usarlos algún día.

Machine Translated by Google



Hay otros nombres que no puedes usar. Por ejemplo, verdadero, falso y nulo son literales. valores, por lo que no pueden ser nombres de variables. Además, hay palabras clave contextuales como módulo en el Capítulo 12. Prepárese para la prueba de estas reglas. Los siguientes ejemplos son legales:

```
identificador largo ok;
float $OK2Identificador;
booleano _alsoOK1d3ntifi3r;
char __SStillOkperoKnotsonice$;
```

Estos ejemplos no son legales:

```
int 3DPointClass; // los identificadores no pueden comenzar con un número
byte hollywood@vine; // @ no es una letra, un dígito, $ o _
Cadena *$café; // * no es una letra, dígito, $ o doble public; // público es _
una palabra reservada
corto _; // no se permite un solo guión bajo
```

### camelCase y Snake_case

Aunque puedes hacer locuras con los nombres de los identificadores, no lo hagas. Java tiene convenciones para que el código sea legible y coherente. Por ejemplo, el caso camello tiene la primera letra de cada palabra en mayúscula. Los nombres de métodos y variables generalmente se escriben en mayúsculas y minúsculas, como toUpper(). Los nombres de clases e interfaces también se escriben en mayúsculas, con la primera letra en mayúscula, como ArrayList.

Otro estilo se llama estuche de serpiente. Simplemente utiliza un guión bajo (_) para separar palabras. Java generalmente usa mayúsculas de serpiente para constantes y valores de enumeración, como NUMBER_FLAGS.

El examen no siempre seguirá estas convenciones para hacer que las preguntas sobre identificadores sean más complicadas. Por el contrario, las preguntas sobre otros temas generalmente siguen convenciones estándar. Le recomendamos que siga estas convenciones en el trabajo.

## Declaración de múltiples variables

También puede declarar e inicializar múltiples variables en la misma declaración. ¿Cuántas variables crees que se declaran e inicializan en el siguiente ejemplo?

```
valla de arena vacía() {
    Cadena s1, s2;
    Cadena s3 = "sí", s4 = "no";
}
```

Machine Translated by Google



Se declararon cuatro variables String : s1, s2, s3 y s4. Puedes declarar muchas variables en la misma declaración siempre que sean todas del mismo tipo. También puede inicializar cualquiera o todos esos valores en línea. En el ejemplo anterior, tenemos dos variables inicializadas: s3 y s4. Las otras dos variables permanecen declaradas pero aún no inicializadas.

Aqui es donde se pone complicado. ¡Presta atención a las cosas complicadas! El examen intentará engañarte. Nuevamente, ¿cuántas variables crees que se declaran e inicializan en el siguiente código?

```
valla de pintura vacía() {
    int i1, i2, i3 = 0;
}
```

Como era de esperar, se declararon tres variables: i1, i2 e i3. Sin embargo, sólo uno de esos valores se inicializó: i3. Los otros dos permanecen declarados pero aún no inicializados. Ese es el truco. Cada fragmento separado por una coma es una pequeña declaración en sí misma. La inicialización de i3 solo se aplica a i3. No tiene nada que ver con i1 o i2 a pesar de estar en la misma declaración. Como verá en la siguiente sección, en realidad no puede usar i1 o i2 hasta que se hayan inicializado.

Otra forma en que el examen podría intentar engañarte es mostrarte un código como esta línea:

```
int num, valor de cadena; // NO COMPILA
```

Este código no se compila porque intenta declarar múltiples variables de diferentes tipos . en el mismo comunicado. El atajo para declarar múltiples variables en la misma declaración es legal solo cuando comparten un tipo.

Legal, válido y compilaciones son sinónimos en el mundo de los exámenes de Java. Intentamos utilizar toda la terminología que pueda encontrar en el examen.

Para asegurarse de comprender esto, vea si puede averiguar cuáles de las siguientes son declaraciones legales:

```
4: booleano b1, b2;
5: Cadena s1 = "1", s2;
6: doble d1, doble d2;
7: eres i1; eres i2;
8: int i3; i4;
```

Las líneas 4 y 5 son legales. Cada uno de ellos declara dos variables. La línea 4 no inicializa ninguna variable y la línea 5 inicializa solo una. La línea 7 también es legal. Aunque int aparece dos veces, cada una está en una declaración separada. Un punto y coma (;) separa las declaraciones en Java. Da la casualidad de que hay dos declaraciones completamente diferentes en la misma línea.

La línea 6 no es legal. Java no le permite declarar dos tipos diferentes en la misma declaración. ¡Espera un minuto! Las variables d1 y d2 son del mismo tipo. Ambos son de tipo doble. Aunque eso es cierto, todavía no está permitido. Si desea declarar varias variables en la misma declaración, deben compartir la misma declaración de tipo y no repetirla.

Machine Translated by Google



La línea 8 no es legal. Nuevamente tenemos dos afirmaciones completamente diferentes en la misma línea. La segunda de la línea 8 no es una declaración válida porque omite el tipo. Cuando vea un punto y coma colocado de manera extraña en el examen, imagine que el código está en líneas separadas y piense si el código se compila de esa manera. En este caso, las dos últimas líneas de código podrían reescribirse de la siguiente manera:

```
eres i1;
int i2;
ent i3;
i4;
```

Si observa la última línea por sí sola, podrá ver fácilmente que la declaración no es válida. Y sí, el examen realmente incluye varias declaraciones en la misma línea, en parte para intentar engañarlo y en parte para incluir más código en la pantalla. En el mundo real, limítese a una declaración por declaración y línea. Tus compañeros de equipo te agradecerán el código legible.

# Inicializando variables

Antes de poder utilizar una variable, necesita un valor. Algunos tipos de variables obtienen este valor establecido automáticamente y otros requieren que el programador lo especifique. En las siguientes secciones, analizamos las diferencias entre los valores predeterminados para las variables locales, de instancia y de clase.

## Creando variables locales

Una variable local es una variable definida dentro de un constructor, método o bloque inicializador. Para simplificar, nos centramos principalmente en las variables locales dentro de los métodos de esta sección, aunque las reglas para los demás son las mismas.

### Variables locales finales

La palabra clave final se puede aplicar a variables locales y equivale a declarar constantes en otros idiomas. Considere este ejemplo:

```
5: int final y = 10;
6: int x = 20;
7: y = x + 10; // NO COMPILA
```

Ambas variables están configuradas, pero y usa la palabra clave final . Por este motivo, la línea 7 genera un error del compilador ya que el valor no se puede modificar.

El modificador final también se puede aplicar a referencias de variables locales. El siguiente ejemplo utiliza un objeto de matriz int[] , que aprenderá en el Capítulo 4.

```
5: final int[] números favoritos = nuevo int[10];
6: Números favoritos[0] = 10;
```

Machine Translated by Google



```
7: Números favoritos[1] = 20;
8: Números favoritos = nulo; // NO COMPILA
```

Observe que podemos modificar el contenido o los datos de la matriz. El error del compilador no es hasta Línea 8, cuando intentamos cambiar el valor de los números favoritos de referencia.

## Variables locales no inicializadas

Las variables locales no tienen un valor predeterminado y deben inicializarse antes de su uso. Además, el compilador informará un error si intenta leer un valor no inicializado. Por ejemplo, el siguiente código genera un error del compilador:

```
4: público int no válido() {
5:     int y = 10;
6:     intx ;
7:     respuesta int = x + y; // NO COMPILA
8:     devolver respuesta;
9: }
```

La variable y se inicializa en 10. Por el contrario, x no se inicializa antes de usarse en la expresión de la línea 7 y el compilador genera un error. El compilador es lo suficientemente inteligente como para reconocer variables que se han inicializado después de su declaración pero antes de ser utilizadas.
He aquí un ejemplo:

```
público int válido() {
    int y = 10;
    intx ; // x se declara aquí
    x = 3; // x se inicializa aquí
    intz ; // z se declara aquí pero nunca se inicializa ni se utiliza
    respuesta int = x + y;
    devolver respuesta;
}
```

En este ejemplo, x se declara, inicializa y utiliza en líneas separadas. Además, z se declara pero nunca se usa, por lo que no es necesario inicializarlo.

El compilador también es lo suficientemente inteligente como para reconocer inicializaciones que son más complejas. En En este ejemplo, hay dos ramas de código:

```
public void findAnswer (verificación booleana) {
    respuesta interna;
    int otra respuesta;
    int sólo una rama;
    si (verificar) {
        sóloUnaRama = 1;
        respuesta = 1;
```

Machine Translated by Google



```
    } demás {
        respuesta = 2;
    }
    System.out.println(respuesta);
    System.out.println(onlyOneBranch); // NO COMPILA
}
```

La variable respuesta se inicializa en ambas ramas de la sentencia if , por lo que el compilador está perfectamente satisfecho. Sabe que, independientemente de si la verificación es verdadera o falsa, el valor de la respuesta se establecerá en algo antes de usarlo. La variable otherAnswer no se inicializa pero nunca se utiliza, y el compilador está igualmente contento. Recuerde, al compilador sólo le preocupa si intenta utilizar variables locales no inicializadas; no importa los que tu nunca usar.

La variable onlyOneBranch se inicializa solo si la verificación es verdadera. El compilador sabe que existe la posibilidad de que la verificación sea falsa, lo que da como resultado un código no inicializado y genera un error del compilador. Aprenderá más sobre la declaración if en el Capítulo 3, “Toma de decisiones”.

En el examen, tenga cuidado con cualquier variable local que esté declarada pero no inicializada en una sola línea. Este es un lugar común en el examen que podría resultar en una respuesta "No compila". Asegúrese de verificar que esté inicializado antes de usarlo en el examen.

## Pasar parámetros de constructor y método

Las variables pasadas a un constructor o método se denominan parámetros del constructor o parámetros del método, respectivamente. Estos parámetros son como variables locales que han sido preinicializadas. Las reglas para inicializar los parámetros del método y del constructor son las mismas, por lo que nos centraremos principalmente en los parámetros del método.

En el ejemplo anterior, check es un parámetro del método.

```
public void findAnswer ( verificación booleana) {}
```

Eche un vistazo al siguiente método checkAnswer() en la misma clase:

```
respuesta de verificación nula pública() {
    valor booleano;
    encontrarRespuesta(valor); // NO COMPILA
}
```

La llamada a findAnswer() no se compila porque intenta utilizar una variable que no está inicializada. Si bien la persona que llama a un método checkAnswer() debe preocuparse por la variable que se está inicializando, una vez dentro del método findAnswer(), podemos asumir que la variable local se ha inicializado con algún valor.

Machine Translated by Google



## Definición de variables de instancia y clase

Las variables que no son variables locales se definen como variables de instancia o como variables de clase. Una variable de instancia, a menudo llamada campo, es un valor definido dentro de una instancia específica de un objeto. Digamos que tenemos una clase Persona con un nombre de variable de instancia de tipo Cadena. Cada instancia de la clase tendría su propio valor de nombre, como Elysia o Sarah. Dos instancias pueden tener el mismo valor de nombre, pero cambiar el valor de una no modifica la otra.

Por otro lado, una variable de clase es aquella que se define en el nivel de clase y se comparte entre todas las instancias de la clase. Incluso puede ser accesible públicamente para clases fuera de la clase y no requiere una instancia para su uso. En nuestro ejemplo anterior de Persona , se podría usar una variable de clase compartida para representar la lista de personas que se encuentran hoy en el zoológico. Puedes decir que una variable es una variable de clase porque tiene la palabra clave static delante. Aprenderá sobre esto en el Capítulo 5. Por ahora, sólo sepa que una variable es una variable de clase si tiene la palabra clave estática en su declaración.

Las variables de instancia y clase no requieren que las inicialices. Tan pronto como declara estas variables, se les asigna un valor predeterminado. El compilador no sabe qué valor usar y por eso quiere el valor más simple que pueda darle al tipo: nulo para un objeto, cero para los tipos numéricos y falso para un booleano. No necesita saber el valor predeterminado de char, pero en caso de que tenga curiosidad, es '\u0000' (NUL).

## Inferir el tipo con var

Tiene la opción de utilizar la palabra clave var en lugar del tipo al declarar variables locales bajo ciertas condiciones. Para utilizar esta función, simplemente escriba var en lugar del tipo primitivo o de referencia. He aquí un ejemplo:

```
Zoológico de clase pública {
    public void whatTypeAmI() {
        var nombre = "Hola";
        tamaño var = 7;
    }
}
```

El nombre formal de esta característica es inferencia de tipo de variable local. Desarmemos eso. Primero viene la variable local. Esto significa exactamente lo que parece. Solo puede utilizar esta función para variables locales. El examen puede intentar engañarte con un código como este:

```
clase pública VarPalabra clave {
    var complicado = "Hola"; // NO COMPILA
}
```

Machine Translated by Google



¡Espera un minuto! Acabamos de aprender la diferencia entre variables de instancia y locales. La variable complicada es una variable de instancia. La inferencia de tipos de variables locales funciona con variables locales y no con variables de instancia.

## Tipo Inferencia de var

Ahora que comprende la parte de la variable local, es hora de continuar con la inferencia de tipos. medio. La buena noticia es que esto también significa lo que parece. Cuando escribe var, le está indicando al compilador que determine el tipo por usted. El compilador mira el código en la línea de la declaración y lo usa para inferir el tipo. Echale un vistazo a éste ejemplo:

```
7: reasignación de anulación pública () {
8:         número de variable = 7;
9:         número = 4;
10:        número = "cinco"; // NO COMPILA
11: }
```

En la línea 8, el compilador determina que queremos una variable int . En la línea 9, no tenemos problemas para asignarle un int diferente . En la línea 10, Java tiene un problema. Le hemos pedido que asigne una cadena a una variable int . Esto no esta permitido. Es equivalente a escribir esto:

```
número int = "cinco";
```

Si conoce un lenguaje como JavaScript, es posible que espere que var signifique una variable que puede adoptar cualquier tipo en tiempo de ejecución. En Java, var sigue siendo un tipo específico definido en tiempo de compilación. No cambia de tipo en tiempo de ejecución.

Para simplificar el análisis de var, asumiremos que una declaración de variable se completa en una sola línea. Podrías insertar un salto de línea entre el nombre de la variable y su valor de inicialización, como en el siguiente ejemplo:

```
7: Declaración pública de ruptura del vacío() {
8:         fue tonto
9:                 = 1;
10: }
```

Este ejemplo es válido y se compila, pero consideramos la declaración y la inicialización. de tontería que esté sucediendo en la misma línea.

## Ejemplos con var

¡Repasemos algunos escenarios más para que el examen no te engañe con este tema! ¿Crees que lo siguiente se compila?

```
3: public void haceThisCompile (verificación booleana) {
4:         pregunta var;
5:         pregunta = 1;
6:         fue respuesta;
```

Machine Translated by Google



```
7:      si (verificar) {
8:          respuesta = 2;
9:      } demás {
10:         respuesta = 3;
11:     }
12:     System.out.println(respuesta);
13: }
```

El código no se compila. Recuerde que para la inferencia de tipos de variables locales, el compilador solo mira la línea con la declaración. Dado que a las preguntas y respuestas no se les asignan valores en las líneas donde están definidas, el compilador no sabe qué hacer con ellas. Por este motivo, las líneas 4 y 6 no se compilan.

Puede que le resulte extraño ya que ambas ramas de if/else asignan un valor. Por desgracia, es no está en la misma línea que la declaración, por lo que no cuenta para var. Compare este comportamiento con lo que vimos hace poco cuando analizamos la bifurcación e inicialización de una variable local en nuestro método findAnswer() .

Ahora sabemos que el valor inicial utilizado para determinar el tipo debe ser parte del mismo declaración. ¿Puedes entender por qué estas dos declaraciones no se compilan?

```
4: dos tipos públicos vacíos () {
5:      int a, var b = 3; // NO COMPILA
6:      varn = cero;              // NO COMPILA
7: }
```

La línea 5 no funcionaría incluso si reemplazaras var con un tipo real. Todos los tipos declarados en una sola línea debe ser del mismo tipo y compartir la misma declaración. No podríamos escribir int a, int v = 3; cualquiera.

La línea 6 es una sola línea. Se le pide al compilador que infiera el tipo de nulo. Podría ser cualquier tipo de referencia. La única elección que puede hacer el compilador es Objeto. Sin embargo, es casi seguro que eso no es lo que pretendía el autor del código. Los diseñadores de Java decidieron que sería mejor no permitir var como nulo que tener que adivinar la intención.

Si bien una var no se puede inicializar con un valor nulo sin un tipo, se le puede reasignar un valor nulo después de declararla, siempre que el tipo de datos subyacente sea un tipo de referencia.

Probemos con otro ejemplo. ¿Ves por qué esto no se compila?

```
public int además(var a, var b) { // NO COMPILA
    devolver a + b;
}
```

En este ejemplo, a y b son parámetros del método. Estas no son variables locales. Estar en el Busque var utilizado con constructores, parámetros de método o variables de instancia. Usar var en uno de estos lugares es un buen truco de examen para ver si estás prestando atención. ¡Recuerde que var solo se usa para la inferencia de tipos de variables locales!

Machine Translated by Google



Hay una última regla que debes tener en cuenta: var no es una palabra reservada y se permite su uso como identificador. Se considera un nombre de tipo reservado. Un nombre de tipo reservado significa que no se puede utilizar para definir un tipo, como una clase, interfaz o enumeración. ¿Crees que esto es legal?

```
var paquete;

clase pública Var {
    var vacía pública() {
        var var = "var";
    }
    Var vacía pública() {
        Var var = nueva Var();
    }
}
```

Lo creas o no, este código se compila. Java distingue entre mayúsculas y minúsculas, por lo que Var no introduce ningún conflicto como nombre de clase. Nombrar una variable local var es legal. ¡No escribas un código que se parezca a este en tu trabajo! Pero comprender por qué funciona le ayudará a prepararse para cualquier pregunta complicada del examen que los creadores del examen puedan plantearle.

**Real World Scenario**

### var en el mundo real

La palabra clave var es excelente para los autores de exámenes porque facilita la escritura de código complicado. Cuando trabajas en un proyecto real, quieres que el código sea fácil de leer.

Una vez que empiece a tener un código similar al siguiente, es hora de considerar el uso de var:

```
PileOfPapersToFileInFilingCabinet pileOfPapersToFile = nuevo
    PileOfPapersToFileInFilingCabinet();
```

Puedes ver cómo acortar esto sería una mejora sin perder información:

```
var pileOfPapersToFile = nuevo PileOfPapersToFileInFilingCabinet();
```

Si alguna vez no está seguro de si es apropiado utilizar var, le recomendamos “Inferencia de tipo de variable local: pautas de estilo”, que está disponible en la siguiente ubicación.

https://openjdk.java.net/projects/amber/LVTIstyle.html

Machine Translated by Google



# Gestión del alcance variable

Has aprendido que las variables locales se declaran dentro de un bloque de código. ¿Cuántas variables ves que están incluidas en este método?

```
comer vacío público (int piezas de queso) {
    int bocadosDeQueso = 1;
}
```

Hay dos variables con alcance local. La variable BitesOfCheese se declara dentro del método. La variable PiecesOfCheese es un parámetro del método. Ninguna variable se puede utilizar fuera de donde está definida.

## Limitación del alcance

Las variables locales nunca pueden tener un alcance mayor que el método en el que están definidas. Sin embargo, pueden tener un alcance menor. Considere este ejemplo:

```
3: public void eatIfHungry (booleano hambriento) {
4:     si (hambriento) {
5:         int bocadosDeQueso = 1;
6:     } // BitesOfCheese queda fuera de alcance aquí
7:     System.out.println(bitesOfCheese); // NO COMPILA
8: }
```

La variable hambre tiene un alcance de todo el método, mientras que la variable muerdeQueso tiene un alcance menor. Solo está disponible para su uso en la declaración if porque se declara dentro de ella. Cuando vea un conjunto de llaves ({}) en el código, significa que ha ingresado un nuevo bloque de código. Cada bloque de código tiene su propio alcance. Cuando hay varios bloques, los combinas de adentro hacia afuera. En nuestro caso, el bloque de declaración if comienza en la línea 4 y termina en la línea 6. El bloque del método comienza en la línea 3 y termina en la línea 8.

Dado que BitesOfCheese se declara en un bloque de declaración if , el alcance se limita a ese bloque. Cuando el compilador llega a la línea 7, se queja de que no sabe nada sobre esto de BitesOfCheese y da un error.

Recuerde que los bloques pueden contener otros bloques. Estos bloques contenidos más pequeños pueden hacer referencia a variables definidas en los bloques de alcance más grandes, pero no al revés. He aquí un ejemplo:

```
16: vacío público eatIfHungry (booleano hambriento) {
17:     si (hambriento) {
18:         int bocadosDeQueso = 1;
19:         {
20:             var pequeñitoBit = verdadero;
21:             System.out.println(bitesOfCheese);
```

Machine Translated by Google



```
22:             }
23:         }
24:         System.out.println(teenyBit); // NO COMPILA
25: }
```

La variable definida en la línea 18 está dentro del alcance hasta que el bloque termina en la línea 23. Usarla en el bloque más pequeño de las líneas 19 a 22 está bien. La variable definida en la línea 20 sale del alcance en la línea 22. No se permite usarla en la línea 24.

## Alcance de seguimiento

El examen intentará engañarlo con varias preguntas sobre el alcance. Probablemente verá una pregunta que parece ser sobre algo complejo y no se puede compilar porque una de las variables está fuera de alcance.

Probemos uno. No se preocupe si aún no está familiarizado con las declaraciones if o los bucles while . No importa lo que haga el código ya que estamos hablando de alcance. Vea si puede averiguar en qué línea entra y sale cada una de las cinco variables locales del alcance:

```
11: public void eatMore (booleano hambriento, int cantidadDeComida) {
12:         int habitaciónEnBelly = 5;
13:         si (hambriento) {
14:             var tiempoParaComer = verdadero;
15:             mientras (cantidadDeComida > 0) {
dieciséis:                 int cantidad consumida = 2;
17:                 roomInBelly = roomInBelly - cantidad consumida;
18:                 cantidadDeComida = cantidadDeComida - cantidadConsumida;
19:             }
20:         }
21:         System.out.println(cantidadDeComida);
22: }
```

Este método compila. El primer paso para determinar el alcance es identificar los bloques. de código. En este caso son tres bloques. Puedes saber esto porque hay tres juegos de llaves. Comenzando desde el conjunto más interno, podemos ver dónde comienza y termina el bloque del bucle while . Repita este proceso a medida que avanzamos con el bloque de declaración if y el bloque de método. La Tabla 1.10 muestra los números de línea en los que comienza y termina cada bloque.

TABLA 1.10 Alcance del seguimiento por bloque

| Línea | Primera línea en bloque | Última línea en bloque |
|---|---|---|
| mientras | 15 | 19 |
| si | 13 | 20 |
| Método | 11 | 22 |

Machine Translated by Google



Ahora que sabemos dónde están los bloques, podemos observar el alcance de cada variable. hambriento y cantidad de comida son parámetros del método, por lo que están disponibles para todo el método. Esto significa que su alcance son las líneas 11 a 22. La variable roomInBelly entra en el alcance en la línea 12 porque ahí es donde se declara. Permanece dentro del alcance durante el resto del método y sale del alcance en la línea 22. La variable timeToEat entra en el alcance en la línea 14 donde se declara. Sale del alcance en la línea 20 donde termina el bloque if . Finalmente, la variable cantidadEaten entra en el alcance de la línea 16 donde se declara. Sale del alcance en la línea 19 donde termina el bloque while .

¡Querrás practicar mucho esta habilidad! La identificación de bloques y alcance variable debe ser algo natural para el examen. La buena noticia es que hay muchos ejemplos de código con los que practicar. Puede consultar cualquier ejemplo de código sobre cualquier tema de este libro y hacer coincidir las llaves.

## Aplicar alcance a las clases

Todo eso fue por variables locales. Afortunadamente, la regla para las variables de ejemplo es más sencilla: están disponibles tan pronto como se definen y duran toda la vida útil del objeto. La regla para las variables de clase, también conocidas como estáticas, es aún más sencilla: entran en el alcance cuando se declaran como los otros tipos de variables. Sin embargo, permanecen dentro del alcance durante toda la vida del programa.

Hagamos un ejemplo más para asegurarnos de que lo dominas. De nuevo, intenta calcular Determine el tipo de las cuatro variables y cuándo entran y salen del alcance.

```
1: ratón de clase pública {
2:         int estático final MAX_LENGTH = 5;
3:         longitud interna;
4:         crecimiento del vacío público (int pulgadas) {
5:             si (longitud < MAX_LENGTH) {
6:                 int nuevoTamaño = largo + pulgadas;
7:                 longitud = nuevoTamaño;
8:             }
9:         }
10: }
```

En esta clase tenemos una variable de clase, MAX_LENGTH; una variable de instancia, longitud; y dos variables locales, pulgadas y nuevoTamaño. La variable MAX_LENGTH es una variable de clase porque tiene la palabra clave estática en su declaración. En este caso, MAX_LENGTH entra en el ámbito de la línea 2 donde se declara. Permanece dentro del alcance hasta que finaliza el programa.

A continuación, la longitud entra en el alcance de la línea 3 donde se declara. Permanece dentro del alcance mientras exista este objeto Mouse . pulgadas entra en el alcance donde se declara en la línea 4. Sale del alcance al final del método en la línea 9. newSize entra en el alcance donde se declara en la línea 6. Dado que está definido dentro del bloque de declaración if , sale del alcance cuando ese bloque termina en la línea 8.

Machine Translated by Google



## Revisión del alcance

¿Tienes todo eso? Repasemos las reglas sobre el alcance:

- Variables locales: en el alcance desde la declaración hasta el final del bloque
- Parámetros del método: dentro del alcance durante la duración del método.
- Variables de instancia: en el alcance desde la declaración hasta que el objeto sea elegible para basura recopilación
- Variables de clase: en el alcance desde la declaración hasta que finaliza el programa

¿No estás seguro de qué es la recolección de basura? Relájese: esa es nuestra siguiente y última sección de este capítulo.

# Destruyendo objetos

Ahora que hemos jugado con nuestros objetos, es hora de guardarlos. Afortunadamente, la JVM se encarga de eso por usted. Java proporciona un recolector de basura para buscar automáticamente objetos que ya no son necesarios.

Recuerde, su código no es el único proceso que se ejecuta en su programa Java. El código Java existe dentro de una JVM, que incluye numerosos procesos independientes del código de su aplicación. Uno de los más importantes es un recolector de basura incorporado.

Todos los objetos Java se almacenan en el montón de la memoria de su programa. El montón, también conocido como almacén gratuito, representa un gran conjunto de memoria no utilizada asignada a su aplicación Java. Si su programa sigue creando instancias de objetos y dejándolos en el montón, eventualmente se quedará sin memoria y fallará. ¡Oh, no! Por suerte, la recogida de basura soluciona este problema. En las siguientes secciones, analizamos la recolección de basura.

## Comprender la recolección de basura

La recolección de basura se refiere al proceso de liberar automáticamente memoria en el montón eliminando objetos a los que ya no se puede acceder en su programa. Hay muchos algoritmos diferentes para la recolección de basura, pero no es necesario conocer ninguno de ellos para el examen.

Como desarrollador, la parte más interesante de la recolección de basura es determinar cuándo se puede recuperar la memoria que pertenece a un objeto. En Java y otros lenguajes, elegible para la recolección de basura se refiere al estado de un objeto en el que ya no se puede acceder a él en un programa y, por lo tanto, no se puede recolectar basura.

¿Significa esto que un objeto que sea elegible para la recolección de basura será recolectado inmediatamente? Definitivamente no. Cuando el objeto realmente se descarta no está bajo su control, pero para el examen, necesitará saber en todo momento qué objetos son elegibles para la recolección de basura.

Piense en la elegibilidad para la recolección de basura como si fuera a enviar un paquete. Puede tomar un artículo, sellarlo en una caja etiquetada y colocarlo en su buzón. Esto es análogo a hacer que un artículo sea elegible para la recolección de basura. Sin embargo, cuando el cartero viene a recogerlo, no está en tu

Machine Translated by Google



control. Por ejemplo, puede ser un feriado postal o podría haber un evento climático severo. Incluso puedes llamar a la oficina de correos y pedirles que vengan a recogerlo de inmediato, pero no hay forma de garantizar cuándo y si esto sucederá realmente. ¡Con suerte, llegarán antes de que su buzón se llene de paquetes!

Java incluye un método integrado para ayudar a admitir la recolección de basura donde puede sugerir esa ejecución de recolección de basura.

```
Sistema.gc();
```

Al igual que la oficina de correos, Java es libre de ignorarte. No se garantiza que este método haga nada.

## Seguimiento de elegibilidad

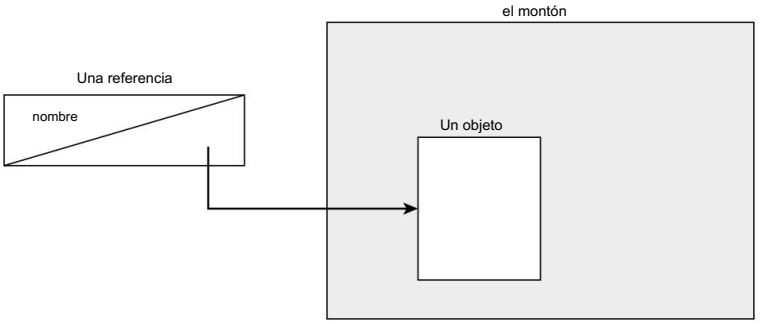
¿Cómo sabe la JVM cuándo un objeto es elegible para la recolección de basura? La JVM espera pacientemente y monitorea cada objeto hasta que determina que el código ya no necesita esa memoria. Un objeto permanecerá en el montón hasta que ya no sea accesible. Ya no se puede acceder a un objeto cuando ocurre una de dos situaciones:

- El objeto ya no tiene ninguna referencia que apunte a él.
- Todas las referencias al objeto han quedado fuera de alcance.

### Objetos versus referencias

No confunda una referencia con el objeto al que se refiere; son dos entidades diferentes. La referencia es una variable que tiene un nombre y se puede utilizar para acceder al contenido de un objeto. Una referencia puede asignarse a otra referencia, pasarse a un método o devolverse desde un método. Todas las referencias son del mismo tamaño, sin importar su tipo.

Un objeto se encuentra en el montón y no tiene nombre. Por lo tanto, no hay forma de acceder a un objeto excepto a través de una referencia. Los objetos vienen en diferentes formas y tamaños y consumen cantidades variables de memoria. Un objeto no se puede asignar a otro objeto y un objeto no se puede pasar a un método ni devolver desde un método. Es el objeto el que se recolecta la basura, no su referencia.



Machine Translated by Google



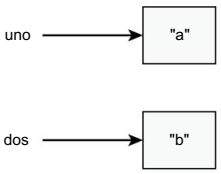
Comprender la diferencia entre una referencia y un objeto contribuye en gran medida a comprender la recolección de basura, el nuevo operador y muchas otras facetas del lenguaje Java.
Mire este código y vea si puede determinar cuándo cada objeto se vuelve elegible por primera vez para la recolección de basura:

```
1: alcance de clase pública {
2:     público estático vacío principal (String [] argumentos) {
3:         Cadena uno, dos;
4:         uno = nueva cadena ("a");
5:         dos = nueva cadena ("b");
6:         uno = dos;
7:         Cadena tres = uno;
8:         uno = nulo;
9:     } }
```

Cuando en el examen te hagan una pregunta sobre la recolección de basura, te recomendamos que dibujes lo que está pasando. Hay muchas cosas que seguir en la cabeza y es fácil cometer un error tonto al tratar de retenerlo todo en la memoria. Intentémoslo juntos ahora. En realidad. Consigue lápiz y papel. Bueno, espera.

¿Tienes ese papel? Bien, comencemos. En la línea 3, escriba uno y dos (solo las palabras; no se necesitan cuadros ni flechas ya que aún no se han agregado objetos al montón). En la línea 4, tenemos nuestro primer objeto. Dibuja un cuadro con la cadena "a" y dibuja una flecha desde la palabra uno hasta ese cuadro. La línea 5 es similar. Dibuja otro cuadro con la cadena "b" esta vez y una flecha de la palabra dos. En este punto, su trabajo debería verse como la Figura 1.4.

FIGURA 1.4 Su dibujo después de la línea 5



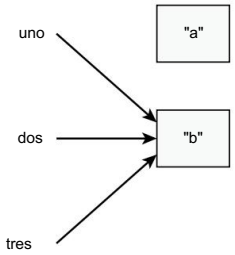

En la línea 6, la variable uno cambia para apuntar a "b". Borre o tache la flecha de uno y dibuje una nueva flecha de uno a "b". En la línea 7, tenemos una nueva variable, así que escribe la palabra tres y dibuja una flecha desde tres hasta "b". Observe que tres apunta a lo que uno está señalando en este momento y no a lo que apuntaba al principio. Por eso estás haciendo dibujos. Es fácil olvidar algo así. En este punto, su trabajo debería verse como la Figura 1.5.

Finalmente, tacha la línea entre uno y "b", ya que la línea 8 establece esta variable en nula.
Ahora, estábamos tratando de averiguar cuándo los objetos fueron elegibles por primera vez para la recolección de basura. En la línea 6, nos deshicimos de la única flecha que apuntaba a "a", haciendo que ese objeto fuera elegible para la recolección de basura. "b" tiene flechas que apuntan hacia él hasta que sale del alcance. Esto significa "b" no sale del alcance hasta el final del método en la línea 9.

Machine Translated by Google



FIGURA 1.5 Su dibujo después de la línea 7




### Formato de código en el examen

No todas las preguntas incluirán declaraciones e importaciones de paquetes. No se preocupe por la falta de declaraciones de paquetes o importaciones a menos que se le pregunte al respecto. Los siguientes son casos comunes en los que no es necesario verificar las importaciones:

- Código que comienza con un nombre de clase
- Código que comienza con una declaración de método
- Código que comienza con un fragmento de código que normalmente estaría dentro de una clase o método
- Código que tiene números de línea que no comienzan con 1

Verás código que no tiene método. Cuando esto suceda, suponga que cualquier código de plomería necesario, como el método main() y la definición de clase, se escribió correctamente. Sólo se le pregunta si la parte del código que se le muestra se compila cuando se coloca en un código circundante válido. Finalmente, recuerde que los espacios en blanco adicionales no importan en la sintaxis de Java. El examen puede utilizar cantidades variables de espacios en blanco para engañarlo.

# Resumen

Java comienza la ejecución del programa con un método main() . La firma más común para este método ejecutado desde la línea de comando es public static void main(String[] args).
Los argumentos se pasan después del nombre de la clase, como en java NameOfClass firstArgument.
Los argumentos se indexan comenzando por 0.

El código Java está organizado en carpetas llamadas paquetes. Para hacer referencia a clases en otros paquetes, utiliza una declaración de importación . Un comodín al final de una declaración de importación significa que desea importar todas las clases de ese paquete. No incluye paquetes que se encuentren dentro de ese. El paquete java.lang es especial porque no es necesario importarlo.

Machine Translated by Google



Para algunos elementos de clase, el orden importa dentro del archivo. La declaración del paquete es lo primero. si está presente. Luego vienen las declaraciones de importación , si están presentes. Luego viene la declaración de clase. Se permite que los campos y métodos estén en cualquier orden dentro de la clase.

Los tipos primitivos son los componentes básicos de los tipos de Java. Se ensamblan en tipos de referencia. Los tipos de referencia pueden tener métodos y se les puede asignar un valor nulo . Los literales numéricos pueden contener guiones bajos (_) siempre que no comiencen ni finalicen el literal y no estén al lado de un punto decimal (.). Las clases contenedoras son tipos de referencia y hay una para cada primitiva. Los bloques de texto permiten crear una cadena en varias líneas usando """.

Declarar una variable implica indicar el tipo de datos y darle un nombre a la variable. Las variables que representan campos en una clase se inicializan automáticamente a sus valores correspondientes 0, nulo o falso durante la creación de instancias del objeto. Las variables locales deben inicializarse específicamente antes de poder usarse. Los identificadores pueden contener letras, números, símbolos de moneda. Los identificadores no o _. pueden comenzar con números. Las declaraciones de variables locales pueden usar la var palabra clave en lugar del tipo real. Cuando se usa var, el tipo se establece una vez en el momento de la compilación y no cambia.

El alcance se refiere a esa parte del código donde se puede acceder a una variable. Hay tres tipos de variables en Java, según su alcance: variables de instancia, variables de clase y variables locales. Las variables de instancia son los campos no estáticos de su clase. Las variables de clase son los campos estáticos dentro de una clase. Las variables locales se declaran dentro de un constructor, método o bloque inicializador.

Los constructores crean objetos Java. Un constructor es un método que coincide con el nombre de la clase y omite el tipo de retorno. Cuando se crea una instancia de un objeto, los campos y bloques de código se inicializan primero. Luego se ejecuta el constructor. Finalmente, la recolección de basura se encarga de eliminar objetos de la memoria cuando ya no se pueden volver a utilizar. Un objeto se vuelve elegible para la recolección de basura cuando no hay más referencias a él o todas sus referencias han salido del alcance.

## Conceptos básicos del examen

Ser capaz de escribir código utilizando un método main() . Un método main() generalmente se escribe como public static void main(String[] args). Se hace referencia a los argumentos comenzando con args[0]. Acceder a un argumento que no se pasó hará que el código genere una excepción.

Comprender el efecto del uso de paquetes e importaciones. Los paquetes contienen clases de Java. Las clases se pueden importar por nombre de clase o comodín. Los comodines no miran a los subdirectorios. En caso de conflicto, las importaciones de nombres de clases tienen prioridad. Empaquetar e importar Las declaraciones son opcionales. Si están presentes, ambos van antes de la declaración de clase en ese orden.

Ser capaz de reconocer un constructor. Un constructor tiene el mismo nombre que la clase. Parece un método sin tipo de retorno.

Machine Translated by Google



Ser capaz de identificar declaraciones e inicializaciones legales e ilegales. Se pueden declarar e inicializar varias variables en la misma declaración cuando comparten un tipo. Las variables locales requieren una inicialización explícita; otros usan el valor predeterminado para ese tipo. Los identificadores pueden contener letras, números, símbolos de moneda o _, aunque no pueden comenzar con números.
Además, no puede definir un identificador que sea solo un carácter de guión bajo. Los literales pueden _. Numérico contener guiones bajos entre dos dígitos, como 1_000, pero no en otros lugares, como _100_.0_.

Comprenda cómo crear bloques de texto. Un bloque de texto comienza con """, la en la primera línea. En siguiente línea comienza con el contenido. La última línea termina con """. Si """ está en su propia línea, se incluye un salto de línea final.

Ser capaz de utilizar var correctamente. Se utiliza una var para una variable local. Una var se inicializa en la misma línea donde se declara y, si bien puede cambiar el valor, no puede cambiar el tipo. una var no se puede inicializar con un valor nulo sin un tipo, ni se puede utilizar en declaraciones de múltiples variables.

Ser capaz de determinar dónde entran y salen las variables del alcance. Todas las variables entran en el alcance cuando se declaran. Las variables locales quedan fuera de alcance cuando finaliza el bloque en el que están declaradas. Las variables de instancia quedan fuera de alcance cuando el objeto es elegible para la recolección de basura. Las variables de clase permanecen dentro del alcance mientras el programa se esté ejecutando.

Sepa cómo identificar cuándo un objeto es elegible para la recolección de basura. Dibuja un diagrama para realizar un seguimiento de las referencias y los objetos mientras rastreas el código. Cuando ninguna flecha apunta a un cuadro (objeto), es elegible para la recolección de basura.

Machine Translated by Google



# Preguntas de revisión

Las respuestas a las preguntas de revisión del capítulo se pueden encontrar en el Apéndice.

1. ¿ Cuáles de los siguientes son métodos de punto de entrada legales que se pueden ejecutar desde el comando ¿línea? (Elija todas las que correspondan).

   A. principal vacío estático privado (String[] args)

   B. principal final estática pública (String[] args)

   C. principal público vacío (cadena [] argumentos)

   D. principal público estático final vacío (String[] args)

   E. principal vacío estático público (String[] args)

   F. principal estática pública (String[] args)

2. ¿Qué opciones de respuesta representan el orden en que las siguientes afirmaciones se pueden ensamblar en un programa que se compilará exitosamente? (Elija todas las que correspondan).

   X: clase Conejo {}
   Y: importar java.util.*; Z:
   animales de paquete;

   A. X, Y, Z B.
   Y, Z, X C. Z,
   Y, X D. Y, X
   E. Z, X F.
   X, Z

   G. Ninguno de los anteriores

3. ¿Cuáles de las siguientes afirmaciones son verdaderas? (Elija todas las que correspondan).

```
conejito de clase pública {
    público estático vacío principal (Cadena [] x) {
        Bollo de conejito = nuevo conejito();
} }
```

   A. Bunny es una clase.

   B. bollo es una clase.

   C. main es una clase.

   D. Bunny es una referencia a un objeto.

   E. bun es una referencia a un objeto.

   F. main es una referencia a un objeto.

   G. El método main() no se ejecuta porque el nombre del parámetro es incorrecto.

Machine Translated by Google



4. ¿ Cuáles de los siguientes son identificadores Java válidos? (Elija todas las que correspondan).

A. _

B. _holaMundo$

C. verdadero

D. java.lang

E. Público

F. 1980_s

G. _P2_

5. ¿ Qué afirmaciones sobre el siguiente programa son correctas? (Elija todas las que correspondan).

```
2: clase pública Oso {
3:     privado Oso pandaBear;
4:     rugido de vacío privado (Oso b) {
5:         System.out.println("¡Rugido!");
6:         oso panda = b;
7:     }
8:     público estático vacío principal (String [] argumentos) {
9:         Oso osomarrón = nuevo Oso();
10:        Oso oso polar = nuevo Oso();
11:        osomarrón.roar(osopolar);
12:        oso polar = nulo;
13:        osomarrón = nulo;
14:        Sistema.gc(); } }
```

R. El objeto creado en la línea 9 es elegible para la recolección de basura después de la línea 13.

B. El objeto creado en la línea 9 es elegible para la recolección de basura después de la línea 14.

C. El objeto creado en la línea 10 es elegible para la recolección de basura después de la línea 12.

D. El objeto creado en la línea 10 es elegible para la recolección de basura después de la línea 13.

E. Se garantiza que se ejecutará la recolección de basura.

F. La recolección de basura puede ejecutarse o no.

G. El código no se compila.

6. Suponiendo que se compila la siguiente clase, ¿cuántas variables definidas en la clase o método están dentro del alcance en la línea marcada en la línea 14?

```
1: camello de clase pública {
2:     { int pelos = 3_000_0; }
3:     largo agua, aire=2;
4:     booleano twoHumps = verdadero;
5:     escupir vacío público (distancia de flotación) {
6:         var ruta = "";
```

Machine Translated by Google



```
7:          { dientes dobles = 32 + distancia++; }
8:          mientras(agua > 0) {
9:              int edad = dos jorobas? 1: 2;
10:             corto i=-1;
11:             para(i=0; i<10; i++) {
12:                 var Privado = 2;
13:             }
14:             // ALCANCE
15:         }
dieciséis:  }
17: }
```

R.2 _

B.3 _

C.4 _

D.5 _

E.6 _

F.7 _

G. Ninguno de los anteriores

7. ¿Cuáles son ciertas sobre este código? (Elija todas las que correspondan).

```
fregadero de cocina de clase pública {
    privado int numForks;

    público estático vacío principal (String [] argumentos) {
        int numCuchillos;
        Sistema.out.print("""
            "# tenedores = " + numForks +
            " # cuchillos = " + numCuchillos +
            # tazas = 0""");
    }
}
```

A. El resultado incluye: # bifurcaciones = 0.

B. La salida incluye: # cuchillos = 0.

C. La salida incluye: # tazas = 0.

D. El resultado incluye una línea en blanco.

E. El resultado incluye una o más líneas que comienzan con espacios en blanco.

F. El código no se compila.

Machine Translated by Google



8. ¿ Cuál de los siguientes fragmentos de código sobre var se compila sin problemas cuando se usa en un ¿método? (Elija todas las que correspondan).

A. var primavera = nulo; B. var caída

= "hojas"; C. var tarde = 2; tarde = nulo;

D. var noche = Integer.valueOf(3); E. var día = 1/0; F. var

invierno = 12, frío; G. var otoño = 2, otoño = 2; H. var

mañana = ""; mañana = nulo;

9. ¿Cuáles de las siguientes son correctas? (Elija todas las que correspondan).

R. Una variable de instancia de tipo float tiene por defecto 0.

B. Una variable de instancia de tipo char tiene como valor predeterminado nulo.

C. Una variable local de tipo doble tiene por defecto 0.0.

D. Una variable local de tipo int tiene el valor predeterminado nulo.

E. Una variable de clase de tipo Cadena tiene por defecto nulo.

F. Una variable de clase de tipo String por defecto es la cadena vacía "".

G. Ninguna de las anteriores.

10. ¿Cuál de las siguientes expresiones, cuando se inserta de forma independiente en la línea en blanco, permite que el código se compile? (Elija todas las que correspondan).

```
public void printMagicData() { var magia
    =; System.out.println(magia);_____

}
```

A. 3_1 B.

1_329_.0

C.3_13.0_D.5_291._2 _ _ _

E. 2_234.0_0 F.

9___6

_1_3_5_0 _

11. Dados los dos archivos de clase siguientes, ¿cuál es la cantidad máxima de importaciones que se pueden eliminar y mantener el código compilado?

```
// Paquete
Water.java acuario;
clase pública Agua { }
```

Machine Translated by Google



```
// Paquete
Tank.java acuario;
importar java.lang.*;
importar java.lang.System;
importar acuario.Agua;
importar acuario.*;
tanque de clase pública
    { impresión vacía pública (agua agua)
    { System.out.println (agua); } }
```

R. 0

B.1 _

c.2 _

D.3 _

E.4 _

F. No compila

12. ¿Qué afirmaciones sobre la siguiente clase son correctas? (Elija todas las que correspondan).

```
1: clase pública Pez Payaso { int
2:      branquias = 0, doble peso = 2; { int aletas
3:      = branquias; } impresión
4:      vacía (int longitud = 3) {
5:          System.out.println(branquias);
6:          System.out.println(peso);
7:          System.out.println(aletas);
8:          System.out.println(longitud);
9: } }
```

A. La línea 2 genera un error del compilador.

B. La línea 3 genera un error del compilador.

C. La línea 4 genera un error del compilador.

D. La línea 7 genera un error del compilador.

E. El código imprime 0.

F. El código imprime 2.0.

G. El código imprime 2.

H. El código imprime 3.

Machine Translated by Google



13. Dadas las siguientes clases, ¿cuál de los siguientes fragmentos se puede insertar de forma independiente en lugar de INSERTAR IMPORTACIONES AQUÍ y compilar el código? (Elija todas las que correspondan).

```
paquete de acuario;
clase pública Agua
    { booleano salado = falso;
}

paquete acuario.gelatinas; agua de
clase pública {
    booleano salado = verdadero;
}

empleado de paquete;
INSERTAR IMPORTACIONES AQUÍ
clase pública WaterFiller {
    Agua agua;
}
```

A. acuario de importación.*; B.
importar acuario. Agua;
importar jaleas.de acuario.*;
C. acuario de importación.*;
importar acuario.gelatinas.Agua; D. acuario de
importación.*;
importar jaleas.de acuario.*;
E. importar acuario. Agua; importar
acuario.gelatinas.Agua; F. Ninguna de estas
importaciones puede hacer que el código se compile.

