

Curso de programación

Java

Mariona Nadal Farré

ANAYA MULTIMEDIA

Manual Imprescindible

Java

Curso de programación

Mariona Nadal Farré

ANAYA MULTIMEDIA

Manual Imprescindible

Realización y adaptación de cubierta: Celia Antón Santos
Diseño de maqueta: Laura Apolonio Guerra
Revisión: Gelsys M. García Lorenzo y Gustavo Pérez
Maquetación: Claudia Valdés-Miranda Cros
Responsable editorial: Eugenio Tuya Feijoó

Primera edición electrónica: 2021

Todos los nombres propios de programas, sistemas operativos, equipos hardware, etc., que aparecen en este libro son marcas registradas de sus respectivas compañías u organizaciones.

Reservados todos los derechos. El contenido de esta obra está protegido por la Ley, que establece penas de prisión y/o multas, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujeren, plagiaren, distribuyeren o comunicaren públicamente, en todo o en parte, una obra literaria, artística o científica, o su transformación, interpretación o ejecución artística fijada en cualquier tipo de soporte o comunicada a través de cualquier medio, sin la preceptiva autorización.

Imágenes no aportadas por la autora: © 2021 iStockphoto LP/ Getty Images

Edición española:

© EDICIONES ANAYA MULTIMEDIA (GRUPO ANAYA, S.A.), 2021
Calle de Juan Ignacio Luca de Tena, 15, 28027 Madrid.

ISBN: 978-84-415-4425-3
Edición digital sobre la 1.ª edición impresa

A todos mis júniores, tanto a aquellos que disfrutaron con el «macetohuerto», como los que lo sufrieron (incluso a los que ni se enteraron de que lo hicimos). ;)

¡Con vosotros empezó todo!



Mariona Nadal Farré

SOBRE LA AUTORA

Mariona Nadal Farré (Barcelona, 1980) es ingeniera informática por la Universidad Politécnica de Madrid (2003).

Cuenta con más de 20 años de experiencia en la práctica del desarrollo de aplicaciones Java en entornos empresariales para grandes clientes internacionales, experiencia que le ha permitido ser formadora en J2EE de jóvenes ingenieros en su primer empleo como programadores.

También es instructora de LinkedIn Learning, donde cuenta con un número creciente de cursos sobre Fundamentos de la Programación y Java.

Su manera de escribir fresca, directa y realista hace sus cursos amenos, claros y útiles, llevándote a través de la práctica a un estilo de programación de fácil mantenimiento y alta empleabilidad.

En http://school.dsrroma.es encontrarás todos sus cursos y más información.

Web DSR School

bit.ly/dsrschoool

Cursos en LinkedIn Learning

bit.ly/lilmnadal

gracias

Gracias a mi madre, Reyes, por regalarme, con gran esfuerzo, mi primer ordenador. A Fernando, por retarme a usarlo más y mejor.

Gracias a Julio, por empujarme a aprender Java fuera del temario oficial y por acompañarme en mi aprendizaje de la programación; y a Carmen, por convertirme en una programadora profesional.

Gracias a quien decidiera que yo me encargara durante un tiempo de las formaciones de los nuevos júniores, y a todos ellos por enseñarme mientras aprendían.

Gracias a Carlos por ficharme como instructora en LinkedIn Learning; y a Eugenio, por ofrecerme la oportunidad de escribir este libro.

Gracias a Gelsys y Claudia, y a Celia y el resto del equipo de Anaya Multimedia, por convertir unos textos en esta obra.

Gracias a mis hermanos, Guillem y Pol, por alejarme del ordenador regularmente.

Gracias también a Eduardo, Pablo, Mike, Ricardo, Xavier y a Quines Mones, a cada uno por sus distintas contribuciones y apoyos durante mis procesos creativos.

Gracias a todas aquellas personas que, aunque no mencione explícitamente, han hecho sus aportaciones, a lo largo de mi vida, que me han llevado a escribir este volumen.