14. ¿Cuáles de las siguientes afirmaciones sobre el fragmento de código son verdaderas? (Elija todas las que correspondan).

```
3: número corto de mascotas
= 5L; 4: int numGranos = 2,0; 5:
Nombre de cadena = "Desaliñado"; 6:
int d = numPets.length(); 7: int e =
numGranos.longitud; 8: int f =
nombre.longitud();
```

Machine Translated by Google



A. La línea 3 genera un error del compilador.

B. La línea 4 genera un error del compilador.

C. La línea 5 genera un error del compilador.

D. La línea 6 genera un error del compilador.

E. La línea 7 genera un error del compilador.

F. La línea 8 genera un error del compilador.

15. ¿Cuáles de las siguientes afirmaciones sobre la recolección de basura son correctas? (Elegir todos eso aplica.)

R. Se garantiza que llamar a System.gc() liberará memoria al destruir objetos elegibles para la recolección de basura.

B. La recolección de basura se realiza según un horario establecido.

C. La recolección de basura permite a la JVM recuperar memoria para otros objetos.

D. La recolección de basura se ejecuta cuando su programa ha consumido la mitad de la memoria disponible.

E. Un objeto puede ser elegible para la recolección de basura pero nunca eliminarse del montón.

F. Un objeto es elegible para la recolección de basura una vez que no se puede acceder a ninguna referencia a él en el programa.

G. Marcar una variable como final significa que su objeto asociado nunca será recolectado como basura.

16. ¿Cuáles son ciertas sobre este código? (Elija todas las que correspondan).

```
var bloque = """
    ardilla \s
    paloma \
    termita""";
System.out.print(en bloques);
```

A. Genera dos líneas.

B. Genera tres líneas.

C. Genera cuatro líneas.

D. Hay una línea con espacios en blanco al final.

E. Hay dos líneas con espacios en blanco al final.

F. Si sangramos cinco caracteres en cada línea, cambiaría la salida.

17. ¿Qué líneas imprime el siguiente programa? (Elija todas las que correspondan).

```
1: botella de agua de clase pública {
2:     marca privada de cuerdas;
3:     booleano privado vacío;
4:     código flotante estático público;
5:     público estático vacío principal (String [] argumentos) {
6:         Botella de agua wb = nuevaBotella de agua();
```

Machine Translated by Google



```
7:      System.out.println("Vacío = " + wb.vacío);
8:      System.out.println("Marca = " + wb.marca);
9:      System.out.println("Código = } } " + código);
10:
```

A. La línea 8 genera un error del compilador.

B. La línea 9 genera un error del compilador.

C. Vacío =

D. Vacío = falso

E. Marca =

F. Marca = nulo

Código G = 0.0

Código H = 0f

18. ¿Cuáles de las siguientes afirmaciones sobre var son verdaderas? (Elija todas las que correspondan).

R. Se puede utilizar una var como parámetro del constructor.

B. El tipo de var se conoce en el momento de la compilación.

C. Una var no se puede utilizar como variable de instancia.

D. Una var se puede utilizar en una declaración de asignación de múltiples variables.

E. El valor de una var no puede cambiar en tiempo de ejecución.

F. El tipo de var no puede cambiar en tiempo de ejecución.

G. La palabra var es una palabra reservada en Java.

19. ¿Cuáles son ciertas sobre el siguiente código? (Elija todas las que correspondan).

```
var num1 = Long.parseLong("100");
var num2 = Long.valueOf("100");
System.out.println(Long.max(num1, num2));
```

R. La salida es 100.

B. La salida es 200.

C. El código no se compila.

D. num1 es una primitiva.

E. num2 es una primitiva.

20. ¿Qué afirmaciones sobre la siguiente clase son correctas? (Elija todas las que correspondan).

```
1: clase pública PoliceBox {
2:     Color de cuerda;
3:     larga edad;
4:     caja de policía vacía pública () {
5:         color = "azul";
6:         edad = 1200;
```

Machine Translated by Google



```
7:      }
8:      principal vacío estático público (cadena [] tiempo) {
9:          var p = nueva PoliceBox();
10:         var q = nueva CajaPolicía();
11:         p.color = "verde";
12:         edad de página = 1400;
13:         pag = q;
14:         System.out.println("Q1="+q.color);
15:         System.out.println("Q2="+q.age);
dieciséis:  System.out.println("P1="+p.color);
17:         System.out.println("P2="+p.age);
18: } }
```

A. Imprime Q1=azul.

B. Imprime Q2=1200.

C. Imprime P1=nulo.

D. Imprime P2=1400.

E. La línea 4 no se compila.

F. La línea 12 no compila.

G. La línea 13 no compila.

H. Ninguna de las anteriores.

21. ¿Cuál es el resultado de ejecutar la siguiente clase?

```
1: salmón de clase pública {
2:      recuento int;
3:      { System.out.print(cuenta+"-"); }
4:      { contar++; }
5:      Salmón público() {
6:          contar = 4;
7:          System.out.print(2+"-");
8:      }
9:      público estático vacío principal (String [] argumentos) {
10:         System.out.print(7+"-");
11:         var s = nuevo Salmón();
12:         System.out.print(s.count+"-"); } }
```

Machine Translated by Google



A.7-0-2-1- _

B.7-0-1- _

C.0-7-2-1- _

D.7-0-2-4- _

E. 0-7-1-

F. La clase no se compila debido a la línea 3.

G. La clase no se compila debido a la línea 4.

H. Ninguna de las anteriores.

22. Dada la siguiente clase, ¿cuál de las siguientes líneas de código puede reemplazar de forma independiente INSERTAR CÓDIGO AQUÍ para compilar el código? (Elija todas las que correspondan).

```
Precio de clase pública {
    admisión pública nula() {
        INSERTAR EL CÓDIGO AQUÍ
        System.out.print(cantidad);
        } }
```

A.int Cantidad = 0b11;

B. cantidad interna = 9L;

C. cantidad int = 0xE;

D. cantidad int = 1_2.0;

E. cantidad doble = 1_0_.0;

F. cantidad int = 0b101;

G. doble cantidad = 9_2.1_2;

H. doble cantidad = 1_2_.0_0;

23. ¿Qué afirmaciones sobre la siguiente clase son verdaderas? (Elija todas las que correspondan).

```
1: río de clase pública {
2:     int Profundidad = 1;
3:     temperatura de flotación = 50,0;
4:     flujo vacío público () {
5:         para (int i = 0; i < 1; i++) {
6:             profundidad interna = 2;
7:             profundidad++;
8:             temperatura--;
9:         }
```

Machine Translated by Google



```
10:         System.out.println(profundidad);
11:         System.out.println(temp); }
12:     principal vacío estático público (cadena... s) {
13:         nuevo río().flujo();
14: } }
```

A. La línea 3 genera un error del compilador.

B. La línea 6 genera un error del compilador.

C. La línea 7 genera un error del compilador.

D. La línea 10 genera un error del compilador.

E. El programa imprime 3 en la línea 10.

F. El programa imprime 4 en la línea 10.

G. El programa imprime 50.0 en la línea 11.

H. El programa imprime 49.0 en la línea 11.

Machine Translated by Google

Capítulo

2

# Operadores

OBJETIVOS DEL EXAMEN OCP CUBIERTOS EN ESTE CAPÍTULO:

Manejo de valores de fecha, hora, texto, numéricos y booleanos.

- Utilice primitivas y clases contenedoras, incluida la API de matemáticas, paréntesis, promoción de tipos y conversión para evaluar expresiones aritméticas y booleanas.

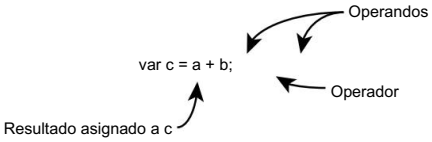
Machine Translated by Google

El capítulo anterior habló mucho sobre la definición de variables, pero ¿qué se puede hacer con una variable una vez creada? Este capítulo presenta los operadores y muestra cómo puede usarlos para combinar variables existentes y crear nuevos valores. Le muestra cómo aplicar operadores a varios tipos de datos primitivos, incluida la introducción de operadores que se pueden aplicar a objetos.

# Comprender los operadores de Java

Antes de entrar en lo divertido, cubramos un poco de terminología. Un operador Java es un símbolo especial que se puede aplicar a un conjunto de variables, valores o literales (denominados operandos). y eso devuelve un resultado. El término operando, que utilizamos a lo largo de este capítulo, se refiere al valor o variable al que se aplica el operador. La Figura 2.1 muestra la anatomía de una operación Java.

FIGURA 2.1 Operación Java




El resultado de la operación se denomina simplemente resultado. La figura 2.1 en realidad contiene una segunda operación, en la que el operador de asignación (=) se utiliza para almacenar el resultado en la variable c.

Estamos seguros de que has estado usando los operadores de suma (+) y resta (-) desde que eras un niño pequeño. Java admite muchos otros operadores que necesita conocer para el examen. Si bien muchos deberían ser revisados por usted, algunos (como los operadores de asignación compuesta) pueden ser nuevos para usted.

## Tipos de operadores

Java admite tres tipos de operadores: unario, binario y ternario. Este tipo de operadores se pueden aplicar a uno, dos o tres operandos, respectivamente. Para el examen necesitas saber

Machine Translated by Google



un subconjunto específico de operadores Java, cómo aplicarlos y el orden en el que deben aplicarse.

Los operadores Java no se evalúan necesariamente de izquierda a derecha. En el siguiente ejemplo, la segunda expresión se evalúa de derecha a izquierda, dados los operadores específicos involucrados:

```
galletas enteras = 4;
doble recompensa = 3 + 2 * --cookies;
System.out.print("El animal del zoológico recibe: "+recompensa+" puntos de recompensa");
```

En este ejemplo, primero disminuye las cookies a 3, luego multiplica el valor resultante por 2 y finalmente suma 3. Luego, el valor se promociona automáticamente de 9 a 9,0 y se asigna a la recompensa. Los valores finales de recompensa y cookies son 9,0 y 3, respectivamente, y se imprime lo siguiente:

```
El animal del zoológico recibe: 9,0 puntos de recompensa
```

Si no seguiste esa evaluación, no te preocupes. Al final de este capítulo, resolviendo el problema Problemas como este deberían ser algo natural.

## Precedencia del operador

Al leer un libro o un periódico, algunos lenguajes escritos se evalúan de izquierda a derecha, mientras que otros se evalúan de derecha a izquierda. En matemáticas, ciertos operadores pueden anular a otros operadores y ser evaluados primero. Determinar qué operadores se evalúan en qué orden se denomina precedencia de operadores. De esta manera, Java sigue más de cerca las reglas de las matemáticas. Considere la siguiente expresión:

```
var perímetro = 2 * altura + 2 * longitud;
```

Apliquemos algunos paréntesis opcionales para demostrar cómo el compilador evalúa esto. declaración:

```
var perímetro = ((2 * altura) + (2 * longitud));
```

El operador de multiplicación (*) tiene mayor prioridad que el operador de suma (+), por lo que la altura y la longitud se multiplican por 2 antes de sumarse. El operador de asignación (=) tiene el orden de precedencia más bajo, por lo que la asignación a la variable de perímetro se realiza en último lugar.

A menos que se anulen con paréntesis, los operadores de Java siguen el orden de operación, enumerado en la Tabla 2.1, por orden decreciente de precedencia de operadores. Si dos operadores tienen el mismo nivel de precedencia, Java garantiza una evaluación de izquierda a derecha para la mayoría de los operadores distintos de los marcados en la tabla.

Machine Translated by Google



TABLA 2.1 Orden de precedencia de operadores

| Operador | Símbolos y ejemplos | Evaluación |
|---|---|---|
| Operadores postunarios | expresión ++, expresión-- | De izquierda a derecha |
| Operadores preunarios | ++expresión, --expresión | De izquierda a derecha |
| Otros operadores unarios | -, !, ~, +, (tipo) | De derecha a izquierda |
| Elenco | (Tipo)referencia | De derecha a izquierda |
| Multiplicación/división/módulo *, /, % | | De izquierda a derecha |
| Suma resta | +, - | De izquierda a derecha |
| Operadores de turno | <<, >>, >>> | De izquierda a derecha |
| Operadores relacionales | <, >, <=, >=, instancia de | De izquierda a derecha |
| Igual a/no igual a | ==, != | De izquierda a derecha |
| Y lógico | & | De izquierda a derecha |
| O exclusivo lógico | ^ | De izquierda a derecha |
| O inclusivo lógico | \| | De izquierda a derecha |
| Condicional Y | && | De izquierda a derecha |
| Condicional O | \|\| | De izquierda a derecha |
| Operadores ternarios | expresión booleana? expresión1: expresión2 | De derecha a izquierda |
| Operadores de Asignación | =, +=, -=, *=, /=, %=, &=, ^=, \|=, <<=, >>=, >>>= | De derecha a izquierda |
| Operador de flecha | -> | De derecha a izquierda |

Recomendamos tener la Tabla 2.1 a mano a lo largo de este capítulo. Para el examen, es necesario memorizar el orden de precedencia en esta tabla. Tenga en cuenta que no se le realizarán pruebas con algunos operadores, como los operadores de turno, aunque le recomendamos que esté al tanto de su existencia.

Machine Translated by Google

El operador de flecha (->), a veces llamado función de flecha u operador lambda, es un operador binario que representa una relación entre dos operandos. Aunque no cubriremos el operador de flecha en este capítulo, lo verá utilizado en expresiones de cambio en el Capítulo 3, “Toma de decisiones”, y en expresiones lambda a partir del Capítulo 8, “Lambdas e interfaces funcionales”.

# Aplicar operadores unarios

Por definición, un operador unario es aquel que requiere exactamente un operando o variable para funcionar. Como se muestra en la Tabla 2.2, a menudo realizan tareas simples, como aumentar una variable numérica en uno o negar un valor booleano .

TABLA 2.2 Operadores unarios

| Operador | Ejemplos | Descripción |
|---|---|---|
| Complemento lógico | !a | Invierte el valor lógico de un booleano |
| Complemento bit a bit | ~b | Invierte todos los 0 y 1 en un número |
| Más | +c | Indica que un número es positivo, aunque en Java se supone que los números son positivos a menos que vayan acompañados de un operador unario negativo. |
| Negación o menos | -d | Indica que un número literal es negativo o niega una expresión |
| Incremento | ++ y f++ | Incrementa un valor en 1 |
| Decremento | --F h-- | Disminuye un valor en 1 |
| Elenco | (Cadena)i | Lanza un valor a un tipo específico |

Aunque la Tabla 2.2 incluye el operador de conversión, posponemos la discusión sobre la conversión hasta la sección "Asignación de valores" más adelante en este capítulo, ya que es allí donde se usa comúnmente.

Machine Translated by Google



## Operadores de complemento y negación

Dado que vamos a trabajar con muchos operadores numéricos en este capítulo, eliminemos primero el booleano . El operador de complemento lógico (!) invierte el valor de una expresión booleana . Por ejemplo, si el valor es verdadero, se convertirá en falso y viceversa. Para ilustrar esto, compare los resultados de las siguientes declaraciones:

```
booleano isAnimalAsleep = false;
System.out.print(isAnimalAsleep); // FALSO
estáAnimalDormido = !isAnimalDormido;
System.out.print(isAnimalAsleep); // verdadero
```

Para el examen, también necesita conocer el operador de complemento bit a bit (~), que invierte todos los ceros y unos de un número. Solo se puede aplicar a tipos numéricos enteros como byte, short, char, int y long. Probemos un ejemplo. Para simplificar, solo mostramos los últimos cuatro bits (en lugar de los 32 bits).

```
valor entero = 3;                    // Almacenado como 0011
complemento int = ~valor;            // Almacenado como 1100
System.out.println(valor);           // 3
System.out.println(complemento); // -4
```

¡Relajarse! No necesitas saber cómo hacer aritmética de bits complicada en el examen, siempre y cuando recuerdes esta regla: para encontrar el complemento bit a bit de un número, multiplícalo por menos uno y luego réstale uno.

```
System.out.println(-1*valor - 1); // -4
System.out.println(-1*complemento - 1); // 3
```

Pasando a operadores más comunes, el operador de negación (-) invierte el signo de una expresión numérica, como se muestra en estas declaraciones:

```
doble temperatura del zoológico = 1,21;
System.out.println(zooTemperature); // 1.21
temperaturazoo = -temperaturazoo;
System.out.println(zooTemperature); // -1.21
temperaturazoo = -(-temperaturazoo);
System.out.println(zooTemperature); // -1.21
```

Observe que en el último ejemplo usamos paréntesis, (), para el operador de negación, -, para aplicar la negación dos veces. Si en lugar de eso hubiéramos escrito, se habría interpretado como el operador de decremento y se habría impreso -2,21. Verá más de ese operador de decremento en breve.

Según la descripción, puede resultar obvio que algunos operadores requieren que la variable o expresión sobre la que actúan sea de un tipo específico. Por ejemplo, no se puede aplicar un operador de negación (-) a una expresión booleana , ni se puede aplicar un operador de complemento lógico (!) a una expresión numérica. Tenga cuidado con las preguntas del examen que intentan hacer esto,

Machine Translated by Google



ya que hacen que el código no se pueda compilar. Por ejemplo, ninguna de las siguientes líneas de código se compilará:

```
int pelícano = !5; // NO COMPILA
pingüino booleano = -verdadero; // NO COMPILA
pavo real booleano = !0; // NO COMPILA
```

La primera declaración no se compilará porque en Java no se puede realizar una inversión lógica de un valor numérico. La segunda declaración no se compila porque no se puede negar numéricamente un valor booleano ; necesitas usar el operador lógico inverso. Finalmente, la última declaración no se compila porque no se puede tomar el complemento lógico de un valor numérico ni se puede asignar un número entero a una variable booleana .

Esté atento a las preguntas del examen que utilizan valores numéricos (como 0 o 1) con expresiones booleanas . A diferencia de otros lenguajes de programación, en Java, 1 y verdadero no están relacionados de ninguna manera, al igual que 0 y falso no están relacionados.

## Operadores de incremento y decremento

Los operadores de incremento y decremento, ++ y --, respectivamente, se pueden aplicar a variables numéricas y tienen un alto orden de precedencia en comparación con los operadores binarios. En otras palabras, suelen aplicarse primero en una expresión.

Los operadores de incremento y decremento requieren especial cuidado porque el orden en el que se adjuntan a su variable asociada puede marcar la diferencia en cómo se procesa una expresión. La tabla 2.3 enumera cada uno de estos operadores.

TABLA 2.3 Operadores de incremento y decremento

| Operador | Descripción de ejemplo | |
|---|---|---|
| Pre-incremento | ++w | Aumenta el valor en 1 y devuelve el nuevo valor. |
| Pre-decremento | --X | Disminuye el valor en 1 y devuelve el nuevo valor. |
| Post-incremento | y++ | Aumenta el valor en 1 y devuelve el valor original. |
| Post-decremento | Con-- | Disminuye el valor en 1 y devuelve el valor original. |

El siguiente fragmento de código ilustra esta distinción:

```
int parkAsistencia = 0;
System.out.println(asistencia al parque); // 0
System.out.println(++parqueAsistencia); // 1
```

Machine Translated by Google



```
System.out.println(asistencia al parque); // 1
System.out.println(asistencia al parque--); // 1
System.out.println(asistencia al parque); // 0
```

El primer operador de incremento previo actualiza el valor de asistencia al parque y las salidas. el nuevo valor de 1. El siguiente operador posterior al decremento también actualiza el valor de parkAttendance pero genera el valor antes de que ocurra el decremento.

Para el examen, es fundamental que conozca la diferencia entre expresiones como parkAttendance++ y ++parkAttendance. Los operadores de incremento y decremento estarán en varias preguntas y la confusión sobre qué valor se devuelve podría hacer que pierda muchos puntos en el examen.

# Trabajar con binario Operadores aritméticos

A continuación, pasamos a los operadores que toman dos operandos, llamados operadores binarios. Los operadores binarios son, con diferencia, los operadores más comunes en el lenguaje Java. Se pueden utilizar para realizar operaciones matemáticas con variables, crear expresiones lógicas y realizar asignaciones de variables básicas. Los operadores binarios suelen combinarse en expresiones complejas con otros operadores binarios; por lo tanto, la precedencia de los operadores es muy importante al evaluar expresiones que contienen operadores binarios. En esta sección, comenzamos con operadores aritméticos binarios; Ampliaremos a otros operadores binarios en secciones posteriores.

## Operadores aritméticos

Los operadores aritméticos son aquellos que operan sobre valores numéricos. Se muestran en la Tabla 2.4.

TABLA 2.4 Operadores aritméticos binarios

| Operador | Descripción de ejemplo |
|---|---|
| Suma | a + b Suma dos valores numéricos |
| Resta | c - d Resta dos valores numéricos |
| Multiplicación | e * f Multiplica dos valores numéricos |
| División | g/h Divide un valor numérico por otro |
| Módulo | i % j Devuelve el resto después de dividir un valor numérico por otro |

Machine Translated by Google



Deberías saber todo menos los módulos de las primeras matemáticas. Sin embargo, si no sabe qué es el módulo, no se preocupe: lo cubriremos en breve. Los operadores aritméticos también incluyen los operadores unarios, ++ y que ya cubrimos. Como habrás notado en la Tabla 2.1, los operadores multiplicativos (*, /, %) tienen un orden de precedencia más alto que los operadores aditivos (+, -). Observa la siguiente expresión:

```
precio int = 2 * 5 + 3 * 4 - 8;
```

Primero, evalúas 2 * 5 y 3 * 4, lo que reduce la expresión a esto:

```
precio int = 10 + 12 - 8;
```

Luego, evalúa los términos restantes en orden de izquierda a derecha, lo que da como resultado un valor de precio de 14. Asegúrese de comprender por qué el resultado es 14 porque probablemente verá este tipo de pregunta de precedencia de operadores en el examen.

Todos los operadores aritméticos se pueden aplicar a cualquier primitiva de Java, a excepción del booleano. Además, sólo los operadores de suma + y += pueden aplicarse a los valores String , lo que da como resultado String concatenación. Aprenderá más sobre estos operadores y cómo se aplican a los valores de cadena en el Capítulo 4, "API principales".

## Agregar paréntesis

Es posible que haya notado que dijimos "A menos que se anule con paréntesis" antes de presentar la Tabla 2.1 sobre la precedencia de operadores. Esto se debe a que puede cambiar el orden de operación explícitamente colocando paréntesis alrededor de las secciones que desea evaluar primero.

### Cambiar el orden de operación

Volvamos al ejemplo de precios anterior . El siguiente fragmento de código contiene los mismos valores y operadores, en el mismo orden, pero con dos conjuntos de paréntesis agregados:

```
precio int = 2 * ((5 + 3) * 4 - 8);
```

Esta vez evaluarías el operador de suma 5 + 3, lo que reduce la expresión a lo siguiente:

```
precio int = 2 * (8 * 4 - 8);
```

Puede reducir aún más esta expresión multiplicando los dos primeros valores entre paréntesis:

```
precio int = 2 * (32 - 8);
```

A continuación, resta los valores dentro del paréntesis antes de aplicar los términos fuera del paréntesis:

```
precio int = 2 * 24;
```

Finalmente, multiplicarías el resultado por 2, lo que daría como resultado un valor de 48 para el precio.

Machine Translated by Google



Los paréntesis pueden aparecer en casi cualquier pregunta del examen que involucre valores numéricos, así que asegúrese de comprender cómo cambian el orden de operación cuando los vea.

Cuando encuentre código en su carrera profesional en el que no esté seguro del orden de operación, no dude en agregar paréntesis opcionales. Si bien a menudo no son necesarios, pueden mejorar la legibilidad, especialmente como verá con los operadores ternarios.

### Verificar la sintaxis de paréntesis

Cuando trabaje con paréntesis, debe asegurarse de que siempre sean válidos y equilibrados. Considere los siguientes ejemplos:

```
paloma larga = 1 + ((3 * 5) / 3;                // NO COMPILA
int blueJay = (9 + 2) + 3) / (2 * 4; // NO COMPILA
```

El primer ejemplo no se compila porque los paréntesis no están equilibrados. Hay un paréntesis izquierdo sin paréntesis derecho coincidente. El segundo ejemplo tiene el mismo número de paréntesis izquierdo y derecho, pero no están equilibrados correctamente. Al leer de izquierda a derecha, un nuevo paréntesis derecho debe coincidir con un paréntesis izquierdo anterior. Asimismo, todos los paréntesis izquierdos deben cerrarse con paréntesis derechos antes del final de la expresión.

Probemos con otro ejemplo:

```
petirrojo corto = 3 + [(4 * 2) + 4];            // NO COMPILA
```

Este ejemplo no se compila porque Java, a diferencia de otros lenguajes de programación, no permite el uso de corchetes, [], en lugar de paréntesis. Si reemplaza los corchetes con paréntesis, el último ejemplo se compilará bien.

## Operadores de división y módulo

Como dijimos anteriormente, el operador de módulo %, puede ser nuevo para usted. El operador de módulo, a veces llamado operador de resto, es simplemente el resto cuando se dividen dos números. Por ejemplo, 9 dividido entre 3 se divide uniformemente y no tiene resto; por lo tanto, el resultado de 9 % 3 es 0. Por otro lado, 11 dividido por 3 no divide exactamente; por lo tanto, el resultado de 11 % 3 es 2.

Los siguientes ejemplos ilustran esta distinción:

```
System.out.println(9/3 ) ;          // 3
System.out.println(9 % 3);          // 0

System.out.println(10/3 ) ; // 3
System.out.println(10 % 3); // 1

System.out.println(11/3 ) ; // 3
System.out.println(11 % 3); // 2
```

Machine Translated by Google



```
System.out.println(12/3 ) ; // 4
System.out.println(12 % 3); // 0
```

Como puede ver, los resultados de la división aumentan solo cuando el valor en el lado izquierdo pasa de 11 a 12, mientras que el valor del resto del módulo aumenta en 1 cada vez que el lado izquierdo aumenta hasta que llega a cero. Para un divisor y dado, la operación del módulo da como resultado un valor entre 0 y (y - 1) para dividendos positivos, o 0, 1, 2 en este ejemplo.

Asegúrese de comprender la diferencia entre división aritmética y módulo. Para entero valores, la división da como resultado el valor mínimo del entero más cercano que cumple la operación, mientras que el módulo es el valor restante. Si escucha la frase valor mínimo, solo significa el valor sin nada después del punto decimal. Por ejemplo, el valor mínimo es 4 para cada uno de los valores 4,0, 4,5 y 4,9999999. A diferencia del redondeo, que cubriremos en el Capítulo 4, simplemente se toma el valor antes del punto decimal, independientemente de lo que haya después del punto decimal.

La operación del módulo no se limita a valores enteros positivos en Java; también se puede aplicar a números enteros negativos y números de punto flotante. Por ejemplo, si el divisor es 5, entonces el valor del módulo de un número negativo está entre -4 y 0. Sin embargo, para el examen no es necesario que puedas tomar el módulo de un número entero negativo o de punto flotante. número.

## Promoción numérica

Ahora que comprende los conceptos básicos de los operadores aritméticos, es vital hablar sobre la promoción numérica primitiva, ya que Java puede hacer cosas que al principio le parezcan inusuales. Como mostramos en el Capítulo 1, “Bloques de construcción”, cada tipo numérico primitivo tiene una longitud de bits. No necesitas saber el tamaño exacto de estos tipos para el examen, pero sí debes saber cuáles son más grandes que otros. Por ejemplo, debes saber que long ocupa más espacio que int, que a su vez ocupa más espacio que short, y así sucesivamente.

Debe memorizar ciertas reglas que Java seguirá al aplicar operadores a tipos de datos:

Reglas de promoción numéricas

1. Si dos valores tienen diferentes tipos de datos, Java promoverá automáticamente uno de los valores al mayor de los dos tipos de datos.
2. Si uno de los valores es integral y el otro es de punto flotante, Java promoverá automáticamente el valor integral al tipo de datos del valor de punto flotante.
3. Los tipos de datos más pequeños, a saber, byte, short y char, se promocionan primero a int cada vez que se usan con un operador aritmético binario de Java con una variable (en lugar de un valor), incluso si ninguno de los operandos es En t.
4. Después de que se haya producido toda la promoción y los operandos tengan el mismo tipo de datos, el valor resultante tendrá el mismo tipo de datos que sus operandos promocionados.

Machine Translated by Google



Las dos últimas reglas son con las que la mayoría de la gente tiene problemas y con las que es probable que tropiecen. estás arriba en el examen. Para la tercera regla, tenga en cuenta que los operadores unarios están excluidos de esta regla. Por ejemplo, aplicar ++ a un valor corto da como resultado un valor corto .

Abordemos algunos ejemplos con fines ilustrativos:

■ ¿ Cuál es el tipo de datos de x * y?

```
entero x = 1;
larga y = 33;
var z = x * y;
```

En este caso, seguimos la primera regla. Dado que uno de los valores es int y el otro es long, y long es mayor que int, el valor int x primero se promueve a long. El resultado z es entonces un valor largo .

■ ¿ Cuál es el tipo de datos de x + y?

```
doble x = 39,21;
flotar y = 2,1;
var z = x + y;
```

En realidad, esta es una pregunta capciosa, ya que la segunda línea no se compila. Como recordará del Capítulo 1, se supone que los literales de punto flotante son dobles a menos que se fijen con una f, como en 2.1f. Si el valor de y se estableciera correctamente en 2.1f, entonces la promoción sería similar al ejemplo anterior, con ambos operandos promovidos a doble y el resultado z sería un valor doble .

■ ¿ Cuál es el tipo de datos de x * y?

```
corto x = 10;
corta y = 3;
var z = x * y;
```

En la última línea, debemos aplicar la tercera regla: que x e y se promoverán a int antes de la operación de multiplicación binaria, lo que dará como resultado una salida de tipo int. Si intentara asignar el valor a una variable corta z sin conversión, entonces el código no se compilaría. Preste mucha atención al hecho de que el resultado resultante no es corto, ya que volveremos a este ejemplo en la próxima sección "Asignación de valores".

■ ¿ Cuál es el tipo de datos de w * x / y?

```
corto w = 14;
flotar x = 13;
doble y = 30;
var z = w * x / y;
```

En este caso, debemos aplicar todas las reglas. En primer lugar, w se promoverá automáticamente a int únicamente porque es corto y se utiliza en una operación aritmética binaria. El valor w promocionado se promoverá automáticamente a flotante para que pueda multiplicarse por x. El resultado de w * x se promoverá automáticamente a doble para que pueda dividirse por y, lo que dará como resultado un valor doble .

Machine Translated by Google



Cuando trabaje con operadores aritméticos en Java, siempre debe tener en cuenta el tipo de datos de las variables, los valores intermedios y los valores resultantes. Debe aplicar la prioridad de los operadores y los paréntesis y trabajar hacia afuera, promoviendo los tipos de datos a lo largo del camino. En la siguiente sección, discutiremos las complejidades de asignar estos valores a variables de un tipo particular.

# Asignar valores

Los errores de compilación de los operadores de tareas a menudo se pasan por alto en el examen, en parte debido a lo sutiles que pueden ser estos errores. Para tener éxito con los operadores de asignación, debe comprender con fluidez cómo maneja el compilador la promoción numérica y cuándo se requiere la conversión. Ser capaz de detectar estos problemas es fundamental para aprobar el examen, ya que los operadores de asignación aparecen en casi todas las preguntas con un fragmento de código.

## Operador de asignación

Un operador de asignación es un operador binario que modifica o asigna la variable del lado izquierdo del operador con el resultado del valor del lado derecho de la ecuación. A diferencia de la mayoría de los demás operadores de Java, el operador de asignación se evalúa de derecha a izquierda.

El operador de asignación más simple es la asignación = , que ya has visto:

```
rebaño int = 1;
```

Esta declaración asigna a la variable rebaño el valor de 1.

Java promoverá automáticamente de tipos de datos más pequeños a más grandes, como vio en la sección anterior sobre operadores aritméticos, pero generará una excepción del compilador si detecta que está intentando convertir de tipos de datos más grandes a más pequeños sin conversión. La Tabla 2.5 enumera el primer operador de asignación que necesita conocer para el examen. Presentamos operadores de asignación adicionales más adelante en esta sección.

TABLA 2.5 Operador de asignación simple

| Operador | Ejemplo | Descripción |
|---|---|---|
| Asignación | int a = 50; | Asigna el valor de la derecha a la variable de la izquierda. |

## Valores de fundición

Parece fácil hasta ahora, ¿verdad? Bueno, realmente no podemos hablar en detalle sobre el operador de asignación hasta que hayamos cubierto el casting. La conversión es una operación unaria en la que un tipo de datos se interpreta explícitamente como otro tipo de datos. La conversión es opcional e innecesaria al convertir a un

Machine Translated by Google



tipo de datos más grande o más amplio, pero es necesario al convertir a un tipo de datos más pequeño o más estrecho. Sin conversión, el compilador generará un error al intentar colocar un tipo de datos más grande dentro de uno más pequeño.

La conversión se realiza colocando el tipo de datos, entre paréntesis, a la izquierda del valor que desea emitir. A continuación se muestran algunos ejemplos de casting:

```
int piel = (int)5;
int cabello = (corto) 2;
Tipo de cadena = (Cadena) "Pájaro";
cola corta = (corta)(4 + 10);
plumas largas = 10(largas); // NO COMPILA
```

Los espacios entre el elenco y el valor son opcionales. Como se muestra en el penúltimo ejemplo, es común que el lado derecho también esté entre paréntesis. Dado que la conversión es una operación unaria, solo se aplicaría al 4 si no incluyéramos 4 + 10 entre paréntesis. El último ejemplo no se compila porque el tipo está en el lado equivocado del valor.

Por un lado, es conveniente que el compilador convierta automáticamente tipos de datos más pequeños en tipos más grandes. Por otro lado, las preguntas de examen son excelentes cuando hacen lo contrario para ver si estás prestando atención. Vea si puede descubrir por qué ninguna de las siguientes líneas de código se compila:

```
huevo flotante = 2,0/9; // NO COMPILA
int renacuajo = (int)5 * 2L; // NO COMPILA
rana corta = 3 - 2,0; // NO COMPILA
```

Todos estos ejemplos implican poner un valor mayor en un tipo de datos más pequeño. No te preocupes si aún no sigues esto; Cubriremos más ejemplos como este en breve.

En este capítulo, la conversión se ocupa principalmente de convertir tipos de datos numéricos en otros tipos de datos. Como verá en capítulos posteriores, la conversión también se puede aplicar a objetos y referencias. Sin embargo, en esos casos no se realiza ninguna conversión. En pocas palabras, convertir un valor numérico puede cambiar el tipo de datos, mientras que convertir un objeto solo cambia la referencia al objeto, no el objeto en sí.

## Revisión de asignaciones primitivas

Vea si puede descubrir por qué cada una de las siguientes líneas no se compila:

```
pez int = 1,0; // NO COMPILA
pájaro corto = 1921222; // NO COMPILA
int mamífero = 9f; // NO COMPILA
reptil largo = 192_301_398_193_810_323; // NO COMPILA
```

La primera declaración no se compila porque estás intentando asignar un doble 1.0 a un valor entero. Aunque el valor es un número entero matemático, al agregar .0, le estás indicando al compilador que lo trate como un doble. La segunda declaración no se compila porque el valor literal 1921222 está fuera del rango corto y el compilador lo detecta. La tercera declaración no se compila porque la f se agregó al final del número

Machine Translated by Google



indica al compilador que trate el número como un valor de punto flotante, pero la asignación es a un int. Finalmente, la última declaración no se compila porque Java interpreta el literal como un int y nota que el valor es mayor de lo que permite int . El literal necesitaría un sufijo L o l para ser considerado largo.

## Aplicar fundición

Podemos arreglar tres de los ejemplos anteriores convirtiendo los resultados a un tipo de datos más pequeño. Recuerde, es necesario convertir primitivas cada vez que se pasa de un tipo de datos numéricos más grande a un tipo de datos numéricos más pequeño, o se convierte de un número de punto flotante a un valor integral.

```
pez int = (int)1.0;
pájaro corto = (corto)1921222; // Almacenado como 20678
int mamífero = (int)9f;
```

¿Qué tal si aplicamos el casting al último ejemplo?

```
reptil largo = (largo)1923013981938103 23; // NO COMPILA
```

Esto todavía no se compila porque el compilador interpreta primero el valor como un int y está fuera de alcance. Lo siguiente corrige este código sin requerir conversión:

```
reptil largo = 1923013981938103 23L;
```

**Real World Scenario**

Desbordamiento y subdesbordamiento

Las expresiones del ejemplo anterior ahora se compilan, aunque hay un costo. El segundo valor, 1.921.222, es demasiado grande para almacenarlo como corto, por lo que se produce un desbordamiento numérico y se convierte en 20.678. El desbordamiento ocurre cuando un número es tan grande que ya no cabe dentro del tipo de datos, por lo que el sistema "se ajusta" al valor negativo más bajo y cuenta desde allí, de manera similar a cómo funciona la aritmética de módulo. También hay un desbordamiento análogo, cuando el número es demasiado bajo para caber en el tipo de datos, como almacenar -200 en un campo de bytes .

Esto está más allá del alcance del examen, pero es algo con lo que debes tener cuidado en tu propio código. Por ejemplo, la siguiente declaración genera un número negativo:

```
Salida.del.sistema.print(2147483647+1); // -2147483648
```

Dado que 2147483647 es el valor int máximo , agregarle cualquier valor estrictamente positivo hará que se ajuste al número negativo más pequeño.

Volvamos a un ejemplo similar de la sección “Promoción numérica” anteriormente en este capítulo.

Machine Translated by Google



```
ratón corto = 10;
hámster corto = 3;
carpincho corto = ratón * hámster; // NO COMPILA
```

Con base en todo lo que has aprendido hasta ahora sobre promoción y casting numéricos, ¿entiendes por qué no se compila la última línea de esta declaración? Como recordará, los valores cortos se promocionan automáticamente a int al aplicar cualquier operador aritmético, siendo el valor resultante de tipo int. Intentar asignar una variable corta con un valor int da como resultado un error del compilador, ya que Java cree que está intentando convertir implícitamente de un tipo de datos más grande a uno más pequeño.

Podemos corregir esta expresión mediante conversión, ya que hay ocasiones en las que es posible que desees anular el comportamiento predeterminado del compilador. En este ejemplo, sabemos que el resultado de 10 * 3 es 30, lo que puede caber fácilmente en una variable corta , por lo que podemos aplicar conversión para convertir el resultado nuevamente a corto:

```
ratón corto = 10;
hámster corto = 3;
carpincho corto = (corto)(ratón * hámster);
```

Al convertir un valor mayor en un tipo de datos más pequeño, le indica al compilador que ignore su comportamiento predeterminado. En otras palabras, le está diciendo al compilador que ha tomado medidas adicionales para evitar el desbordamiento o el desbordamiento insuficiente. También es posible que en su aplicación y escenario particulares, el desbordamiento o el desbordamiento resulten en valores aceptables.

Por último, pero no menos importante, la conversión puede aparecer en cualquier parte de una expresión, no solo en la asignación. mento. Por ejemplo, echemos un vistazo a una forma modificada del ejemplo anterior:

```
ratón corto = 10;
hámster corto = 3;
carpincho corto = ratón (corto) * hámster;                    // NO COMPILA
```

Entonces, ¿qué está pasando en la última línea? Bueno, ¿recuerdas cuando dijimos que el casting era una operación unaria? Eso significa que la conversión en la última línea se aplica al mouse y solo al mouse . Una vez completada la conversión, ambos operandos se promueven a int ya que se usan con el operador de multiplicación binaria (*), lo que convierte el resultado en un int y provoca un error de compilación.

¿Qué pasa si cambiamos la última línea a la siguiente?

```
carpincho corto = 1 + (corto)(ratón * hámster); // NO COMPILA
```

En el ejemplo, la conversión se realiza correctamente, pero el valor resultante se modifica automáticamente. promovido a int porque se usa con el operador aritmético binario (+).

## Valores de conversión frente a variables

Revisando nuestra tercera regla promocional numérica, el compilador no requiere conversión cuando trabaja con valores literales que se ajustan al tipo de datos. Considere estos ejemplos:

```
sombrero de bytes = 1;
guantes de bytes = 7 * 10;
bufanda corta = 5;
botas cortas = 2 + 1;
```

Machine Translated by Google



Todas estas declaraciones se compilan sin problemas. Por otra parte, ninguna de estas afirmaciones recoge:

```
botas cortas = 2 + sombrero; // NO COMPILA
guantes de bytes = 7 * 100; // NO COMPILA
```

La primera declaración no se compila porque es una variable, no un valor, y ambos operandos se promocionan automáticamente a int. Al trabajar con valores, el compilador tenía suficiente información para determinar la intención del escritor. Sin embargo, cuando se trabaja con variables, existe ambigüedad sobre cómo proceder, por lo que el compilador informa un error. La segunda expresión no se compila porque 700 desencadena un desbordamiento del byte, que tiene un valor máximo de 127.

## Operadores de asignación compuesta

Además del operador de asignación simple (=), Java admite numerosos operadores de asignación compuestos. Para el examen, debe estar familiarizado con los operadores compuestos de la Tabla 2.6.

TABLA 2.6 Operadores de asignación compuesta

| Operador | Ejemplo | Descripción |
|---|---|---|
| tarea adicional | un += 5 | Agrega el valor de la derecha a la variable de la izquierda y asigna la suma a la variable |
| tarea de resta | b -= 0.2 | Resta el valor de la derecha de la variable de la izquierda y asigna la diferencia a la variable |
| Asignación de multiplicación | c*= 100 | Multiplica el valor de la derecha con la variable de la izquierda y asigna el producto a la variable |
| Asignación de división | re/= 4 | Divide la variable de la izquierda por el valor de la derecha y asigna el cociente a la variable |

Los operadores compuestos son en realidad formas glorificadas del operador de asignación simple, con una operación aritmética o lógica incorporada que aplica los lados izquierdo y derecho de la declaración y almacena el valor resultante en la variable en el lado izquierdo de la declaración. Por ejemplo, las dos declaraciones siguientes después de la declaración de camello y jirafa son equivalentes cuando se ejecutan de forma independiente:

```
int camello = 2, jirafa = 3;
camello = camello * jirafa; // Operador de asignación simple
camello *= jirafa; // operador de asignación compuesta
```

Machine Translated by Google



El lado izquierdo del operador compuesto sólo se puede aplicar a una variable que ya está definida y no se puede utilizar para declarar una nueva variable. En este ejemplo, si camel no estuviera definido, la expresión camel *= jirafa no se compilaría.

Los operadores compuestos son útiles para algo más que una taquigrafía: también pueden evitar tener que convertir explícitamente un valor. Por ejemplo, considere lo siguiente. ¿Puedes entender por qué la última línea no se compila?

```
cabra larga = 10;
int oveja = 5;
oveja = oveja * cabra; // NO COMPILA
```

De la sección anterior, debería poder detectar el problema en la última línea. Estamos intentando asignar un valor largo a una variable int . Esta última línea podría arreglarse con una conversión explícita a (int), pero hay una mejor manera de usar el operador de asignación compuesta:

```
cabra larga = 10;
int oveja = 5;
oveja *= cabra;
```

El operador compuesto primero convertirá oveja en un largo, aplicará la multiplicación de dos valores largos y luego convertirá el resultado en un int. A diferencia del ejemplo anterior, en el que el compilador informó un error, el compilador convertirá automáticamente el valor resultante al tipo de datos del valor en el lado izquierdo del operador compuesto.

## Valor de retorno de los operadores de asignación

Una última cosa que debe saber sobre los operadores de asignación es que el resultado de una asignación es una expresión en sí misma igual al valor de la asignación. Por ejemplo, el siguiente fragmento de código es perfectamente válido, aunque tenga un aspecto un poco extraño:

```
lobo largo = 5;
coyote largo = (lobo=3);
System.out.println(lobo); // 3
System.out.println(coyote); // 3
```

La clave aquí es que (wolf=3) hace dos cosas. Primero, establece el valor de la variable. wolf es 3. En segundo lugar, devuelve un valor de la asignación, que también es 3.

A los creadores del examen les gusta insertar el operador de asignación (=) en medio de una expresión y usar el valor de la tarea como parte de una expresión más compleja. Por ejemplo, no se sorprenda si ve una declaración if en el examen similar a la siguiente:

```
booleano saludable = falso;
si (saludable = verdadero)
    System.out.print("¡Bien!");
```

Si bien esto puede parecer una prueba de si saludable es verdadero, en realidad está asignando a saludable un valor de verdadero. El resultado de la asignación es el valor de la asignación, lo cual es cierto,

Machine Translated by Google



dando como resultado la impresión de este fragmento ¡Bueno!. Cubriremos esto con más detalle en la próxima sección "Operadores de igualdad".