Y, por supuesto, también gracias a todo el personal involucrado en la impresión, distribución, exposición, reposición, venta, recomendación, reparto… sin su trabajo no sería posible que tengas hoy en tus manos este manual.

Índice de contenidos

Cómo usar este libro 16
Introducción 20

PARTE 1. Estructuras básicas 23

1. Mi primer programa 25
Introducción . . . 26
Creando una clase en Java . . . 26
El método main en Java . . . 28
Ejecutar el método main de Java desde la línea de comandos . . . 30
¿Qué puede salir mal con el método main de Java? . . . 31
Mostrando mensajes al usuario en Java (System.out.println) . . . 35
Test y ejercicios . . . 36
Soluciones . . . 37

2. Argumentos, variables, métodos y operadores 41
Introducción . . . 42
Argumentos . . . 42
Los argumentos del main y las secuencias . . . 42
¿Cómo ejecutar el main con argumentos de texto? . . . 44
Variables . . . 46
Declaración y uso de variables . . . 46
Tipos de las variables en Java . . . 48
Métodos . . . 49
Declaración de métodos . . . 49
Uso de métodos . . . 51

Operadores . . . 53
Instrucción de asignación . . . 53
Operadores aritméticos . . . 53
Operadores relacionales . . . 55
Operadores lógicos . . . 57
Operadores bit a bit . . . 58
Test y ejercicios . . . 59
Soluciones . . . 63

3. Estructuras condicionales 73
Introducción . . . 74
Una condición: si (if)... si no (else)... . . . 74
Varias condiciones: si no si (else if)... . . . 75
Constantes . . . 78
Muchas condiciones: switch . . . 80
Cuando no hace falta el if . . . 81
El operador ternario . . . 82
Test y ejercicios . . . 83
Soluciones . . . 86

4. Estructuras iterativas 95
Introducción . . . 96
Bucle «para cada» elemento de una colección (for each) . . . 96
Bucle «para» (for) . . . 97
Bucle «mientras» se cumpla la condición (while) . . . 98
Bucle «haz mientras» se cumpla la condición (do while) . . . 100
La clase Scanner . . . 102
Bucles anidados . . . 104
Caracteres especiales . . . 105
Test y ejercicios . . . 107
Soluciones . . . 110

5. Proyecto «piedra, papel, tijeras» 121
Introducción . . . 122
Presentación del programa . . . 123
Preparar el esqueleto y las constantes necesarias . . . 123
Mostrar instrucciones al usuario . . . 124
Generar jugada del ordenador . . . 125
Recoger jugada del usuario . . . 126
Interpretar jugada del usuario . . . 127
Calcular el ganador de la jugada . . . 130
Mostrar el resultado de la jugada . . . 133
Hacer el juego repetitivo . . . 134
Mejoras y evoluciones . . . 137

PARTE 2. Orientación a objetos 139

6. Diseño orientado a objetos 141
Introducción . . . 142
Requisitos del proyecto . . . 142
Extracción de conceptos . . . 143
Boceto del modelo conceptual . . . 145
Representación del modelo conceptual en UML . . . 147
Test . . . 148
Soluciones . . . 149

7. Clases y objetos 151
Introducción . . . 152
Paquetes . . . 152
Interfaces . . . 155
Las interfaces de sOOPer . . . 155
Clases abstractas . . . 158
Las clases abstractas de sOOPer . . . 158
Clases «normales» . . . 162
Las clases de sOOPer . . . 162
Enumerados . . . 165
Los enumerados de sOOPer . . . 165
Objetos . . . 167
Los objetos de sOOPer . . . 167
Los objetos en la máquina virtual de Java . . . 170
Test . . . 172
Soluciones . . . 173