# Comparando valores

El último conjunto de operadores binarios gira en torno a la comparación de valores. Se pueden utilizar para comprobar si dos valores son iguales, comprobar si un valor numérico es menor o mayor que otro y realizar aritmética booleana. Lo más probable es que haya utilizado muchos de los operadores de esta sección en su experiencia de desarrollo.

## Operadores de igualdad

Determinar la igualdad en Java puede ser una tarea no trivial ya que existe una diferencia semántica entre "dos objetos son iguales" y "dos objetos son equivalentes". Lo complica aún más el hecho de que para las primitivas numéricas y booleanas no existe tal distinción.

La tabla 2.7 enumera los operadores de igualdad. El operador igual (==) y el operador no igual (!=) comparan dos operandos y devuelven un valor booleano que determina si las expresiones o valores son iguales o no iguales, respectivamente.

TABLA 2.7 Operadores de igualdad

| Ejemplo de operador | Aplicar a primitivas | Aplicar a objetos |
|---|---|---|
| Igualdad a == 10 | Devuelve verdadero si los dos valores representan el mismo valor | Devuelve verdadero si los dos valores hacen referencia al mismo objeto |
| Desigualdad b != 3.14 | Devuelve verdadero si los dos valores representan diferentes valores | Devuelve verdadero si los dos valores no hacen referencia al mismo objeto |

El operador de igualdad se puede aplicar a valores numéricos, valores booleanos y objetos (incluidos cadenas y nulos). Al aplicar el operador de igualdad, no se pueden mezclar estos tipos. Cada uno de los siguientes resultados produce un error del compilador:

```
mono booleano = verdadero == 3; // NO COMPILA
mono booleano = false != "Uva"; // NO COMPILA
gorila booleano = 10,2 == "Koko"; // NO COMPILA
```

Preste mucha atención a los tipos de datos cuando vea un operador de igualdad en el examen. Como se mencionó en la sección anterior, los creadores del examen también tienen la costumbre de mezclar operadores de asignación y operadores de igualdad.

```
oso booleano = falso;
```

Machine Translated by Google



```
polar booleano = (oso = verdadero);
System.out.println(polar); // verdadero
```

A primera vista, podría pensar que el resultado debería ser falso, y si la expresión fuera (bear == true), entonces estaría en lo cierto. Sin embargo, en este ejemplo, la expresión asigna el valor verdadero y , como vio en la sección sobre operadores de asignación, la asignación en sí tiene el valor de la asignación . Por lo tanto, a polar también se le asigna un valor verdadero y la salida es verdadera.

Para la comparación de objetos, el operador de igualdad se aplica a las referencias a los objetos, no a los objetos a los que apuntan. Dos referencias son iguales si y sólo si apuntan al mismo objeto o ambas apuntan a nulo. Echemos un vistazo a algunos ejemplos:

```
var lunes = nuevo archivo("programa.txt");
var martes = nuevo archivo("programa.txt");
var miércoles = martes;
System.out.println(lunes == martes);          // FALSO
System.out.println(martes == miércoles); // verdadero
```

Aunque todas las variables apuntan a la misma información de archivo, solo dos referencias, martes y miércoles, son iguales en términos de == ya que apuntan al mismo objeto.

Espera, ¿cuál es la clase Archivo ? En este ejemplo, así como durante el examen, es posible que se le presenten nombres de clases que no le resulten familiares, como Archivo. Muchas veces puedes responder preguntas sobre estas clases sin conocer los detalles específicos de estas clases. En el ejemplo anterior, debería poder responder preguntas que indiquen lunes y martes.
Son dos objetos separados y distintos porque se utiliza la palabra clave new , incluso si no está familiarizado con los tipos de datos de estos objetos.

En algunos idiomas, comparar nulo con cualquier otro valor siempre es falso, aunque esto No es el caso en Java.

```
System.out.print(nulo == nulo); // verdadero
```

En el capítulo 4, continuaremos la discusión sobre la igualdad de objetos introduciendo lo que significa que dos objetos diferentes sean equivalentes. También cubriremos la igualdad de cadenas y mostraremos cómo este puede ser un tema no trivial.

## Operadores relacionales

Pasamos ahora a los operadores relacionales, que comparan dos expresiones y devuelven un valor booleano . La Tabla 2.8 describe los operadores relacionales que necesita conocer para el examen.

Machine Translated by Google



TABLA 2.8 Operadores relacionales

| Operador | Ejemplo | Descripción |
|---|---|---|
| Menos que | un < 5 | Devuelve verdadero si el valor de la izquierda es estrictamente menor que el valor de la derecha |
| Menos que o igual a | segundo <= 6 | Devuelve verdadero si el valor de la izquierda es menor o igual que el valor de la derecha |
| Mas grande que | c > 9 | Devuelve verdadero si el valor de la izquierda es estrictamente mayor que el valor de la derecha |
| Mayor qué o igual a | 3 >= d | Devuelve verdadero si el valor de la izquierda es mayor o igual que el valor de la derecha |
| Comparación de tipos e instancia de | Cadena | Devuelve verdadero si la referencia en el lado izquierdo es una instancia del tipo en el lado derecho (clase, interfaz, registro, enumeración, anotación) |

## Operadores de comparación numérica

Los primeros cuatro operadores relacionales de la tabla 2.8 se aplican sólo a valores numéricos. Si los dos operandos numéricos no son del mismo tipo de datos, se promueve el más pequeño, como se analizó anteriormente.

Veamos ejemplos de estos operadores en acción:

```
int gibbonNumFeet = 2, wolfNumFeet = 4, avestruzNumFeet = 2;
System.out.println(gibbonNumFeet < wolfNumFeet); // verdadero
System.out.println(gibbonNumFeet <= wolfNumFeet); // verdadero
System.out.println(gibbonNumFeet >= avestruzNumFeet); // verdadero
System.out.println(gibbonNumFeet > avestruzNumFeet); // FALSO
```

Observe que el último ejemplo genera falso, porque aunque gibbonNumFeet y ostrichNumFeet tiene el mismo valor, gibbonNumFeet no es estrictamente mayor que ostrichNumFeet.

## instancia de operador

El último operador relacional que necesita conocer para el examen es el operador instancia de , que se muestra en la Tabla 2.8. Es útil para determinar si un objeto arbitrario es miembro de una clase o interfaz particular en tiempo de ejecución.

¿Por qué no sabrías qué clase o interfaz es un objeto? Como veremos en el Capítulo 6, “Diseño de clases”, Java admite el polimorfismo. Por ahora, todo lo que necesitas saber es

Machine Translated by Google



Los objetos se pueden pasar usando una variedad de referencias. Por ejemplo, todas las clases heredan de java.lang.Object. Esto significa que cualquier instancia puede asignarse a una referencia de Objeto . Por ejemplo, ¿cuántos objetos se crean y utilizan en el siguiente fragmento de código?

```
Entero zooTime = Integer.valueOf(9);
Número num = zooTime;
Objeto obj = zooTime;
```

En este ejemplo, sólo se crea un objeto en la memoria, pero hay tres referencias diferentes a él porque Integer hereda tanto Número como Objeto. Esto significa que puede llamar a instancia de cualquiera de estas referencias con tres tipos de datos diferentes y devolverá verdadero para cada una de ellas.

Donde el polimorfismo suele entrar en juego es cuando se crea un método que toma un tipo de datos con muchas subclases posibles. Por ejemplo, imaginemos que tenemos una función que abre el zoológico e imprime la hora. Como entrada, toma un Número como parámetro de entrada.

```
openZoo público vacío (Número de tiempo) {}
```

Ahora, queremos que la función agregue las horas al final de la salida si el valor es un tipo de número entero, como un número entero; de lo contrario, simplemente imprime el valor.

```
openZoo público vacío (Número de tiempo) {
    si ( instancia de tiempo de entero)
        System.out.print((Entero)hora + "En punto");
    demás
        System.out.print(hora);
}
```

Ahora tenemos un método que puede manejar de forma inteligente tanto valores enteros como otros. Un buen ejercicio que le queda al lector es agregar comprobaciones para otros tipos de datos numéricos como Corto, Largo, Doble, etc.

Observe que convertimos el valor entero en este ejemplo. Es común usar la conversión con instancia de cuando se trabaja con objetos que pueden ser de varios tipos diferentes, ya que la conversión le brinda acceso a campos disponibles solo en las clases más específicas. Se considera una buena práctica de codificación utilizar el operador instancia de antes de convertir un objeto a un tipo más limitado.

Para el examen, solo necesita concentrarse en cuándo se usa instancia de con clases e interfaces. Aunque se puede utilizar con otros tipos de alto nivel, como registros, enumeraciones y anotaciones, no es común.

## Instancia no válida

Un área en la que el examen podría intentar hacerle tropezar es el uso de instancia de con tipos incompatibles. Por ejemplo, es imposible que Número contenga un valor de Cadena , por lo que lo siguiente provoca un error de compilación:

```
openZoo público vacío (Número de tiempo) {
    if(instancia de tiempo de String) // NO COMPILA
        System.out.print(hora);
}
```

Machine Translated by Google



Si el compilador puede determinar que una variable no se puede convertir a una clase específica, informa un error.

### nulo y el operador de instancia

¿ Qué sucede si llamas a instancia de una variable nula ? Para el examen, debe saber que llamar a instancia de un literal nulo o una referencia nula siempre devuelve falso.

```
System.out.print (instancia nula de objeto); // FALSO

Objeto noObjectHere = nulo;
System.out.print(noObjectHere instancia de String); // FALSO
```

Los ejemplos anteriores imprimen falso. Casi no importa cuál sea el lado derecho de la expresión. Decimos “casi” porque hay excepciones. Este ejemplo no se compila, ya que se usa nulo en el lado derecho del operador instancia de :

```
System.out.print (instancia nula de nulo); // NO COMPILA
```

Aunque puede parecer que has aprendido todo lo que hay sobre el operador de instancia , ¡hay mucho más por venir! En el Capítulo 3, presentamos la coincidencia de patrones con el operador instancia de , que se agregó oficialmente en Java 16. En el Capítulo 7, “Más allá de las clases”, presentamos el polimorfismo con mucho más detalle y mostramos cómo aplicar estas reglas a las interfaces.

## Operadores logicos

Si has estudiado informática, es posible que ya te hayas encontrado antes con operadores lógicos. Si no, no entre en pánico: los cubriremos en detalle en esta sección.

Los operadores lógicos (&), (|) y (^) se pueden aplicar tanto a números como a booleanos . tipos de datos; se enumeran en la Tabla 2.9. Cuando se aplican a tipos de datos booleanos , se les denomina operadores lógicos. Alternativamente, cuando se aplican a tipos de datos numéricos, se les conoce como operadores bit a bit, ya que realizan comparaciones bit a bit de los bits que componen el número. Sin embargo, para el examen no es necesario saber nada sobre comparaciones numéricas bit a bit, por lo que dejaremos ese aspecto educativo para otros libros.

TABLA 2.9 Operadores lógicos

| Operador | Descripción de ejemplo | |
|---|---|---|
| Y lógico | a y b | El valor es verdadero sólo si ambos valores son verdaderos. |
| Inclusivo lógico OR c \| d | | El valor es verdadero si al menos uno de los valores es verdadero. |
| O exclusivo lógico e | ^ f | El valor es verdadero sólo si un valor es verdadero y el otro es falso. |

Machine Translated by Google



Debe familiarizarse con las tablas de verdad de la Figura 2.2 , donde xey son Se supone que son tipos de datos booleanos .

FIGURA 2.2 Las tablas de verdad lógicas para &, | y ^

Y (x e y)

| | y = verdadero | y = false |
|---|---|---|
| x = verdadero | verdadero | FALSO |
| x = FALSO | FALSO | FALSO |

INCLUSIVO O (x | y)

| | y = verdadero | y = false |
|---|---|---|
| x = verdadero | verdadero | verdadero |
| x = FALSO | verdadero | FALSO |

EXCLUSIVO O (x ^ y)

| | y = verdadero | y = false |
|---|---|---|
| x = verdadero | FALSO | verdadero |
| x = FALSO | verdadero | FALSO |

A continuación se ofrecen algunos consejos que le ayudarán a recordar esta tabla:

■ AND sólo es verdadero si ambos operandos son verdaderos.

■ El OR inclusivo solo es falso si ambos operandos son falsos. ■ OR exclusivo sólo es verdadero si los operandos son diferentes.

Echemos un vistazo a algunos ejemplos:

```
ojos booleanosCerrados = verdadero;
respiración booleana Lentamente = verdadero;

booleano en reposo = ojos cerrados | respirando lentamente; booleano
dormido = ojos cerrados y respirando lentamente; booleano despierto
= ojos cerrados ^ respirando lentamente; System.out.println(en
reposo); // verdadero System.out.println(dormido); //
verdadero System.out.println(despierto);
                                                    // FALSO
```

Deberías probarlos tú mismo, cambiando los valores de ojos cerrados y respirando lentamente y estudiando los resultados.

# Operadores condicionales

A continuación, presentamos los operadores condicionales, && y ||, en la Tabla 2.10.

Machine Translated by Google



TABLA 2.10 Operadores condicionales

| Operador | Descripción de ejemplo |
|---|---|
| Condicional Y | a && b El valor es verdadero solo si ambos valores son verdaderos. Si el lado izquierdo es falso, entonces el lado derecho no será evaluado. |
| Condicional O | c \|\| d El valor es verdadero si al menos uno de los valores es verdadero. Si el lado izquierdo es verdadero, entonces el lado derecho no será evaluado. |

Los operadores condicionales, a menudo llamados operadores de cortocircuito, son casi idénticos a los operadores lógicos, & y |, excepto que es posible que el lado derecho de la expresión nunca se evalúe si el resultado final puede determinarse por el lado izquierdo de la expresión. Por ejemplo, considere la siguiente afirmación:

```
hora int = 10;
booleano zooOpen = verdadero || (hora < 4);
System.out.println(zooOpen); // verdadero
```

Con referencia a las tablas de verdad, el valor zooOpen puede ser falso sólo si ambos lados de la expresión son falsos. Como sabemos que el lado izquierdo es verdadero, no es necesario evaluar el lado derecho, ya que ningún valor de hora hará que este código se imprima como falso. En otras palabras, la hora podría haber sido -10 u 892; el resultado habría sido el mismo. ¡Pruébelo usted mismo con diferentes valores por hora!

## Evitar una excepción NullPointerException

Un ejemplo más común de dónde se utilizan operadores condicionales es verificar si hay valores nulos. objetos antes de realizar una operación. En el siguiente ejemplo, si duck es nulo, el programa generará una NullPointerException en tiempo de ejecución:

```
if(duck!=null & duck.getAge()<5) { // Podría lanzar una NullPointerException
    // Hacer algo
}
```

El problema es que el operador lógico AND (&) evalúa ambos lados de la expresión. Podríamos agregar una segunda declaración if , pero esto podría resultar difícil de manejar si tenemos muchas variables que verificar. Una solución fácil de leer es utilizar el operador AND condicional (&&):

```
if(pato!=null && pato.getAge()<5) {
    // Hacer algo
}
```

En este ejemplo, si duck es nulo, el condicional evita una NullPointerException que nunca se lance, ya que nunca se alcanza la evaluación de duck.getAge() < 5 .

Machine Translated by Google



### Comprobación de efectos secundarios no realizados

Tenga cuidado con el comportamiento de cortocircuito en el examen, ya que se sabe que las preguntas alteran una variable en el lado derecho de la expresión que tal vez nunca se alcance. Esto se conoce como efecto secundario no realizado. Por ejemplo, ¿cuál es el resultado del siguiente código?

```
int conejo = 6;
conejito booleano = (conejo >= 6) || (++conejo <= 7);
System.out.println(conejo);
```

Como conejo >= 6 es verdadero, el operador de incremento en el lado derecho de la expresión nunca se evalúa, por lo que el resultado es 6.

# Tomar decisiones con el Operador ternario

El operador final con el que debería estar familiarizado para el examen es el operador condicional, ? :, también conocido como operador ternario. Se destaca porque es el único operador que toma tres operandos. El operador ternario tiene la siguiente forma:

```
expresión booleana? expresión1: expresión2
```

El primer operando debe ser una expresión booleana y el segundo y tercer operando pueden ser cualquier expresión que devuelva un valor. La operación ternaria es en realidad una forma condensada de una declaración combinada if y else que devuelve un valor. Cubrimos las declaraciones if/else con mucho más detalle en el Capítulo 3, por lo que por ahora solo usaremos ejemplos simples.

Por ejemplo, considere el siguiente fragmento de código que calcula la cantidad de comida para un búho:

```
int búho = 5;
comida interna;
si(búho < 2) {
    comida = 3;
} demás {
    comida = 4;
}
System.out.println(comida); // 4
```

Compare el fragmento de código anterior con el siguiente fragmento de código del operador ternario:

```
int búho = 5;
int comida = búho < 2 ? 3 : 4;
System.out.println(comida); // 4
```

Machine Translated by Google



Estos dos fragmentos de código son equivalentes. Tenga en cuenta que suele ser útil para mejorar la legibilidad agregar paréntesis alrededor de las expresiones en operaciones ternarias, aunque ciertamente no es necesario hacerlo. Sin embargo, es especialmente útil cuando se utilizan varios operadores ternarios juntos. Considere las siguientes dos expresiones equivalentes:

```
int comida1 = búho <4? búho > 2 ? 3:4:5;
int comida2 = (búho < 4 ? ((búho > 2) ? 3 : 4): 5);
```

Si bien son equivalentes, encontramos que la segunda afirmación es mucho más legible. Dicho esto, es posible que el examen pueda utilizar varios operadores ternarios en una sola línea.

Para el examen, debes saber que no es necesario que la segunda y tercera expresión en operaciones ternarias tengan el mismo tipo de datos, aunque sí entra en juego cuando se combina con el operador de asignación. Compare las dos declaraciones que siguen a la declaración de variable:

```
rayas internas = 7;

System.out.print((rayas > 5)? 21: "Cebra");

int animal = (rayas < 9)? 3: "Caballo"; // NO COMPILA
```

Ambas expresiones evalúan valores booleanos similares y devuelven un int y un String, aunque solo se compilará el primero. A System.out.print() no le importa que las expresiones sean de tipos completamente diferentes, porque puede convertir ambas en valores de Objeto y llamar a toString() en ellas. Por otro lado, el compilador sabe que "Horse" tiene un tipo de datos incorrecto y no se puede asignar a un int; por lo tanto, no permite compilar el código.

### Expresión ternaria y efectos secundarios no realizados

Como vimos con los operadores condicionales, una expresión ternaria puede contener un efecto secundario no realizado, ya que solo una de las expresiones del lado derecho será evaluada en tiempo de ejecución. Ilustremos este principio con el siguiente ejemplo:

```
int oveja = 1;
int zzz = 1;
int dormir = zzz<10? oveja++ : zzz++;
System.out.print(oveja + "," + zzz); // 2,1
```

Observe que como la expresión booleana de la izquierda era verdadera, sólo se incrementó oveja . Compare el ejemplo anterior con la siguiente modificación:

Machine Translated by Google



```
int oveja = 1;
int zzz = 1;
int dormir = oveja>=10? oveja++ : zzz++;
System.out.print(oveja + "," + zzz); // 1,2
```

Ahora que la expresión booleana de la izquierda se evalúa como falsa, solo se incrementa zzz . De esta manera vemos cómo los cambios en un operador ternario pueden no aplicarse si no se utiliza la expresión particular.

Para el examen, tenga cuidado con cualquier pregunta que incluya una expresión ternaria en la que se modifique una variable en una de las expresiones del lado derecho.

# Resumen

Este capítulo cubrió una amplia variedad de temas sobre operadores Java para operadores unarios, binarios y ternarios. Con suerte, la mayoría de estos operadores fueron revisados por usted. Si no, es necesario estudiarlos en detalle. Es importante que comprenda cómo utilizar todos los operadores Java requeridos cubiertos en este capítulo y sepa cómo la precedencia de los operadores y los paréntesis influyen en la forma en que se interpreta una expresión particular.

Es probable que haya numerosas preguntas en el examen que parezcan probar una cosa, como NIO.2 o el manejo de excepciones, cuando en realidad la respuesta está relacionada con el mal uso de un operador en particular que hace que la aplicación no se pueda compilar. Cuando vea un operador que involucra números en el examen, siempre verifique que se utilicen los tipos de datos apropiados y que coincidan entre sí cuando corresponda.

Los operadores se utilizan durante todo el examen, en casi todos los ejemplos de código, por lo que cuanto mejor comprenda este capítulo, más preparado estará para el examen.

## Conceptos básicos del examen

Ser capaz de escribir código que utilice operadores Java. Este capítulo cubrió una amplia variedad de símbolos de operador. Vuelva atrás y revíselos varias veces para familiarizarse con ellos durante el resto del libro.

Ser capaz de reconocer qué operadores están asociados con qué tipos de datos. Algunos operadores pueden aplicarse sólo a primitivas numéricas, algunos sólo a valores booleanos y otros sólo a objetos. Es importante que observe cuando un operador y sus operandos no coinciden, ya que es probable que este problema surja en un par de preguntas del examen.

Machine Translated by Google



Comprenda cuándo se requiere casting o se produce una promoción numérica. Siempre que mezcle operandos de dos tipos de datos diferentes, el compilador debe decidir cómo manejar el tipo de datos resultante. Cuando convierte de un tipo de datos más pequeño a uno más grande, la promoción numérica se aplica automáticamente. Cuando se convierte de un tipo de datos más grande a uno más pequeño, se requiere conversión.

Comprender la precedencia de los operadores de Java. La mayoría de los operadores Java con los que trabajará son binarios, pero el número de expresiones suele ser mayor que dos. Por lo tanto, debe comprender el orden en el que Java evaluará cada símbolo de operador.

Ser capaz de escribir código que utilice paréntesis para anular la precedencia de los operadores. Puede utilizar paréntesis en su código para cambiar manualmente el orden de prioridad.

Machine Translated by Google



# Preguntas de revisión

Las respuestas a las preguntas de revisión del capítulo se pueden encontrar en el Apéndice.

1. ¿Cuál de los siguientes operadores Java se puede utilizar con variables booleanas? (Elegir todos eso aplica.)

   R. ==

   B. +

   C.- _

   D. !

   mi %

   F ~

   G. Transmitir con (booleano)

2. ¿Qué tipo (o tipos) de datos permitirán que se compile el siguiente fragmento de código? (Elegir todos eso aplica.)

   manzanas de bytes
   = 5; naranjas cortas =
   _____ 10; plátanos = manzanas + naranjas;

   A.int _

   B. largo

   booleano _

   D. doble

   E. corto

   F byte

3. ¿Qué cambio, cuando se aplica de forma independiente, permitiría compilar el siguiente fragmento de código? (Elija todas las que correspondan).

   3: oreja larga = 10; 4:
   int audición = 2 * oído;

   R. Sin cambios; se compila tal cual.

   B. Transmitir oreja en la línea 4 hasta int.

   C. Cambie el tipo de datos de oreja en la línea 3 a corto.

   D. Eche 2 * ear en la línea 4 hasta int.

   E. Cambiar el tipo de datos de audiencia en la línea 4 a corto.

   F. Cambie el tipo de datos de audiencia en la línea 4 a largo.

Machine Translated by Google



4. ¿Cuál es el resultado del siguiente fragmento de código?

```
3: canino booleano = verdadero, lobo = verdadero;
4: int dientes = 20;
5: canino = (dientes! = 10) ^ (lobo = falso);
6: System.out.println(canino+", "+dientes+", "+lobo);
```

A. cierto, 20, verdadero

B. verdadero, 20, falso

C. falso, 10, verdadero

D falso, 20, falso

E. El código no se compilará debido a la línea 5.

F. Ninguna de las anteriores.

5. ¿ Cuáles de los siguientes operadores están clasificados en orden creciente o en el mismo orden de precedencia? Supongamos que el operador + es una suma binaria, no una forma unaria. (Elija todas las que correspondan).

A. +, *, %, --

B. ++, (int), *

C. =, ==, !

D. (corto), =, !, *

E. *, /, %, +, ==

F. !, ||, y

GRAMO. ^, +, =, +=

6. ¿Cuál es el resultado del siguiente programa?

```
1: clase pública CandyCounter {
2:     static long addCandy(doble fruta, vegetales flotantes) {
3:         devolver (int)frutas+verduras;
4:     }
5:
6:     público estático vacío principal (String [] argumentos) {
7:         System.out.print(addCandy(1.4, 2.4f) + ", ");
8:         System.out.print(addCandy(1.9, (flotante)4) + ", ");
9:         System.out.print(addCandy((long)(int)(short)2, (float)4)); } }
```

R. 4, 6, 6.0

B.3 , 5, 6

C.3 , 6, 6

D.4 , 5, 6

E. El código no se compila debido a la línea 9.

F. Ninguna de las anteriores.

Machine Translated by Google



7. ¿Cuál es el resultado del siguiente fragmento de código?

```
int ph = 7, vis = 2;
booleano claro = vis > 1 & (vis < 9 || ph < 2); booleano
seguro = (vis > 2) && (ph++ > 1); booleano
sabroso = 7 <= --ph;
System.out.println(claro + "-" + seguro + "-" + sabroso);
```

A. verdadero-verdadero-verdadero

B. verdadero-verdadero-falso

C. verdadero-falso-verdadero

D. verdadero-falso-falso

E. falso-verdadero-verdadero

F. falso-verdadero-falso

G. falso-falso-verdadero

H. falso-falso-falso

8. ¿Cuál es el resultado del siguiente fragmento de código?

```
4: int cerdo = (corto)4; 5:
cerdo = cerdo++;
6: cabra larga = (int)2;
7: cabra -= 1,0; 8:
Sistema.out.print(cerdo +          "  -  "      + cabra);
```

R.4 - 1

B.4 - 2

C.5 - 1

D.5 - 2

E. El código no se compila debido a la línea 7.

F. Ninguna de las anteriores.

9. ¿Cuáles son los resultados únicos del siguiente fragmento de código? (Elija todas las que correspondan).

```
int a = 2, b = 4, c = 2;
System.out.println(a > 2 ? --c : b++);
System.out.println(b = (a!=c ? a : b++));
System.out.println(a > b? b < c? b: 2: 1);
```

R.1 _

b.2 _

C.3 _

D.4 _

E.5 _

F.6 _

G. El código no se compila.

Machine Translated by Google



10. ¿Cuáles son los resultados únicos del siguiente fragmento de código? (Elija todas las que correspondan).

```
altura corta = 1, peso = 3; cebra
corta = (byte) peso * (byte) altura; doble buey = 1 +
altura * 2 + peso; jirafa larga = 1 + 9 %
altura + 1; System.out.println(cebra);
System.out.println(ox);
System.out.println(jirafa);
```

R.1 _

b.2 _

C.3 _

D.4 _

E.5 _

F.6 _

G. El código no se compila.

11. ¿Cuál es el resultado del siguiente código?

```
11: int muestra1 = (2 * 4) % 3; 12:
int muestra2 = 3 * 2 % 3; 13: int
muestra3 = 5 * (1 % 2); 14:
System.out.println(muestra1 + ", " + muestra2 + ", " + muestra3);
```

A. 0, 0, 5 B. 1, 2,

10 C. 2, 1, 5 D. 2,

0, 5 E. 3, 1, 10 F.

3, 2, 6 G. El

código no se

compila.

12. El operador _____ aumenta un valor y devuelve el valor original, mientras que el operador _____ disminuye un valor y devuelve el nuevo valor.

A. post-incremento, post-incremento B. pre-

decremento, post-decremento C. post-

incremento, post-decremento D. post-

incremento, pre-decremento E. pre-

incremento, pre-decremento F. pre-

incremento , post-decremento

Machine Translated by Google



13. ¿Cuál es el resultado del siguiente fragmento de código?

```
booleano soleado = verdadero, lloviendo = falso, domingo =
verdadero; booleano goingToTheStore = soleado y lloviendo ^
domingo; booleano goingToTheZoo = domingo && !
lloviendo; booleano quedarse en casa = !(yendoALaTienda &&
yendoAlZoo); System.out.println(yendoALaTienda + "-" + yendoAlZoo
    + "-" +quedarseEnCasa);
```

A. verdadero-falso-falso

B. falso-verdadero-falso

C. verdadero-verdadero-verdadero

D. falso-verdadero-verdadero

E. falso-falso-falso

F. verdadero-verdadero-falso

G. Ninguno de los anteriores

14. ¿Cuáles de las siguientes afirmaciones son correctas? (Elija todas las que correspondan).

R. El valor de retorno de una expresión de operación de asignación puede ser nulo.

B. El operador de desigualdad (!=) se puede utilizar para comparar objetos.

C. El operador de igualdad (==) se puede utilizar para comparar un valor booleano con un valor numérico. valor.

D. Durante el tiempo de ejecución, & y | Los operadores pueden hacer que sólo el lado izquierdo de la expresión se muestre. ser evaluado.

E. El valor de retorno de una expresión de operación de asignación es el valor de la variable recién asignada.

F. En Java, 0 y false se pueden usar indistintamente.

G. El operador de complemento lógico (!) no se puede utilizar para invertir valores numéricos.

15. ¿Qué operadores toman tres operandos o valores? (Elija todas las que correspondan).

R. =

B. &&

C. *=

D. ? :

E. y

F. ++

G. /

Machine Translated by Google



16. ¿Cuántas líneas del siguiente código contienen errores de compilación?

```
nota int = 1 * 2 + (larga)3; melodía
corta = (byte)(doble)(nota *= 2); canción doble =
melodía; sinfonía flotante =
(flotante)((canción == 1_000f)? canción * 2L: canción);
```

R. 0

B.1 _

c.2 _

D.3 _

E.4 _

17. Dado el siguiente fragmento de código, ¿cuáles son los valores de las variables después de ejecutarlo?
(Elija todas las que correspondan).

```
int boletos tomados = 1; int
boletos vendidos = 3;
entradas vendidas += 1 + entradas tomadas
++; entradas tomadas
*= 2; entradasVendidas += (largo)1;
```

A. las entradas vendidas son 8.

B. entradas tomadas es 2.

C. entradas vendidas son 6.

D. los boletos tomados son 6.

E. entradas vendidas son 7.

F. entradas tomadas es 4.

G. El código no se compila.

18. ¿Cuál de los siguientes se puede utilizar para cambiar el orden de operación en una expresión?
(Elija todas las que correspondan).

R. [ ]

B. < >

C. ( )

D. \ / E. { }

F. --

Machine Translated by Google



19. ¿Cuál es el resultado de ejecutar el siguiente fragmento de código? (Elija todas las que correspondan).

```
3: int inicio = 7; 4:
fin int = 4; 5: fin
+= ++inicio; 6: inicio
= (byte)(Byte.MAX_VALUE + 1);
```

A. el inicio es 0.

B. el inicio es -128.

C. el inicio es 127.

D. el final es 8.

E. el final es 11.

F. el final es 12.

G. El código no se compila.

H. El código se compila pero genera una excepción en tiempo de ejecución.

20. ¿Cuáles de las siguientes afirmaciones sobre operadores unarios son verdaderas? (Elija todas las que correspondan).

R. Los operadores unarios siempre se ejecutan antes que cualquier binario o ternario numérico circundante. operadores.

B. El operador - se puede utilizar para invertir un valor booleano.

C. El operador de preincremento (++) devuelve el valor de la variable antes de que se realice el incremento. aplicado.

D. El operador post-decremento (--) devuelve el valor de la variable antes del decremento Está aplicado.

E. ¡El! El operador no se puede utilizar en valores numéricos.

F. Ninguna de las anteriores

21. ¿Cuál es el resultado de ejecutar el siguiente fragmento de código?

```
int miNúmeroFavorito = 8; int
pájaro = ~miNúmeroFavorito; int
plano = -miNúmeroFavorito; var
superman = pájaro == avión? 5:10;
System.out.println(pájaro + "," + avión + "," + --superman);
```

A. -7,-8,9 B.

-7,-8,10 C. -8,-8,4

D. -8,-8,5 E.

-9,-8,9 F. -9, -8,10

G. Ninguno de los anteriores

Machine Translated by Google

Capítulo 3

# Tomando decisiones

OBJETIVOS DEL EXAMEN OCP CUBIERTOS EN ESTE CAPÍTULO:

Controlar el flujo del programa

- Crear estructuras de control de flujo de programa que incluyan if/else, cambiar declaraciones y expresiones, bucles y break y con declaraciones continuas

Utilizando el enfoque orientado a objetos de Java

- Implementar polimorfismo y diferenciar el tipo de objeto del tipo de referencia. Realizar conversión de tipos, identificar tipos de objetos utilizando el operador de instancia y la coincidencia de patrones.

Machine Translated by Google

Como muchos lenguajes de programación, Java se compone principalmente de variables, operadores y declaraciones reunidos en algún orden lógico. En el último capítulo, cubrimos cómo crear y manipular variables. Sin embargo, escribir software es algo más que gestionar variables; se trata de crear aplicaciones que puedan tomar decisiones inteligentes. En este capítulo, presentamos las diversas declaraciones de toma de decisiones disponibles para usted dentro del idioma. Este conocimiento le permitirá construir funciones complejas y estructuras de clases que verá a lo largo de este libro.

# Crear declaraciones para la toma de decisiones

Los operadores de Java le permiten crear muchas expresiones complejas, pero están limitados en la forma en que pueden controlar el flujo del programa. Imagine que desea que un método se ejecute solo bajo ciertas condiciones que no se pueden evaluar hasta el tiempo de ejecución. Por ejemplo, en días lluviosos, un zoológico debe recordar a sus clientes que traigan un paraguas, o en un día de nieve, es posible que el zoológico deba cerrar. El software no cambia, pero el comportamiento del software sí debería, dependiendo de las entradas suministradas en el momento. En esta sección, analizamos las declaraciones de toma de decisiones que incluyen if y else, junto con la nueva función de coincidencia de patrones.

## Declaraciones y bloques

Como recordará del Capítulo 1, “Bloques de construcción”, una declaración de Java es una unidad de ejecución completa en Java, terminada con un punto y coma (;). En este capítulo, le presentamos varias declaraciones de flujo de control de Java. Las declaraciones de flujo de control dividen el flujo de ejecución mediante el uso de toma de decisiones, bucles y ramificaciones, lo que permite a la aplicación ejecutar selectivamente segmentos particulares de código.

Estas declaraciones se pueden aplicar a expresiones individuales, así como a un bloque de código Java. Como se describe en el Capítulo 1, un bloque de código en Java es un grupo de cero o más declaraciones entre llaves equilibradas ({}) y se puede usar en cualquier lugar donde se permita una sola declaración. Por ejemplo, los dos fragmentos siguientes son equivalentes: el primero es una expresión única y el segundo es un bloque que contiene la misma declaración:

```
// declaración única
patrocinadores++;
```

Machine Translated by Google



```
// Declaración dentro de un bloque
{
    patrocinadores++;
}
```

Una declaración o un bloque a menudo sirve como objetivo de una declaración de toma de decisiones. Para Por ejemplo, podemos anteponer la declaración if de toma de decisiones a estos dos ejemplos:

```
// declaración única
si(boletos tomados > 1)
    patrocinadores++;

// Declaración dentro de un bloque
si(boletos tomados > 1)
{
    patrocinadores++;
}
```

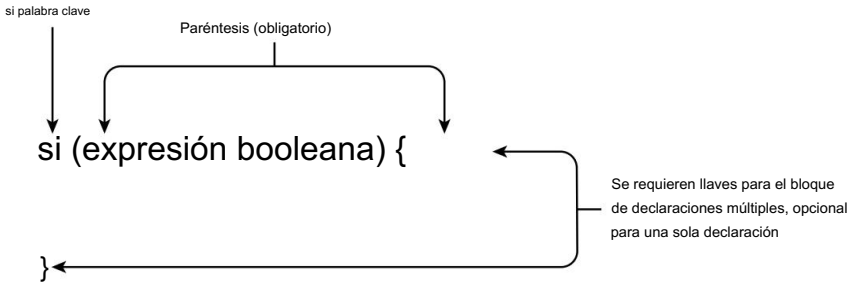
Nuevamente, ambos fragmentos de código son equivalentes. Sólo recuerde que el objetivo de una declaración de toma de decisiones puede ser una sola declaración o un bloque de declaraciones. Durante el resto del capítulo, utilizamos ambos formularios para prepararlo mejor para lo que verá en el examen.

Si bien los dos ejemplos anteriores son equivalentes, a menudo se prefiere el uso estilístico de bloques, incluso si el bloque tiene solo una declaración. La segunda forma tiene la ventaja de que puedes insertar rápidamente nuevas líneas de código en el bloque, sin modificar la estructura circundante.

# La declaración si

A menudo, queremos ejecutar un bloqueo sólo en determinadas circunstancias. La declaración if , como se muestra en la Figura 3.1, logra esto al permitir que nuestra aplicación ejecute un bloque de código particular si y solo si una expresión booleana se evalúa como verdadera en tiempo de ejecución.

FIGURA 3.1 La estructura de una declaración if



Machine Translated by Google



Por ejemplo, imaginemos que tuviéramos una función que usara la hora del día, un valor entero de 0 a 23, para mostrar un mensaje al usuario:

```
si(horaDía < 11)
    System.out.println("Buenos días");
```

Si la hora del día es inferior a las 11, se mostrará el mensaje. Ahora digamos que también queremos incrementar algún valor, MorningGreetingCount, cada vez que se imprime el saludo. Podríamos escribir la declaración if dos veces, pero afortunadamente Java nos ofrece un enfoque más natural usando un bloque:

```
if(horaDía < 11) {
    System.out.println("Buenos días");
    mañanaSaludoCount++;
}
```

**Ver sangría y llaves**

Un área en la que los redactores del examen intentarán hacerle tropezar es si las declaraciones sin llaves ({}). Por ejemplo, eche un vistazo a esta forma ligeramente modificada de nuestro ejemplo:

```
si(horaDía < 11)
    System.out.println("Buenos días");
    mañanaSaludoCount++;
```

Según la sangría, es posible que se sienta inclinado a pensar que la variable MorningGreetingCount solo se incrementará si hourOfDay es menor que 11, pero eso no es lo que hace este código. Ejecutará la declaración de impresión solo si se cumple la condición, pero siempre ejecutará la operación de incremento.

Recuerde que en Java, a diferencia de otros lenguajes de programación, las pestañas son solo espacios en blanco y no se evalúan como parte de la ejecución. Cuando vea una declaración de flujo de control en una pregunta, asegúrese de rastrear las llaves de apertura y cierre del bloque, ignorando cualquier sangría que pueda encontrar.

## La declaración else

Ampliemos un poco nuestro ejemplo. ¿Qué pasa si queremos mostrar un mensaje diferente si son las 11 a.m. o más tarde? ¿Podemos hacerlo usando sólo las herramientas que tenemos? ¡Por supuesto que podemos!

```
if(horaDía < 11) {
    System.out.println("Buenos días");
```

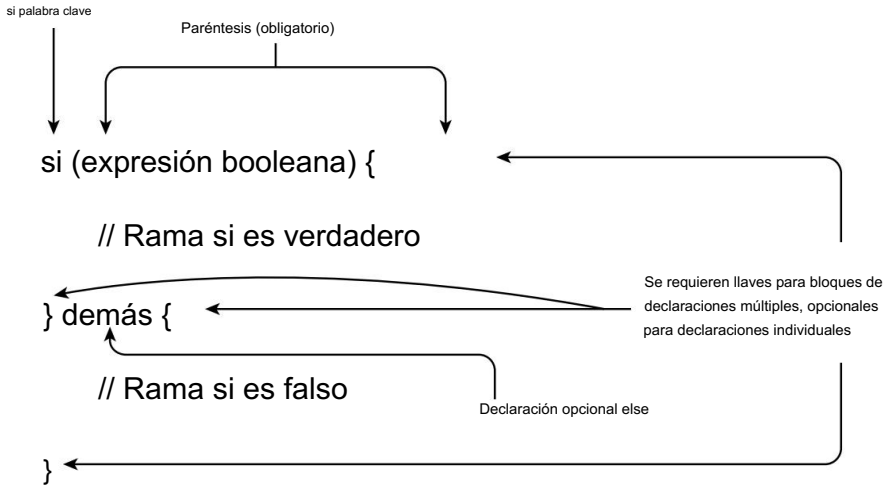
Machine Translated by Google



```
} if(horaDía>= 11)
    { System.out.println("Buenas tardes");
}
```

Sin embargo, esto parece un poco redundante, ya que estamos realizando una evaluación en hourOfDay . dos veces. Afortunadamente, Java nos ofrece un enfoque más útil en forma de declaración else , como se muestra en la Figura 3.2.

FIGURA 3.2 La estructura de una declaración else




Volvamos a este ejemplo:

```
if(horaDía < 11)
    { System.out.println("Buenos días");
} else System.out.println("Buenas tardes");
```

Ahora nuestro código realmente se bifurca entre una de las dos opciones posibles, y la evaluación booleana se realiza solo una vez. El operador else toma una declaración o un bloque de declaraciones, de la misma manera que la declaración if . De manera similar, podemos agregar declaraciones if adicionales a un bloque else para llegar a un ejemplo más refinado:

```
if(horaDía < 11)
    { System.out.println("Buenos días"); } else
if(horaDía < 15) {
    System.out.println("Buenas tardes");
} demás {
    System.out.println("Buenas noches");
}
```

Machine Translated by Google



En este ejemplo, el proceso Java continuará la ejecución hasta que encuentre una declaración if que se evalúe como verdadera. Si ninguna de las dos primeras expresiones es verdadera, ejecutará el código final del bloque else .

### Verificar que la declaración if se evalúe como una expresión booleana

Otra forma común en que el examen puede intentar desviarlo es proporcionando un código donde la expresión booleana dentro de la declaración if no sea en realidad una expresión booleana . Por ejemplo, eche un vistazo a las siguientes líneas de código:

```
int horaDía = 1;
if(hourOfDay) { // NO COMPILA
    ...
}
```

Esta declaración puede ser válida en otros lenguajes de programación y scripting, pero no en Java, donde 0 y 1 no se consideran valores booleanos .

## Código de acortamiento con coincidencia de patrones

Java 16 introdujo oficialmente la coincidencia de patrones con declaraciones if y el operador instancia de . La coincidencia de patrones es una técnica para controlar el flujo de un programa que solo ejecuta una sección de código que cumple con ciertos criterios. Se utiliza junto con declaraciones if para un mayor control del programa.

Si la coincidencia de patrones es nueva para usted, tenga cuidado de no confundirla con la clase Java Pattern o las expresiones regulares (regex). Si bien la coincidencia de patrones puede incluir el uso de expresiones regulares para el filtrado, son conceptos no relacionados.

La coincidencia de patrones es una nueva herramienta a su disposición para reducir el texto repetitivo en su código. El código repetitivo es código que tiende a duplicarse en una sección de código una y otra vez de manera similar. Muchas de las mejoras más recientes del lenguaje Java se centran en reducir el código repetitivo.

Para comprender por qué se agregó esta herramienta, considere el siguiente código que toma un Número instancia y lo compara con el valor 5. Si no ha visto Número o Integer, solo necesita saber que Integer hereda de Número por ahora. ¡Los verás mucho en este libro!

```
void compareIntegers(Número número) {
    if(número instancia de entero) {
        Datos enteros = número (entero);
```

Machine Translated by Google



```
        System.out.print(data.compareTo(5));
    }
}
```

La conversión es necesaria ya que el método compareTo() está definido en Integer, pero no en Number.

Código que primero verifica si una variable es de un tipo particular y luego la convierte inmediatamente en ese tipo es extremadamente común en el mundo Java. Es tan común que los autores de Java decidieron implementar una sintaxis más corta:

```
void compareIntegers(Número número) {
    if(número de instancia de datos enteros) {
        System.out.print(data.compareTo(5));
    }
}
```

Los datos variables en este ejemplo se denominan variables de patrón. Tenga en cuenta que este código también evita cualquier posible ClassCastException porque la operación de conversión se ejecuta solo si el operador implícito de instancia devuelve verdadero.

### Reasignación de variables de patrón

Si bien es posible, es una mala práctica reasignar una variable de patrón, ya que hacerlo puede generar ambigüedad sobre lo que está y lo que no está dentro del alcance.

```
    if(número de instancia de datos enteros) {
        datos = 10;
    }
```

La reasignación se puede evitar con un modificador final , pero es mejor no reasignar la variable en absoluto.

```
    if(número de instancia de datos enteros finales ) {
        datos = 10; // NO COMPILA
    }
```

## Variables y expresiones de patrón

La coincidencia de patrones incluye expresiones que se pueden utilizar para filtrar datos, como en el siguiente ejemplo:

```
void printIntegersGreaterThan5(Número número) {
    if(número de instancia de datos enteros && data.compareTo(5)>0)
        System.out.print(datos);
}
```

Machine Translated by Google



Podemos aplicar varios filtros o patrones para que la sentencia if se ejecute sólo en circunstancias específicas. Observe que estamos usando la variable de patrón en una expresión en la misma línea en la que se declara.

## Subtipos

El tipo de variable de patrón debe ser un subtipo de la variable en el lado izquierdo de la expresión. Tampoco puede ser del mismo tipo. Esta regla no existe para la instancia tradicional de expresiones de operador, sin embargo. Considere los dos usos siguientes del operador instancia de :

```
Valor entero = 123;
if(valor instancia de entero) {}
if(valor instancia de datos enteros) {} // NO COMPILA
```

Mientras que la segunda línea se compila, la última línea no se compila debido a la coincidencia de patrones. requiere que el tipo de variable de patrón Integer sea un subtipo estricto de Integer.

### Limitaciones de la aplicación de subtipos

El compilador tiene algunas limitaciones a la hora de aplicar tipos de coincidencia de patrones cuando mezclamos clases e interfaces, lo que tendrá más sentido después de leer el Capítulo 7, "Más allá de las clases". Por ejemplo, dada la lista de interfaz y número de clase no final , esto se compila aunque no estén relacionados:

```
Valor numérico = 123;
if(valor instancia de lista) {}
if(valor instancia de datos de lista) {}
```

## Alcance del flujo

El compilador aplica el alcance del flujo cuando trabaja con la coincidencia de patrones. Alcance del flujo significa que la variable solo está dentro del alcance cuando el compilador puede determinar definitivamente su tipo. El alcance del flujo se diferencia de cualquier otro tipo de alcance en que no es estrictamente jerárquico como el alcance de postura, clase o local. Lo determina el compilador en función de la ramificación y el flujo del programa.

Dada esta información, ¿puedes ver por qué lo siguiente no se compila?

```
void printIntegersOrNumbersGreaterThan5(Número número) {
    if(número de instancia de datos enteros || data.compareTo(5)>0)
        System.out.print(datos);
}
```

Machine Translated by Google



Si la entrada no hereda Integer, la variable de datos no está definida. Dado que el compilador no puede garantizar que los datos sean una instancia de Integer, los datos no están dentro del alcance y el código no se compila.

¿Qué pasa con este ejemplo?

```
void printIntegerTwice(Número número) {
    si (número de instancia de datos enteros)
        System.out.print(data.intValue());
    System.out.print(data.intValue()); // NO COMPILA
}
```

Dado que es posible que la entrada no haya heredado Integer, los datos ya no están dentro del alcance después de la declaración if . Ah, entonces podrías estar pensando que la variable de patrón solo está dentro del alcance dentro de la declaración if , ¿verdad? ¡Bueno no exactamente! Considere el siguiente ejemplo que compila:

```
void printOnlyIntegers(Número número) {
    if (!(número de instancia de datos enteros))
        devolver;
    System.out.print(data.intValue());
}
```

Puede que le sorprenda saber que este código se compila. ¡Eek! ¿Que esta pasando aqui? El método devuelve si la entrada no hereda Integer. Esto significa que cuando se llega a la última línea del método, la entrada debe heredar Integer y, por lo tanto, los datos permanecen dentro del alcance incluso después de que finaliza la declaración if .

### Alcance del flujo y otras ramas

Si el último ejemplo de código te confunde, no te preocupes: ¡no estás solo! Otra forma de pensarlo es reescribir la lógica a algo equivalente que use una declaración else :

```
void printOnlyIntegers(Número número) {
    if (!(número de instancia de datos enteros))
        devolver;
    demás
        System.out.print(data.intValue());
}
```

Ahora podemos ir un paso más allá e invertir las ramas if y else invirtiendo la expresión booleana :

```
void printOnlyIntegers(Número número) {if (número
    instancia de datos enteros)
```

Machine Translated by Google



```
            System.out.print(data.intValue());
        demás
            devolver;
    }
```

Nuestro nuevo código es equivalente al original y demuestra mejor cómo el compilador pudo determinar que los datos estaban dentro del alcance solo cuando el número es un número entero.

Asegúrese de comprender cómo funciona el alcance del flujo. En particular, es posible utilizar una variable de patrón fuera de la declaración if , pero sólo cuando el compilador pueda determinar definitivamente su tipo.

# Aplicar declaraciones de cambio

¿Qué pasa si tenemos muchas ramas o caminos posibles para un solo valor? Por ejemplo, es posible que queramos imprimir un mensaje diferente según el día de la semana. Ciertamente podríamos lograr esto con una combinación de siete declaraciones if o else , pero eso tiende a crear un código largo, difícil de leer y, a menudo, no divertido de mantener:

```
public void printDayOfWeek(int día) {
    si(día == 0)
        System.out.print("Domingo");
    de lo contrario si (día == 1)
        System.out.print("Lunes");
    de lo contrario si (día == 2)
        System.out.print("Martes");
    de lo contrario si (día == 3)
        System.out.print("Miércoles");
    ...
}
```

Afortunadamente, Java, junto con muchos otros lenguajes, ofrece un enfoque más limpio. En esta sección presentamos la declaración switch , junto con la expresión switch más nueva para controlar el flujo del programa.

## La declaración de cambio

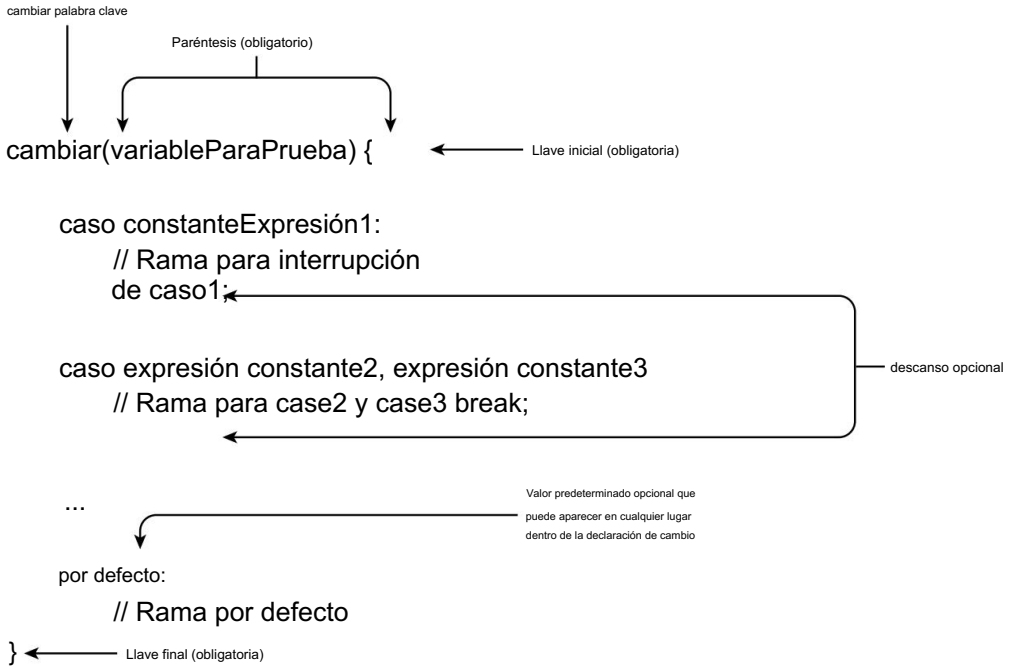
Una declaración de cambio , como se muestra en la Figura 3.3, es una estructura compleja de toma de decisiones en la que se evalúa un único valor y el flujo se redirige a la primera rama coincidente, conocida como declaración de caso . Si no se encuentra ninguna declaración de caso que coincida con el valor, se agregará una opción opcional.

Machine Translated by Google



Se llamará a la declaración predeterminada . Si dicha opción predeterminada no está disponible, se omitirá toda la declaración de cambio . Observe en la Figura 3.3 que los valores de caso se pueden combinar en una sola declaración de caso usando comas.

FIGURA 3.3 La estructura de una declaración de cambio




Debido a que las declaraciones de cambio pueden ser más largas que la mayoría de las declaraciones de toma de decisiones, la El examen puede presentar una sintaxis de cambio no válida para ver si está prestando atención.

### Combinando valores de casos

¿Notas algo nuevo en la Figura 3.3? A partir de Java 14, los valores de casos ahora se pueden combinar:

```
cambiar(animal) {
    caso 1,2: System.out.print("León"); caso 3:
    System.out.print("Tigre");
}
```

Machine Translated by Google



Antes de Java 14, el código equivalente habría sido el siguiente:

```
cambiar(animal) {
    caso 1: caso 2: System.out.print("León");
    caso 3:         System.out.print("Tigre");
}
```

Como verá en breve, las expresiones de cambio pueden reducir el código repetitivo aún más.

Vea si puede descubrir por qué cada una de las siguientes declaraciones de cambio no se compila:

```
mes entero = 5;

cambiar de mes { // NO COMPILA
    caso 1: System.out.print("Enero");
}

cambiar(mes) // NO COMPILA
    caso 1: System.out.print("Enero");

switch(mes) { caso
    1: 2: System.out.print("Enero"); // NO COMPILA
}
```

La primera declaración de cambio no se compila porque faltan paréntesis alrededor de la variable de cambio . La segunda declaración no se compila porque faltan llaves alrededor del cuerpo del conmutador . La tercera declaración no se compila porque se debe usar una coma (,) para separar declaraciones de casos combinadas , no dos puntos (:).

Una última nota que debe tener en cuenta para el examen: no se requiere una declaración de cambio para contener declaraciones de casos . Por ejemplo, esta afirmación es perfectamente válida:

```
cambiar(mes) {}
```

Volviendo a nuestro método printDayOfWeek() , podemos reescribirlo para usar un estado de cambio . ment en lugar de declaraciones if/else :

```
public void printDayOfWeek(int día) { cambiar(día)
    {
        caso 0:
            System.out.print("Domingo"); romper;

        caso 1:
            System.out.print("Lunes"); romper;

        caso 2:
            System.out.print("Martes"); romper;
```

Machine Translated by Google



```
        caso 3:
            System.out.print("Miércoles");
            romper;
        caso 4:
            System.out.print("Jueves");
            romper;
        caso 5:
            System.out.print("Viernes");
            romper;
        caso 6:
            System.out.print("Sábado");
            romper;
        por defecto:
            System.out.print("Valor no válido");
            romper;
    } }
```

Para simplificar, simplemente imprimimos un mensaje si el valor no es válido. Si conoce las excepciones o ya ha leído el Capítulo 11, “Excepciones y localización”, podría tener más sentido generar una excepción en la rama predeterminada si no se encuentra ninguna coincidencia.

## Salir con declaraciones de descanso

Si echa un vistazo a nuestra implementación anterior de printDayOfWeek() , verá una declaración de interrupción al final de cada caso y sección predeterminada . Una declaración de interrupción finaliza la declaración de cambio y devuelve el control de flujo al proceso adjunto. En pocas palabras, finaliza la declaración de cambio inmediatamente.

Las declaraciones break son opcionales, pero sin ellas el código ejecutará cada rama después de una declaración case coincidente , incluidas las declaraciones predeterminadas que encuentre. sin descanso declaraciones en cada rama, el orden de las declaraciones de caso y predeterminado ahora es extremadamente importante. ¿Qué crees que se imprime lo siguiente cuando se llama a printSeason(2) ?

```
printSeason public void(int mes) {
    cambiar(mes) {
        caso 1, 2, 3: caso      System.out.print("Invierno");
        4, 5, 6:                System.out.print("Primavera");
        por defecto:            System.out.print("Desconocido");
        caso 7, 8, 9: caso      System.out.print("Verano");
        10, 11, 12: System.out.print("Caída");
    } }
```

¡Imprime todo!

```
InviernoPrimaveraDesconocidoVeranoOtoño
```

Machine Translated by Google



Coincide con la primera declaración de caso y ejecuta todas las ramas en el orden en que se encuentran, incluida la declaración predeterminada . Es común, aunque ciertamente no obligatorio, utilizar una declaración de interrupción después de cada declaración de caso .

A los creadores del examen les gustan los ejemplos de cambio a los que les falta un descanso ¡declaraciones! Al evaluar declaraciones de cambio en el examen, siempre considere que se pueden visitar varias ramas en una sola ejecución.

## Seleccionar tipos de datos del interruptor

Como se muestra en la Figura 3.3, una declaración de cambio tiene una variable de destino que no se evalúa hasta el tiempo de ejecución. El tipo de este destino puede incluir tipos de datos primitivos seleccionados (int, byte, short, char) y sus clases contenedoras asociadas (Entero, Byte, Corto, Carácter). La siguiente es una lista de todos los tipos de datos admitidos por declaraciones de cambio :

- int y entero
- byte y byte
- corto y corto
- caracteres y caracteres
- Cadena
- valores de enumeración
- var (si el tipo se resuelve en uno de los tipos anteriores)

Para este capítulo, sólo necesita saber que una enumeración, o enumeración, representa un valor fijo. conjunto de constantes, como días de la semana, meses del año, etc. Cubrimos las enumeraciones con más detalle en el Capítulo 7, y mostramos cómo pueden definir variables, métodos y constructores.

Tenga en cuenta que booleano, largo, flotante y doble están excluidos de las declaraciones de cambio , al igual que sus clases booleanas, largas, flotantes y dobles asociadas. Las razones son variadas, como que los valores booleanos tengan un rango de valores demasiado pequeño y que los números de punto flotante tengan un rango de valores bastante amplio. Sin embargo, para el examen, sólo necesita saber que no están permitidos en las declaraciones de cambio .

## Determinación de valores de casos aceptables

No se puede utilizar cualquier variable o valor en una declaración de caso . Primero, los valores en cada declaración de caso deben ser valores constantes en tiempo de compilación del mismo tipo de datos que el cambio. valor. Esto significa que solo puede utilizar literales, constantes de enumeración o variables constantes finales del mismo tipo de datos. Por constante final queremos decir que la variable debe marcarse con el modificador final e inicializarse con un valor literal en la misma expresión en la que se declara. Por ejemplo, no puede tener un valor de declaración de caso que requiera ejecutar un

Machine Translated by Google



método en tiempo de ejecución, incluso si ese método siempre devuelve el mismo valor. Por estos motivos, solo se compilan las declaraciones del primer y último caso del siguiente ejemplo:

```
final int getCookies() { retorno 4; }
void alimentarAnimales() {
    plátanos int finales = 1;
    int manzanas = 2;
    int númeroDeAnimales = 3;
    cookies int finales = getCookies();
    cambiar(númeroDeAnimales) {
        plátanos caso:
        caso de manzanas:          // NO COMPILA
        case getCookies(): // NO COMPILA
        galletas de caso:          // NO COMPILA
        caso 3 * 5:
    } }
```

La variable bananas está marcada como final y su valor se conoce en tiempo de compilación, por lo que es válido. La variable manzanas no está marcada como final, aunque se conoce su valor, por lo que no está permitida. Las siguientes dos declaraciones de caso , con valores getCookies() y cookies, no se compilan porque los métodos no se evalúan hasta el tiempo de ejecución, por lo que no se pueden usar como el valor de una declaración de caso , incluso si uno de los valores se almacena en una variable final . . La última declaración de caso , con valor 3 * 5, se compila, ya que se permiten expresiones como valores de caso , siempre que el valor se pueda resolver en tiempo de compilación. También deben poder encajar en el tipo de datos del conmutador sin una conversión explícita. Entraremos en eso con más detalle en breve.

A continuación, el tipo de datos para las declaraciones de caso debe coincidir con el tipo de datos de la variable de cambio . Por ejemplo, no puede tener una declaración case de tipo String si la variable de declaración switch es de tipo int, ya que los tipos son incomparables.

## La expresión del interruptor

Nuestra segunda implementación de printDayOfWeek() fue mejorada pero aún es bastante larga. Observe que había mucho código repetitivo, junto con numerosas declaraciones de interrupción . ¿Podemos hacerlo mejor? Sí, gracias a las nuevas expresiones de cambio que se agregaron oficialmente a Java 14.

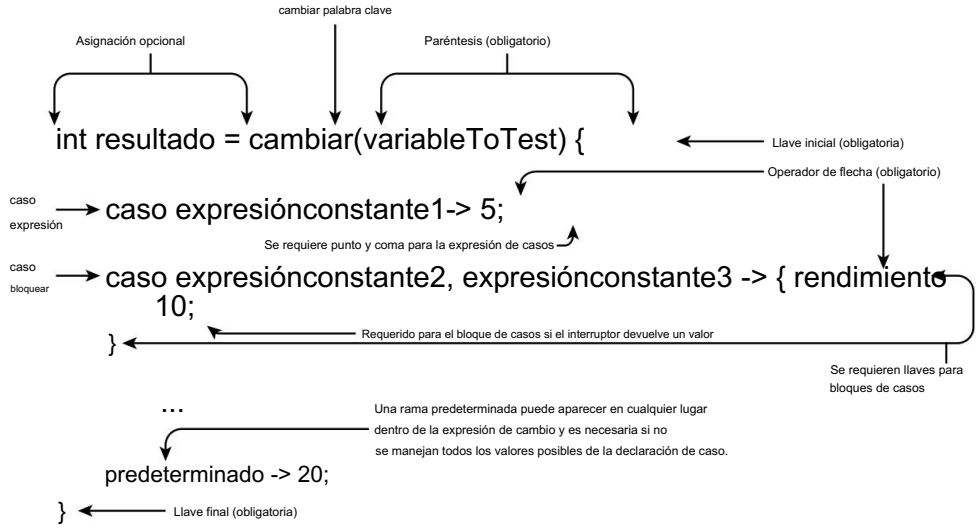
Una expresión de cambio es una forma mucho más compacta de declaración de cambio , capaz de devolver un valor. Eche un vistazo a la nueva sintaxis en la Figura 3.4.

Debido a que una expresión de cambio es una forma compacta, ¡están sucediendo muchas cosas en la Figura 3.4! Para empezar, ahora podemos asignar el resultado de una expresión de cambio a un resultado variable . Para que esto funcione, todas las ramas de casos y predeterminadas deben devolver un tipo de datos que sea compatible con la asignación. La expresión switch admite dos tipos de ramas: una expresión y un bloque. Cada uno tiene diferentes reglas sintácticas sobre cómo debe crearse. Más sobre estos temas en breve.

Machine Translated by Google



FIGURA 3.4 La estructura de una expresión de cambio




Al igual que una declaración de cambio tradicional , una expresión de cambio admite cero o muchas ramas de casos y una rama predeterminada opcional . Ambos también admiten la nueva característica que permite combinar valores de casos con una sola declaración de caso usando comas. Sin embargo, a diferencia de una declaración de cambio tradicional , las expresiones de cambio tienen reglas especiales sobre cuándo se requiere la rama predeterminada .

Recuerde del Capítulo 2, “Operadores”, que -> es el operador de flecha. Si bien el operador de flecha se usa comúnmente en expresiones lambda, cuando se usa en una expresión de cambio , las ramas de casos no son lambdas.

Podemos reescribir nuestro método printDayOfWeek() anterior de una manera mucho más concisa usando expresiones de caso :

```
public void printDayOfWeek(int día) { var
    resultado = switch(día) { caso 0
        -> "Domingo"; caso 1
        -> "lunes"; caso 2 ->
        "martes"; caso 3 ->
        "miércoles"; caso 4 ->
        "jueves"; caso 5 ->
        "viernes"; caso 6 ->
        "sábado";
```

Machine Translated by Google



```
        predeterminado -> "Valor no válido";
    };
    System.out.print(resultado);
}
```

Compare este código con la declaración de cambio que escribimos anteriormente. Ambos realizan la misma tarea, pero se ha eliminado gran parte del código repetitivo, dejando el comportamiento que más nos interesa.

Observe que se requiere un punto y coma después de cada expresión de cambio . Por ejemplo, el siguiente código no se compila. ¿Cuantos puntos y coma faltan?

```
resultado var = cambiar(oso) {
    caso 30 -> "Grizzly"
    predeterminado -> "Panda"
}
```

La respuesta es tres. Cada caso o expresión predeterminada requiere un punto y coma, además de la asignación misma. Lo siguiente corrige el código:

```
resultado var = cambiar(oso) {
    caso 30 -> "Oso pardo";
    predeterminado -> "Panda";
};
```

Como se muestra en la Figura 3.4, las declaraciones de caso pueden tomar múltiples valores, separados por comas. Reescribamos nuestro método printSeason() de antes usando una expresión de cambio :

```
printSeason public void(int mes) {
    cambiar(mes) {
        caso 1, 2, 3          -> System.out.print("Invierno");
        caso 4, 5, 6          -> System.out.print("Primavera");
        caso 7, 8, 9          -> System.out.print("Verano");
        caso 10, 11, 12 -> System.out.print("Caída");
    } }
```

Al llamar a printSeason(2) se imprime el valor único Winter. Esta vez no tenemos que preocuparnos por las declaraciones de interrupción , ya que solo se ejecuta una rama.

La mayoría de las veces, una expresión de cambio devuelve un valor, aunque printSeason() muestra una en la que el tipo de retorno es nulo. Como el tipo es nulo, no se puede asignar a una variable. En el examen, es más probable que vea una expresión de cambio que devuelva un valor, pero debe tener en cuenta que es posible.

Machine Translated by Google



Todas las reglas anteriores sobre tipos de datos de cambio y valores de casos aún se aplican, aunque tenemos algunas reglas nuevas. No se preocupe si estas reglas son nuevas para usted o si nunca antes ha visto la palabra clave de rendimiento ; los discutiremos en las siguientes secciones.

1. Todas las ramas de una expresión de cambio que no generan una excepción deben devolver un tipo de datos consistente (si la expresión de cambio devuelve un valor).
2. Si la expresión de cambio devuelve un valor, entonces cada rama que no sea una expresión debe arrojar un valor.
3. Se requiere una rama predeterminada a menos que se cubran todos los casos o no se devuelva ningún valor.

Cubriremos la última regla en breve, pero observe que nuestro ejemplo printSeason() no contiene una rama predeterminada . Dado que la expresión de cambio no devuelve un valor ni lo asigna a una variable, es completamente opcional.

Java 17 también admite la coincidencia de patrones dentro de expresiones de cambio , pero como se trata de una función de Vista previa, no está dentro del alcance del examen.

## Devolver tipos de datos consistentes

La primera regla para utilizar una expresión de cambio es probablemente la más sencilla. No puede devolver tipos de datos aleatorios o incompatibles. Por ejemplo, ¿puedes ver por qué tres de las líneas del siguiente código no se compilan?

```
medida interna = 10;
int tamaño = cambiar(medida) {
    caso 5 -> 1;
    caso 10 -> (corto)2;
    predeterminado -> 5;
    caso 20 -> "3";          // NO COMPILA
    caja 40 -> 4L;           // NO COMPILA
    caso 50 -> nulo; // NO COMPILA
};
```

Observe que la segunda expresión de caso devuelve un short, pero eso se puede convertir implícitamente a un int. De esta manera, los valores deben ser coherentes con el tamaño, pero no todos tienen que ser del mismo tipo de datos. Las últimas tres expresiones case no se compilan porque cada una devuelve un tipo que no se puede asignar a la variable int .

## Aplicar un bloque de caso

Una expresión de cambio admite tanto una expresión como un bloque en mayúsculas y minúsculas y por defecto sucursales. Al igual que un bloque normal, un bloque de caso es aquel que está rodeado por llaves ({}). También incluye una declaración de rendimiento si la expresión de cambio devuelve un valor. Por ejemplo, lo siguiente utiliza una combinación de expresiones de casos y bloques:

Machine Translated by Google



```
pez int = 5;
longitud interna = 12;
var nombre = cambiar(pez) {
    caso 1 -> "Pez dorado";
    caso 2 -> {rendimiento "Trucha";}
    caso 3 -> {
        if(longitud > 10) produce "Pez globo";
        de lo contrario, obtenga "Verde";
    }
    predeterminado -> "Pez espada";
};
```

La palabra clave de rendimiento es equivalente a una declaración de retorno dentro de una expresión de cambio y se usa para evitar ambigüedad sobre si pretendía salir del bloque o método alrededor de la expresión de cambio .

Refiriéndose a nuestra segunda regla para expresiones de cambio , las declaraciones de rendimiento no son opcionales si el La declaración de cambio devuelve un valor. ¿Puedes ver por qué las siguientes líneas no se compilan?

```
10: pez int = 5;
11: longitud interna = 12;
12: var nombre = cambiar(pez) {
13:     caso 1 -> "Pez dorado";
14:     caso 2 -> {} // NO COMPILA
15:     caso 3 -> {
dieciséis:         if(longitud > 10) produce "Pez globo";
17:     } // NO COMPILA
18:     predeterminado -> "Pez espada";
19: };
```

La línea 14 no se compila porque no devuelve un valor usando rendimiento. La línea 17 tampoco se compila. Si bien el código devuelve un valor para una longitud mayor que 10, no devuelve un valor si la longitud es menor o igual a 10. No importa que la longitud esté configurada en 12; todas las ramas deben producir un valor dentro del bloque de casos .

## Ver punto y coma en expresiones de cambio

A diferencia de una declaración de cambio normal , una expresión de cambio se puede usar con el operador de asignación y requiere un punto y coma al hacerlo. Además, se requieren puntos y coma para las expresiones de casos , pero no se pueden usar con bloques de casos .

```
var nombre = cambiar(pez) {
    caso 1 -> "Pez dorado"          // NO COMPILA (falta punto y coma)
```

Machine Translated by Google



```
        caso 2 -> {rendimiento "Trucha";}; // NO COMPILA (punto y coma adicional)
        ...
    } // NO COMPILA (falta punto y coma)
```

Un poco confuso, ¿verdad? Es sólo una de esas cosas que tienes que entrenarte para detectar en el examen.

## Cubriendo todos los valores posibles

La última regla sobre las expresiones de cambio es probablemente con la que es más probable que el examen intente engañarlo: una expresión de cambio que devuelve un valor debe manejar todos los valores de entrada posibles. Y como viste antes, cuando no devuelve un valor, es opcional.

Probemos esto. Dado el siguiente código, ¿cuál es el valor de tipo si canis es 5?

```
Tipo de cadena = switch(canis) { // NO COMPILA
    caso 1 -> "perro";
    caso 2 -> "lobo";
    caso 3 -> "coyote";
};
```

No hay una rama de caso que cubra 5 (o 4, -1, 0, etc.), entonces, ¿la expresión de cambio debería devolver nulo, la cadena vacía, indefinido o algún otro valor? Al agregar expresiones de cambio al lenguaje Java, los autores decidieron que este comportamiento no sería compatible. Cada expresión de cambio debe manejar todos los valores posibles de la variable de cambio . Como desarrollador, hay dos formas de abordar esto:

- Agregar una rama predeterminada .
- Si la expresión de cambio toma un valor de enumeración, agregue una rama de caso para cada valor de enumeración posible.

En la práctica, la primera solución es la que se utiliza con mayor frecuencia. La segunda solución se aplica sólo a las expresiones de cambio que toman una enumeración. Puede intentar escribir declaraciones de casos para todos los valores int posibles , ¡pero le prometemos que no funciona! El compilador no permite incluso tipos más pequeños como byte , a pesar de que solo hay 256 valores posibles.

Para enumeraciones, la segunda solución funciona bien cuando el número de valores de enumeración es relativamente pequeño. Por ejemplo, considere la siguiente definición y método de enumeración:

```
enum Temporada {INVIERNO, PRIMAVERA, VERANO, OTOÑO}

String getWeather (valor de temporada) {
    interruptor de retorno (valor) {
        caso INVIERNO -> "Frío";
        caso PRIMAVERA -> "Lluvioso";
        caso VERANO -> "Caliente";
        caso OTOÑO -> "Cálido";
    };
}
```

Machine Translated by Google



Dado que se cubren todas las posibles permutaciones de Temporada , no se requiere una rama predeterminada en esta expresión de cambio . Sin embargo, puedes incluir una rama predeterminada opcional , incluso si cubres todos los valores conocidos.

¿Qué pasa si usas una enumeración con tres valores y luego alguien agrega un cuarto valor? Cualquier expresión de cambio que utilice la enumeración sin una rama predeterminada de repente no se podrá compilar. Si esto se hiciera con frecuencia, ¡es posible que tenga mucho código que corregir! Por este motivo, considere incluir una rama predeterminada en cada expresión de cambio , incluso aquellas que involucran valores de enumeración.

# Escribir bucles while

Una práctica común al escribir software es realizar la misma tarea varias veces.
Podrías usar las estructuras de decisión que hemos presentado hasta ahora para lograr esto, pero será una cadena bastante larga de declaraciones if o else , especialmente si tienes que ejecutar lo mismo 100 veces o más.

¡Entra en bucles! Un bucle es una estructura de control repetitiva que puede ejecutar una declaración de código varias veces seguidas. Al utilizar variables a las que se les pueden asignar nuevos valores, cada repetición de la declaración puede ser diferente. El siguiente bucle se ejecuta exactamente 10 veces:

```
contador int = 0;
mientras (contador < 10) {
    precio doble = contador * 10;
    System.out.println(precio);
    contador++;
}
```

Si no sigue este código, no entre en pánico: lo cubriremos en breve. En esta sección, analizaremos el bucle while y sus dos formas. En la siguiente sección, pasamos a los bucles for , que tienen sus raíces en los bucles while .

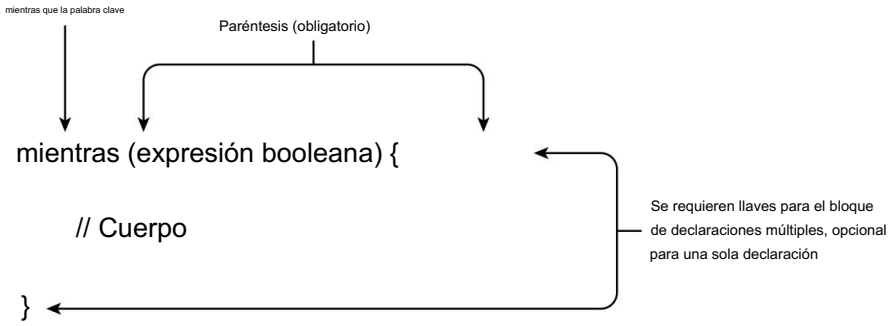
## La declaración mientras

La estructura de control repetitiva más simple en Java es la declaración while , descrita en la Figura 3.5. Como todas las estructuras de control de repetición, tiene una condición de terminación, implementada como una expresión booleana , que continuará mientras la expresión se evalúe como verdadera.

Machine Translated by Google



FIGURA 3.5 La estructura de una declaración while




Como se muestra en la Figura 3.5, un bucle while es similar a una declaración if en que está compuesto de una expresión booleana y una declaración, o un bloque de declaraciones. Durante la ejecución, la expresión booleana se evalúa antes de cada iteración del bucle y sale si la evaluación devuelve falso.

Veamos cómo se puede usar un bucle para modelar un ratón comiendo:

```
int habitaciónEnBelly = 5;
public void eatCheese(int bocadosDeQueso) {
    mientras (bitesOfCheese > 0 && roomInBelly > 0) {
        bocadosDeQueso--;
        roomInBelly--;
    }
    System.out.println(bitesOfCheese+"quedan trozos de queso");
}
```

Este método toma una cantidad de comida (en este caso, queso) y continúa hasta que el ratón no tiene espacio en su vientre o no queda comida para comer. Con cada iteración del ciclo, el ratón "come" un bocado de comida y pierde una parte de su vientre. Al utilizar una declaración booleana compuesta , se asegura de que el ciclo while pueda finalizar para cualquiera de las condiciones.

Una cosa para recordar es que un bucle while puede terminar después de su primera evaluación de la expresión booleana . Por ejemplo, ¡cuántas veces no está lleno! impreso en el siguiente ejemplo?

```
int completo = 5;
mientras (completo < 5) {
    System.out.println("¡No lleno!");
    completo++;
}
```

¿La respuesta? ¡Cero! En la primera iteración del bucle, se alcanza la condición y el bucle sale. Es por eso que los bucles while se usan a menudo en lugares donde se espera cero o más ejecuciones del bucle. En pocas palabras, es posible que el cuerpo del bucle no se ejecute en absoluto o que se ejecute muchas veces.

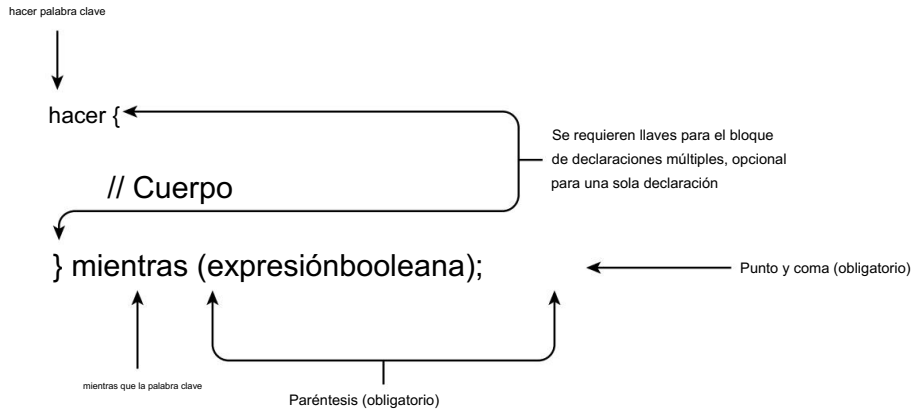
Machine Translated by Google



## La declaración hacer/mientras

La segunda forma que puede adoptar un bucle while se llama bucle do/ while , que, al igual que un bucle while , es una estructura de control de repetición con una condición de terminación y una declaración, o un bloque de declaraciones, como se muestra en la Figura 3.6.

FIGURA 3.6 La estructura de una declaración do/ while




Sin embargo, a diferencia de un bucle while , un bucle do/ while garantiza que la declaración o bloque se ejecutará al menos una vez. Por ejemplo, ¿cuál es el resultado de las siguientes declaraciones?

```
int lagarto = 0;
hacer {
    lagarto++;
} mientras (falso);
System.out.println(lagarto); // 1
```

Java ejecutará primero el bloque de instrucciones y luego verificará la condición del bucle. Incluso aunque el ciclo sale de inmediato, el bloque de instrucciones aún se ejecuta una vez y el programa imprime 1.

## Bucles infinitos

¡Lo más importante que debe tener en cuenta cuando utilice estructuras de control de repetición es asegurarse de que siempre terminen! No terminar un bucle puede provocar numerosos problemas en la práctica, incluidas excepciones de desbordamiento, pérdidas de memoria, rendimiento lento e incluso datos incorrectos. Echemos un vistazo a un ejemplo:

```
pluma int = 2;
int cerdos = 5;
mientras(bolígrafo < 10)
    cerdos++;
```

Machine Translated by Google



Quizás notes un problema evidente con esta afirmación: nunca terminará. La variable pen nunca se modifica, por lo que la expresión (pen < 10) siempre se evaluará como verdadera. El resultado es que el bucle nunca terminará, creando lo que comúnmente se conoce como bucle infinito. Un bucle infinito es un bucle cuya condición de terminación nunca se alcanza durante el tiempo de ejecución.

Cada vez que escriba un bucle, debe examinarlo para determinar si la condición de terminación siempre se cumple bajo alguna condición. Por ejemplo, un bucle en el que no cambian variables entre dos ejecuciones sugiere que es posible que no se cumpla la condición de terminación. Las variables del bucle siempre deben moverse en una dirección particular.

En otras palabras, asegúrese de que la condición del bucle o las variables de las que depende la condición en adelante, están cambiando entre ejecuciones. Luego, asegúrese de que finalmente se alcance la condición de terminación en todas las circunstancias. Como aprenderá en la última sección de este capítulo, un bucle también puede salir bajo otras condiciones, como una declaración de interrupción .

# Construyendo bucles

Aunque las declaraciones while y do/ while son bastante poderosas, algunas tareas son tan comunes al escribir software que se crearon tipos especiales de bucles; por ejemplo, iterar sobre una declaración exactamente 10 veces o iterar sobre una lista de nombres . Podrías realizar fácilmente estas tareas con varios bucles while que has visto hasta ahora, pero normalmente requieren mucho código repetitivo. ¿No sería fantástico si hubiera una estructura de bucle que pudiera hacer lo mismo en una sola línea de código?

Con esto, presentamos la estructura de control de repetición más conveniente, para bucles. Hay dos tipos de bucles for , aunque ambos utilizan la misma palabra clave for . El primero se conoce como bucle for básico y el segundo a menudo se denomina bucle for mejorado . Para mayor claridad, a lo largo del libro nos referimos a ellos como bucle for y bucle for-each, respectivamente.

## El bucle for

Un bucle for básico tiene la misma expresión booleana condicional y declaración, o bloque de declaraciones, que los bucles while , así como dos secciones nuevas: un bloque de inicialización y una declaración de actualización . La figura 3.7 muestra cómo se distribuyen estos componentes.

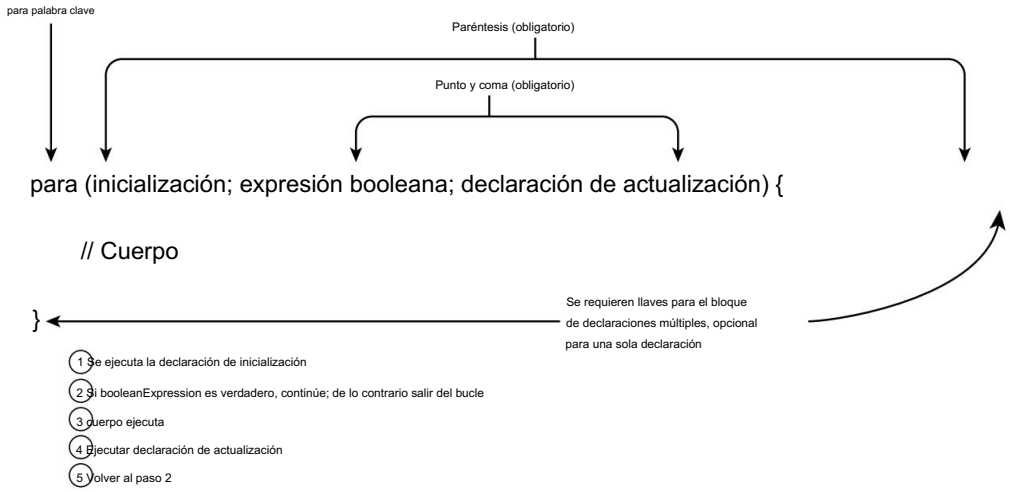
Aunque la Figura 3.7 puede parecer un poco confusa y casi arbitraria al principio, la organización de los componentes y el flujo nos permiten crear declaraciones extremadamente poderosas en una sola línea que de otro modo ocuparían varias líneas con un bucle while . Cada una de las tres secciones está separada por un punto y coma. Además, las secciones de inicialización y actualización pueden contener varias declaraciones, separadas por comas.

Las variables declaradas en el bloque de inicialización de un bucle for tienen un alcance limitado y solo son accesibles dentro del bucle for . Tenga cuidado con las preguntas del examen en las que se declara una variable dentro del bloque de inicialización de un bucle for y luego se lee fuera del bucle. Por ejemplo, este código no se compila porque se hace referencia a la variable de bucle i fuera del bucle:

Machine Translated by Google



FIGURA 3.7 La estructura de un bucle for básico




```
para(int i=0; i < 10; i++)
    System.out.println("El valor es: "+i);
System.out.println(i); // NO COMPILA
```

Alternativamente, las variables declaradas antes del bucle for y a las que se les asigna un valor en el bloque de inicialización pueden usarse fuera del bucle for porque su alcance precede a la creación del bucle for .

```
ent yo;
para(i=0; i < 10; i++)
    System.out.println("El valor es: "+i);
System.out.println(i);
```

Echemos un vistazo a un ejemplo que imprime los primeros cinco números, comenzando con cero:

```
para(int i = 0; i < 5; i++) {
    System.out.print(i + " ");
}
```

La variable local i se inicializa primero a 0. La variable i solo está dentro del alcance durante la duración del bucle y no está disponible fuera del bucle una vez que se ha completado. como un rato bucle, la condición booleana se evalúa en cada iteración del bucle antes de que se ejecute el bucle. Como devuelve verdadero, el bucle se ejecuta y genera 0 seguido de un espacio. A continuación, el bucle ejecuta la sección de actualización, que en este caso aumenta el valor de i a 1. Luego, el bucle evalúa la expresión booleana por segunda vez y el proceso se repite varias veces, imprimiendo lo siguiente:

```
0 1 2 3 4
```

Machine Translated by Google



En la quinta iteración del ciclo, el valor de i llega a 4 y se incrementa en 1 para llegar a 5. En la sexta iteración del ciclo, se evalúa la expresión booleana y, dado que (5 < 5) devuelve falso, el ciclo termina sin ejecutar el cuerpo del bucle de declaración.

**Real World Scenario**

¿Por qué me sumo a Loops?

Puede notar que es una práctica común nombrar una variable de bucle for i. Mucho antes de que existiera Java, los programadores comenzaron a usar i como abreviatura de variable de incremento, y la práctica existe hoy, ¡aunque muchos de esos lenguajes de programación ya no lo hacen! Para bucles dobles o triples, donde ya se usa i , a menudo se usan las siguientes letras del alfabeto, j y k.

## Elementos de impresión al revés

Supongamos que desea imprimir los mismos primeros cinco números desde cero como lo hicimos en la sección anterior, pero esta vez en orden inverso. El objetivo entonces es imprimir 4 3 2 1 0.

¿Cómo lo harías tú? Una implementación inicial podría verse como la siguiente:

```
for (var contador = 5; contador > 0; contador--) {
    System.out.print(contador + " ");
}
```

Si bien este fragmento genera cinco valores distintos y se parece a nuestro primer bucle for Por ejemplo, no genera los mismos cinco valores. En cambio, este es el resultado:

```
5 4 3 2 1
```

Espera, ¡eso no es lo que queríamos! Queríamos 4 3 2 1 0. Comienza con 5, porque eso es el primer valor que se le asigna. Arreglemos eso comenzando con 4 :

```
for (var contador = 4; contador > 0; contador--) {
    System.out.print(contador + " ");
}
```

¿Qué imprime esto ahora? Imprime lo siguiente:

```
4 3 2 1
```

¡Tan cerca! El problema es que termina en 1, no en 0, porque le dijimos que saliera tan pronto como el valor no fuera estrictamente mayor que 0. Si queremos imprimir el mismo 0 al 4 que en nuestro primer ejemplo, necesitamos actualizar el condición de terminación, así:

Machine Translated by Google



```
for (var contador = 4; contador >= 0; contador--) {
    System.out.print(contador + " ");
}
```

¡Finalmente! Tenemos un código que ahora imprime 4 3 2 1 0 y coincide con el reverso de nuestro ejemplo de bucle for en la sección anterior. En su lugar podríamos haber usado contador > -1 como condición de terminación del bucle en este ejemplo, aunque el contador >= 0 tiende a ser más legible.

Para el examen, tendrás que saber leer bucles for hacia adelante y hacia atrás . Cuando vea un bucle for en el examen, preste mucha atención a la variable del bucle y a las operaciones si se utiliza el operador de decremento, -- . Si bien incrementar desde 0 en un bucle for suele ser sencillo, disminuir tiende a ser menos intuitivo. De hecho, si ve un bucle for con un operador de decremento en el examen, debe asumir que están intentando poner a prueba sus conocimientos sobre las operaciones de bucle.

## Trabajar con bucles for

Aunque la mayoría de los bucles for que probablemente encontrará en su experiencia de desarrollo profesional estarán bien definidos y serán similares a los ejemplos anteriores, hay una serie de variaciones y casos extremos que podrá ver en el examen. Debe familiarizarse con los siguientes cinco ejemplos; Es probable que se vean variaciones de estos en el examen.

Abordemos algunos ejemplos con fines ilustrativos:

1. Creando un bucle infinito

```
para( ; ; )
    System.out.println("Hola mundo");
```

Aunque parezca que este bucle for no se compila, de hecho se compilará y ejecutará sin problemas . En realidad, es un bucle infinito que imprimirá la misma declaración repetidamente.
Este ejemplo refuerza el hecho de que los componentes del bucle for son opcionales.
Tenga en cuenta que los puntos y comas que separan las tres secciones son obligatorios, ya que for() sin punto y coma no se compilará.

2. Agregar varios términos a la declaración for

```
int x = 0;
for(largo y = 0, z = 4; x < 5 && y < 10; x++, y++) {
    System.out.print(y + " "); }
System.out.print(x + " ");
```

Este código demuestra tres variaciones del bucle for que quizás no hayas visto.
Primero, puede declarar una variable, como x en este ejemplo, antes de que comience el ciclo y usarla después de que se complete. En segundo lugar, su bloque de inicialización, expresión booleana y declaraciones de actualización pueden incluir variables adicionales que pueden o no hacer referencia a cada una.

Machine Translated by Google



otro. Por ejemplo, z se define en el bloque de inicialización y nunca se usa. Finalmente, la declaración de actualización puede modificar múltiples variables. Este código imprimirá lo siguiente cuando se ejecute:

```
0 1 2 3 4 5
```

3. Redeclarar una variable en el bloque de inicialización

```
int x = 0;
for(int x = 4; x < 5; x++) // NO COMPILA
    Sistema.out.print(x +       " ");
```

Este ejemplo es similar al anterior, pero no se compila debido al bloque de inicialización. La diferencia es que x se repite en el bloque de inicialización después de haber sido declarado antes del ciclo, lo que hace que el compilador se detenga debido a una declaración de variable duplicada. Podemos arreglar este bucle eliminando la declaración de x desde el bucle for de la siguiente manera:

```
int x = 0;
para(x = 0; x < 5; x++)
    Sistema.out.print(x +       " ");
```

Tenga en cuenta que esta variación ahora se compilará porque el bloque de inicialización simplemente asigna un valor a x y no lo declara.

4. Uso de tipos de datos incompatibles en el bloque de inicialización

```
int x = 0;
for(long y = 0, int z = 4; x < 5; x++) // NO COMPILA
    Impresión.de.salida.del.sistema(y +   " ");
```

Al igual que el tercer ejemplo, este código no se compilará, aunque esta vez por un motivo diferente. Todas las variables del bloque de inicialización deben ser del mismo tipo. En el ejemplo de términos múltiples, y y z eran ambos largos, por lo que el código se compiló sin problemas; pero en este ejemplo, tienen tipos diferentes, por lo que el código no se compilará.

5. Uso de variables de bucle fuera del bucle

```
para (largo y = 0, x = 4; x < 5 && y < 10; x++, y++)
    Impresión.de.salida.del.sistema(y +   " ");
Sistema.out.print(x); // NO COMPILA
```

Ya cubrimos esto al comienzo de esta sección, pero es tan importante para aprobar el examen que lo discutiremos nuevamente aquí. Si observa, x se define en el bloque de inicialización del bucle y luego se usa después de que termina el bucle. Dado que x solo tenía como alcance el bucle, usarlo fuera del bucle provocará un error del compilador.