8. Relaciones orientadas a objetos 175
Introducción . . . 176
Herencia . . . 176
La herencia en sOOPer . . . 177
Sobrecarga y sobrescritura . . . 183
Sobrescritura de toString en el sOOPer . . . 184
Algoritmo de distribución de productos en contenedores en sOOPer . . . 188
Reparto de responsabilidades . . . 190
Reparto de responsabilidades en sOOPer . . . 191
Diagrama de secuencia . . . 195
Diagrama de secuencia del método addProducto . . . 195
Detección y corrección de problemas . . . 196
Detección y corrección de problemas en sOOPer . . . 197
Probado no exhaustivo de la implementación . . . 203

Modificadores de Java 207
Ejemplo estático 209
Test 212
Soluciones 213

9. Proyecto «macetohuerto» 215

Introducción 216
Presentación del proyecto 216
Extracción de conceptos del macetohuerto 218
Estructura de paquetes del macetohuerto 219
Interfaces del macetohuerto 219
Interfaz IHuerto 219
Interfaz IMaceta 220
Enumerado FormaMaceta 220
Interfaz IPlanta 220
Enumerado Familia 221
Enumerado Especie 221
Clases de la estructura de macetas del macetohuerto 221
Clase abstracta Maceta 221
Clase MacetaRectangular 224
Clase MacetaTubular 225
Clases de la estructura de plantas del macetohuerto 226
Clase abstracta Planta 226
Clase abstracta PlantaAromatica 229
Clase abstracta PlantaFruto 229
Clase abstracta PlantaHoja 230
Clase abstracta PlantaRaiz 230
Clase Hinojo 230
Clase Lechuga 231
Clase Perejil 231
Clase Tomate 232
Clase TomateCherry 232
Nueva implementación de la clase Tomate 232
Clase Zanahoria 233
El huerto y sus procesos 233
Clase Huerto 233
Método plantar de la clase Maceta 235
Método esCompatible de la clase Planta 236
Método tengoEspacio de la clase Planta 236
Método tengoEspacio de la clase PlantaRaiz 237
Pruebas 238
Clase Sistema 238

PARTE 3. Buenas prácticas 243

10. Manejo de excepciones 245
Introducción . . . 246
¿Qué son las excepciones? . . . 246
¿Qué es una excepción? . . . 246
Tipos de excepciones . . . 247
Naturaleza de las excepciones . . . 253
API de las excepciones . . . 254
Ejemplo práctico . . . 256
¿Cómo manejar las excepciones? . . . 259
¿De dónde vienen las excepciones? throw . . . 259
¿Son contagiosas? throws . . . 260
¿Pero no se pueden tratar? try / catch . . . 261
¿Pero no se pueden tratar? (Más complicado) . . . 262
¿Siempre hay que tratarlas? . . . 265
¡Finalmente! finally . . . 267
Con recursos (try) . . . 268
Multicaptura (catch) . . . 270
Ejemplo práctico . . . 270
Malas prácticas . . . 279
Comérsela con patatas . . . 279
Comérsela con lechuga . . . 281
Perder la memoria histórica . . . 283
Generalizar . . . 284
Conclusión . . . 286
Test y ejercicios . . . 286
Soluciones . . . 291

11. Depuración 295
Introducción . . . 296
Cómo depurar en Eclipse . . . 296
Depurando Supermercado . . . 297
Para qué sirve depurar . . . 300
Test . . . 301
Soluciones . . . 302

12. Test unitarios 305
Introducción . . . 306
Test Driven Development . . . 306
JUnit . . . 307
Esqueleto de un test unitario . . . 308
Primeros test unitarios . . . 310

Detección de regresiones 312
Elección de los test unitarios adecuados 314
Testando cosas que van mal 318
Objetos maquetados (mocks) 322
Interfaz SuperCalculadora 322
Clase SuperCalculadoraImpl 322
Clase de test SuperCalculadoraTest 323
Configuración para utilizar Mockito 323
Clase de test SuperCalculadoraTest utilizando Mockito 324
Test 325
Soluciones 326

13. Trazas de ejecución 329

Introducción 330
Niveles de prioridad 330
Configuración 331
Uso 332
Buenas prácticas 332
Ejemplo práctico 333
Dependencias 333
Configuración 334
Clase de ejemplo App.java 335
Salida por consola 336
Ficheros de logs 337
Test y ejercicios 338
Soluciones 339