Machine Translated by Google



### Modificar variables de bucle

Como regla general, se considera una mala práctica de codificación modificar las variables del bucle debido a la imprevisibilidad del resultado, como en los siguientes ejemplos:

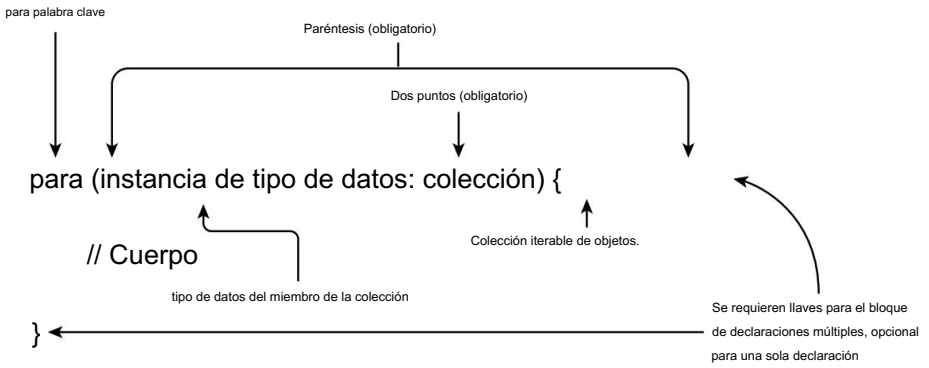
```
para(int i=0; i<10; i++)
    yo = 0;

para(int j=1; j<10; j++)
    j++;
```

También tiende a hacer que el código sea difícil de seguir para otras personas.

## El bucle para cada uno

El bucle for-each es una estructura especializada diseñada para iterar sobre matrices y varias clases de Collections Framework, como se presenta en la Figura 3.8.

FIGURA 3.8 La estructura de un bucle para cada mejorado




La declaración de bucle for-each se compone de una sección de inicialización y un objeto a ser iterado. El lado derecho del bucle for-each debe ser uno de los siguientes:

- Una matriz Java integrada
- Un objeto cuyo tipo implementa java.lang.Iterable

Cubrimos lo que significa implementar en el Capítulo 7, pero por ahora sólo necesita saber que el lado derecho debe ser una matriz o colección de elementos, como una Lista o un Conjunto. Para el examen, debes saber que esto no incluye todas las clases del Marco de Colecciones.

Machine Translated by Google



o interfaces, pero solo aquellos que implementan o amplían esa interfaz de Colección . Por ejemplo, Map no se admite en un bucle for-each, aunque Map incluye métodos que devuelven instancias de Collection .

El lado izquierdo del bucle for-each debe incluir una declaración para una instancia de una variable cuyo tipo es compatible con el tipo de matriz o colección en el lado derecho de la declaración. En cada iteración del bucle, a la variable nombrada en el lado izquierdo de la declaración se le asigna un nuevo valor de la matriz o colección en el lado derecho de la declaración.

Compare estos dos métodos que imprimen los valores de una matriz, uno usando un método tradicional bucle for y el otro usando un bucle for-each:

```
public void printNames(String[] nombres) {
    for(int contador=0; contador<nombres.longitud; contador++)
        System.out.println(nombres[contador]);
}

public void printNames(String[] nombres) {
    para (var nombre: nombres)
        System.out.println(nombre);
}
```

El bucle for-each es mucho más corto, ¿no? Ya no tenemos una variable de bucle contador que debamos crear, incrementar y monitorear. Como usar un bucle for en lugar de un while bucle, los bucles for-each están destinados a reducir el código repetitivo, haciendo que el código sea más fácil de leer/escribir y permitiéndole concentrarse en las partes de su código que realmente importan.

También podemos usar un bucle for-each en una Lista, ya que implementa Iterable.

```
public void printNames(List<String> nombres) {
    para (var nombre: nombres)
        System.out.println(nombre);
}
```

Cubrimos los genéricos en detalle en el Capítulo 9, “Colecciones y genéricos”. Para este capítulo, usted sólo necesito saber que en cada iteración, un bucle for-each asigna una variable con el mismo tipo que el argumento genérico. En este caso, el nombre es de tipo String.

Hasta ahora, todo bien. ¿Qué pasa con los siguientes ejemplos?

```
Pájaros de cuerda = "Jay";
for(String bird: pájaros) // NO COMPILA
    System.out.print(pájaro + " ");

String[] perezosos = new String[3];
para (int perezoso: perezosos)        // NO COMPILA
    System.out.print(perezoso + " ");
```

El primer bucle for-each no se compila porque String no se puede utilizar en el lado derecho de la declaración. Si bien una cadena puede representar una lista de caracteres, en realidad tiene que ser una matriz o implementar Iterable. El segundo ejemplo no se compila porque el tipo de bucle en el lado izquierdo de la declaración es int y no coincide con el tipo esperado de Cadena.

Machine Translated by Google



# Controlar el flujo con ramificaciones

Los últimos tipos de estructuras de flujo de control que cubrimos en este capítulo son declaraciones de ramificación. Hasta ahora hemos estado tratando con bucles simples que terminaban sólo cuando su valor booleano expresión evaluada como falsa. Ahora le mostramos otras formas en que los bucles podrían terminar o bifurcarse, y verá que la ruta tomada durante el tiempo de ejecución puede no ser tan sencilla como en los ejemplos anteriores.

## Bucles anidados

Antes de pasar a las declaraciones de ramificación, debemos introducir el concepto de bucles anidados. Un bucle anidado es un bucle que contiene otro bucle, incluidos los bucles while, do/ while, for y for-each. Por ejemplo, considere el siguiente código que itera sobre una matriz bidimensional, que es una matriz que contiene otras matrices como miembros. Cubrimos los arreglos multidimensionales en detalle en el Capítulo 4, “API principales”, pero por ahora, supongamos que la siguiente manera es cómo declararía un arreglo bidimensional:

```
int[][] myComplexArray = {{5,2,1,3},{3,9,8,9},{5,7,12,7}};

for(int[] miArraySimple: miArrayComplejo) {
    for(int i=0; i<mySimpleArray.length; i++) {
        System.out.print(mySimpleArray[i]+"\t");
    }
    Sistema.out.println();
}
```

Observe que en este ejemplo mezclamos intencionalmente un bucle for y un bucle for-each. El bucle exterior se ejecutará un total de tres veces. Cada vez que se ejecuta el bucle exterior, el bucle interior se ejecuta cuatro veces. Cuando ejecutamos este código, vemos el siguiente resultado:

```
5    2    1    3
3    9    8    9
5    7    12   7
```

Los bucles anidados pueden incluir while y do/ while, como se muestra en este ejemplo. Vea si puede determinar qué generará este código:

```
int hambrientoHipopótamo = 8;
while(hipopótamo hambriento>0) {
    hacer {
        hipopótamo hambriento -= 2;
    } while (hipopótamo hambriento>5);
    hipopótamo hambriento--;
    System.out.print(hipopótamo hambriento+", ");
}
```

Machine Translated by Google



La primera vez que se ejecuta este bucle, el bucle interno se repite hasta que el valor de hambrientoHippopotamus sea 4. Luego, el valor se reducirá a 3, y ese será el resultado al final de la primera iteración del bucle externo.

En la segunda iteración del bucle exterior, la sentencia do/ while interna se ejecutará una vez, aunque hambrientoHippopotamus ya no sea mayor que 5. Como recordarás, las sentencias do/ while siempre ejecutan el cuerpo al menos una vez. Esto reducirá el valor a 1, que el operador de decremento reducirá aún más en el bucle externo a 0. Una vez que el valor llegue a 0, el bucle externo terminará. El resultado es que el código generará lo siguiente:

```
3, 0,
```

Los ejemplos del resto de esta sección incluyen muchos bucles anidados. También encontrará bucles anidados en el examen, por lo que cuanto más practique con ellos, más preparado estará.

## Agregar etiquetas opcionales

Una cosa que omitimos intencionalmente cuando presentamos declaraciones if , declaraciones de cambio y bucles es que todas pueden tener etiquetas opcionales. Una etiqueta es un puntero opcional al encabezado de una declaración que permite que el flujo de la aplicación salte a ella o salga de ella. Es un identificador único seguido de dos puntos (:). Por ejemplo, podemos agregar etiquetas opcionales a uno de los ejemplos anteriores:

```
int[][] myComplexArray = {{5,2,1,3},{3,9,8,9},{5,7,12,7}};

OUTER_LOOP: for(int[] mySimpleArray: myComplexArray) {
    INNER_LOOP: for(int i=0; i<mySimpleArray.length; i++) {
        System.out.print(mySimpleArray[i]+"\t");
    }
    Sistema.out.println();
}
```

Las etiquetas siguen las mismas reglas de formato que los identificadores. Para facilitar la lectura, comúnmente se expresan usando letras mayúsculas en Snake_case con guiones bajos entre las palabras. Cuando se trata de un solo bucle, las etiquetas no agregan ningún valor, pero como aprenderá en la siguiente sección, son extremadamente útiles en estructuras anidadas.

Mientras este tema no esté en el examen, es posible agregar etiquetas opcionales para controlar y bloquear declaraciones. Por ejemplo, el compilador permite lo siguiente, aunque es extremadamente poco común:

```
int rana = 15;
BAD_IDEA: si(rana>10)
AUN_PEOR_IDEA: {
    rana++;
}
```

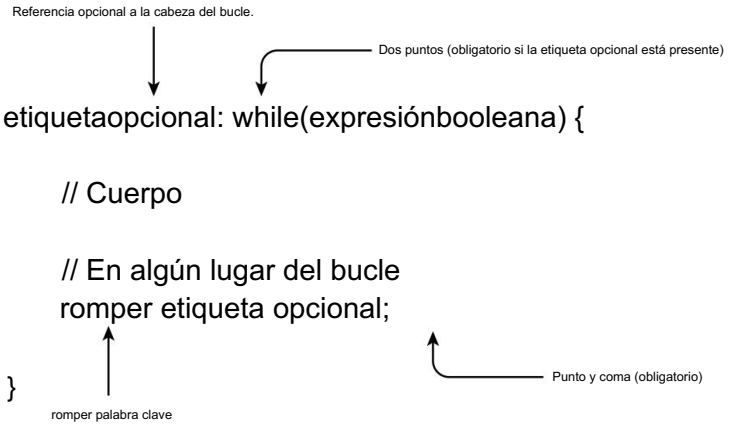
Machine Translated by Google



## La declaración de ruptura

Como vio al trabajar con declaraciones de cambio , una declaración de interrupción transfiere el flujo de control a la declaración adjunta. Lo mismo se aplica a una instrucción break que aparece dentro de un bucle while, do/ while o for , ya que finalizará el bucle antes, como se muestra en la Figura 3.9.

FIGURA 3.9 La estructura de una declaración de ruptura




Observe en la Figura 3.9 que la declaración break puede tomar un parámetro de etiqueta opcional . Sin un parámetro de etiqueta, la instrucción break terminará el bucle interno más cercano que se encuentra actualmente en proceso de ejecución. El parámetro de etiqueta opcional nos permite salir de un bucle externo de nivel superior. En el siguiente ejemplo, buscamos la primera posición del índice de matriz (x,y) de un número dentro de una matriz bidimensional sin clasificar:

```
clase pública FindInMatrix {
    público estático vacío principal (String [] argumentos) {
        int[][] lista = {{1,13},{5,2},{2,2}};
        int valorbúsqueda = 2;
        int posiciónX = -1;
        posición intY = -1;
        PARENT_LOOP: for(int i=0; i<lista.longitud; i++) {
            for(int j=0; j<lista[i].length; j++) {
                if(lista[i][j]==valorbúsqueda) {
                    posiciónX = i;
```

Machine Translated by Google



```
                    posiciónY = j;
                    romper PARENT_LOOP;
                }
            }
        }
        if(posiciónX==-1 || posiciónY==-1) {
            System.out.println("Valor "+valorbúsqueda+" no encontrado");
        } demás {
            System.out.println("Valor "+searchValue+" encontrado en:                    "  +
                "("+posiciónX+","+posiciónY+")");
        }
    }
}
```

Cuando se ejecute, este código generará lo siguiente:

Valor 2 encontrado en: (1,1)

En particular, observe la declaración break PARENT_LOOP. Esta declaración saldrá de toda la estructura del bucle tan pronto como se encuentre el primer valor coincidente. Ahora, imagina lo que sucedería si reemplazáramos el cuerpo del bucle interno con lo siguiente:

```
                if(lista[i][j]==valorbúsqueda) {
                    posiciónX = i;
                    posiciónY = j;
                    romper;
                }
```

¿Cómo cambiaría esto nuestro flujo y cambiaría la producción? En lugar de salir cuando se encuentra el primer valor coincidente, el programa ahora solo saldría del bucle interno cuando se cumpliera la condición. En otras palabras, la estructura encontraría el primer valor coincidente del último bucle interno que contenga el valor, lo que daría como resultado el siguiente resultado:

Valor 2 encontrado en: (2,0)

Finalmente, ¿qué pasaría si eliminamos la ruptura por completo?

```
                if(lista[i][j]==valorbúsqueda) {
                    posiciónX = i;
                    posiciónY = j;
                }
```

En este caso, el código buscaría el último valor en toda la estructura que tuviera el valor coincidente. La salida se vería así:

Valor 2 encontrado en: (2,1)

Machine Translated by Google

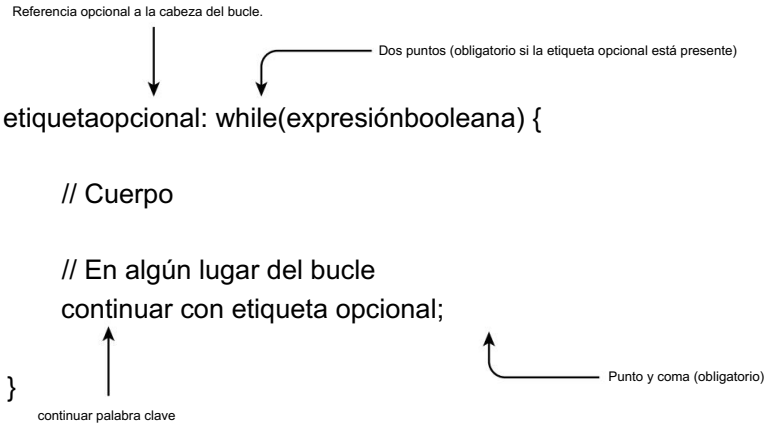


Puede ver en este ejemplo que usar una etiqueta en una instrucción break en un bucle anidado, o no usar la instrucción break en absoluto, puede hacer que la estructura del bucle se comporte de manera muy diferente.

## La declaración de continuación

Ampliemos ahora nuestra discusión sobre el control de bucle avanzado con la instrucción continuar , una instrucción que hace que el flujo finalice la ejecución de la iteración del bucle actual, como se muestra en la Figura 3.10.

FIGURA 3.10 La estructura de una declaración de continuación




Puede notar que la sintaxis de la instrucción continuar refleja la de la interrupción. declaración. De hecho, las declaraciones son idénticas en cómo se utilizan, pero con resultados diferentes. Mientras que la instrucción break transfiere el control a la instrucción adjunta, la instrucción continue transfiere el control a la expresión booleana que determina si el ciclo debe continuar.
En otras palabras, finaliza la iteración actual del bucle. Además, al igual que la instrucción break , la instrucción continue se aplica al bucle interno más cercano en ejecución, utilizando instrucciones de etiqueta opcionales para anular este comportamiento.

Echemos un vistazo a un ejemplo. Imaginemos que tenemos un cuidador del zoológico que se supone debe limpiar el primer leopardo en cada uno de los cuatro establos, pero se salta el establo b por completo.

```
horario de limpieza de clase pública {
    público estático vacío principal (String [] argumentos) {
        LIMPIEZA: for(char establos = 'a'; establos<='d'; establos++) {
            for(int leopardo = 1; leopardo<4; leopardo++) {
                if(establos=='b' || leopardo==2) {
```

Machine Translated by Google



```
6:                  continuar LIMPIEZA;
7:              }
8:              System.out.println("Limpieza: "+establos+","+leopardo);
9: } } } }
```

Con la estructura definida, el bucle devolverá el control al bucle principal cada vez que el primer valor sea b o el segundo valor sea 2. En la primera, tercera y cuarta ejecuciones del bucle externo, el bucle interno imprime una declaración exactamente una vez y luego sale en el siguiente bucle interno cuando leopardo es 2. En la segunda ejecución del bucle externo, el bucle interno sale inmediatamente sin imprimir nada ya que b se encuentra de inmediato. Se imprime lo siguiente:

```
Limpieza: a,1
Limpieza: c,1
Limpieza: d,1
```

Ahora, imagine que eliminamos la etiqueta CLEANING en la instrucción continuar para que el control se devuelve al bucle interior en lugar del exterior. La línea 6 queda como sigue:

```
6:                  continuar;
```

Esto corresponde a que el cuidador del zoológico limpie a todos los leopardos excepto aquellos etiquetados con 2 o en el establo b. La salida es entonces la siguiente:

```
Limpieza: a,1
Limpieza: a,3
Limpieza: c,1
Limpieza: c,3
Limpieza: d,1
Limpieza: d,3
```

Finalmente, si eliminamos la declaración continue y la declaración if asociada por completo Al eliminar las líneas 5 a 7, llegamos a una estructura que genera todos los valores, como esta:

```
Limpieza: a,1
Limpieza: a,2
Limpieza: a,3
Limpieza: b,1
Limpieza: b,2
Limpieza: b,3
Limpieza: c,1
Limpieza: c,2
Limpieza: c,3
Limpieza: d,1
Limpieza: d,2
Limpieza: d,3
```

Machine Translated by Google



## La declaración de devolución

Dado que este libro no debería ser su primera incursión en la programación, esperamos que haya encontrado métodos que contengan declaraciones de retorno . De todos modos, cubrimos cómo diseñar y crear métodos que los utilicen en detalle en el Capítulo 5, "Métodos".

Por ahora, sin embargo, debería estar familiarizado con la idea de que la creación de métodos y el uso de declaraciones de retorno se pueden utilizar como una alternativa al uso de etiquetas y declaraciones de interrupción . Por ejemplo, eche un vistazo a esta reescritura de nuestra clase FindInMatrix anterior:

```
clase pública FindInMatrixUsingReturn {
    privado estático int[] searchForValue(int[][] lista, int v) {
        for (int i = 0; i < lista.longitud; i++) {
            for (int j = 0; j < lista[i].longitud; j++) {
                si (lista[i][j] == v) {
                    devolver nuevo int[] {i,j};
                }
            }
        }
        devolver nulo;
    }

    público estático vacío principal (String [] argumentos) {
        int[][] lista = { { 1, 13 }, { 5, 2 }, { 2, 2 } };
        int valorbúsqueda = 2;
        int[] resultados = searchForValue(lista,searchValue);

        si (resultados == nulo) {
            System.out.println("Valor " + valor de búsqueda + " extraviado");
        } demás {
            System.out.println("Valor " "(" + + valor de búsqueda + " encontrado en: " +
                resultados[0] + "," + resultados[1] + ")");
        }
    }
}
```

Esta clase es funcionalmente la misma que la primera clase FindInMatrix que vimos anteriormente usando romper. Si necesita un control más detallado del bucle con múltiples pausas y continuar declaraciones, la primera clase probablemente sea mejor. Dicho esto, encontramos que el código sin etiquetas ni declaraciones de interrupción es mucho más fácil de leer y depurar. Además, hacer que la lógica de búsqueda sea una función independiente hace que el código sea más reutilizable y que la llamada al método main() sea mucho más fácil de leer.

Machine Translated by Google



Para el examen, necesitará conocer ambas formas. Solo recuerde que las declaraciones de retorno se pueden usar para salir de bucles rápidamente y pueden conducir a un código más legible en la práctica, especialmente cuando se usan con bucles anidados.

## Código inalcanzable

Una faceta de romper, continuar y regresar que debe tener en cuenta es que cualquier código colocado inmediatamente después de ellos en el mismo bloque se considera inalcanzable y no se compilará. Por ejemplo, el siguiente fragmento de código no se compila:

```
int fecha de verificación = 0;
mientras(verificarfecha<10) {
    comprobarFecha++;
    si(verificarfecha>100) {
        romper;
        comprobarFecha++; // NO COMPILA
    }
}
```

Aunque no es lógicamente posible que la declaración if se evalúe como verdadera en este ejemplo de código, el compilador nota que tiene declaraciones inmediatamente después de la interrupción y no podrá compilar con "código inalcanzable" como motivo. Lo mismo ocurre con las declaraciones continuar y devolver , como se muestra en los dos ejemplos siguientes:

```
int minuto = 1;
MIRAR: mientras(minuto>2) {
    si(minuto++>2) {
        continuar MIRAR;
        System.out.print(minuto); // NO COMPILA
    }
}

hora int = 2;
cambiar(hora) {
    caso 1: regreso; hora++; // NO COMPILA
    caso 2:
}
```

Una cosa que hay que recordar es que no importa si el bucle o la estructura de decisión realmente visitan la línea de código. Por ejemplo, el bucle podría ejecutarse cero o infinitas veces en tiempo de ejecución. Independientemente de la ejecución, el compilador informará un error si encuentra algún código que considere inalcanzable, en este caso cualquier declaración inmediatamente después de una interrupción, continuación o declaración de devolución .

Machine Translated by Google



## Revisión de ramificaciones

Concluimos esta sección con la Tabla 3.1, que le ayudará a recordar cuándo se permiten etiquetas, declaraciones de interrupción y continuación en Java. Aunque con fines ilustrativos nuestros ejemplos utilizan estas declaraciones en bucles anidados, también se pueden utilizar dentro de bucles individuales.

TABLA 3.1 Uso de declaraciones de control

| | Etiquetas de soporte | Rotura de soporte | Soporte continúa | Rendimiento de soporte |
|---|---|---|---|---|
| mientras | Sí | Sí | Sí | No |
| hacer/mientras sí | | Sí | Sí | No |
| para | Sí | Sí | Sí | No |
| cambiar | Sí | Sí | No | Sí |

Por último, pero no menos importante, todos los centros de pruebas deben ofrecer algún tipo de papel de desecho o pizarra para usar durante el examen. Le recomendamos encarecidamente que utilice estas ayudas de prueba si encuentra preguntas complejas que involucran bucles anidados y declaraciones de ramificación.

Algunas de las preguntas que consumen más tiempo y que puede ver en el examen podrían involucrar bucles anidados con muchas ramificaciones. A menos que detecte un error obvio del compilador, le recomendamos omitir estas preguntas y volver a ellas al final. Recuerde, ¡todas las preguntas del examen tienen el mismo peso!

# Resumen

Este capítulo presentó cómo tomar decisiones inteligentes en Java. Cubrimos construcciones básicas para la toma de decisiones, como if, else y switch , y mostramos cómo usarlas para cambiar la ruta del proceso en tiempo de ejecución. También presentamos características más nuevas en el lenguaje Java, incluida la coincidencia de patrones y las expresiones de cambio , ambas diseñadas para reducir el código erplate.

Luego trasladamos nuestra discusión a las estructuras de control de repetición, comenzando con los bucles while y do/ while . Mostramos cómo usarlos para crear procesos que se repiten varias veces y también mostramos lo importante que es asegurarse de que eventualmente terminen. Recuerde que la mayoría de estas estructuras requieren la evaluación de una expresión booleana particular para completarse.

Machine Translated by Google



A continuación, cubrimos las estructuras de control de repetición extremadamente convenientes: los bucles for y for each. Si bien su sintaxis es más compleja que la tradicional while o do/ while bucles, son extremadamente útiles en la codificación diaria y le permiten crear expresiones complejas en una sola línea de código. Con un bucle for-each, no es necesario escribir explícitamente una expresión booleana , ya que el compilador crea una por usted. Para mayor claridad, nos referimos a un bucle for mejorado como bucle for-each, pero sintácticamente ambos se escriben usando la palabra clave for .

Concluimos este capítulo analizando opciones de control avanzadas y cómo se puede mejorar el flujo mediante bucles anidados junto con declaraciones break, continue y return . Tenga cuidado con las preguntas del examen que utilizan bucles anidados, especialmente aquellas con etiquetas, y verifique que se estén usando correctamente.

Este capítulo es especialmente importante porque es probable que al menos un componente de este capítulo aparezca en cada pregunta del examen con código de muestra. Muchas de las preguntas del examen se centran en el uso sintáctico adecuado de las estructuras, ya que serán una gran fuente de preguntas que terminan en "No compila". Debería poder responder correctamente todas las preguntas de repaso o comprender completamente aquellas que respondió incorrectamente antes de pasar a capítulos posteriores.

## Conceptos básicos del examen

Comprender las declaraciones de control de decisiones if y else . Las declaraciones if y else aparecen con frecuencia a lo largo del examen en preguntas no relacionadas con el control de decisiones, así que asegúrese de comprender completamente estos componentes básicos de Java.

Aplicar coincidencia de patrones y alcance de flujo. La coincidencia de patrones se puede utilizar para reducir el código de caldera que involucra una declaración if , un operador de instancia y una operación de conversión usando una variable de patrón. También puede incluir un patrón o filtro después de la declaración de la variable del patrón. La coincidencia de patrones utiliza un alcance de flujo en el que la variable de patrón está dentro del alcance siempre que el compilador pueda determinar definitivamente su tipo.

Comprenda las declaraciones de cambio y su uso adecuado. Debería poder detectar una declaración de cambio mal formada en el examen. El valor de cambio y el tipo de datos deben ser compatibles con las declaraciones de caso , y los valores de las declaraciones de caso deben evaluarse como constantes de tiempo de compilación. Finalmente, en tiempo de ejecución, una instrucción de cambio se bifurca al primer caso coincidente , o se establece por defecto si no hay coincidencia, o sale por completo si no hay coincidencia ni rama predeterminada . Luego, el proceso continúa en cualquier caso procesal o declaración de incumplimiento hasta que se llega a una declaración de ruptura o devolución .

Utilice expresiones de cambio correctamente. Discerna las diferencias entre expresiones de cambio y declaraciones de cambio . Comprenda cómo escribir expresiones de cambio correctamente, incluido el uso adecuado de punto y coma, escribiendo expresiones de casos y bloques que produzcan un valor consistente, y asegurándose de que todos los valores posibles de la variable de cambio sean manejados por la expresión de cambio.

Machine Translated by Google



Escribe bucles while . Conozca la estructura sintáctica de todos los bucles while y do/ while . En particular, sepa cuándo utilizar uno frente al otro.

Ser capaz de utilizar bucles for . Debe estar familiarizado con los bucles for y for each y saber cómo escribirlos y evaluarlos. Cada bucle tiene sus propias propiedades y estructuras especiales.
Debe saber cómo utilizar bucles for-each para iterar sobre listas y matrices.

Comprenda cómo la pausa, la continuación y el retorno pueden cambiar el control del flujo. Sepa cómo cambiar el control de flujo dentro de una declaración aplicando una pausa, una continuación o un retorno. declaración. También sepa qué declaraciones de control pueden aceptar declaraciones de interrupción y cuáles pueden aceptar declaraciones de continuación . Finalmente, debe comprender cómo funcionan estas declaraciones dentro de bucles integrados o declaraciones de cambio .

Machine Translated by Google



# Preguntas de revisión

Las respuestas a las preguntas de revisión del capítulo se pueden encontrar en el Apéndice.

1. ¿Cuál de los siguientes tipos de datos se puede utilizar en una expresión de cambio? (Elegir todos eso aplica.)

   una enumeración

   B.int _

   C. Byte D.

   largo E.

   Cadena

   f char

   G. tener

   H. doble

2. ¿Cuál es el resultado del siguiente fragmento de código? (Elija todas las que correspondan).

```
3: temperatura interna = 4;
4: humedad prolongada = -temperatura + temperatura * 3;
5: si (temperatura>=4) 6:
si (humedad < 6) System.out.println("Demasiado bajo"); 7:
else System.out.println("Justo a la derecha"); 8:
else System.out.println("Demasiado alto");
```

   A. Demasiado bajo

   B. Justo C. Demasiado

   alto D. Se lanza

   una excepción NullPointerException en tiempo de ejecución.

   E. El código no se compilará debido a la línea 7.

   F. El código no se compilará debido a la línea 8.

3. ¿Cuáles de los siguientes tipos de datos están permitidos en el lado derecho de una expresión para cada uno? (Elija todas las que correspondan).

   A. Doble[][]

   B. Objeto C.

   Mapa

   D. Lista

   E. Cadena F.

   char[]

   G. Excepción

   H. conjunto

Machine Translated by Google



4. ¿Cuál es el resultado de llamar a printReptile(6)?

```
void printReptile(int categoría) { var tipo =
    switch(categoría) { caso 1,2 ->
            "Serpiente"; caso 3,4 ->
            "Lagarto"; caso 5,6 ->
            "Tortuga"; caso 7,8 ->
            "Cocodrilo";
        };
        System.out.print(tipo);
}
```

A. serpiente

B. lagarto

C. Tortuga

D. Alligator E.

TurtleAlligator F. Ninguno de

los anteriores

5. ¿Cuál es el resultado del siguiente fragmento de código?

```
Lista<Integer> misNúmerosFavoritos = nueva ArrayList<>();
misNúmerosFavoritos.add(10);
misNúmerosFavoritos.add(14); for
(var a: misNúmerosFavoritos)
      { System.out.print(a + ", "); romper;

}

for (int b: misNúmerosFavoritos) {
      continuar;
      Sistema.out.print(b + ", ");
}

para (Objeto c: misNúmerosFavoritos)
      Sistema.out.print(c + ", ");
```

R. Se compila y ejecuta sin problemas, pero no produce ningún resultado.

B. 10, 14, C. 10,

10, 14, D. 10, 10, 14,

10, 14, E. Exactamente una línea de

código no se compila.

F. No se compilan exactamente dos líneas de código.

G. Tres o más líneas de código no se compilan.

H. El código contiene un bucle infinito y no termina.

Machine Translated by Google



6. ¿ Qué afirmaciones sobre las estructuras de decisión son verdaderas? (Elija todas las que correspondan).

R. Se puede ejecutar un bucle for-each en cualquier objeto de Collections Framework.

B. Se garantiza que el cuerpo de un bucle while se ejecutará al menos una vez.

C. La expresión condicional de un bucle for se evalúa antes de la primera ejecución del cuerpo del bucle.

D. Una expresión de cambio que toma una cadena y asigna el resultado a una variable requiere una rama predeterminada.

E. Se garantiza que el cuerpo de un bucle do/ while se ejecutará al menos una vez.

F. Una declaración if puede tener varias declaraciones else correspondientes.

7. Suponiendo que el clima es una matriz no vacía bien formada, ¿qué fragmento de código, cuando se inserta de forma independiente en el espacio en blanco del siguiente código, imprime todos los elementos del clima? (Elija todas las que correspondan).

```
impresión vacía privada (int [] clima) {
            ———————————————————) {
      for( System.out.println(clima[i]);
      }
}
```

A. int i=tiempo.longitud; i>0; i--

B.int i=0; i<=tiempo.longitud-1; ++yo

C. var w: clima

D. int i=tiempo.longitud-1; i>=0; i--

mi.int i=0, int j=3; i<tiempo.longitud; ++yo

F.int i=0; ++i<10 && i<tiempo.longitud;

G. Ninguno de los anteriores

8. ¿Cuál es el resultado de llamar a printType(11)?

```
31: tipo de impresión nulo (objeto o) {
32:     si (o instancia de murciélago entero) {
33:         System.out.print("int");
34:     } else if(o instancia de Integer bat && bat < 10) {
35:         System.out.print("pequeño int");
36:     } else if(o instancia de Long bat || bat <= 20) {
37:         System.out.print("largo");
38:     } por defecto {
39:         System.out.print("desconocido");
40:     }
41: }
```

Machine Translated by Google



A.int _

B. pequeño int

c. largo

D. desconocido

E. No se imprime nada.

F. El código contiene una línea que no se compila.

G. El código contiene dos líneas que no se compilan.

H. Ninguna de las anteriores

9. ¿ Qué declaraciones, cuando se insertan de forma independiente en el siguiente espacio en blanco, harán que el código imprima 2 en tiempo de ejecución? (Elija todas las que correspondan).

```
recuento int = 0;
CONEJITO: para(int fila = 1; fila <=3; fila++)
    CONEJO: for(int col = 0; col <3 ; col++) {
        si((col + fila) % 2 == 0)
            ____________________ ;
        contar++;
    }
System.out.println(recuento);
```

A. romper CONEJITO

B. romper CONEJO

C. continuar CONEJITO

D. continuar CONEJO

E. romper

f continuar

G. Ninguna de las anteriores, ya que el código contiene un error de compilación.

10. Dado el siguiente método, ¿cuántas líneas contienen errores de compilación? (Elegir todos eso aplica.)

```
10: DayOfWeek privado getWeekDay (día int, jueves int final) {
11:     int otroDía = día;
12:     int domingo = 0;
13:     cambiar (otro día) {
14:         por defecto:
15:         caso 1: continuar; caso
dieciséis:  jueves: retorno DayOfWeek.THURSDAY;
17:         caso 2,10: descanso;
```

Machine Translated by Google



```
18:             caso Domingo: retorno DayOfWeek.SUNDAY;
19:             caso DayOfWeek.MONDAY: return DayOfWeek.MONDAY;
20:         }
21:         devolver DíaDeLaSemana.VIERNES;
22: }
```

R. Ninguno, el código se compila sin problemas.

B.1 _

c.2 _

D.3 _

E.4 _

F.5 _

G.6 _

H. El código se compila pero puede producir un error en tiempo de ejecución.

11. ¿Cuál es el resultado de llamar a printLocation(Animal.MAMMAL)?

```
10: clase Zoológico {
11:         enum Animal {AVE, PESCADO, MAMÍFERO}
12:         Ubicación de impresión vacía (Animal a) {
13:             tipo largo = cambiar(a) {
14:                 caso PÁJARO -> 1;
15:                 caso PESCADO -> 2;
dieciséis:          caso MAMÍFERO -> 3;
17:                 predeterminado -> 4;
18:             };
19:             System.out.print(tipo);
20:         } }
```

R.3 _

B.4 _

C.34 _

D. El código no se compila debido a la línea 13.

E. El código no se compila debido a la línea 17.

F. Ninguna de las anteriores

12. ¿Cuál es el resultado del siguiente fragmento de código?

```
3: int cantar = 8, graznido = 2, notas = 0;
4: mientras(canta > graznir) {
5:         eso--;
6:         graznido += 2;
```

Machine Translated by Google



```
7:      notas += cantar + graznido;
8: }
9: System.out.println(notas);
```

R.11 _

B.13 _

C.23 _

D.33 _

E.50 _

F. El código no se compilará debido a la línea 7.

13. ¿Cuál es el resultado del siguiente fragmento de código?

```
2: booleano keepGoing = verdadero;
3: resultado int = 15, metros = 10; 4: hacer
{metros--;
5:      if(metros==8)
6:      keepGoing = false; resultado -= 2; 8: }
7:      mientras sigues;
9: System.out.println(resultado);
```

R.7 _

B.9 _

C.10 _

D.11 _

E.15 _

F. El código no se compilará debido a la línea 6.

G. El código no se compila por un motivo diferente.

14. ¿Qué afirmaciones sobre el siguiente fragmento de código son correctas? (Elija todas las que correspondan).

```
para (var pingüino: nuevo int [2])
    System.out.println(pingüino); var
avestruz = nuevo personaje[3]; para (var
emu: avestruz)
    System.out.println(emu);
Lista<Integer> loros = new ArrayList<Integer>(); para (var
guacamayo: loros)
    System.out.println(guacamayo);
```

Machine Translated by Google



A. El tipo de datos de Penguin es entero.

B. El tipo de datos de pingüino es int.

C. El tipo de datos de emu no está definido.

D. El tipo de datos de emu es Carácter.

E. El tipo de datos de guacamayo es Lista.

F. El tipo de datos de guacamayo es entero.

G. Ninguna de las anteriores, ya que el código no se compila.

15. ¿Cuál es el resultado del siguiente fragmento de código?

```
carácter final a = 'A', e = 'E'; grado
de carbón = 'B';
cambiar (grado) {
    por defecto:
    caso a:
    caso 'B': 'C': System.out.print("excelente "); caso 'D':
    System.out.print("bueno "); romper;
    caso e:
    caso 'F': System.out.print("no es bueno ");
}
```

A. excelente

B. muy bueno C.

bueno D.

no bueno E. El

código no se compila porque el tipo de datos de una o más declaraciones de casos sí lo hace no coincide con el tipo de datos de la variable de cambio.

F. Ninguna de las anteriores

16. Dada la siguiente matriz, ¿qué fragmentos de código imprimen los elementos en orden inverso al de cómo se declaran? (Elija todas las que correspondan).

```
char[] lobo = {'W', 'e', 'b', 'b', 'y'};
```

A.

```
int q = lobo.longitud;
for( ; ; )
    { System.out.print(lobo[--q]); si(q==0)
    romper;
}
```

B.

```
para(int m=wolf.length-1; m>=0; --m)
    System.out.print(lobo[m]);
```

Machine Translated by Google



C.

```
para(int z=0; z<wolf.length; z++)
    System.out.print(lobo[lobo.longitud-z]);
```

D.

```
int x = lobo.longitud-1;
para(int j=0; x>=0 && j==0; x--)
    System.out.print(lobo[x]);
```

Y.

```
final int r = lobo.longitud; para(int
w = r-1; r>-1; w = r-1)
    System.out.print(lobo[w]);
```

F.

```
para(int i=wolf.length; i>0; --i)
    System.out.print(lobo[i]);
```

G. Ninguno de los anteriores

17. ¿Qué números distintos se imprimen cuando se ejecuta el siguiente método? (Elegir todos eso aplica.)

```
cuenta vacía privadaAttendees() { int
    participantes = 4, animales = 2, artistas = -1; while((participantes =
    participantes+1) < 10) {} hacer {} while (animales++ <= 1);
    for( ; artistas<2; artistas+=2) {}



    System.out.println(participantes);
    System.out.println(animales);
    System.out.println(artistas);
}
```

R.6 _

B.3 _

C.4 _

D.5 _

E.10 _

F.9 _

G. El código no se compila.

H. Ninguna de las anteriores

Machine Translated by Google



18. ¿Qué afirmaciones sobre la coincidencia de patrones y el alcance del flujo son correctas? (Elegir todos eso aplica.)

A. La coincidencia de patrones con una declaración if se implementa utilizando el operador de instancia.

B. La coincidencia de patrones con una declaración if se implementa utilizando el operador instancia.

C. La coincidencia de patrones con una declaración if se implementa utilizando el operador instancia de.

D. No se puede acceder a la variable de patrón después de la declaración if en la que se declara.

E. El alcance del flujo significa que una variable de patrón solo es accesible si el compilador puede discernir su tipo.

F. La coincidencia de patrones se puede utilizar para declarar una variable con una declaración else.

19. ¿Cuál es el resultado del siguiente fragmento de código?

```
doble iguana = 0;
hacer {
        serpiente int = 1;
        System.out.print(serpiente++ + " ");
        iguana--;
} mientras (serpiente <= 5);
System.out.println(iguana);
```

R. 1 2 3 4 -4,0

B. 1 2 3 4 -5,0

C. 1 2 3 4 5 -4,0

D. 0 1 2 3 4 5 -5,0

E. El código no se compila.

F. El código se compila pero produce un bucle infinito en tiempo de ejecución.

G. Ninguno de los anteriores

20. ¿ Qué declaraciones, cuando se insertan en los siguientes espacios en blanco, permiten que el código se compile y ¿Correr sin entrar en un bucle infinito? (Elija todas las que correspondan).

```
altura interna = 1;
L1: mientras (altura++ <10) {
        humedad larga = 12;
        L2: hacer {
            if(humedad-- % 12 == 0) int _____;
            temperatura = 30;
            L3: para( ; ) {   ;
                temperatura++;
                si(temperatura>50) _____;
            }
        } mientras (humedad > 4);
}
```

Machine Translated by Google



A. romper L2 en la línea 8; continúa L2 en la línea 12

B. continuar en la línea 8; continúa en la línea 12

C. romper L3 en la línea 8; romper L1 en la línea 12

D. continuar L2 en la línea 8; continúa L3 en la línea 12

E. continuar L2 en la línea 8; continúa L2 en la línea 12

F. Ninguna de las anteriores, ya que el código contiene un error de compilación.

21. Un mínimo de cuántas líneas deben corregirse antes del siguiente método ¿Se compilará?

```
21: void findZookeeper (identificación larga) {
22:         System.out.print(interruptor(id) {
23:             caso 10 -> {"Jane"}
24:             caso 20 -> {rendimiento "Lisa";};
25:             caso 30 -> "Kelly";
26:             caso 30 -> "Sara";
27:             predeterminado -> "No asignado";
28:         });
29: }
```

A. Cero

B. Uno

C. Dos

D. Tres

E. Cuatro

F. cinco

22. ¿Cuál es el resultado del siguiente fragmento de código? (Elija todas las que correspondan).

```
2: var tailFeathers = 3;
3: var final uno = 1;
4: cambiar (tailFeathers) {
5:         caso uno: System.out.print(3 + " ");
6:         predeterminado: caso 3: System.out.print(5 + " ");
7: }
8: mientras (colaFeathers > 1) {
9:         System.out.print(--tailFeathers + " "); }
```

R.3 _

B.5 1

C.5 2

D.3 5 1

E.5 2 1

F. El código no se compilará debido a las líneas 3 a 5.

G. El código no se compilará debido a la línea 6.

Machine Translated by Google



23. ¿Cuál es el resultado del siguiente fragmento de código?

```
15: int pingüino = 50, tortuga = 75; 16: booleano
mayor = pingüino >= tortuga; 17: if (mayor = verdadero)
System.out.println("Éxito"); 18: else System.out.println("Error"); 19: else
if(pingüino!= 50) System.out.println("Otro");
```

A. Éxito

B. Fracaso

C. Otros

D. El código no se compilará debido a la línea 17.

E. El código se compila pero genera una excepción en tiempo de ejecución.

F. Ninguna de las anteriores

24. ¿ Cuáles de los siguientes son posibles tipos de datos para amigos que permitirían que el código ¿compilar? (Elija todas las que correspondan).

```
para(var amigo en amigos) {
    System.out.println(amigo);
}
```

Un conjunto

B. Mapa

C. Cadena D.

int[]

E. Colección

F. Constructor de cadenas

G. Ninguno de los anteriores

25. ¿Cuál es el resultado del siguiente fragmento de código?

```
6: Instrumento de cuerda = "violín"; 7: cadena
final CELLO = "violonchelo";
8: Cuerda viola = "viola"; 9: int p = -1;
10: interruptor
(instrumento) { caso "bajo": pausa;
11:         caso VIOLONCHELO : p++;
12:         predeterminado: p++;
13:         caso "VIOLÍN": p+
14:         +; caso "viola" : ++p;
15:         romper;
dieciséis: }
17: Impresión.de.salida.del.sistema(p);
```

Machine Translated by Google



R. -1

b.0 _

c.1 _

D.2 _

E.3 _

F. El código no se compila.

26. ¿Cuál es el resultado del siguiente fragmento de código? (Elija todas las que correspondan).

```
9: int w = 0, r = 1;
10: Nombre de cadena = "";
11: mientras (w < 2) {
12:         nombre += "A";
13:         hacer {
14:             nombre += "B";
15:             if(nombre.longitud()>0) nombre += "C";
dieciséis:          de lo contrario romper;
17:         } mientras (r <=1);
18:         r++; w++; }
19: System.out.println(nombre);
```

A.ABC _

B. ABCABC

C. ABCABCABC

D. La línea 15 contiene un error de compilación.

E. La línea 18 contiene un error de compilación.

F. El código se compila pero nunca finaliza en tiempo de ejecución.

G. El código se compila pero arroja una NullPointerException en tiempo de ejecución.

27. ¿Qué se imprime en el siguiente fragmento de código?

```
23: byte anfibio = 1;
24: Nombre de cadena = "Rana";
25: Color de cuerda = cambiar(anfibio) {
26:         caso 1 -> { producir "Rojo"; } caso
27:         2 -> { if(name.equals("Rana")) produce "Verde"; }
28:         caso 3 -> { rendimiento "Púrpura"; }
29:         predeterminado -> lanzar nueva RuntimeException();
30: };
31: Sistema.out.print(color);
```

Machine Translated by Google



Un rojo

B. verde

C. Púrpura

D. rojopúrpura

E. Se lanza una excepción en tiempo de ejecución.

F. El código no se compila.

28. ¿Cuál es el resultado de llamar a getFish("goldie")?

```
40: void getFish (pez objeto) {
41:     if (!(instancia de pez de String guppy))
42:         System.out.print("¡Come!");
43:     else if (!(instancia de pez de String guppy)) {
44:         lanzar nueva RuntimeException();
45:     }
46:     System.out.print("¡Nadar!");
47: }
```

R. ¡Come!

B. ¡Nadar!

C. ¡Come! seguido de una excepción.

D. ¡Comer! ¡Nadar!

E. Se imprime una excepción.

F. Ninguna de las anteriores

29. ¿Cuál es el resultado del siguiente código?

```
1: clase pública PrintIntegers {
2:     público estático vacío principal (String [] argumentos) {
3:         int y = -2;
4:         hacer System.out.print(++y + " ");
5:         mientras(y <= 5);
6: } }
```

R. -2 -1 0 1 2 3 4 5

segundo -2 -1 0 1 2 3 4

-1 0 1 2 3 4 5 6

-1 0 1 2 3 4 5

E. El código no se compilará debido a la línea 5.

F. El código contiene un bucle infinito y no termina.

Machine Translated by Google

Capítulo

4

# API principales

OBJETIVOS DEL EXAMEN OCP CUBIERTOS EN ESTE CAPÍTULO:

Manejo de valores de fecha, hora, texto, numéricos y booleanos.

- Utilice primitivas y clases contenedoras, incluida la API de matemáticas, paréntesis, promoción de tipos y conversión para evaluar expresiones aritméticas y booleanas.
- Manipular texto, incluidos bloques de texto, utilizando String y String clases de constructor
- Manipular fecha, hora, duración, período, instante y zona horaria objetos que utilizan la API de fecha y hora

Trabajar con matrices y colecciones

- Crear matrices Java, colecciones List, Set, Map y Deque, y agregar, eliminar, actualizar, recuperar y ordenar sus elementos

Machine Translated by Google

En el contexto de una interfaz de programación de aplicaciones (API), una interfaz se refiere a un grupo de clases o definiciones de interfaz Java que le brindan acceso a la funcionalidad.

En este capítulo, aprenderá sobre muchas estructuras de datos centrales en Java, junto con las API más comunes para acceder a ellas. Por ejemplo, String y StringBuilder, junto con sus API asociadas, se utilizan para crear y manipular datos de texto. Luego cubrimos las matrices. Finalmente, exploramos las API matemáticas y de fecha/hora.

# Crear y manipular cadenas

La clase String es una clase tan fundamental que sería difícil escribir código sin ella. Después de todo, ni siquiera puedes escribir un método main() sin usar la clase String .
Una cadena es básicamente una secuencia de caracteres; aquí hay un ejemplo:

```
Nombre de cadena = "Fluffy";
```

Como aprendió en el Capítulo 1, “Bloques de construcción”, este es un ejemplo de un tipo de referencia. También aprendió que los tipos de referencia se crean utilizando la nueva palabra clave. Espera un minuto. Algo le falta al ejemplo anterior: ¡no tiene novedades ! En Java, estos dos fragmentos crean una cadena:

```
Nombre de cadena = "Fluffy";
Nombre de cadena = nueva cadena ("Fluffy");
```

Ambos le brindan una variable de referencia llamada nombre que apunta al objeto String "Fluffy".
Son sutilmente diferentes, como verá más adelante en este capítulo. Por ahora, recuerde que la clase String es especial y no es necesario crear una instancia con new.

Además, los bloques de texto son otra forma de crear una cadena. Para revisar, este bloque de texto es el Igual que las variables anteriores:

```
Nombre de cadena = """
                   Esponjoso""";
```

Dado que una cadena es una secuencia de caracteres, probablemente no le sorprenderá saber que implementa la interfaz CharSequence. Esta interfaz es una forma general de representar varias clases, incluidas String y StringBuilder. Aprenderá más sobre las interfaces en el Capítulo 7, "Más allá de las clases".

En esta sección, analizamos la concatenación, los métodos comunes y el encadenamiento de métodos.

Machine Translated by Google



# Concatenando

En el Capítulo 2, “Operadores”, aprendiste a sumar números. 1 + 2 es claramente 3. Pero, ¿qué es "1" + "2"? Es "12" porque Java combina los dos objetos String . Colocando una cuerda antes de la otra Cadena y combinarlos se llama concatenación de cadenas. A los creadores del examen les gusta la concatenación de cadenas porque el operador + se puede usar de dos maneras dentro de la misma línea de código. No hay muchas reglas que saber para esto, pero hay que conocerlas bien:

1. Si ambos operandos son numéricos, + significa suma numérica.
2. Si alguno de los operandos es una cadena, + significa concatenación.
3. La expresión se evalúa de izquierda a derecha.

Ahora veamos algunos ejemplos:

```
System.out.println(1 + 2);          // 3
System.out.println("a" + "b");      // ab
System.out.println("a" + "b" + 3);  //ab3
System.out.println(1 + 2 + "c");    // 3c
System.out.println("c" + 1 + 2);    // c12
System.out.println("c" + nulo);     // cnull
```

El primer ejemplo utiliza la primera regla. Ambos operandos son números, por lo que usamos la suma normal. El segundo ejemplo es la concatenación de cadenas simple, descrita en la segunda regla. Las comillas de String sólo se utilizan en el código; no obtienen resultados.

El tercer ejemplo combina la segunda y tercera reglas. Como comenzamos por la izquierda, Java descubre qué se evalúa "a" + "b" . Ese ya lo conoces: es "ab". Luego Java mira la expresión restante de "ab" + 3. La segunda regla nos dice que concatenemos ya que uno de los operandos es una Cadena.

En el cuarto ejemplo, comenzamos con la tercera regla, que nos dice que consideremos 1 + 2. Ambos operandos son numéricos, por lo que la primera regla nos dice que la respuesta es 3. Luego tenemos 3 + "c", que usa la segunda regla para darnos "3c". ¿Notas que las tres reglas se usan en una línea?

El quinto ejemplo muestra la importancia de la tercera regla. Primero tenemos "c" + 1, que utiliza la segunda regla para darnos "c1". Luego tenemos "c1" + 2, que usa nuevamente la segunda regla para darnos "c12".

Finalmente, el último ejemplo muestra cómo se representa nulo como una cadena cuando se concatena o impreso, dándonos "cnull".

El examen lleva el engaño un paso más allá e intentará engañarte con algo como esto:

```
int tres = 3;
Cadena cuatro = "4";
System.out.println(1 + 2 + tres + cuatro);
```

Cuando vea esto, tómelo con calma, recuerde las tres reglas y asegúrese de verificar los tipos de variables. En este ejemplo, comenzamos con la tercera regla, que nos dice que consideremos 1 + 2.

Machine Translated by Google



La primera regla nos da 3. Luego, tenemos 3 + tres. Como tres es de tipo int, todavía usamos la primera regla, lo que nos da 6. Luego, tenemos 6 + cuatro. Dado que cuatro es de tipo Cadena, cambiamos a la segunda regla y obtenemos una respuesta final de "64". Cuando vea preguntas como esta, tómese su tiempo y verifique los tipos. Ser metódico vale la pena.

Hay una cosa más que debes saber sobre la concatenación, pero es fácil. En este ejemplo, usted

Sólo tengo que recordar lo que hace += . Tenga en cuenta que s += "2" significa lo mismo que s = s + "2".

```
4: var s = "1"; 5:          // s actualmente tiene "1"
s += "2"; 6:                // s actualmente tiene "12"
s+= 3; 7:                   // s actualmente contiene "123"
System.out.println(s);      // 123
```

En la línea 5, estamos “agregando” dos cadenas, lo que significa que las concatenamos. La línea 6 intenta engañarte sumando un número, pero es como si escribiéramos s = s + 3. Sabemos que una cadena “más” cualquier otra cosa significa usar concatenación.

Para revisar las reglas una vez más: use la suma numérica si hay dos números involucrados, use en caso contrario, concatene y evalúe de izquierda a derecha. ¿Ya has memorizado estas tres reglas? ¡Asegúrate de hacerlo antes del examen!

## Métodos de cadena importantes

La clase String tiene docenas de métodos. Afortunadamente, sólo necesitas saber unos pocos para el examen. Los creadores del examen eligen la mayoría de los métodos que utilizan los desarrolladores en el mundo real.

Para todos estos métodos, debe recordar que una cadena es una secuencia de caracteres y Java cuenta desde 0 cuando se indexa. La Figura 4.1 muestra cómo se indexa cada carácter de la cadena "animales" .

FIGURA 4.1 Indexación de una cadena

| 0 123 4 | | | | 5 | 6 |
|---|---|---|---|---|---|
| animar un | | | | yo | s |

También necesita saber que una Cadena es inmutable o inmutable. Esto significa que llamar a un método en una Cadena devolverá un objeto Cadena diferente en lugar de cambiar el valor de la referencia. En este capítulo, utilizará objetos inmutables. En el Capítulo 6, “Diseño de clases”, aprenderá a crear sus propios objetos inmutables.

Veamos varios métodos de la clase String . Muchos de ellos son sencillos. ward, por lo que no los discutiremos en detalle. Necesita saber cómo utilizar estos métodos.

## Determinando la longitud

El método length() devuelve el número de caracteres de la cadena. La firma del método es la siguiente:

```
longitud int pública ()
```

Machine Translated by Google



El siguiente código muestra cómo usar length():

```
var nombre = "animales";
System.out.println(nombre.longitud()); // 7
```

Esperar. ¿Tiene salida 7? ¿No te acabamos de decir que Java cuenta desde 0? La diferencia es ese conteo cero ocurre solo cuando se usan índices o posiciones dentro de una lista. Al determinar el tamaño o la longitud total, Java vuelve a utilizar el conteo normal.

## Obtener un solo personaje

El método charAt() le permite consultar la cadena para averiguar qué carácter está en un índice específico. La firma del método es la siguiente:

```
char público charAt ( índice int)
```

El siguiente código muestra cómo utilizar charAt():

```
var nombre = "animales";
System.out.println(nombre.charAt(0)); // a
System.out.println(nombre.charAt(6)); // s
System.out.println(nombre.charAt(7)); // excepción
```

Dado que los índices comienzan a contar con 0, charAt(0) devuelve el "primer" carácter en el secuencia. De manera similar, charAt(6) devuelve el “séptimo” carácter de la secuencia. Sin embargo, charAt(7) es un problema. Solicita el “octavo” carácter de la secuencia, pero sólo hay siete caracteres presentes. Cuando algo sale mal y Java no sabe cómo solucionarlo, genera una excepción, como se muestra aquí. Encontrará más información sobre las excepciones en el Capítulo 11, “Excepciones y localización”.

```
java.lang.StringIndexOutOfBoundsException: índice de cadena fuera de rango: 7
```

## Encontrar un índice

El método indexOf() mira los caracteres de la cadena y encuentra el primer índice que coincide con el valor deseado. El método indexOf puede funcionar con un carácter individual o una cadena completa como entrada. También puede comenzar desde una posición solicitada. Recuerda que un char se puede pasar a un tipo de parámetro int . En el examen, solo verá un carácter pasado a los parámetros denominados ch. Las firmas del método son las siguientes:

```
índice int público de (int ch)

público int indexOf(int ch, int fromIndex)
índice int público de (cadena cadena)
public int indexOf(String str, int fromIndex)
```

El siguiente código le muestra cómo utilizar indexOf():

```
var nombre = "animales";
System.out.println(nombre.indexOf('a'));          // 0
```

Machine Translated by Google



```
System.out.println(nombre.indexOf("al"));        // 4
System.out.println(nombre.indexOf('a', 4));      // 4
System.out.println(nombre.indexOf("al", 5));     // -1
```

Como los índices comienzan con 0, la primera 'a' coincide en esa posición. La segunda declaración busca una cadena más específica, por lo que coincide más tarde. La tercera declaración dice que Java ni siquiera debería mirar los caracteres hasta llegar al índice 4. La declaración final no encuentra nada porque comienza a buscar después de que se produjo la coincidencia. A diferencia de charAt(), el indexOf()
El método no genera una excepción si no puede encontrar una coincidencia, sino que devuelve –1. Como los índices comienzan con 0, la persona que llama sabe que –1 no podría ser un índice válido. Esto lo convierte en un valor común para un método para indicarle a la persona que llama que no se encuentra ninguna coincidencia.

## Obtener una subcadena

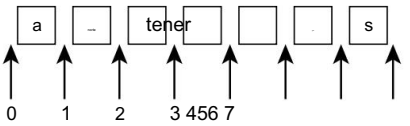
El método substring() también busca caracteres en una cadena. Devuelve partes de la cadena.
El primer parámetro es el índice con el que comenzar para la cadena devuelta. Como es habitual, se trata de un índice de base cero. Hay un segundo parámetro opcional, que es el índice final en el que desea detenerse.

Observe que dijimos "detenerse en" en lugar de "incluir". Esto significa el parámetro endIndex
Se permite pasar uno del final de la secuencia si desea detenerse al final de la secuencia. Sin embargo, eso sería redundante, ya que en ese caso podría omitir el segundo parámetro por completo. En su propio código, desea evitar esta redundancia. Sin embargo, no se sorprenda si el examen lo utiliza. Las firmas del método son las siguientes:

```
subcadena de cadena pública (int comenzarIndex)
subcadena de cadena pública (int startIndex, int endIndex)
```

Es útil pensar en los índices de forma un poco diferente para los métodos de subcadena. Fingir los índices están justo antes del personaje al que señalarían. La figura 4.2 ayuda a visualizar esto. Observe cómo la flecha con el 0 apunta al carácter que tendría índice 0. La flecha con el 1 apunta entre caracteres con índices 0 y 1. Hay siete caracteres en la Cadena. Dado que Java utiliza índices de base cero, esto significa que el último carácter tiene un índice de 6. La flecha con los 7 apunta inmediatamente después de este último carácter. Esto le ayudará a recordar que endIndex no da una excepción fuera de límites cuando pasa del final de la cadena.

FIGURA 4.2 Índices para una subcadena




El siguiente código muestra cómo utilizar la subcadena():

```
var nombre = "animales";
System.out.println(nombre.subcadena(3)); // malos
System.out.println(nombre.substring(nombre.indexOf('m'))); // malos
```

Machine Translated by Google



```
System.out.println(nombre.subcadena(3, 4));    // metro
System.out.println(nombre.subcadena(3, 7));    // malos
```

El método substring() es el método String más complicado del examen. El primer ejemplo dice para llevar los caracteres que comienzan con el índice 3 hasta el final, lo que nos da "mals". El segundo ejemplo hace lo mismo, pero llama a indexOf() para obtener el índice en lugar de codificarlo. Esta es una práctica común al codificar porque es posible que no conozca el índice de antemano.

El tercer ejemplo dice que se deben tomar los caracteres que comienzan con el índice 3 hasta el carácter del índice 4, pero sin incluirlo. Esta es una forma complicada de decir que queremos una cadena. con un carácter: el del índice 3. Esto da como resultado "m". El último ejemplo dice que debemos tomar los caracteres que comienzan con el índice 3 hasta llegar al índice 7. Dado que el índice 7 es el mismo que el final de la cadena, es equivalente al primer ejemplo.

Esperamos que no haya sido demasiado confuso. Los siguientes ejemplos son menos obvios:

```
System.out.println(nombre.subcadena(3, 3)); // cuerda vacía
System.out.println(nombre.subcadena(3, 2)); // excepción
System.out.println(nombre.subcadena(3, 8)); // excepción
```

El primer ejemplo de este conjunto imprime una cadena vacía. La solicitud es para los caracteres que comienzan con el índice 3 hasta llegar al índice 3. Como comenzamos y terminamos con el mismo índice, no hay caracteres intermedios. El segundo ejemplo de este conjunto genera una excepción porque los índices no pueden estar al revés. Java sabe perfectamente que nunca llegará al índice 2 si comienza con el índice 3. El tercer ejemplo dice que continúe hasta el octavo carácter. No existe una octava posición, por lo que Java genera una excepción. Por supuesto, tampoco hay un séptimo carácter, pero al menos existe la posición invisible del “fin de cadena”.

Repasemos esto una vez más ya que substring() es muy complicado. El método devuelve la cadena. a partir del índice solicitado. Si se solicita un índice final, se detiene justo antes de ese índice. De lo contrario, irá al final de la cadena.

## Caso de ajuste

Uf. Después de ese ejercicio mental, ¡es bueno tener métodos que actúen exactamente como suenan! Estos métodos facilitan la conversión de sus datos. Las firmas del método son las siguientes:

```
cadena pública a minúsculas()
Cadena pública en mayúsculas()
```

El siguiente código muestra cómo utilizar estos métodos:

```
var nombre = "animales";
System.out.println(nombre.toUpperCase());        // ANIMALES
System.out.println("Abc123".toLowerCase()); // abc123
```

Estos métodos hacen lo que dicen. El método toUpperCase() convierte los caracteres en minúscula a mayúsculas en la cadena devuelta. El método toLowerCase() convierte los caracteres en mayúsculas a minúsculas en la cadena devuelta. Estos métodos dejan en paz cualquier carácter que no sean letras. Además, recuerde que las cadenas son inmutables, por lo que la cadena original permanece igual.

Machine Translated by Google



## Comprobando la igualdad

El método equals() comprueba si dos objetos String contienen exactamente los mismos caracteres en el mismo orden. El método equalsIgnoreCase() comprueba si dos cadenas
Los objetos contienen los mismos caracteres, con la excepción de que ignora el caso de los caracteres. Las firmas del método son las siguientes:

```
público booleano es igual (Objeto obj)
booleano público es igual a IgnoreCase (String str)
```

Es posible que hayas notado que equals() toma un Objeto en lugar de una Cadena. Esto se debe a que el método es el mismo para todos los objetos. Si pasa algo que no es una cadena, simplemente devolverá falso. Por el contrario, el método equalsIgnoreCase() solo se aplica a objetos String , por lo que puede tomar el tipo más específico como parámetro.

En Java, los valores de cadena distinguen entre mayúsculas y minúsculas. Eso significa que "abc" y "ABC" se consideran diferentes. valores ent. Teniendo esto en cuenta, el siguiente código muestra cómo utilizar estos métodos:

```
System.out.println("abc".equals("ABC")); // FALSO
System.out.println("ABC".equals("ABC")); // verdadero
System.out.println("abc".equalsIgnoreCase("ABC")); // verdadero
```

Este ejemplo debería ser bastante intuitivo. En el primer ejemplo, los valores no son exactamente los mismos. En el segundo son exactamente iguales. En el tercero, difieren solo por caso, pero está bien porque llamamos al método que ignora las diferencias por caso.

### Anulando toString(), igual a(Objeto) y hashCode()

Saber cómo anular correctamente toString(), equals(Object) y hashCode()
Fue parte de los exámenes de certificación de Java en el pasado. Como desarrollador profesional de Java, sigue siendo importante que conozca al menos las reglas básicas para anular cada uno de estos métodos:

- toString(): el método toString() se llama cuando intenta imprimir un objeto o concatenar el objeto con un String. Por lo general, se reemplaza con una versión que imprime una descripción única de la instancia utilizando sus campos de instancia.
- igual(Objeto): el método igual(Objeto) se utiliza para comparar objetos, con la implementación predeterminada simplemente usando el operador == . Debe anular el método igual (Objeto) cada vez que desee comparar convenientemente la igualdad de elementos, especialmente si esto requiere verificar numerosos campos.
- hashCode(): cada vez que anules igual(Objeto), debes anular hashCode() para ser consistente. Esto significa que para dos objetos cualesquiera, si a.equals(b) es verdadero, entonces a.hashCode()==b.hashCode() también debe ser verdadero. Si no son consistentes, esto podría generar datos no válidos y efectos secundarios en colecciones basadas en hash como HashMap y HashSet.

Todos estos métodos proporcionan una implementación predeterminada en Object, pero si desea hacer un uso inteligente de ellos, debe anularlos.

Machine Translated by Google



## Buscando subcadenas

A menudo, es necesario buscar una cadena más grande para determinar si contiene una subcadena. Los métodos comienza con () y termina con () miran si el valor proporcionado coincide con parte de la cadena. El método contiene() no es tan particular; busca coincidencias en cualquier parte de la cadena. Las firmas del método son las siguientes:

```
booleano público comienza con ( prefijo de cadena)
booleano público termina con ( sufijo de cadena)
booleano público contiene (CharSequence charSeq)
```

El siguiente código muestra cómo utilizar estos métodos:

```
System.out.println("abc".startsWith("a")); // verdadero
System.out.println("abc".startsWith("A")); // FALSO

System.out.println("abc".endsWith("c")); // verdadero
System.out.println("abc".endsWith("a")); // FALSO

System.out.println("abc".contains("b")); // verdadero
System.out.println("abc".contains("B")); // FALSO
```

Una vez más, no hay nada sorprendente aquí. Java está realizando una verificación que distingue entre mayúsculas y minúsculas de los valores proporcionados. Tenga en cuenta que el método contiene() es un método conveniente, por lo que no es necesario escribir str.indexOf(otherString) != -1.

## Reemplazo de valores

El método replace() realiza una búsqueda simple y reemplaza en la cadena. Hay una versión que toma parámetros char así como una versión que toma parámetros CharSequence . Las firmas del método son las siguientes:

```
Reemplazo de cadena pública (char oldChar, char newChar)
Reemplazo de cadena pública (objetivo de CharSequence, reemplazo de CharSequence)
```

El siguiente código muestra cómo utilizar estos métodos:

```
System.out.println("abcabc".replace('a', 'A')); // AbcAbc
System.out.println("abcabc".replace("a", "A")); // AbcAbc
```

El primer ejemplo utiliza la firma del primer método y pasa parámetros char . El segundo El ejemplo utiliza la firma del segundo método y pasa parámetros de cadena .

## Eliminar espacios en blanco

Estos métodos eliminan espacios en blanco del principio y/o final de una cadena. la tira() y los métodos trim() eliminan los espacios en blanco del principio y del final de una cadena. En términos del examen, los espacios en blanco se componen de espacios junto con los caracteres \t (tabulación) y \n (nueva línea). Otros caracteres, como \r (retorno de carro), también se incluyen en lo que se recorta. El método strip() hace todo lo que hace trim() , pero es compatible con Unicode.

Machine Translated by Google

No es necesario saber acerca de Unicode para el examen. Pero si desea probar la diferencia, uno de los caracteres de espacio en blanco Unicode es el siguiente:

```
char ch = '\u2000';
```

Además, el método stripLeading() elimina los espacios en blanco del principio de la cadena y los deja al final. El método stripTrailing() hace lo contrario. Elimina los espacios en blanco del final de la cadena y lo deja al principio. Las firmas del método son las siguientes:

```
tira de cadena pública ()
tira de cadena públicaLeading ()
tira de cadena públicaTrailing ()
recorte de cadena pública ()
```

El siguiente código muestra cómo utilizar estos métodos:

```
System.out.println("abc".strip());                    // a B C
System.out.println("\tab c\n".strip());               // a B C

Texto de cadena = " abc\t ";
System.out.println(text.trim().length()); // 3
System.out.println(text.strip().length()); // 3
System.out.println(text.stripLeading().length()); // 5
System.out.println(text.stripTrailing().length());// 4
```

Primero, recuerda que \t es un solo carácter. La barra invertida escapa de la t para representar una pestaña. El primer ejemplo imprime la cadena original porque no hay espacios en blanco al principio ni al final. El segundo ejemplo elimina la pestaña inicial, los espacios posteriores y la nueva línea final. Deja los espacios que hay en medio de la cuerda.

Los ejemplos restantes simplemente imprimen el número de caracteres restantes. Puedes ver eso trim() y strip() dejan los mismos tres caracteres "abc" porque eliminan los espacios en blanco iniciales y finales. El método stripLeading() solo elimina el carácter de espacio en blanco al principio de la cadena. Deja la pestaña y el espacio al final. El método stripTrailing() elimina estos dos caracteres al final pero deja el carácter al principio de la cadena.

## Trabajar con sangría

Ahora que Java admite bloques de texto, es útil tener métodos que se ocupen de la sangría. Ambos son un poco complicados, ¡así que lee atentamente!

```
Sangría de cadena pública (int númeroEspacios)
tira de cadena públicaIndent ()
```

Machine Translated by Google



El método indent() agrega la misma cantidad de espacios en blanco al comienzo de cada línea si pasa un número positivo. Si pasa un número negativo, intenta eliminar esa cantidad de caracteres de espacio en blanco del principio de la línea. Si pasa cero, la sangría no cambiará.

Si llama a indent() con un número negativo e intenta eliminar más espacios en blanco de los que están presentes al principio de la línea, Java eliminará todos los que pueda encontrar.

Esto parece bastante sencillo. Sin embargo, sangría() también normaliza los espacios en blanco. ¿Qué significa normalizar los espacios en blanco? Primero, se agrega un salto de línea al final de la cadena, si aún no está allí. En segundo lugar, cualquier salto de línea se convierte al formato \n . Independientemente de si su sistema operativo utiliza \r\n (Windows) o\n (Mac/ Unix), Java se estandarizará en \n por usted.

El método stripIndent() es útil cuando se creó una cadena con concatenación en lugar de usar un bloque de texto. Elimina todos los espacios en blanco incidentales. Esto significa que todas las líneas que no están en blanco se desplazan hacia la izquierda, de modo que se elimina la misma cantidad de caracteres de espacio en blanco de cada línea y el primer carácter que queda no está en blanco. Al igual que sangría(), \r\n se convierte en \n. Sin embargo, el método stripIndent() no agrega un salto de línea final si falta.

Bueno, eran muchas reglas. La tabla 4.1 proporciona una referencia para que sean más fáciles de recordar.

TABLA 4.1 Reglas para indent() y stripIndent()

| Método | cambio de sangría | Normaliza los saltos de línea existentes | Agrega un salto de línea al final si falta |
|---|---|---|---|
| sangría(n) donde n > 0 | Agrega n espacios al comienzo de cada línea | Sí | Sí |
| sangría(n) donde n == 0 | Sin cambios | Sí | Sí |
| sangría(n) donde n < 0 | Elimina hasta n espacios de cada línea donde se elimina el mismo número de caracteres de cada línea que no está en blanco | Sí | Sí |
| tiraIndent() | Elimina todos los espacios en blanco incidentales principales | Sí | No |

Machine Translated by Google



El siguiente código muestra cómo utilizar estos métodos. No te preocupes si los resultados no son que esperas. Te explicamos cada uno.

```
10: donde bloque =      """
11:                     a
12:                       b
13:                     C""";
14: cal viva =            " un"
15:                   +   "b\n"
dieciséis:            +   " C";
17: System.out.println(block.length()); // 6
18: System.out.println(concat.length()); // 9
19: System.out.println(block.indent(1).length()); // 10
20: System.out.println(concat.indent(-1).length()); // 7
21: System.out.println(concat.indent(-4).length()); // 6
22: System.out.println(concat.stripIndent().length()); // 6
```

Las líneas 10 a 16 crean cadenas similares utilizando un bloque de texto y una cadena normal, respectivamente. Decimos "similar" porque concat tiene un carácter de espacio en blanco al principio de cada línea, mientras que block no.

La línea 17 cuenta los seis caracteres del bloque, que son las tres letras, el espacio en blanco antes de b y el \n después de a y b. La línea 18 cuenta los nueve caracteres en concat, que son las tres letras, un espacio en blanco antes de a, dos espacios en blanco antes de b, un espacio en blanco antes de c y \n después de a y b. Cuéntelos usted mismo. Si no comprende qué caracteres se cuentan, todo se volverá más confuso.

En la línea 19, le pedimos a Java que agregue un único espacio en blanco a cada una de las tres líneas del bloque. Sin embargo, el resultado dice que agregamos 4 caracteres en lugar de 3 ya que la longitud pasó de 6 a 10. Este misterioso carácter adicional se debe a la normalización de la terminación de línea.
Como el bloque de texto no tiene un salto de línea al final, ¡ indent() agrega uno!

En la línea 20, eliminamos un carácter de espacio en blanco de cada una de las tres líneas de concat. Esto da una longitud de siete. Comenzamos con nueve, nos deshicimos de tres caracteres y agregamos una nueva línea final normalizada.

En la línea 21, le pedimos a Java que elimine cuatro espacios en blanco de las mismas tres líneas. Como no hay cuatro espacios en blanco, Java hace lo mejor que puede. El espacio único se elimina antes de a y c. Ambos espacios se eliminan antes de b. La longitud de seis debería tener sentido aquí; Eliminamos un carácter más aquí que en la línea 20.

Finalmente, la línea 22 usa el método stripIndent() . Todas las líneas tienen al menos un carácter de espacio en blanco. Como no todos tienen dos caracteres de espacio en blanco, el método solo elimina un carácter por línea. Dado que stripIndent() no agrega ninguna nueva línea , la longitud es seis, que es tres menos que las nueve originales.

Machine Translated by Google



## Traduciendo fugas

Cuando escapamos caracteres, usamos una sola barra invertida. Por ejemplo, \t es una pestaña. Si no queremos este comportamiento, agregamos otra barra invertida para escapar de la barra invertida, por lo que \\t es la cadena literal \t. El método TranslateEscapes() toma estos literales y los convierte en el carácter de escape equivalente. La firma del método es la siguiente:

cadena pública traducirEscapes()

El siguiente código muestra cómo utilizar estos métodos:

```
var cadena = "1\\t2";
System.out.println(cadena); // 1\t2
System.out.println(str.translateEscapes()); // 1 2
```

La primera línea imprime la cadena literal \t porque se escapa la barra invertida. La segunda línea imprime una pestaña real desde que tradujimos el escape. Este método se puede utilizar para secuencias de escape como \t (tabulación), \n (nueva línea), \s (espacio), \" (comilla doble) y \' (comilla simple).

## Comprobación de cadenas vacías o en blanco

Java proporciona métodos convenientes para saber si una cadena tiene una longitud de cero o contiene solo caracteres de espacio en blanco. Las firmas del método son las siguientes:

booleano público está vacío ()
booleano público está en blanco()

El siguiente código muestra cómo utilizar estos métodos:

```
System.out.println(" ".isEmpty()); // FALSO
System.out.println("".isEmpty()); // verdadero
System.out.println(" ".isBlank()); // verdadero
System.out.println("".isBlank()); // verdadero
```

La primera línea imprime falso porque la cadena no está vacía; tiene un espacio en blanco. La segunda línea imprime verdadero porque esta vez no hay caracteres en la cadena. Las dos últimas líneas se imprimen como verdaderas porque no hay más caracteres que espacios en blanco.

## Formatear valores

Existen métodos para formatear valores de cadena utilizando indicadores de formato. Dos de los métodos toman la cadena de formato como parámetro y el otro usa una instancia para ese valor. Un método requiere una configuración regional, que aprenderá en el Capítulo 11.

Los parámetros del método se utilizan para construir una cadena formateada en una sola llamada al método. en lugar de mediante muchas operaciones de formato y concatenación. Devuelven una referencia a la instancia a la que se les llama para que las operaciones se puedan encadenar. Las firmas del método son las siguientes:

formato de cadena estática pública (formato de cadena, argumentos de objeto...)
Formato de cadena estática pública (localización local, formato de cadena, argumentos de objeto...)
Cadena pública formateada ( argumentos de objeto...)

Machine Translated by Google



El siguiente código muestra cómo utilizar estos métodos:

```
var nombre = "Kate";
var ID de pedido = 5;

// Todo impreso: Hola Kate, el pedido 5 está listo
System.out.println("Hola "+nombre+", el pedido "+orderId+" está listo");
System.out.println(String.format("Hola %s, el pedido %d está listo", nombre, orderId));

System.out.println("Hola %s, el pedido %d está listo"
    .formatted(nombre, orderId));
```

En las operaciones format() y formatted() , los parámetros se insertan y se formatean mediante símbolos en el orden en que se proporcionan en vararg. La Tabla 4.2 enumera los que usted debe saber para el examen.

TABLA 4.2 Símbolos de formato comunes

| Símbolo | Descripción |
|---|---|
| %s | Se aplica a cualquier tipo, comúnmente valores de cadena . |
| %d | Se aplica a valores enteros como int y long. |
| %F | Se aplica a valores de punto flotante como float y double |
| %norte | Inserta un salto de línea usando el separador de línea dependiente del sistema. |

El siguiente ejemplo utiliza los cuatro símbolos de la Tabla 4.2:

```
var nombre = "James";
puntuación var = 90,25;
vartotal = 100;
System.out.println("%s:%n Puntuación: %f de %d"
    .formatted(nombre, puntuación, total));
```

Esto imprime lo siguiente:

```
Jaime:
    Puntuación: 90,250000 sobre 100
```

La combinación de tipos de datos puede provocar excepciones en tiempo de ejecución. Por ejemplo, los siguientes lanzamientos una excepción porque se utiliza un número de punto flotante cuando se espera un valor entero:

```
var str = "Comida: %d toneladas".formatted(2.0); // excepción de conversión de formato ilegal
```

Machine Translated by Google



### Usando formato() con banderas

Además de admitir símbolos, Java también admite indicadores opcionales entre % y el carácter de símbolo. En el ejemplo anterior, el número de punto flotante se imprimió como 90,250000. De forma predeterminada, %f muestra exactamente seis dígitos después del decimal. Si desea mostrar solo un dígito después del decimal, puede usar %.1f en lugar de %f. El formato() El método se basa en redondear en lugar de truncar al acortar números. Por ejemplo, 90.250000 se mostrará como 90.3 (no 90.2) cuando se pase a format() con %.1f.

El método format() también admite dos funciones adicionales. Puede especificar la longitud total de la salida utilizando un número antes del símbolo decimal. De forma predeterminada, el método llenará el espacio vacío con espacios en blanco. También puedes llenar el espacio vacío con ceros colocando un solo cero antes del símbolo decimal. Los siguientes ejemplos utilizan corchetes, [], para mostrar el inicio/final del valor formateado:

```
donde pi = 3,14159265359;
System.out.format("[%f]",pi); // [3.141593]
System.out.format("[%12.8f]",pi); // [3.14159265]
System.out.format("[%012f]",pi); // [00003.141593]
System.out.format("[%12.2f]",pi); // [3.14]
System.out.format("[%.3f]",pi);                    // [3.142]
```

El método format() admite muchos otros símbolos y banderas. No necesita conocer ninguno de ellos para el examen más allá de lo que ya hemos comentado.

## Encadenamiento de métodos

¿Listo para reunir todo lo que acabas de aprender? Es común llamar a varios métodos como se muestra aquí:

```
var inicio = "AniMaL ";
var recortado = start.trim(); var                    // "Animal"
minúscula = trimmed.toLowerCase(); resultado         // "animal"
var = minúscula.replace('a', 'A');                   // "Animal"
System.out.println(resultado);
```

Esto es solo una serie de métodos String . Cada vez que se llama a uno, se pone el valor devuelto. en una nueva variable. Hay cuatro valores de cadena en el camino y se genera AnimAl .

Sin embargo, en el examen, hay una tendencia a meter tanto código como sea posible en un pequeño espacio. Verá código utilizando una técnica llamada encadenamiento de métodos. He aquí un ejemplo:

```
Resultado de cadena = "AniMaL ".trim().toLowerCase().replace('a', 'A');
System.out.println(resultado);
```

Machine Translated by Google



Este código es equivalente al ejemplo anterior. También crea cuatro objetos String y genera AnimAl. Para leer código que utiliza encadenamiento de métodos, comience por la izquierda y evalúe el primer método. Luego llame al siguiente método con el valor devuelto del primer método. Continúe hasta llegar al punto y coma.

¿Cuál crees que es el resultado de este código?

```
5: Cadena a = "abc";
6: Cadena b = a.toUpperCase();
7: b = b.replace("B", "2").replace('C', '3');
8: System.out.println("a=" + a);
9: System.out.println("b=" + b);
```

En la línea 5, configuramos a para que apunte a "abc" y nunca apuntamos a nada más. Como nada del código de las líneas 6 y 7 cambia a, el valor sigue siendo "abc".

Sin embargo, b es un poco más complicado. La línea 6 tiene b apuntando a "ABC", lo cual es sencillo. En la línea 7, tenemos encadenamiento de métodos. Primero, se llama "ABC".replace("B", "2") . Esto devuelve "A2C". A continuación, se llama a "A2C".replace('C', '3') . Esto devuelve "A23". Finalmente, b cambia para apuntar a esta cadena devuelta. Cuando se ejecuta la línea 9, b es "A23".

# Usando la clase StringBuilder

Un programa pequeño puede crear muchos objetos String muy rápidamente. Por ejemplo, ¿cuántos objetos crees que crea este fragmento de código?

```
10: Cadena alfa = "";
11: para (char actual = 'a'; actual <= 'z'; actual ++)
12:         alfa += actual;
13: System.out.println(alfa);
```

Se crea una instancia de la cadena vacía en la línea 10 y luego la línea 12 agrega una "a". Sin embargo, debido a que el objeto String es inmutable, se asigna un nuevo objeto String a alfa y el objeto "" pasa a ser elegible para la recolección de basura. La siguiente vez que se realiza el ciclo, a alfa se le asigna un nuevo objeto String , "ab", y el objeto "a" pasa a ser elegible para la recolección de basura. La siguiente iteración asigna alfa a "abc" y el objeto "ab" pasa a ser elegible para la recolección de basura, y así sucesivamente.

Esta secuencia de eventos continúa, y después de 26 iteraciones a través del ciclo, un total de 27 Se crean instancias de objetos, la mayoría de los cuales son inmediatamente elegibles para la recolección de basura.

Esto es muy ineficiente. Por suerte, Java tiene una solución. La clase StringBuilder crea un Cadena sin almacenar todos esos valores de cadena provisionales . A diferencia de la clase String , StringBuilder no es inmutable.

```
15: StringBuilder alfa = nuevo StringBuilder();
16: para (char actual = 'a'; actual <= 'z'; actual ++)
17:         alfa.append(actual);
18: System.out.println(alfa);
```

Machine Translated by Google



En la línea 15, se crea una instancia de un nuevo objeto StringBuilder . La llamada a append() en la línea 17 agrega un carácter al objeto StringBuilder cada vez que pasa por el bucle for , agregando el valor de current al final de alfa. Este código reutiliza el mismo StringBuilder sin crear una cadena provisional cada vez.

En el código antiguo, es posible que veas referencias a StringBuffer. Funciona de la misma manera, excepto que admite subprocesos, sobre los cuales aprenderá en el Capítulo 13, “Simultaneidad”. StringBuffer no está en el examen. Funciona más lento que StringBuilder, así que simplemente use StringBuilder.

En esta sección, veremos la creación de un StringBuilder y el uso de sus métodos comunes.

## Mutabilidad y encadenamiento

Estamos seguros de que lo habrás notado en el ejemplo anterior, pero StringBuilder no es inmutable. De hecho, le dimos 27 valores diferentes en el ejemplo (un espacio en blanco más la suma de cada letra del alfabeto). Es probable que el examen intente engañarlo con respecto a String y StringBuilder. siendo mutable.

El encadenamiento hace que esto sea aún más interesante. Cuando encadenamos llamadas al método String , el resultado fue una nueva Cadena con la respuesta. El encadenamiento de métodos StringBuilder no funciona de esta manera. En cambio, StringBuilder cambia su propio estado y devuelve una referencia a sí mismo. Veamos un ejemplo para que esto quede más claro:

```
4: StringBuilder sb = nuevo StringBuilder ("inicio");
5: sb.append("+medio"); 6:                    // sb = "inicio+medio"
StringBuilder igual = sb.append("+end");      // "inicio+medio+final"
```

La línea 5 agrega texto al final de sb. También devuelve una referencia a sb, que se ignora. Línea 6 también agrega texto al final de sb y devuelve una referencia a sb. Esta vez la referencia se almacena en el mismo. Esto significa que sb y el mismo apuntan al mismo objeto e imprimirían el mismo valor.

El examen no siempre hará que el código sea fácil de leer al tener solo un método por línea. ¿Qué crees que imprime este ejemplo?

```
4: StringBuilder a = nuevo StringBuilder ("abc");
5: StringBuilder b = a.append("de");
6: b = b.append("f").append("g");
7: System.out.println("a=" + a);
8: System.out.println("b=" + b);
```

¿Dijiste que ambos imprimen "abcdefg"? Bien. Aquí sólo hay un objeto StringBuilder . Lo sabemos porque new StringBuilder() se llama solo una vez. En la línea 5, hay dos variables que hacen referencia a ese objeto, que tiene un valor "abcde". En la línea 6, esas dos variables todavía se refieren al mismo objeto, que ahora tiene un valor de "abcdefg".

Por cierto, la tarea de volver a b no hace absolutamente nada. b ya está apuntando a ese StringBuilder.

Machine Translated by Google



## Creando un constructor de cadenas

Hay tres formas de construir un StringBuilder:

```
StringBuilder sb1 = nuevo StringBuilder();
StringBuilder sb2 = nuevo StringBuilder("animal");
StringBuilder sb3 = nuevo StringBuilder(10);
```

El primero dice crear un StringBuilder que contenga una secuencia vacía de caracteres y asignar sb1 para que apunte a él. El segundo dice crear un StringBuilder que contenga un valor específico y asignar sb2 para que apunte a él. Los dos primeros ejemplos le dicen a Java que administre los detalles de implementación. El ejemplo final le dice a Java que tenemos una idea de qué tan grande será el valor final y que nos gustaría que StringBuilder reserve una cierta capacidad, o número de espacios, para los caracteres.

## Métodos importantes de StringBuilder

Al igual que con String, no cubriremos todos los métodos de la clase StringBuilder . Estos son los que podrías ver en el examen.

# Usando métodos comunes

Estos cuatro métodos funcionan exactamente igual que en la clase String . Asegúrese de poder identificar el resultado de este ejemplo:

```
var sb = new StringBuilder("animales");
Cadena sub = sb.substring(sb.indexOf("a"), sb.indexOf("al"));
int len = sb.longitud();
char ch = sb.charAt(6);
System.out.println(sub + " " +solo + " " + canal);
```

La respuesta correcta es anim 7 s. Las llamadas al método indexOf() devuelven 0 y 4, respectivamente. El método substring() devuelve la cadena que comienza con el índice 0 y termina justo antes del índice 4.

El método length() devuelve 7 porque es el número de caracteres en String Builder en lugar de un índice. Finalmente, charAt() devuelve el carácter en el índice 6. Aquí comenzamos con 0 porque nos referimos a índices. Si esto no le resulta familiar, regrese y lea la sección sobre Cadenas nuevamente.

Observe que substring() devuelve una cadena en lugar de un StringBuilder. Por eso no se cambia alguien . El método substring() es en realidad solo un método que pregunta sobre el estado de StringBuilder.

# Agregar valores

El método append() es, con diferencia, el método más utilizado en StringBuilder. De hecho, se usa con tanta frecuencia que comenzamos a usarlo sin comentarios. Por suerte, este método no

Machine Translated by Google



exactamente lo que parece: agrega el parámetro al StringBuilder y devuelve una referencia al StringBuilder actual. Una de las firmas del método es la siguiente:

agregar StringBuilder público (String str)

Observe que dijimos una de las firmas del método. Hay más de 10 firmas de métodos que parecen similares pero toman diferentes tipos de datos como parámetros, como int, char, etc.
Todos esos métodos se proporcionan para que puedas escribir código como este:

```
var sb = new StringBuilder().append(1).append('c');
sb.append("-").append(verdadero);
System.out.println(sb);                    // 1c-verdadero
```

Buen método de encadenamiento, ¿no? El método append() se llama directamente después del constructor.
Al tener todas estas firmas de métodos, puede simplemente llamar a append() sin tener que convertir primero su parámetro a una Cadena .

## Insertar datos

El método insert() agrega caracteres al StringBuilder en el índice solicitado y devuelve una referencia al StringBuilder actual. Al igual que append(), existen muchas firmas de métodos para diferentes tipos. Aquí hay uno:

inserción pública de StringBuilder (int offset, String str)

Preste atención al desplazamiento en estos ejemplos. Es el índice donde queremos insertar el parámetro solicitado.

```
3: var sb = new StringBuilder("animales");
4: sb.insert(7, "-"); // sb = animales 5: sb.insert(0, "-"); // sb =
-animales 6: sb.insert(4, "-"); // sb = -animales 7: System.out.println(sb);
```

La línea 4 dice que se inserte un guión en el índice 7, que resulta ser el final de la secuencia de caracteres. La línea 5 dice que se inserte un guión en el índice 0, que resulta ser el comienzo.
Finalmente, la línea 6 dice que se inserte un guión justo antes del índice 4. Los creadores del examen intentarán hacerle tropezar con esto. A medida que agregamos y eliminamos caracteres, sus índices cambian. Cuando vea una pregunta relacionada con este tipo de operaciones, dibuje lo que está sucediendo utilizando los materiales de escritura disponibles para no confundirse.

## Eliminar contenidos

El método eliminar() es lo opuesto al método insertar() . Elimina caracteres de la secuencia y devuelve una referencia al StringBuilder actual. El método deleteCharAt() es útil cuando desea eliminar solo un carácter. Las firmas del método son las siguientes:

eliminación pública de StringBuilder (int startIndex, int endIndex)
público StringBuilder eliminarCharAt ( índice int )

Machine Translated by Google



El siguiente código muestra cómo utilizar estos métodos:

```
var sb = nuevo StringBuilder("abcdef");
sb.delete(1, 3); // alguien = admitir
sb.deleteCharAt(5); // excepción
```

Primero, eliminamos los caracteres que comienzan con el índice 1 y terminan justo antes del índice 3. Esto nos da adef. A continuación, le pedimos a Java que elimine el carácter en la posición 5. Sin embargo, el valor restante tiene solo cuatro caracteres, por lo que genera una excepción StringIndexOutOfBoundsException.

El método eliminar() es más flexible que otros cuando se trata de índices de matrices. Si especifica un segundo parámetro que está más allá del final de StringBuilder, Java simplemente asumirá que se refiere al final. Eso significa que este código es legal:

```
var sb = nuevo StringBuilder("abcdef");
sb.delete(1, 100); // alguien = a
```

### Reemplazo de porciones

El método replace() funciona de manera diferente para StringBuilder que para String. La firma del método es la siguiente:

```
reemplazo público de StringBuilder (int startIndex, int endIndex, String newString)
```

El siguiente código muestra cómo utilizar este método:

```
var builder = new StringBuilder("paloma sucia");
constructor.replace(3, 6, "sty");
System.out.println(constructor); // pocilga sucia
```

Primero, Java elimina los caracteres que comienzan con el índice 3 y terminan justo antes del índice 6. Esto nos da suciedad de cerdo. Luego Java inserta el valor "sty" en esa posición.

En este ejemplo, la cantidad de caracteres eliminados e insertados es la misma. Sin embargo, no hay ninguna razón para que así sea. ¿Qué crees que hace esto?

```
var builder = new StringBuilder("paloma sucia");
constructor.replace(3, 100, "");
System.out.println(constructor);
```

Imprime "cerdo". Recuerde, el método consiste primero en realizar una eliminación lógica. El reemplazo() El método permite especificar un segundo parámetro que está más allá del final de StringBuilder. Eso significa que sólo quedan los primeros tres caracteres.

## invertir

Después de todo eso, es hora de utilizar un método sencillo y agradable. El método reverse() hace exactamente lo que parece: invierte los caracteres en las secuencias y devuelve una referencia al StringBuilder actual. La firma del método es la siguiente:

```
reversa pública StringBuilder ()
```

Machine Translated by Google



El siguiente código muestra cómo utilizar este método:

```
var sb = nuevo StringBuilder("ABC");
sb.reverse();
System.out.println(sb);
```

Como era de esperar, esto imprime CBA. Este método no es tan interesante. Quizás a los creadores del examen les guste incluirlo para animarlo a escribir el valor en lugar de depender de la memoria para los índices.

### Trabajando con toString()

La clase Object contiene un método toString() del que muchas clases proporcionan implementaciones personalizadas. La clase StringBuilder es una de ellas.

El siguiente código muestra cómo utilizar este método:

```
var sb = nuevo StringBuilder("ABC");
Cadena s = sb.toString();
```

A menudo, StringBuilder se utiliza internamente con fines de rendimiento, pero el resultado final debe ser un String. Por ejemplo, tal vez sea necesario pasarlo a otro método que espere una cadena.

# Entendiendo la igualdad

En el Capítulo 2, aprendiste cómo usar == para comparar números y que las referencias a objetos se refieren al mismo objeto. En esta sección, veremos lo que significa que dos objetos sean equivalentes o iguales. También analizamos el impacto del grupo String en la igualdad.

## Comparando iguales() y ==

Considere el siguiente código que usa == con objetos:

```
var uno = nuevo StringBuilder();
var dos = nuevo StringBuilder();
var tres = uno.append("a");
System.out.println(uno == dos); // FALSO
System.out.println(uno == tres); // verdadero
```

Dado que este ejemplo no trata con primitivas, sabemos que debemos buscar si las referencias se refieren al mismo objeto. Las variables uno y dos son completamente

Machine Translated by Google



Separe los objetos StringBuilder , dándonos dos objetos. Por lo tanto, la primera declaración impresa nos da falsedad. Las tres variables son más interesantes. Recuerde cómo StringBuilder ¿Métodos les gusta devolver la referencia actual para el encadenamiento? Esto significa que uno y tres apuntan al mismo objeto, y la segunda declaración impresa nos da verdadero.

Viste antes que equals() usa igualdad lógica en lugar de igualdad de objetos para Objetos de cadena :

```
var x = "Hola mundo";
var z =     "Hola mundo".trim();
System.out.println(x.equals(z)); // verdadero
```

Esto funciona porque los autores de la clase String implementaron un método estándar llamado igual() para verificar los valores dentro de la cadena en lugar de la referencia de la cadena en sí.
Si una clase no tiene un método igual () , Java determina si las referencias apuntan al mismo objeto, que es exactamente lo que hace == .

En caso de que se lo pregunte, los autores de StringBuilder no implementaron iguales (). Si llama a equals() en dos instancias de StringBuilder , verificará la igualdad de referencia. Puedes llamar a toString() en StringBuilder para obtener un String para verificar la igualdad.

Finalmente, el examen podría intentar engañarte con una pregunta como esta. ¿Puedes adivinar por qué el ¿El código no se compila?

```
var nombre = "a";
var constructor = new StringBuilder("a");
System.out.println(nombre == constructor); // NO COMPILA
```

Recuerde que == está verificando la igualdad de referencias de objetos. El compilador es lo suficientemente inteligente. saber que dos referencias no pueden apuntar al mismo objeto cuando son de tipos completamente diferentes.

## El grupo de cuerdas

Dado que las cadenas están en todas partes en Java, consumen mucha memoria. En algunas aplicaciones de producción, pueden utilizar una gran cantidad de memoria en todo el programa. Java se da cuenta de que muchas cadenas se repiten en el programa y resuelve este problema reutilizando las comunes. El grupo de cadenas, también conocido como grupo interno, es una ubicación en la máquina virtual Java (JVM) que recopila todas estas cadenas.

El grupo de cadenas contiene valores literales y constantes que aparecen en su programa.
Por ejemplo, "nombre" es un literal y, por lo tanto, va al grupo de cadenas. El miObjeto.
El método toString() devuelve una cadena pero no un literal, por lo que no ingresa al grupo de cadenas.

Veamos ahora el escenario más complejo y confuso, la igualdad de cadenas , que se logra en parte debido a la forma en que la JVM reutiliza los literales de cadenas .

```
var x = "Hola mundo";
var y = "Hola mundo";
System.out.println(x == y);                    // verdadero
```

Machine Translated by Google



Recuerde que una cadena es inmutable y los literales se agrupan. La JVM creó solo uno literal en la memoria. Las variables xey apuntan ambas a la misma ubicación en la memoria; por lo tanto, la declaración resulta verdadera. Se vuelve aún más complicado. Considere este código:

```
var x = "Hola mundo";
var z =     "Hola mundo".trim();
System.out.println(x == z); // FALSO
```

En este ejemplo, no tenemos dos literales de cadena iguales . Aunque x y z se evalúan como la misma cadena, se calcula una en tiempo de ejecución. Como no es lo mismo en tiempo de compilación, se crea un nuevo objeto String . Probemos con otro. ¿Qué crees que se produce aquí?

```
var singleString = "hola mundo";
var concat = "hola";
concat += "mundo";
System.out.println(singleString == concat); // FALSO
```

Esto imprime falso. Llamar a += es como llamar a un método y da como resultado una nueva cadena. Incluso puedes forzar el problema creando una nueva cadena:

```
var x = "Hola mundo";
var y = new String("Hola mundo");
System.out.println(x == y); // FALSO
```

El primero dice que se utilice el grupo de cadenas normalmente. El segundo dice: “No, JVM, realmente no quiero que uses el grupo de cadenas. Por favor, cree un nuevo objeto para mí aunque sea menos eficiente”.

También puedes hacer lo contrario y decirle a Java que use el grupo de cadenas. El método intern() utilizará un objeto en el grupo de cadenas si hay uno presente.

```
pasante de cadena pública ()
```

Si el literal aún no está en el grupo de cadenas, Java lo agregará en este momento.

```
var nombre = "Hola mundo";
var nombre2 = new String("Hola mundo").intern();
System.out.println(nombre == nombre2); // verdadero
```

Primero le decimos a Java que use el grupo de cadenas normalmente para el nombre. Luego, para nombre2, le decimos a Java que cree un nuevo objeto usando el constructor pero que lo interna y use el grupo de cadenas de todos modos. Dado que ambas variables apuntan a la misma referencia en el grupo de cadenas, podemos usar el operador == .

Probemos con otro. ¿Qué crees que esto imprime? Ten cuidado. Es complicado.

```
15: var primero = "rata" + 1;
16: var segundo = "r" + "a" + "t" + "1";
17: var tercero = "r" + "a" + "t" + nueva cadena ("1");
18: System.out.println(primero == segundo);
```

Machine Translated by Google



```
19: System.out.println(primero == segundo.intern());
20: System.out.println(primero == tercero);
21: System.out.println(primero == tercero.intern());
```

En la línea 15, tenemos una constante en tiempo de compilación que se coloca automáticamente en el grupo de cadenas como "rat1". En la línea 16, tenemos una expresión más complicada que también es una constante de tiempo de compilación. Por lo tanto, el primero y el segundo comparten la misma referencia del grupo de cadenas. Esto hace que las líneas 18 y 19 se impriman verdaderas.

En la línea 17, tenemos un constructor String . Esto significa que ya no tenemos una constante de tiempo de compilación y el tercero no apunta a una referencia en el grupo de cadenas. Por lo tanto, la línea 20 imprime falso. En la línea 21, la llamada intern() busca en el grupo de cadenas. Java nota que primero apunta a la misma cadena e imprime verdadero.

Cuando escribes programas, no querrás crear una cadena de una cadena ni utilizar el método intern() . Para el examen es necesario saber qué están permitidos y cómo se comportan.

Recuerde nunca usar intern() o == para comparar objetos String en su código. El único momento en el que deberías tener que lidiar con esto es en el examen.

# Comprender las matrices

Hasta ahora, nos hemos referido a las clases String y StringBuilder como una "secuencia de caracteres". Esto es cierto. Se implementan utilizando una variedad de caracteres. Una matriz es un área de memoria en el montón con espacio para una cantidad designada de elementos. Una cadena se implementa como una matriz con algunos métodos que es posible que desee utilizar al tratar con caracteres específicamente. Un StringBuilder se implementa como una matriz donde el objeto de la matriz se reemplaza con un objeto de matriz nuevo y más grande cuando se queda sin espacio para almacenar todos los caracteres. Una gran diferencia es que una matriz puede ser de cualquier otro tipo de Java. Si no quisiéramos usar un String por algún motivo, podríamos usar una matriz de primitivas char directamente:

```
letras char[];
```

Esto no sería muy conveniente porque perderíamos todas las propiedades especiales String nos da, como escribir "Java". Tenga en cuenta que las letras son una variable de referencia y no una primitiva. El tipo char es primitivo. Pero char es lo que entra en la matriz y no el tipo de matriz en sí. La matriz en sí es de tipo char[]. Puede leer mentalmente los corchetes ([]) como "matriz".

En otras palabras, una matriz es una lista ordenada. Puede contener duplicados. En esta sección, analizamos la creación de una matriz de primitivas y objetos, la clasificación, la búsqueda, los varargs y las matrices multidimensionales.

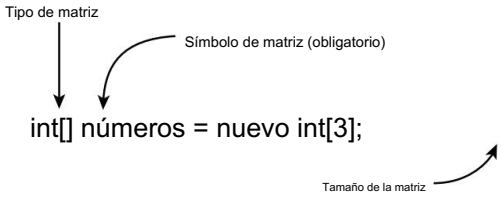
Machine Translated by Google



# Creando una matriz de primitivas

La forma más común de crear una matriz se muestra en la Figura 4.3. Especifica el tipo de matriz (int) y el tamaño (3). Los paréntesis indican que se trata de una matriz.

FIGURA 4.3 La estructura básica de una matriz




Cuando utiliza este formulario para crear una instancia de una matriz, todos los elementos se configuran con el valor predeterminado para ese tipo. Como aprendiste en el Capítulo 1, el valor predeterminado de un int es 0. Dado que los números es una variable de referencia, apunta al objeto de matriz, como se muestra en la Figura 4.4. Como puede ver, el valor predeterminado para todos los elementos es 0. Además, los índices comienzan con 0 y cuentan hacia arriba, tal como lo hicieron para una Cadena.

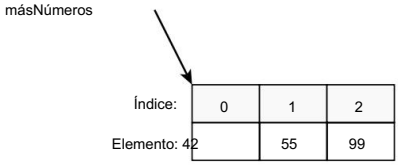
FIGURA 4.4 Una matriz vacía

Otra forma de crear una matriz es especificar todos los elementos con los que debe comenzar:

```
int[] másNúmeros = nuevo int[] {42, 55, 99};
```

En este ejemplo, también creamos una matriz int de tamaño 3. Esta vez, especificamos los valores iniciales de esos tres elementos en lugar de usar los valores predeterminados. La Figura 4.5 muestra cómo se ve esta matriz.

FIGURA 4.5 Una matriz inicializada



Machine Translated by Google



Java reconoce que esta expresión es redundante. Dado que está especificando el tipo de matriz en el lado izquierdo del signo igual, Java ya conoce el tipo. Y como estás especificando los valores iniciales, ya conoce el tamaño. Como atajo, Java te permite escribir esto:

```
int[] másNúmeros = {42, 55, 99};
```

Este enfoque se llama matriz anónima. Es anónimo porque no especifica el tipo ni el tamaño.

Finalmente, puede escribir [] antes o después del nombre, y agregar un espacio es opcional. Esto significa que estas cinco declaraciones hacen exactamente lo mismo:

```
int[] numAnimales;
int [] numAnimales2;
int []numAnimales3;
int numAnimales4[];
int numAnimales5 [];
```

La mayoría de la gente usa el primero. Podrías ver cualquiera de estos en el examen, así que acostúmbrate a ver los corchetes en lugares extraños.

### Múltiples "matrices" en declaraciones

¿Qué tipos de variables de referencia crees que crea el siguiente código?

```
int[] identificadores, tipos;
```

La respuesta correcta son dos variables de tipo int[]. Esto parece bastante lógico. Después de todo, int a, b; creó dos variables int . ¿Qué pasa con este ejemplo?

```
int identificadores [], tipos;
```

Todo lo que hicimos fue mover los corchetes, pero cambió el comportamiento. Esta vez obtenemos una variable de tipo int[] y una variable de tipo int. Java ve esta línea de código y piensa algo como esto: “Quieren dos variables de tipo int. El primero se llama ids[]. Este es un int[] llamado ids. El segundo se llama simplemente tipos. Sin corchetes, por lo que es un número entero regular”.

No hace falta decir que no deberías escribir código con este aspecto. Pero necesitas entender soportarlo para el examen.

## Crear una matriz con variables de referencia

Puede elegir cualquier tipo de Java como tipo de matriz. Esto incluye clases que usted mismo crea. Echemos un vistazo a un tipo integrado con String:

```
String[] bugs = { "grillo", "escarabajo", "mariquita" };
Cadena [] alias = errores;
```

Machine Translated by Google



```
System.out.println(bugs.equals(alias)); // verdadero
System.out.println(bugs.toString());    // [Ljava.lang.String;@160bc7c0
```

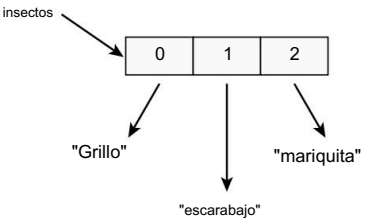
Podemos llamar a equals() porque una matriz es un objeto. Devuelve verdadero debido a la igualdad de referencia. El método equals() en matrices no mira los elementos de la matriz. Recuerde, esto también funcionaría incluso en un int[] . El tipo int es primitivo; int[] es un objeto.

La segunda declaración impresa es aún más interesante. ¿Qué diablos es [Ljava.lang. Cadena;@160bc7c0? No es necesario que sepa esto para el examen, pero [L significa que es una matriz, java.lang.String es el tipo de referencia y 160bc7c0 es el código hash. Obtendrá números y letras diferentes cada vez que lo ejecute, ya que se trata de una referencia.

Java proporciona un método que imprime muy bien una matriz: matrices. toString(bugs) imprimiría [grillo, escarabajo, mariquita].

Asegúrese de comprender la Figura 4.6. El array no asigna espacio para el String objetos. En cambio, asigna espacio para una referencia a dónde están realmente almacenados los objetos.

FIGURA 4.6 Una matriz que apunta a cadenas




Como repaso rápido, ¿a qué crees que apunta este conjunto?

```
Nombres de clase pública {
    Nombres de cadena [];
}
```

Nos tienes. Fue una revisión del Capítulo 1 y no nuestra discusión sobre matrices. La respuesta es nulo. El código nunca creó una instancia de la matriz, por lo que es solo una variable de referencia para nulo. Intentemos eso de nuevo: ¿a qué crees que apunta esta matriz?

```
Nombres de clase pública {
    Nombres de cadena[] = nueva cadena[2];
}
```

Es una matriz porque tiene corchetes. Es una matriz de tipo String ya que ese es el tipo mencionado en la declaración. Tiene dos elementos porque la longitud es 2. Cada una de esas dos ranuras actualmente es nula pero tiene el potencial de apuntar a un objeto String .

Machine Translated by Google



¿Recuerdas el casting del capítulo anterior cuando querías forzar un tipo más grande a uno más pequeño? Puedes hacer eso con matrices también:

```
3: Cadena [] cadenas = { "stringValue" };
4: Objeto [] objetos = cadenas;
5: String[] otra vezStrings = (String[]) objetos;
6: otra vezStrings[0] = nuevo StringBuilder(); // NO COMPILA
7: objetos[0] = nuevo StringBuilder(); // ¡Cuidadoso!
```

La línea 3 crea una matriz de tipo String. La línea 4 no requiere conversión porque Object es un tipo más amplio que String. En la línea 5, se necesita un reparto porque nos estamos moviendo a un tipo más específico. La línea 6 no se compila porque String[] solo permite objetos String y StringBuilder no es un String.

La línea 7 es donde esto se pone interesante. Desde el punto de vista del compilador, esto está bien. Un objeto StringBuilder claramente puede ir en un Objeto []. El problema es que en realidad no tenemos un Objeto []. Tenemos una Cadena [] a la que se hace referencia desde una variable Objeto [] . En tiempo de ejecución, el código genera una ArrayStoreException. No es necesario que memorice el nombre de esta excepción, pero sí debe saber que el código generará una excepción.

## Usando una matriz

Ahora que sabes cómo crear una matriz, intentemos acceder a una:

```
4: String[] mamíferos = {"mono", "chimpancé", "burro"};
5: System.out.println(mamíferos.longitud); // 3
6: System.out.println(mamíferos[0]); // mono
7: System.out.println(mamíferos[1]); // chimpance
8: System.out.println(mamíferos[2]); // burro
```

La línea 4 declara e inicializa la matriz. La línea 5 nos dice cuántos elementos puede contener la matriz. El resto del código imprime la matriz. Observe que los elementos están indexados comenzando con 0. Esto debería resultarle familiar a String y StringBuilder, que también comienzan a contar con 0. Esas clases también cuentan la longitud como el número de elementos. Tenga en cuenta que no hay paréntesis después de la longitud ya que no es un método. ¡Cuidado con errores de compilación como los siguientes en el examen!

```
4: String[] mamíferos = {"mono", "chimpancé", "burro"};
5: System.out.println(mamíferos.longitud()); // NO COMPILA
```

Para asegurarte de que entiendes cómo funciona la longitud , ¿qué crees que implica?

```
var pájaros = nueva cadena[6];
System.out.println(pájaros.longitud);
```

La respuesta es 6. Aunque los seis elementos de la matriz son nulos, todavía quedan seis de ellos. a ellos. El atributo de longitud no considera lo que hay en la matriz; solo considera cuántos espacios se han asignado.

Machine Translated by Google



Es muy común utilizar un bucle al leer o escribir en una matriz. Este bucle establece cada elemento de números en cinco más que el índice actual:

```
5: números var = nuevo int[10];
6: para (int i = 0; i < números.longitud; i++)
7:         números[i] = i + 5;
```

La línea 5 simplemente crea una instancia de una matriz con 10 ranuras. La línea 6 es un bucle for que utiliza un extremo patrón común. Comienza en el índice 0, que es donde también comienza una matriz. Continúa, uno a la vez, hasta que llega al final de la matriz. La línea 7 establece el elemento actual de números.

El examen comprobará si está siendo observador al intentar acceder a elementos que no están en la matriz. ¿Puedes decir por qué cada uno de estos arroja una excepción ArrayIndexOutOfBoundsException ?

¿Para nuestra gama de tamaño 10?

```
números[10] = 3;

números[números.longitud] = 5;

para (int i = 0; i <= números.longitud; i++)
    números[i] = i + 5;
```

El primero es intentar ver si sabes que los índices comienzan con 0. Dado que tenemos 10 elementos en nuestra matriz, esto significa que solo los números [0] hasta los números [9] son válidos. El segundo ejemplo supone que eres lo suficientemente inteligente como para saber que 10 no es válido y lo disfrazas usando el campo de longitud . Sin embargo, la longitud es siempre uno más que el índice máximo válido. Finalmente, el bucle for usa incorrectamente <= en lugar de <, que también es una forma de referirse a ese décimo elemento.

## Clasificación

Java facilita la clasificación de una matriz al proporcionar un método de clasificación, o más bien, un conjunto de métodos de clasificación. Así como StringBuilder te permitió pasar casi cualquier cosa a append(), puedes pasar casi cualquier matriz a Arrays.sort().

Las matrices requieren una importación. Para usarlo, debe tener cualquiera de los dos estados siguientes comentarios en tu clase:

```
importar java.util.*;              // importar el paquete completo, incluidas las matrices
importar java.util.Arrays;         // importar solo matrices
```

Hay una excepción, aunque no aparece con frecuencia en el examen. Puede escribir java.util.Arrays cada vez que se use en la clase en lugar de especificarlo como una importación.

Recuerde que si se le muestra un fragmento de código, puede asumir que las importaciones necesarias están ahí. Este sencillo ejemplo ordena tres números:

```
int[] números = {6, 9, 1};
Arrays.sort(números);
para (int i = 0; i < números.longitud; i++)
    System.out.print(números[i] + " ");
```

Machine Translated by Google



El resultado es 1 6 9, como deberías esperar. Observe que recorrimos la salida para imprimir los valores en la matriz. Simplemente imprimir la variable de matriz directamente daría el molesto hash de [I@2bd9c3e7. Alternativamente, podríamos haber impreso Arrays.toString(numbers) en lugar de usar el bucle. Eso tendría salida [1, 6, 9].

Intente esto nuevamente con tipos de cadena :

```
Cadena[] cadenas = { "10", "9", "100" };
Arrays.sort(cadenas);
para (Cadena s: cadenas)
    System.out.print(s + " ");
```

Esta vez el resultado puede no ser el esperado. Este código genera 10 100 9. El problema es que String ordena en orden alfabético y 1 ordena antes de 9. (Los números ordenan antes que las letras y las mayúsculas antes que las minúsculas). En el Capítulo 9, "Colecciones y genéricos", aprenderá a cree órdenes de clasificación personalizados utilizando algo llamado comparador.

¿Notaste que introdujimos el bucle for mejorado en este ejemplo? Ya que no estamos usando el índice, no necesitamos el bucle for tradicional . Sin embargo, eso no impedirá que los creadores del examen lo utilicen, por lo que nos aseguraremos de utilizar ambos para mantenerte alerta.

# buscando

Java también proporciona una forma cómoda de buscar, pero sólo si la matriz ya está ordenada. La tabla 4.3 cubre las reglas para la búsqueda binaria.

TABLA 4.3 Reglas de búsqueda binaria

| Guión | Resultado |
|---|---|
| Elemento de destino encontrado en una matriz ordenada | índice de partido |
| Elemento de destino no encontrado en la matriz ordenada | Valor negativo que muestra uno más pequeño que el negativo del índice, donde se debe insertar una coincidencia para preservar el orden ordenado |
| matriz sin clasificar | Una sorpresa; este resultado no está definido |

Probemos estas reglas con un ejemplo:

```
3: int[] números = {2,4,6,8};
4: System.out.println(Arrays.binarySearch(números, 2)); // 0
5: System.out.println(Arrays.binarySearch(números, 4)); // 1
6: System.out.println(Arrays.binarySearch(números, 1)); // -1
7: System.out.println(Arrays.binarySearch(números, 3)); // -2
8: System.out.println(Arrays.binarySearch(números, 9)); // -5
```

Machine Translated by Google



Tome nota del hecho de que la línea 3 es una matriz ordenada. Si no fuera así, no podríamos aplicar ninguna de las otras reglas. La línea 4 busca el índice de 2. La respuesta es el índice 0. La línea 5 busca el índice de 4, que es 1.

La línea 6 busca el índice de 1. Aunque 1 no está en la lista, la búsqueda puede determinar que debe insertarse en el elemento 0 para preservar el orden de clasificación. Dado que 0 ya significa algo para los índices de matrices, Java necesita restar 1 para darnos la respuesta de –1. La línea 7 es similar. Aunque 3 no está en la lista, sería necesario insertarlo en el elemento 1 para preservar el orden. Negamos y restamos 1 por coherencia, obteniendo –1 –1, también conocido como –2.

Finalmente, la línea 8 quiere decirnos que se debe insertar 9 en el índice 4. Nuevamente negamos y restamos 1, obteniendo –4 –1, también conocido como –5.

¿Qué crees que sucede en este ejemplo?

```
5: int[] números = nuevo int[] {3,2,1};
6: System.out.println(Arrays.binarySearch(números, 2));
7: System.out.println(Arrays.binarySearch(números, 3));
```

Tenga en cuenta que en la línea 5, la matriz no está ordenada. Esto significa que la salida no se definirá. Al probar este ejemplo, la línea 6 dio correctamente 1 como resultado. Sin embargo, la línea 7 dio una respuesta incorrecta. Los creadores del examen no esperarán que usted sepa qué valores incorrectos aparecen. Tan pronto como vea que la matriz no está ordenada, busque una opción de respuesta sobre resultados impredecibles.

En el examen, necesita saber qué arroja una búsqueda binaria en varios escenarios. Extrañamente, no necesita saber por qué aparece "binario" en el nombre. En caso de que tenga curiosidad, una búsqueda binaria divide la matriz en dos partes iguales (recuerde, 2 es binario) y determina en qué mitad se encuentra el objetivo. Repite este proceso hasta que solo queda un elemento.

## Comparando

Java también proporciona métodos para comparar dos matrices y determinar cuál es "más pequeña". Primero cubrimos el método compare() y luego pasamos a no coincidir(). Estos métodos están sobrecargados para tomar una variedad de parámetros.

### Usando comparar()

Hay un montón de reglas que debes conocer antes de llamar a compare(). Afortunadamente, estas son las mismas reglas que necesitas conocer en el Capítulo 9 al escribir un Comparador.

Primero necesitas aprender qué significa el valor de retorno. No necesitas saber exactamente valores de retorno, pero necesita saber lo siguiente:

- Un número negativo significa que la primera matriz es más pequeña que la segunda.
- Un cero significa que las matrices son iguales.
- Un número positivo significa que la primera matriz es mayor que la segunda.

He aquí un ejemplo:

```
System.out.println(Arrays.compare(nuevo int[] {1}, nuevo int[] {2}));
```

Machine Translated by Google



Este código imprime un número negativo. Debería ser bastante intuitivo que 1 es menor que 2, haciendo la primera matriz más pequeña.

Ahora que sabes cómo comparar un valor único, veamos cómo comparar matrices de diferentes longitudes:

- Si ambas matrices tienen la misma longitud y los mismos valores en cada lugar en el mismo orden, devuelve cero.
- Si todos los elementos son iguales pero la segunda matriz tiene elementos adicionales al final, devuelve un número negativo.
- Si todos los elementos son iguales, pero la primera matriz tiene elementos adicionales al final, devuelve un número positivo.
- Si el primer elemento que difiere es más pequeño en la primera matriz, devuelve un número negativo.
- Si el primer elemento que difiere es mayor en la primera matriz, devuelve un número positivo.

Finalmente, ¿qué significa más pequeño? Aquí hay algunas reglas más que se aplican aquí y para compareTo(), que se ve en el Capítulo 8, “Lambdas e interfaces funcionales”:

- nulo es menor que cualquier otro valor.
- Para los números, se aplica el orden numérico normal.
- Para cadenas, una es más pequeña si es prefijo de otra.
- Para cadenas/caracteres, los números son más pequeños que las letras.
- Para cadenas/caracteres, las mayúsculas son más pequeñas que las minúsculas.

La tabla 4.4 muestra ejemplos de estas reglas en acción.

TABLA 4.4 Ejemplos de Arrays.compare()

| Primera matriz | Segunda matriz | Resultado | Razón |
|---|---|---|---|
| nuevo int[] {1, 2} | nuevo int[] {1} | Número positivo | El primer elemento es el mismo, pero la primera matriz es más larga. |
| nuevo int[] {1, 2} | nuevo int[] {1, 2} | Cero | Coincidencia exacta |
| new String[] {"a"} | new String[] {"aa"} | Número negativo | El primer elemento es una subcadena del segundo. |
| new String[] {"a"} | new String[] {"A"} | Número positivo | Las mayúsculas son más pequeñas que las minúsculas. |
| new String[] {"a"} | new String[] {null} | El número positivo | nulo es menor que una carta. |

Machine Translated by Google



Finalmente, este código no se compila porque los tipos son diferentes. Al comparar dos matrices, deben ser del mismo tipo de matriz.

```
System.out.println(Arrays.compare(
    nuevo int[] {1}, nueva cadena[] {"a"})); // NO COMPILA
```

### Usando falta de coincidencia()

Ahora que está familiarizado con compare(), es hora de aprender sobre la falta de coincidencia(). Si las matrices son iguales, mismatch() devuelve -1. De lo contrario, devuelve el primer índice en el que difieren. ¿Puedes averiguar qué imprimen?

```
System.out.println(Arrays.mismatch(nuevo int[] {1}, nuevo int[] {1}));
System.out.println(Arrays.mismatch(nueva cadena[] {"a"},
    nueva cadena[] {"A"}));
System.out.println(Arrays.mismatch(new int[] {1, 2}, new int[] {1}));
```

En el primer ejemplo, las matrices son las mismas, por lo que el resultado es -1. En el segundo ejemplo, las entradas en el elemento 0 no son iguales, por lo que el resultado es 0. En el tercer ejemplo, las entradas en el elemento 0 son iguales, por lo que seguimos buscando. El elemento en el índice 1 no es igual. O, más específicamente, una matriz tiene un elemento en el índice 1 y la otra no. Por tanto, el resultado es 1.

Para asegurarse de comprender los métodos compare() y mismatch() , estudie la Tabla 4.5. Si no comprende por qué todos los valores están ahí, regrese y estudie esta sección nuevamente.

TABLA 4.5 Igualdad versus comparación versus desajuste

| Método | Cuando las matrices contienen los mismos datos | Cuando las matrices son diferentes |
|---|---|---|
| es igual() | verdadero | FALSO |
| comparar() | 0 | Número positivo o negativo |
| discordancia() | -1 | Índice cero o positivo |

## Usando métodos con Varargs

Cuando usted mismo crea una matriz, se parece a lo que hemos visto hasta ahora. Cuando se pasa uno a su método, hay otra forma en que puede verse. Aquí hay tres ejemplos con un método main() :

```
principal vacío estático público (String [] argumentos)
principal vacío estático público (argumentos de cadena [])
public static void main(String... args) // varargs
```

Machine Translated by Google



El tercer ejemplo utiliza una sintaxis llamada varargs (argumentos variables), que vio en Capítulo 1. Aprende cómo llamar a un método usando varargs en el Capítulo 5, “Métodos”. Por ahora, todo lo que necesitas saber es que puedes usar una variable definida usando varargs como si fuera una matriz normal. Por ejemplo, args.length y args[0] son legales.

# Trabajar con matrices multidimensionales

Las matrices son objetos y, por supuesto, los componentes de las matrices pueden ser objetos. No lleva mucho tiempo, comparando esos dos hechos, preguntarse si los arreglos pueden contener otros arreglos y, por supuesto, pueden hacerlo.

## Creando una matriz multidimensional

Múltiples separadores de matriz es todo lo que se necesita para declarar matrices con múltiples dimensiones. Puede ubicarlos con el tipo o nombre de la variable en la declaración, tal como antes:

```
int[][] vars1; // matriz 2D
int vars2 [][]; // matriz 2D
int[] vars3[]; // matriz 2D
int[] vars4 [], espacio [][]; // una matriz 2D Y 3D
```

Los dos primeros ejemplos no son nada sorprendentes y declaran una matriz bidimensional (2D). El tercer ejemplo también declara una matriz 2D. No hay ninguna buena razón para usar este estilo aparte de confundir a los lectores con su código. El último ejemplo declara dos matrices en la misma línea. Sumando los corchetes, vemos que vars4 es una matriz 2D y el espacio es una matriz 3D.
Nuevamente, no hay otra razón para usar este estilo que no sea confundir a los lectores de su código. Sin embargo, a los creadores del examen les gusta intentar confundirte. ¡Afortunadamente, estás al tanto de ellos y no dejarás que esto te suceda!

Puede especificar el tamaño de su matriz multidimensional en la declaración si lo desea:

```
Cadena [][] rectángulo = nueva Cadena [3] [2];
```

El resultado de esta declaración es una matriz rectangular con tres elementos, cada uno de los cuales se refiere a una matriz de dos elementos. Puede pensar en el rango direccionable como [0][0] a [2][1], pero no lo considere una estructura de direcciones como [0,0] o [2,1].

Ahora supongamos que establecemos uno de estos valores:

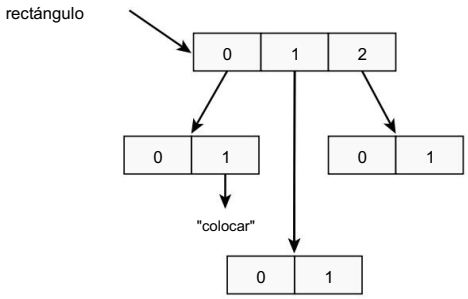
```
rectángulo[0][1] = "conjunto";
```

Puede visualizar el resultado como se muestra en la Figura 4.7. Esta matriz está escasamente poblada porque tiene muchos valores nulos . Puedes ver que el rectángulo todavía apunta a una matriz de tres elementos y que tenemos tres matrices de dos elementos. También puede seguir el rastro desde la referencia hasta el valor que apunta a una Cadena. Comienza en el índice 0 en la matriz superior.
Luego vas al índice 1 en la siguiente matriz.

Machine Translated by Google



FIGURA 4.7 Una matriz multidimensional escasamente poblada

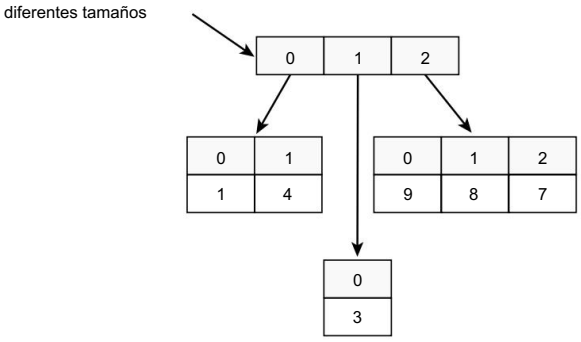



Si bien esa matriz tiene forma rectangular, no es necesario que lo sea. Considere éste:

```
int[][] tamaños diferentes = {{1, 4}, {3}, {9,8,7}};
```

Todavía comenzamos con una serie de tres elementos. Sin embargo, esta vez los elementos del siguiente Los niveles son todos de diferentes tamaños. Uno tiene una longitud de 2, el siguiente tiene una longitud de 1 y el último tiene una longitud de 3. Consulte la Figura 4.8. Esta vez la matriz es de primitivas, por lo que se muestran como si estuvieran en la matriz misma.

FIGURA 4.8 Una matriz multidimensional asimétrica




Otra forma de crear una matriz asimétrica es inicializar solo la primera dimensión de una matriz y definir el tamaño de cada componente de la matriz en una declaración separada:

```
int [][] args = nuevo int[4][];
argumentos[0] = nuevo int[5];
argumentos[1] = nuevo int[3];
```

Esta técnica revela lo que realmente se obtiene con Java: matrices de matrices que, adecuadamente gestionados, ofrecen un efecto multidimensional.

Machine Translated by Google



## Usando una matriz multidimensional

La operación más común en una matriz multidimensional es recorrerla. Este ejemplo imprime una matriz 2D:

```
var dosD = nuevo int[3][2];
for(int i = 0; i < dosD.longitud; i++) {
    for(int j = 0; j < dosD[i].longitud; j++)
        System.out.print(twoD[i][j] + " "); // imprimir elemento
    Sistema.out.println(); // hora de una nueva fila
}
```

Tenemos dos bucles aquí. El primero usa el índice i y pasa por el primer subarreglo para twoD. El segundo usa una variable de bucle diferente, j. Es importante que sean nombres de variables diferentes para que los bucles no se mezclen. El bucle interno analiza cuántos elementos hay en la matriz de segundo nivel. El bucle interior imprime el elemento y deja un espacio para facilitar la lectura. Cuando se completa el bucle interior, el bucle exterior pasa a una nueva línea y repite el proceso para el siguiente elemento.

Todo este ejercicio sería más fácil de leer con el bucle for mejorado .

```
for(int[] interior: dosD) {
    para (int num: interno)
        System.out.print(núm + " ");
    Sistema.out.println();
}
```

Le concederemos que no se trata de menos líneas, sino que cada línea es menos compleja y no hay variables de bucle ni condiciones de terminación que puedan confundirse.

# Calcular con API matemáticas

No debería sorprender que las computadoras sean buenas para calcular números. Java viene con una poderosa clase de Matemáticas con muchos métodos para hacerte la vida más fácil. Aquí solo cubrimos algunos de los más comunes que tienen más probabilidades de aparecer en el examen. Cuando realice sus propios proyectos, consulte el Javadoc matemático para ver qué otros métodos pueden ayudarle.

Preste especial atención a los tipos de devolución en las preguntas de matemáticas. ¡Son una excelente oportunidad para engañar!

## Encontrar el mínimo y el máximo

Los métodos min() y max() comparan dos valores y devuelven uno de ellos.

Las firmas del método para min() son las siguientes:

```
público estático doble min (doble a, doble b)
flotador estático público mínimo (flotador a, flotador b)
```

Machine Translated by Google



```
público estático int min (int a, int b)
público estático largo min (largo a, largo b)
```

Hay cuatro métodos sobrecargados, por lo que siempre tendrás una API disponible con el mismo tipo. Cada método devuelve el que sea menor entre aob . El método max() funciona de la misma manera, excepto que devuelve el valor mayor.

A continuación se muestra cómo utilizar estos métodos:

```
int primero = Math.max(3, 7); int          // 7
segundo = Math.min(7, -9); // -9
```

La primera línea devuelve 7 porque es más grande. La segunda línea devuelve -9 porque es más pequeña. Recuerda desde el colegio que los valores negativos son más pequeños que los positivos.

## Redondear números

El método round() elimina la parte decimal del valor y elige el siguiente número superior si corresponde. Si la parte fraccionaria es .5 o superior, redondeamos hacia arriba.

Las firmas del método para round() son las siguientes:

```
ronda larga estática pública (doble número)
ronda int estática pública (número flotante)
```

Hay dos métodos sobrecargados para garantizar que haya suficiente espacio para almacenar un redondeado doble si es necesario. A continuación se muestra cómo utilizar este método:

```
largo bajo = Math.round(123.45); // 123
largo alto = Math.round(123.50); // 124
int fromFloat = Math.round(123.45f); // 123
```

La primera línea devuelve 123 porque 0,45 es menor que la mitad. La segunda línea devuelve 124 porque la parte fraccionaria apenas es la mitad. La última línea muestra que un flotador explícito desencadena la firma del método que devuelve un int.

## Determinar el techo y el piso

El método ceil() toma un valor doble . Si es un número entero, devuelve el mismo valor. Si tiene algún valor fraccionario, se redondea al siguiente número entero. Por el contrario, el método Floor() descarta cualquier valor después del decimal.

Las firmas del método son las siguientes:

```
techo doble estático público (doble número)
piso doble estático público (doble num)
```

A continuación se muestra cómo utilizar estos métodos:

```
doble c = Math.ceil(3.14); // 4.0
doble f = Math.floor(3.14); // 3.0
```

Machine Translated by Google



La primera línea devuelve 4.0 porque cuatro es el número entero, sólo que mayor. La segunda línea regresa 3.0 porque es el número entero, sólo que más pequeño.

## Calcular exponentes

El método pow() maneja exponentes. Como recordarás de tu clase de matemáticas de la escuela primaria, 32 significa tres al cuadrado. Esto es 3 * 3 o 9. También se permiten exponentes fraccionarios. Dieciséis elevado a 0,5 significa la raíz cuadrada de 16, que es 4. (No te preocupes, no tendrás que calcular raíces cuadradas en el examen).

La firma del método es la siguiente:

```
potencia doble estática pública (número doble, exponente doble)
```

A continuación se muestra cómo utilizar este método:

```
doble cuadrado = Math.pow(5, 2); // 25.0
```

Observe que el resultado es 25,0 en lugar de 25, ya que es un doble. Nuevamente, no te preocupes; el examen no le pedirá que haga ningún cálculo complicado.

## Generando números aleatorios

El método random() devuelve un valor mayor o igual a 0 y menor que 1. La firma del método es la siguiente:

```
público estático doble aleatorio()
```

A continuación se muestra cómo utilizar este método:

```
doble número = Math.random();
```

Como es un número aleatorio, no podemos saber el resultado de antemano. Sin embargo, podemos descartar ciertos números. Por ejemplo, no puede ser negativo porque es menor que 0. No puede ser 1,0 porque no es menor que 1.

Mientras no esté en el examen, es común usar la clase Random para generar números pseudoaleatorios. Permite generar números de diferentes tipos.

# Trabajar con fechas y horas

Java proporciona varias API para trabajar con fechas y horas. También hay un antiguo java.util.Date clase, pero no está en el examen. Necesita una declaración de importación para trabajar con las clases de fecha y hora modernas. Para usarlo, agregue esta importación a su programa:

```
importar java.time.*;          // importar clases de tiempo
```

Machine Translated by Google



**Día versus fecha**

En inglés americano, la palabra fecha se utiliza para representar dos conceptos diferentes. A veces, es la combinación mes/día/año cuando algo sucedió, como el 1 de enero de 2000.
A veces, es el día del mes, como por ejemplo “La fecha de hoy es el día 6”.

Así es; las palabras día y fecha se utilizan a menudo como sinónimos. Esté atento a esto en el examen, especialmente si vive en un lugar donde la gente es más precisa acerca de esta distinción.

En las siguientes secciones, analizamos la creación y manipulación de fechas y horas, incluidas zonas horarias y horario de verano.

## Crear fechas y horas

En el mundo real, solemos hablar de fechas y zonas horarias como si la otra persona estuviera cerca de nosotros. Por ejemplo, si me dices: "Te llamaré el martes por la mañana a las 11:00", asumimos que las 11:00 significan lo mismo para ambos. Pero si yo vivo en Nueva York y tú vives en California, debemos ser más específicos. California es tres horas más temprano que Nueva York porque los estados están en diferentes zonas horarias. En su lugar, diría: "Te llamaré a las 11:00 EST (hora estándar del este) el martes por la mañana".

Al trabajar con fechas y horas, lo primero que debe hacer es decidir cuánta información necesita. El examen le ofrece cuatro opciones:

LocalDate Contiene solo una fecha, sin hora ni zona horaria. Un buen ejemplo de fecha local es tu cumpleaños este año. Es tu cumpleaños durante un día completo, sin importar la hora que sea.

LocalTime Contiene solo una hora: sin fecha ni zona horaria. Un buen ejemplo de LocalTime es la medianoche. Es medianoche a la misma hora todos los días.

LocalDateTime Contiene una fecha y una hora, pero no una zona horaria. Un buen ejemplo de LocalDateTime es "la medianoche de la víspera de Año Nuevo". La medianoche del 2 de enero no es tan especial, lo que hace que la fecha sea relativamente poco importante, y claramente una hora después de la medianoche tampoco es tan especial.

ZonedDateTime Contiene una fecha, hora y zona horaria. Un buen ejemplo de ZonedDateTime es "una conferencia telefónica a las 9:00 a. m. EST". Si vives en California, tendrás que levantarte muy temprano ya que la llamada es a las 6:00 am hora local.

Obtienes instancias de fecha y hora usando un método estático :

```
System.out.println(LocalDate.now());
System.out.println(LocalTime.now());
System.out.println(LocalDateTime.now());
System.out.println(ZonedDateTime.now());
```

Machine Translated by Google



Cada una de las cuatro clases tiene un método estático llamado now(), que proporciona la fecha actual. y tiempo. Su resultado dependerá de la fecha/hora en que lo ejecute y de dónde viva. Los autores viven en los Estados Unidos, por lo que el resultado se verá similar al siguiente cuando se ejecute el 25 de octubre a las 9:13 a. m.:

```
2021–10–25
09:13:07.768
2021–10–25T09:13:07.768
2021–10–25T09:13:07.769–05:00[América/Nueva_York]
```

La clave es el tipo de información en el resultado. La primera línea contiene sólo una fecha y ninguna hora. El segundo contiene sólo una hora y ninguna fecha. La hora muestra horas, minutos, segundos y fracciones de segundo. El tercero contiene tanto una fecha como una hora. La salida utiliza T para separar la fecha y la hora al convertir LocalDateTime en una cadena. Finalmente, el cuarto agrega el desplazamiento de zona horaria y la zona horaria. Nueva York está a cuatro zonas horarias de la hora media de Greenwich (GMT).

La hora media de Greenwich es una zona horaria en Europa que se utiliza como zona horaria cero cuando se habla de compensaciones. Es posible que también hayas oído hablar del Tiempo Universal Coordinado, que es un estándar de zona horaria. Se abrevia como UTC, como resultado de un compromiso entre los nombres en inglés y francés. (Eso no es un error tipográfico. ¡UTC no es en realidad el acrónimo adecuado en ninguno de los idiomas!) UTC usa la misma zona horaria cero que GMT.

Primero, intentemos averiguar qué tan separados están estos momentos en el tiempo. Observe cómo India tiene un desfase de media hora, no una hora completa. Para abordar un problema como este, se resta la zona horaria de la hora. Esto le da el equivalente GMT de la hora:

```
2022–06–20T06:50+05:30[Asia/Calcuta]        // GMT 2022–06–20 01:20
2022–06–20T07:50-05:00[EE.UU./Este]         // GMT 2022–06–20 12:50
```

Recuerda que debes sumar al restar un número negativo. Después de convertir a GMT, puede ver que la hora del este de EE. UU. está 11 horas y media por detrás de la hora de Calcuta.

El desplazamiento de la zona horaria se puede enumerar de diferentes maneras: +02:00, GMT+2 y UTC+2 significan lo mismo. Es posible que veas a cualquiera de ellos en el examen.

Si tiene problemas para recordar esto, intente memorizar un ejemplo en el que la hora Las zonas están separadas por algunas zonas y recuerda la dirección. En Estados Unidos, la mayoría de la gente sabe que la costa este está tres horas por delante de la costa oeste. Y la mayoría de la gente sabe que Asia está por delante de Europa. Simplemente no cruce la zona horaria cero en el ejemplo que elija recordar. El cálculo funciona de la misma manera, pero no es una gran ayuda para la memoria.

Machine Translated by Google



**Espera, no vivo en los Estados Unidos.**

El examen reconoce que los examinados viven en todo el mundo y no le preguntará sobre los detalles de los formatos de fecha y hora de EE. UU. Dicho esto, nuestros ejemplos utilizan formatos de fecha y hora de EE. UU., al igual que las preguntas del examen. Solo recuerda que el mes viene antes de la fecha. Además, Java tiende a utilizar un reloj de 24 horas, aunque en Estados Unidos se utiliza un reloj de 12 horas.
reloj de hora con am/pm

Ahora que sabe cómo crear la fecha y hora actuales, veamos otras fechas y horas específicas. Para empezar, creemos solo una fecha sin hora. Ambos ejemplos crean la misma fecha:

```
var fecha1 = LocalDate.of(2022, Mes.ENERO, 20);
var fecha2 = LocalDate.of(2022, 1, 20);
```

Ambos pasan en el año, mes y fecha. Aunque es bueno utilizar las constantes Mes (para hacer que el código sea más fácil de leer), puede pasar el número int del mes directamente. Simplemente use el número del mes de la misma manera que lo haría si estuviera escribiendo la fecha en la vida real.

Las firmas del método son las siguientes:

```
fecha local estática pública de (int año, int mes, int día del mes)
fecha local estática pública de (int año, mes mes, int día del mes)
```

Hasta ahora os hemos estado diciendo continuamente que Java cuenta empezando por 0. Bueno, los meses son una excepción. Durante meses en los nuevos métodos de fecha y hora, Java cuenta a partir de 1, tal como lo hacemos los humanos.

Al crear un tiempo, puedes elegir qué tan detallado quieres que sea. Puede especificar solo la hora y los minutos, o puede incluir la cantidad de segundos. Incluso puedes incluir nanosegundos si quieres ser muy preciso. (Un nanosegundo es una milmillonésima de segundo, aunque probablemente no sea necesario ser tan específico).

```
var tiempo1 = LocalTime.of(6, 15); var          // hora y minuto
tiempo2 = LocalTime.of(6, 15, 30); var tiempo3  // + segundos
= LocalTime.of(6, 15, 30, 200);                 // + nanosegundos
```

Estos tres tiempos son todos diferentes pero con un minuto de diferencia entre sí. El método signa. turas son las siguientes:

```
hora local estática pública de (hora int, minuto int)
hora local estática pública de (int hora, int minuto, int segundo)
Hora local estática pública de (int hora, int minuto, int segundo, int nanos)
```

Machine Translated by Google



Puede combinar fechas y horas en un solo objeto:

```
var fechaHora1 = LocalDateTime.of(2022, Mes.ENERO, 20, 6, 15, 30);
var fechaHora2 = LocalDateTime.of(fecha1, hora1);
```

La primera línea de código muestra cómo puede especificar toda la información sobre LocalDateTime directamente en la misma línea. La segunda línea de código muestra cómo puede crear objetos LocalDate y LocalTime por separado primero y luego combinarlos para crear un objeto LocalDateTime .

Hay muchas firmas de métodos ya que hay más combinaciones. La siguiente las firmas de métodos utilizan valores enteros:

```
público estático LocalDateTime de (int año, int mes,
        int día del mes, int hora, int minuto)
público estático LocalDateTime de (int año, int mes,
        int díaDeMes, int hora, int minuto, int segundo)
público estático LocalDateTime de (int año, int mes,
        int díaDeMes, int hora, int minuto, int segundo, int nanos)
```

Otros toman una referencia del mes :

```
público estático LocalDateTime de (int año, mes mes,
        int día del mes, int hora, int minuto)
público estático LocalDateTime de (int año, mes mes,
        int díaDeMes, int hora, int minuto, int segundo)
público estático LocalDateTime de (int año, mes mes,
        int díaDeMes, int hora, int minuto, int segundo, int nanos)
```

Finalmente, se toman una fecha local y una hora local existentes:

```
LocalDateTime estático público de (fecha LocalDate, hora LocalTime)
```

Para crear ZonedDateTime, primero necesitamos obtener la zona horaria deseada. Usaremos EE. UU./Este en nuestros ejemplos:

```
var zona = ZoneId.of("EE.UU./Este");
var zonado1 = ZonedDateTime.of(2022, 1, 20,
        6, 15, 30, 200, zona);
var zoned2 = ZonedDateTime.of(fecha1, hora1, zona);
var zoned3 = ZonedDateTime.of(dateTime1, zona);
```

Comenzamos obteniendo el objeto de zona horaria. Luego utilizamos uno de tres enfoques para crear ZonedDateTime. El primero pasa todos los campos individualmente. No recomendamos este enfoque: hay demasiados números y es difícil de leer. Un mejor enfoque es pasar un objeto LocalDate y un objeto LocalTime , o un objeto LocalDateTime .

Machine Translated by Google



Aunque existen otras formas de crear un ZonedDateTime, sólo necesitas saber tres para el examen:

```
pública estática ZonedDateTime de (int año, int mes,
    int día del mes, int hora, int minuto, int segundo,
    int nanos, zona ZoneId)
pública estática ZonedDateTime de (fecha LocalDate, hora LocalTime,
    ZonaId de zona)
pública estática ZonedDateTime de (LocalDateTime dateTime, zona ZoneId)
```

Observe que no hay una opción para pasar la enumeración Mes . Además, no utilizamos un constructor en ninguno de los ejemplos. Las clases de fecha y hora tienen constructores privados junto con métodos estáticos que devuelven instancias. Esto se conoce como patrón de fábrica. Los creadores del examen pueden arrojarle algo como esto:

```
var d = nueva Fecha Local(); // NO COMPILA
```

No caigas en esto. No se le permite construir un objeto de fecha u hora directamente.
Otro truco es lo que sucede cuando pasas números no válidos a of(), por ejemplo:

```
var d = LocalDate.of(2022, Mes.ENERO, 32) // DateTimeException
```

No es necesario saber la excepción exacta que se genera, pero es clara:

```
java.time.DateTimeException: valor no válido para DayOfMonth
        (valores válidos 1-28/31): 32
```

## Manipulación de fechas y horas

Agregar una fecha es fácil. Las clases de fecha y hora son inmutables. Recuerde asignar los resultados de estos métodos a una variable de referencia para que no se pierdan.

```
12: var fecha = LocalDate.of(2022, Mes.ENERO, 20);
13: System.out.println(fecha); // 2022–01–20
14: fecha = fecha.plusDays(2);
15: System.out.println(fecha); 16:          // 2022–01–22
fecha = fecha.plusSemanas(1);
17: System.out.println(fecha); 18:          // 2022–01–29
fecha = fecha.plusMeses(1);
19: System.out.println(fecha); 20:          // 2022–02–28
fecha = fecha.plusYears(5);
21: System.out.println(fecha);              // 2027–02–28
```

Este código es bueno porque hace exactamente lo que parece. Comenzamos con el 20 de enero de 2022. En la línea 14, le agregamos dos días y lo reasignamos a nuestra variable de referencia. En la línea 16, agregamos una semana. Este método nos permite escribir código más claro que plusDays(7). ahora fecha es el 29 de enero de 2022. En la línea 18, agregamos un mes. Esto nos llevaría al 29 de febrero de 2022.

Machine Translated by Google



Sin embargo, 2022 no es un año bisiesto. (2020 y 2024 son años bisiestos). Java es lo suficientemente inteligente como para darse cuenta de que el 29 de febrero de 2022 no existe y, en su lugar, nos da el 28 de febrero de 2022. Finalmente, la línea 20 suma cinco años.

El 29 de febrero existe sólo en un año bisiesto. Los años bisiestos son años que son múltiplos de 4 o 400, pero no otros múltiplos de 100. Por ejemplo, 2000 y 2016 son años bisiestos, pero 2100 no.

También existen métodos sencillos y agradables para retroceder en el tiempo. Esta vez, trabajemos con Fecha y hora local:

```
22: var fecha = LocalDate.of(2024, Mes.ENERO, 20);
23: var tiempo = LocalTime.of(5, 15);
24: var dateTime = LocalDateTime.of(fecha, hora);
25: System.out.println(fechaHora); 26:          // 2024–01–20T05:15
fechaHora = fechaHora.minusDays(1);
27: System.out.println(fechaHora); 28:          // 2024–01–19T05:15
fechaHora = fechaHora.minusHoras(10);
29: System.out.println(fechaHora); // 2024–01–18T19:15
30: fechaHora = fechaHora.minusSeconds(30);
31: System.out.println(fechaHora); // 2024–01–18T19:14:30
```

La línea 25 imprime la fecha original del 20 de enero de 2024, a las 5:15 am. La línea 26 resta un día completo, lo que nos lleva al 19 de enero de 2024, a las 5:15 am. La línea 28 resta 10 horas, lo que muestra que la fecha cambiará si las horas hacen que se ajuste, y nos lleva al 18 de enero de 2024, a las 19:15 (19:15). Finalmente, la línea 30 resta 30 segundos. Puede ver que, de repente, el valor de la pantalla comienza a mostrar segundos. Java es lo suficientemente inteligente como para ocultar los segundos y nanosegundos cuando no los estamos usando.

Es común que los métodos de fecha y hora estén encadenados. Por ejemplo, sin las declaraciones impresas, el ejemplo anterior podría reescribirse de la siguiente manera:

```
var fecha = LocalDate.of(2024, Mes.ENERO, 20);
var tiempo = LocalTime.of(5, 15);
var fechaHora = LocalDateTime.of(fecha, hora)
        .minusDays(1).minusHours(10).minusSeconds(30);
```

Cuando tienes que hacer muchas manipulaciones, este encadenamiento resulta útil. Hay
Hay dos formas en que los creadores del examen pueden intentar engañarte. ¿Qué crees que esto imprime?

```
var fecha = LocalDate.of(2024, Mes.ENERO, 20);
fecha.plusDays(10);
System.out.println(fecha);
```

Imprime el 20 de enero de 2024. Agregar 10 días fue inútil porque el programa ignoró el resultado. Siempre que vea tipos inmutables, preste atención para asegurarse de que el valor de retorno

Machine Translated by Google



de una llamada a un método no se ignora. El examen también puede evaluar si recuerda qué incluye cada uno de los objetos de fecha y hora. ¿Ves lo que está mal aquí?

```
var fecha = LocalDate.of(2024, Mes.ENERO, 20);
fecha = fecha.plusMinutos(1); // NO COMPILA
```

LocalDate no contiene hora. Esto significa que no puede agregarle minutos. Esto puede ser complicado en una secuencia encadenada de operaciones de suma/resta, así que asegúrese de saber qué métodos de la Tabla 4.6 se pueden invocar y en qué tipos.

TABLA 4.6 Métodos en LocalDate, LocalTime, LocalDateTime y ZonedDateTime

| | ¿ Puedo llamar a LocalDate? | ¿ Se puede llamar en LocalTime? | ¿ Se puede llamar a LocalDateTime o ZonedDateTime? |
|---|---|---|---|
| másAños() menosAños() | Sí | No | Sí |
| másMeses() menosMeses() | Sí | No | Sí |
| másSemanas() menos Semanas() | Sí | No | Sí |
| másDías() menosDías() | Sí | No | Sí |
| másHoras() menosHoras() | No | Sí | Sí |
| másMinutos() menosMinutos() | No | Sí | Sí |
| másSegundos() menosSegundos() | No | Sí | Sí |
| másNanos() menosNanos() | No | Sí | Sí |

## Trabajar con períodos

¡Ahora sabes lo suficiente como para hacer algo divertido con las citas! Nuestro zoológico realiza actividades de enriquecimiento animal para brindarles a los animales algo agradable que hacer. El jefe del cuidador del zoológico tiene

Machine Translated by Google



Decidió cambiar los juguetes cada mes. Este sistema continuará durante tres meses para ver cómo funciona.

```
público estático vacío principal (String [] argumentos) {
    var inicio = LocalDate.of(2022, Mes.ENERO, 1);
    var end = LocalDate.of(2022, Mes.MARZO, 30);
    realizarEnriquecimientoAnimal(inicio, fin);
}
vacío estático privado performAnimalEnrichment (inicio de fecha local, final de fecha local) {
    var hasta = inicio;
    while (upTo.isBefore(end)) { // comprueba si aún está antes del final
        System.out.println("dar juguete nuevo: + upTo");
        hasta = hasta.plusMeses(1); // agregar un mes
} }
```

Este código funciona bien. Agrega un mes a la fecha hasta que llega a la fecha de finalización. El problema es que este método no se puede reutilizar. Nuestro cuidador del zoológico quiere probar diferentes horarios para ver cuál funciona mejor.

LocalDate y LocalDateTime tienen un método para convertirse en valores largos , equivalentes a la cantidad de milisegundos que han pasado desde el 1 de enero de 1970, lo que se conoce como época. ¿Qué tiene de especial esta fecha? Eso es lo que Unix empezó a usar para los estándares de fecha, por lo que Java lo reutilizó.

Afortunadamente, Java tiene una clase Period que podemos pasar. Este código hace lo mismo que el ejemplo anterior:

```
público estático vacío principal (String [] argumentos) {
    var inicio = LocalDate.of(2022, Mes.ENERO, 1);
    var end = LocalDate.of(2022, Mes.MARZO, 30);
    var periodo = Periodo.deMeses(1); // crear un punto
    realizarEnriquecimientoAnimal(inicio, fin, punto);
}
vacío estático privado performAnimalEnrichment (inicio de LocalDate, final de LocalDate,
    Período período ) {                // usa el punto genérico

    var hasta = inicio;
    mientras (hasta.isBefore(fin)) {
        System.out.println("dar juguete nuevo: + upTo");
        hasta = hasta.plus(punto); // agrega el punto
} }
```

Machine Translated by Google



El método puede agregar un período de tiempo arbitrario que se pasa. Esto nos permite reutilizar el mismo método durante diferentes períodos de tiempo a medida que nuestro cuidador del zoológico cambia de opinión.

Hay cinco formas de crear una clase Period :

```
var anualmente = Periodo.deAños(1); // cada 1 año
var trimestral = Periodo.deMeses(3); // cada 3 meses
var cadaTresSemanas = Periodo.deSemanas(3); // cada 3 semanas
var cadaOtroDía = Period.ofDays(2); // cada 2 días
var cadaAñoAndASemana = Periodo.de(1, 0, 7); // cada año y 7 días
```

Hay un inconveniente. No puede encadenar métodos al crear un período. El siguiente código parece equivalente al ejemplo de eachYearAndAWeek , pero no lo es. Solo se utiliza el último método porque los métodos Period.of son métodos estáticos .

```
var incorrecto = Periodo.deAños(1).deSemanas(1); // cada semana
```

Este complicado código es realmente como escribir lo siguiente:

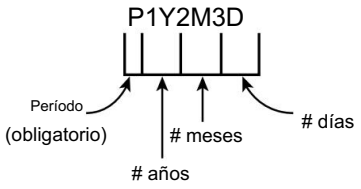
```
var incorrecto = Periodo.deAños(1);
incorrecto = Periodo.de Semanas(1);
```

¡Esto claramente no es lo que pretendías! Es por eso que el método of() te permite pasar el número de años, meses y días. Todos ellos están incluidos en el mismo período. Recibirá una advertencia del compilador sobre esto. Las advertencias del compilador le indican que algo anda mal o es sospechoso sin fallar la compilación.

El método of() sólo tarda años, meses y días. La posibilidad de utilizar otro método de fábrica para pasar semanas es simplemente una conveniencia. Como se puede imaginar, el período real se almacena en términos de años, meses y días. Cuando imprime el valor, Java muestra cualquier parte distinta de cero utilizando el formato que se muestra en la Figura 4.9.

FIGURA 4.9 Formato de período

```
System.out.println(Periodo.de(1,2,3));
```




Como puede ver, la P siempre comienza la Cadena para mostrar que es una medida de período. Luego viene el número de años, el número de meses y el número de días. Si alguno de ellos es cero, se omite.

¿Puedes averiguar qué produce esto?

```
System.out.println(Periodo.deMeses(3));
```

Machine Translated by Google



La salida es P3M. Recuerde que Java omite cualquier medida que sea cero. Lo último Lo que hay que saber sobre Period es con qué objetos se puede utilizar. Veamos un código:

```
3: fecha var = LocalDate.of(2022, 1, 20);
4: var tiempo = LocalTime.of(6, 15);
5: var fechaHora = LocalDateTime.of(fecha, hora);
6: período var = período.de meses (1);
7: System.out.println(fecha.más(período)); 8:                    // 2022–02–20
System.out.println(dateTime.plus(período)); // 2022–02–20T06:15
9: System.out.println(tiempo.más(período)); // Excepción
```

Las líneas 7 y 8 funcionan como se esperaba. Suman un mes al 20 de enero de 2022, dándonos febrero 20 de enero de 2022. El primero tiene solo la fecha y el segundo tiene tanto la fecha como la hora.

La línea 9 intenta agregar un mes a un objeto que solo tiene una hora. Esto no funcionará. Java genera una excepción UnsupportedTemporalTypeException y se queja de que intentamos utilizar una unidad no compatible: meses.

Como puedes ver, debes prestar atención al tipo de objetos de fecha y hora en cada lugar donde los veas.

## Trabajar con duraciones

Probablemente ya hayas notado que un Período es un día o más de tiempo. También está la Duración, que está pensada para unidades de tiempo más pequeñas. Para Duración, puede especificar la cantidad de días, horas, minutos, segundos o nanosegundos. Y sí, podrías pasar 365 días para formar un año, pero realmente no deberías hacerlo: para eso está el Periodo .

Convenientemente, Duración funciona más o menos de la misma manera que Período, excepto que se usa con objetos que tienen tiempo. La duración se genera comenzando con PT, que puede considerarse como un período de tiempo. Una duración se almacena en horas, minutos y segundos. El número de segundos incluye fracciones de segundo.

Podemos crear una Duración usando varias granularidades diferentes:

```
var diario = Duración.deDías(1); var                    // PT24H
por hora = Duración.deHoras(1); var                     // PT1H
cada minuto = Duración.de minutos(1); var               // PT1M
cadaDiezSecondos = Duración.deSeconds(10); // PT10S
var eachMilli = Duración.ofMillis(1); // PT0.001S
var cadaNano = Duración.ofNanos(1); // PT0.000000001S
```

La duración no tiene un método de fábrica que requiera varias unidades como lo hace Period . Si Si quieres que algo suceda cada hora y media, especificas 90 minutos.

La duración incluye otro método de fábrica más genérico. Se necesita un número y una Unidad Temporal. La idea es, digamos, algo así como "5 segundos". Sin embargo, TemporalUnit es una interfaz. Por el momento, sólo existe una implementación denominada ChronoUnit.

Machine Translated by Google



El ejemplo anterior podría reescribirse así:

```
var diario = Duración.de(1, ChronoUnit.DAYS);
var por hora = Duración.de(1, ChronoUnit.HORAS);
var cada minuto = Duración.de(1, ChronoUnit.MINUTES);
var cadaDiezSecondos = Duración.de(10, ChronoUnit.SECONDS);
var eachMilli = Duración.de(1, ChronoUnit.MILLIS);
var cadaNano = Duración.de(1, ChronoUnit.NANOS);
```

ChronoUnit también incluye algunas unidades convenientes como ChronoUnit.HALF_DAYS para representan 12 horas.

**ChronoUnit para las diferencias**

ChronoUnit es una excelente manera de determinar qué tan separados están dos valores temporales . Temporal incluye LocalDate, LocalTime, etc. ChronoUnit está en java. paquete time.temporal .

```
var uno = LocalTime.of(5, 15);
var dos = LocalTime.of(6, 30);
var fecha = LocalDate.of(2016, 1, 20);
System.out.println(ChronoUnit.HOURS.between(uno, dos));          // 1
System.out.println(ChronoUnit.MINUTES.between(one, two)); // 75
System.out.println(ChronoUnit.MINUTES.between(one, date)); // FechaHoraException
```

La primera declaración impresa muestra que entre truncamientos en lugar de rondas. El segundo muestra lo fácil que es contar en diferentes unidades. Simplemente cambie el tipo de ChronoUnit . Lo último nos recuerda que Java generará una excepción si mezclamos lo que se puede hacer con los objetos de fecha y hora.

Alternativamente, puedes truncar cualquier objeto con un elemento de tiempo. Por ejemplo:

```
Hora local hora = Hora local.of(3,12,45);
System.out.println(hora); // 03:12:45
LocalTime truncado = time.truncatedTo(ChronoUnit.MINUTES);
System.out.println(truncado); // 03:12
```

Este ejemplo pone a cero cualquier campo menor que minutos. En nuestro caso, se deshace de los segundos.

Usar una duración funciona de la misma manera que usar un período. Por ejemplo:

```
7: fecha var = LocalDate.of(2022, 1, 20);
8: var tiempo = LocalTime.of(6, 15);
```

Machine Translated by Google



```
9: var dateTime = LocalDateTime.of(fecha, hora);
10: duración var = Duración.deHoras(6);
11: System.out.println(dateTime.plus(duración)); // 2022–01–20T12:15
12: System.out.println(tiempo.más(duración)); // 12:15
13: Sistema.out.println(
14:         fecha.plus(duración)); // Excepción de tipo temporal no admitida
```

La línea 11 muestra que podemos agregar horas a LocalDateTime, ya que contiene una hora. Línea 12 también funciona, ya que lo único que tenemos es un tiempo. La línea 13 falla porque no podemos agregar horas a un objeto que no contiene una hora.

Intentémoslo de nuevo, pero esta vez agreguemos 23 horas.

```
7: fecha var = LocalDate.of(2022, 1, 20);
8: var tiempo = LocalTime.of(6, 15);
9: var dateTime = LocalDateTime.of(fecha, hora);
10: duración var = Duración.deHoras(23);
11: System.out.println(dateTime.plus(duración)); // 2022–01–21T05:15
12: System.out.println(tiempo.más(duración)); // 05:15
13: Sistema.out.println(
14:         fecha.plus(duración)); // Excepción de tipo temporal no admitida
```

Esta vez vemos que Java avanza más allá del final del día. La línea 11 pasa al día siguiente ya que pasamos la medianoche. La línea 12 no tiene día, por lo que el tiempo simplemente avanza, como en un reloj real.

## Período versus duración

Recuerde que Periodo y Duración no son equivalentes. Este ejemplo muestra un Período y una Duración de la misma duración:

```
var fecha = LocalDate.of(2022, 5, 25);
var periodo = Periodo.deDías(1);
var días = Duración.deDías(1);

System.out.println(fecha.plus(período)); // 2022–05–26
System.out.println(fecha.plus(días));                    // Unidad no compatible: Segundos
```

Dado que estamos trabajando con LocalDate, debemos usar Period. Duración tiene unidades de tiempo, incluso si no las vemos, y están destinadas sólo a objetos con tiempo. Asegúrese de poder completar la Tabla 4.7 para identificar qué objetos pueden usar Periodo y Duración.

Machine Translated by Google



TABLA 4.7 Dónde utilizar Duración y período

| | ¿ Se puede usar con Periodo? | ¿ Se puede usar con Duración? |
|---|---|---|
| Fecha local | Sí | No |
| Fecha y hora local | Sí | Sí |
| Hora local | No | Sí |
| ZonaDateTime | Sí | Sí |

# Trabajar con instantes

La clase Instantánea representa un momento específico en la zona horaria GMT. Supongamos que desea ejecutar un temporizador:

```
var ahora = Instantáneo.now();
// hacer algo que requiera mucho tiempo
var let = Instantáneo.ahora();

var duración = Duración.entre(ahora, más tarde);
System.out.println(duración.toMillis()); // Devuelve el número de milisegundos
```

En nuestro caso, "algo que llevó mucho tiempo" fue poco más de un segundo y el programa imprimió 1025.

Si tienes un ZonedDateTime, puedes convertirlo en un Instant:

```
var fecha = LocalDate.of(2022, 5, 25);
var tiempo = LocalTime.of(11, 55, 00);
var zona = ZoneId.of("EE.UU./Este");
var zonedDateTime = ZonedDateTime.of(fecha, hora, zona);
var instantáneo = zonedDateTime.toInstant(); // 2022–05–25T15:55:00Z
System.out.println(zonedDateTime); // 2022–05–25T11:55–04:00[EE.UU./Este]
System.out.println(instantáneo); // 202–05–25T15:55:00Z
```

Las dos últimas líneas representan el mismo momento en el tiempo. ZonedDateTime incluye _ una zona horaria. El Instante elimina la zona horaria y la convierte en un Instante de tiempo en GMT.

No se puede convertir un LocalDateTime en un Instant. Recuerda que un Instante es un punto en el tiempo. Un LocalDateTime no contiene una zona horaria y, por lo tanto, no se reconoce universalmente en todo el mundo como el mismo momento en el tiempo.

Machine Translated by Google



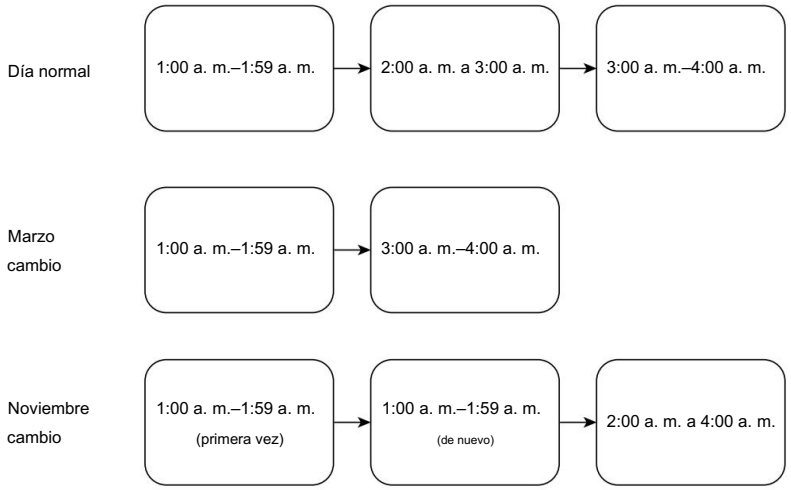
# Contabilización del horario de verano

Algunos países observan el horario de verano. Aquí es donde los relojes se ajustan una hora dos veces al año para aprovechar mejor la luz solar. No todos los países participan, y los que sí lo hacen utilizan fines de semana diferentes para el cambio. Sólo tienes que trabajar con el horario de verano de EE. UU. en el examen, y eso es lo que describimos aquí.

La pregunta le permitirá saber si una fecha/hora mencionada cae en un fin de semana en el que está previsto cambiar los relojes. Si no se menciona en alguna pregunta, puedes asumir que es un fin de semana normal. El acto de adelantar o retroceder el reloj se produce a las 2:00 am, que cae muy temprano en la mañana del domingo.

La figura 4.10 muestra lo que sucede con los relojes. Cuando cambiamos nuestros relojes en marzo, el tiempo avanza de 1:59 am a 3:00 am. Cuando cambiamos nuestros relojes en noviembre, el tiempo retrocede y experimentamos la hora de 1:00 am a 1:59 am dos veces. Los niños aprenden esto como “Avanzar en primavera y retroceder en otoño”.

FIGURA 4.10 Cómo funciona el horario de verano




Por ejemplo, el 13 de marzo de 2022, adelantamos nuestros relojes una hora y saltamos de las 2:00 am a las 3:00 am. Esto significa que ese día no son las 2:30 am. Si quisiéramos saber la hora una hora después de la 1:30, serían las 3:30.

```
var fecha = LocalDate.of(2022, Mes.MARZO, 13);
var tiempo = LocalTime.of(1, 30);
var zona = ZoneId.of("EE.UU./Este");
var dateTime = ZonedDateTime.of(fecha, hora, zona);
System.out.println(fechaHora); // 2022–03-13T01:30-05:00[EE.UU./Este]
System.out.println(dateTime.getHour()); // 1
System.out.println(dateTime.getOffset()); // -05:00
```

Machine Translated by Google



```
fechaHora = fechaHora.plusHours(1);
System.out.println(fechaHora); // 2022–03-13T03:30-04:00[EE.UU./Este]
System.out.println(dateTime.getHour()); // 3
System.out.println(dateTime.getOffset()); // -04:00
```

Observe que dos cosas cambian en este ejemplo. El tiempo salta de 1:30 a 3:30. La compensación UTC también cambia. ¿Recuerdas cuando calculamos la hora GMT restando la zona horaria de la hora? Puedes ver que pasamos de las 6:30 GMT (1:30 menos –5:00) a las 7:30 GMT (3:30 menos –4:00). Esto muestra que la hora realmente cambió una hora desde el punto de vista GMT. Imprimimos los campos de hora y compensación por separado para darle énfasis.

Del mismo modo, en noviembre, una hora después de la 1:30 am inicial también es la 1:30 am porque a las 2:00 am repetimos la hora. Esta vez, intente calcular usted mismo la hora GMT para las tres horas para confirmar que realmente nos movemos solo una hora a la vez.

```
var fecha = LocalDate.of(2022, Mes.NOVIEMBRE, 6);
var tiempo = LocalTime.of(1, 30);
var zona = ZoneId.of("EE.UU./Este");
var dateTime = ZonedDateTime.of(fecha, hora, zona);
System.out.println(fechaHora); // 2022-11-06T01:30-04:00[EE.UU./Este]

fechaHora = fechaHora.plusHours(1);
System.out.println(fechaHora); // 2022-11-06T01:30-05:00[EE.UU./Este]

fechaHora = fechaHora.plusHours(1);
System.out.println(fechaHora); // 2022-11-06T02:30-05:00[EE.UU./Este]
```

¿Lo obtuviste? Pasamos de las 5:30 GMT a las 6:30 GMT, a las 7:30 GMT.

Finalmente, intentar crear un tiempo que no existe simplemente avanza:

```
var fecha = LocalDate.of(2022, Mes.MARZO, 13);
var tiempo = LocalTime.of(2, 30);
var zona = ZoneId.of("EE.UU./Este");
var dateTime = ZonedDateTime.of(fecha, hora, zona);
System.out.println(fechaHora);                // 2022–03–13T03:30–04:00[EE.UU./Este]
```

Java es lo suficientemente inteligente como para saber que no son las 2:30 am esa noche y cambia a el desplazamiento GMT apropiado.

Sí, es molesto que Oracle espere que usted sepa esto incluso si no se encuentra en los Estados Unidos o, de hecho, en una parte de los Estados Unidos que no sigue el horario de verano. Los creadores del examen están en los Estados Unidos y decidieron que todos necesitan saber cómo funcionan las zonas horarias estadounidenses.

Machine Translated by Google



# Resumen

En este capítulo, aprendiste que una Cadena es una secuencia inmutable de caracteres. Llamar explícitamente al constructor es opcional. El operador de concatenación (+) crea una nueva Cadena con el contenido de la primera Cadena seguido del contenido de la segunda Cadena. Si cualquiera de los operandos involucrados en la expresión + es una Cadena, se usa la concatenación; de lo contrario, se utiliza la suma.
Los literales de cadena se almacenan en el grupo de cadenas. La clase String tiene muchos métodos.

Por el contrario, un StringBuilder es una secuencia mutable de caracteres. La mayoría de los métodos devuelven una referencia al objeto actual para permitir el encadenamiento de métodos. La clase StringBuilder tiene muchos métodos.

Llamar == en objetos String comprobará si apuntan al mismo objeto en el grupo.
Llamar a == en las referencias de StringBuilder comprobará si apuntan al mismo objeto StringBuilder .
Llamar a equals() en objetos String comprobará si la secuencia de caracteres es la misma. Llamar a equals() en objetos StringBuilder verificará si apuntan al mismo objeto en lugar de mirar los valores internos.

Una matriz es un área de memoria de tamaño fijo en el montón que tiene espacio para primitivas o punteros a objetos. El tamaño lo especificas al crearlo. Por ejemplo, int[] a = nuevo int[6];. Los índices comienzan con 0 y se hace referencia a los elementos mediante [0]. Las matrices.
El método sort() ordena una matriz. Arrays.binarySearch() busca una matriz ordenada y devuelve el índice de una coincidencia. Si no se encuentra ninguna coincidencia, niega la posición donde debería insertarse el elemento y resta 1. Arrays.compare() y Arrays.mismatch()
compruebe si dos matrices son equivalentes. Los métodos que se pasan varargs (...) se pueden usar como si se pasara una matriz normal. En una matriz multidimensional, las matrices de segundo nivel y más allá pueden tener diferentes tamaños.

La clase Math proporciona una serie de métodos estáticos para realizar óperas matemáticas. ciones. Por ejemplo, puede obtener mínimos o máximos. Puedes redondear o incluso generar números aleatorios. Algunos métodos funcionan con cualquier primitiva numérica y otros solo funcionan con el doble.

LocalDate contiene solo una fecha, LocalTime contiene solo una hora y LocalDateTime contiene tanto una fecha como una hora. Los tres tienen constructores privados y se crean usando LocalDate.now() o LocalDate.of() (o los equivalentes para esa clase). Las fechas y horas se pueden manipular utilizando los métodos plusXXX o minusXXX . La clase Period representa una cantidad de días, meses o años para agregar o restar de LocalDate o LocalDateTime. Las clases de fecha y hora son todas inmutables, lo que significa que se debe utilizar el valor de retorno.

Machine Translated by Google



## Conceptos básicos del examen

Ser capaz de determinar la salida del código usando String. Conozca las reglas para concatenar con String y cómo utilizar métodos String comunes . Sepa que una cadena es inmutable. Preste especial atención al hecho de que los índices están basados en cero y que la subcadena() El método sube la cadena hasta justo antes del índice del segundo parámetro.

Ser capaz de determinar la salida del código utilizando StringBuilder. Sepa que un StringBuilder es mutable y cómo utilizar métodos comunes de StringBuilder . Conoce esa subcadena() no cambia el valor de StringBuilder, mientras que append(), delete() e insert() sí lo cambian. Tenga en cuenta también que la mayoría de los métodos StringBuilder devuelven una referencia a la instancia actual de StringBuilder.

Comprenda la diferencia entre == y es igual (). == comprueba la igualdad de objetos. es igual() Depende de la implementación del objeto al que se llama. Para la clase String , equals() verifica los caracteres dentro de ella.

Ser capaz de determinar la salida del código utilizando matrices. Saber declarar y crear instancias de matrices unidimensionales y multidimensionales. Poder acceder a cada elemento y saber cuando un índice está fuera de límites. Reconocer resultados correctos e incorrectos al buscar y clasificar.

Identificar los tipos de retorno de los métodos matemáticos . Dependiendo de la primitiva pasada, Math Los métodos pueden devolver resultados primitivos diferentes.

Reconocer usos no válidos de fechas y horas. LocalDate no contiene campos de hora y LocalTime no contiene campos de fecha. Esté atento a las operaciones que se realizan en el momento equivocado. También esté atento a sumar o restar tiempo e ignorar el resultado. Siéntase cómodo con las matemáticas de fechas, incluidas las zonas horarias y el horario de verano.

Machine Translated by Google



# Preguntas de revisión

Las respuestas a las preguntas de revisión del capítulo se pueden encontrar en el Apéndice.

1. ¿Qué genera el siguiente código? (Elija todas las que correspondan).

```
1: pez de clase pública {
2:     público estático vacío principal (String [] argumentos) {
3:         int numPez = 4;
4:         String fishType = "atún";
5:         Cadena otroPez = numFish + 1;
6:         System.out.println(otroPez + "" + tipo de pez);
7:         System.out.println(numFish + "" + 1);
8: } }
```

R.4 1

B.5 _

c.5 atún

D.5atún _

E.51atún _

F. El código no se compila.

2. ¿Cuáles de estas declaraciones de matrices no son legales? (Elija todas las que correspondan).

A. int[][] puntuaciones = new int[5][];

B. Objeto[][][] cubículos = nuevo Objeto[3][0][5];

C. Judías verdes[] = judías nuevas[6];

D. java.util.Date[] fechas[] = nuevo java.util.Date[2][];

E. int[][] tipos = nuevo int[];

F. int[][] java = nuevo int[][];

3. Tenga en cuenta que el 13 de marzo de 2022 es el fin de semana en el que avanzamos y el 6 de noviembre de 2022 es cuando retrocedemos para el horario de verano. ¿Cuál de los siguientes puede llenar el espacio en blanco sin que el código arroje una excepción? (Elija todas las que correspondan).

```
var zona = ZoneId.of("EE.UU./Este");
fecha var = ____________________________ ;
var tiempo = LocalTime.of(2, 15);
var z = ZonedDateTime.of(fecha, hora, zona);
```

A. Fecha local.de (2022, 3, 13)

B. Fecha local de (2022, 3, 40)

C. Fecha local.de (2022, 11, 6)

Machine Translated by Google



D. Fecha local.de (2022, 11, 7)

E. Fecha local.de (2023, 2, 29)

F. LocalDate.of(2022, MonthEnum.MARZO, 13);

4. ¿Cuáles de los siguientes resultados genera este código? (Elija todas las que correspondan).

3: var s = "Hola"; 4:
var t = nueva(s) cadena(s); 5:
if ("Hola".equals(s)) System.out.println("uno"); 6: if (t == s)
System.out.println("dos"); 7: if (t.intern() == s)
System.out.println("tres"); 8: if ("Hola" == s)
System.out.println("cuatro"); 9: if ("Hola".intern() == t)
System.out.println("cinco");

uno _

B. dos

C. tres

D. cuatro

mi cinco

F. El código no se compila.

G. Ninguno de los anteriores

5. ¿Cuál es el resultado del siguiente código?

7: var sb = nuevo StringBuilder(); 8:
sb.append("aaa").insert(1, "bb").insert(4, "ccc"); 9:
System.out.println(sb);

A. abbaaccc

B. abbaccca

C. bbaaaccc

D. bbaaccca

E. Una línea vacía

F. El código no se compila.

6. ¿Cuántas de estas líneas contienen un error de compilación? (Elija todas las que correspondan). 23: doble uno
= Math.pow(1, 2); 24: int dos =
Math.round(1.0); 25: flotador tres =
Math.random(); 26: var dobles = nuevo doble[] {uno, dos, tres};

R. 0

B.1 _

Machine Translated by Google



c.2 _

D.3 _

E.4 _

7. ¿Cuál de estas afirmaciones es cierta para los dos valores? (Elija todas las que correspondan).

2022–08–28T05:00 GMT-04:00
2022–08–28T09:00 GMT-06:00

R. La primera fecha/hora es anterior.

B. La segunda fecha/hora es anterior.

C. Ambas fechas y horas son iguales.

D. Las fechas y horas tienen dos horas de diferencia.

E. Las fechas y horas tienen una diferencia de seis horas.

F. Las fechas y horas tienen una diferencia de 10 horas.

8. ¿Cuál de los siguientes devuelve 5 cuando se ejecuta de forma independiente? (Elija todas las que correspondan).

cadena var = "12345"; var
constructor = nuevo StringBuilder("12345");

A. constructor.charAt(4)

B. builder.replace(2, 4, "6").charAt(3)

C. builder.replace(2, 5, "6").charAt(2)

D. cadena.charAt(5)

E. string.length F.

string.replace("123", "1").charAt(2)

G. Ninguno de los anteriores

9. ¿Cuáles de las siguientes afirmaciones son verdaderas sobre las matrices? (Elija todas las que correspondan).

R. El primer elemento es el índice 0.

B. El primer elemento es el índice 1.

C. Las matrices tienen un tamaño fijo.

D. Las matrices son inmutables.

E. Llamar a equals() en dos matrices diferentes que contienen siempre los mismos valores primitivos devuelve verdadero.

F. Llamar a equals() en dos matrices diferentes que contienen siempre los mismos valores primitivos devuelve falso.

G. Llamar a equals() en dos matrices diferentes que contienen los mismos valores primitivos puede devolver verdadero o falso.

Machine Translated by Google



10. ¿Cuántas de estas líneas contienen un error de compilación? (Elija todas las que correspondan).

```
23: int uno = Math.min(5, 3); 24: dos largos
= Math.round(5.5); 25: doble tres =
Math.floor(6.6); 26: var dobles = nuevo doble[] {uno,
dos, tres};
```

R. 0

B.1 _

c.2 _

D.3 _

E.4 _

11. ¿Cuál es el resultado del siguiente código? var fecha

```
= LocalDate.of(2022, 4, 3); fecha.plusDays(2);
fecha.plusHoras(3);
System.out.println(fecha.getAño()
+ " " + fecha.getMonth()
+ " " + fecha.getDayOfMonth());
```

A. 4 DE MARZO DE 2022

B. 2022 6 DE MARZO

C. 3 DE ABRIL DE 2022

D. 5 DE ABRIL DE 2022

E. El código no se compila.

F. Se lanza una excepción de tiempo de ejecución.

12. ¿Qué genera el siguiente código? (Elija todas las que correspondan). var

```
números = "012345678".indent(1); números =
números.stripLeading();
System.out.println(numbers.substring(1, 3));
System.out.println(numbers.substring(7, 7));
System.out.print(números.substring(7));
```

R.12 _

B.123 _

C.7 _

D.78 _

E. Una línea en blanco

F. El código no se compila.

G. Se lanza una excepción.

Machine Translated by Google



13. ¿Cuál es el resultado del siguiente código?

```
león de clase pública {
    rugido público vacío (String rugido1, StringBuilder rugido2)
        { rugido1.concat("!!!");
        rugido2.append("!!!");

    } público estático vacío principal (String[] args) {
        var rugido1 = "rugido";
        var rugido2 = new StringBuilder("rugido"); nuevo
        León().roar(rugido1, rugido2);
        System.out.println(rugido1 +        " "        + rugido2);
} }
```

A. rugido rugido

B. rugido rugido!!!

C. rugido!!! rugido

D. rugido!!! ¡¡¡rugido!!!

E. Se lanza una excepción.

F. El código no se compila.

14. Dado lo siguiente, ¿quién puede completar correctamente el espacio en blanco? (Elija todas las que correspondan).

```
var fecha = LocalDate.now(); var
tiempo = LocalTime.now(); var
fechaHora = LocalDateTime.now(); var zonaId
= ZoneId.systemDefault(); var ZonedDateTime
= ZonedDateTime.of(dateTime, zonaId);
instante instante =        ____________________________________ ;
```

A. Instantáneo.ahora()

B. nuevo instantáneo()

C. fecha.toInstant()

D. dateTime.toInstant()

E. time.toInstant()

F.zonedDateTime.toInstant ()

15. ¿Cuál es el resultado de lo siguiente? (Elija todas las que correspondan).

```
var arr = new String[] { "CERDO", "cerdo", "123"};
Arrays.sort(arr);
System.out.println(Arrays.toString(arr));
System.out.println(Arrays.binarySearch(arr, "Pippa"));
```

Machine Translated by Google



A. [cerdo, CERDO, 123]

B. [CERDO, cerdo, 123]

C. [123, CERDO, cerdo]

D. [123, cerdo, CERDO]

mi -3

-2 _

G. Los resultados de binarioSearch() no están definidos en este ejemplo.

16. ¿Qué se incluye en el resultado del siguiente código? (Elija todas las que correspondan).

```
var base = "oveja\noveja\\t"; int
longitud = base.longitud(); int sangría
= base.indent(2).length(); int traducir =
base.translateEscapes().length();

var formateado = "%s %s %s".formatted(longitud, sangría, traducción);
System.out.format(formateado);
```

R.10 _

B.11 _

C.12 _

D.13 _

E.14 _

F.15 _

G.16 _

17. ¿Cuáles de estas afirmaciones son verdaderas? (Elija todas las que correspondan). var letras = new StringBuilder("abcdefg"); A.

cards.substring(1, 2) devuelve una cadena de un solo carácter.

B. cards.substring(2, 2) devuelve una cadena de un solo carácter.

C. cards.substring(6, 5) devuelve una cadena de un solo carácter.

D. cards.substring(6, 6) devuelve una cadena de un solo carácter.

E. cards.substring(1, 2) genera una excepción.

F. cards.substring(2, 2) genera una excepción.

G. cards.substring(6, 5) genera una excepción.

H. cards.substring(6, 6) genera una excepción.

18. ¿Cuál es el resultado del siguiente código? (Elija todas las que correspondan).

```
13: Cadena s1 = """
14:         ronroneo""";
```

Machine Translated by Google



```
15: Cadena s2 = "";
dieciséis:
17: s1.toUpperCase(); 18:
s1.recortar(); 19:
s1.substring(1, 3); 20: s1 +=
"dos";
21:
22: s2 += 2; 23:
s2 += 'c'; 24: s2
+= falso;
25:
26: si ( s2 == "2cfalse") System.out.println("=="); 27: if
( s2.equals("2cfalse")) System.out.println("equals"); 28: System.out.println(s1.length());
```

R.2 _

B.4 _

C.7 _

D.10 _

E. ==

F. es igual a G.

Se lanza una excepción.

H. El código no se compila.

19. ¿Cuál de los siguientes completa el espacio en blanco para imprimir un número entero positivo? (Elija todas las que correspondan).

```
String[] s1 = { "Camello", "Pavo real", "Llama"}; Cadena[] s2
= { "Camello", "Llama", "Pavo real"}; Cadena[] s3 =
{ "Camello"}; Cadena[] s4 =
{ "Camello", nulo}; System.out.println(Arrays.
                                        _____ );
```

A. comparar (s1, s2)

B. falta de coincidencia (s1, s2)

C. comparar (s3, s4)

D. falta de coincidencia (s3, s4)

E. comparar (s4, s4)

F. falta de coincidencia (s4, s4)

Machine Translated by Google



20. Tenga en cuenta que el 13 de marzo de 2022 es el fin de semana en el que los relojes se adelantan al horario de verano. ¿Cuál es el resultado de lo siguiente? (Elija todas las que correspondan).

```
var fecha = LocalDate.of(2022, Mes.MARZO, 13);
var tiempo = LocalTime.of(1, 30);
var zona = ZoneId.of("EE.UU./Este");
var dateTime1 = ZonedDateTime.of(fecha, hora, zona);
var fechaHora2 = fechaHora1.plus(1, ChronoUnit.HORAS);

diferencia larga = ChronoUnit.HOURS.between(dateTime1, dateTime2);
int hora = dateTime2.getHour();
desplazamiento booleano = dateTime1.getOffset()
    == dateTime2.getOffset();
System.out.println("diff = " + diferencia);
System.out.println("hora = " + hora);
System.out.println("desplazamiento = " + compensación);
```

A. diferencia = 1

B. diferencia = 2

C. hora = 2

D. hora = 3

E. desplazamiento = verdadero

F. El código no se compila.

G. Se lanza una excepción de tiempo de ejecución.

21. ¿Cuál de los siguientes puede llenar el espacio en blanco para imprimir avaJ? (Elija todas las que correspondan).

```
3: var rompecabezas = new StringBuilder("Java");
4: rompecabezas. _____ ;
5: System.out.println(rompecabezas);
```

A. revertir()

B. append("vaJ$").substring(0, 4)

C. append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length() - 1)

D. append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length())

E. Ninguna de las anteriores

22. ¿Cuál es el resultado del siguiente código?

```
var fecha = LocalDate.of(2022, Mes.ABRIL, 30);
fecha.plusDays(2);
fecha.plusAños(3);
System.out.println(date.getYear() + + " " + fecha.getMonth()
    + " " fecha.getDayOfMonth());
```

Machine Translated by Google



A. 30 DE ABRIL DE 2022

B.2022 2 DE MAYO

C.2025 2 DE ABRIL

D. 2025 30 DE ABRIL

E.2025 2 DE MAYO

F. El código no se compila.

G. Se lanza una excepción de tiempo de ejecución.