14. Proyecto «Gestión de récords» 341

Introducción 342
Funcionalidad básica del programa 343
Clases RecordsManager 343
Validaciones 345
Clase RecordsManager 346
Clase PlayerNameTooShortException 348
Clase ScoreTooLowException 348
Control de errores 349
Clase RecordsManager 349
puntuaciones.txt 351
Salida por consola 352
Fichero de salida (records.txt) 352
Test unitarios 353
Clase RecordsManagerTest 353
Mejoras y evoluciones 355

PARTE 4. Datos en Java 357

15. Estructuras de datos 359

Introducción . . . 360
Cadenas de texto . . . 360
Clase String . . . 361
Clase Character . . . 362
Clases StringBuffer y StringBuilder . . . 363
Internacionalización y localización . . . 363
Clase MessageFormat . . . 364
Números . . . 366
Autoboxing y unboxing . . . 366
Grandes números y alta precisión . . . 367
Clase NumberFormat . . . 368
Fechas . . . 369
Fechas a la antigua . . . 369
Fechas a partir de Java 8 (paquete java.time) . . . 371
Colecciones . . . 373
Tipos genéricos . . . 374
Listas . . . 375
Conjuntos . . . 376
Funciones hash . . . 377
Diccionarios . . . 379
Test y ejercicios . . . 382
Soluciones . . . 384

16. Bases de datos: Mapeo Objeto-Relacional 393

Introducción . . . 394
Herramientas necesarias . . . 395
Configuración del proyecto . . . 396
Entidades . . . 399
Entidad Pedido . . . 401
Objetos de Acceso a Datos (DAO) . . . 401
Interfaz Dao . . . 402
Gestor de entidades . . . 403
El DAO abstracto . . . 405
Los DAO del gestor . . . 407
Consultas simples . . . 409
Consulta del pedido más reciente . . . 410
Consulta de pedidos de la semana pasada . . . 413
Relaciones 1:N . . . 415
Entidad Albaran . . . 415
El DAO de Albaran . . . 418

Relaciones 1:N 418
Entidad Factura 418
El DAO de Factura 420
Relaciones M:N 420
Entidad Producto 421
Gestor de Pedidos 422
Relaciones M:N bidireccionales 423
Consultas con criterios de búsqueda 425
Pros y contras de criterios y queries 425
Recuperando datos sin consultas. 426
Clase de prueba SinQueries 426
Las queries que no vemos 427
Test 428
Soluciones 430

17. Expresiones lambda y Streams 433

Introducción 434
Expresiones lambda 434
Calculadora lambda 434
Streams 436
Cuentas varias 436
Test y ejercicios. 437
Soluciones 438

18. Proyecto «otra reunión más» 441

Introducción 442
Código de base. 442
Cartel de sala 442
Clase ReunionDao. 443
Clase CartelSala 443
Informe de reunión 445
Clase InformeReunion 445
Informes de todas las reuniones. 447
Clase InformeReuniones 447
Mejoras y evoluciones 451
Esto es solo el principio 451

Índice alfabético 452

Cómo usar este libro

A quién va dirigido y qué es necesario para empezar

Este libro va dirigido principalmente a programadores noveles de Java, que comienzan a dar sus primeros pasos en el mundo de la programación. Empezamos desde cero para, paso a paso, lograr el desarrollo de pequeños programas.

Los programadores con cierta experiencia también pueden usar este libro para afianzar algunos conceptos más complejos, yendo directamente al capítulo que necesiten o, para practicar test y ejercicios, yendo al final de cada capítulo o a los capítulos finales de cada parte.

Hablo de programadores noveles y de programadores con cierta experiencia, pero ¿verdad que no he dicho programadores jóvenes o mayores? Porque poco importa la edad para empezar a programar. Este libro está pensado para ayudar a aprender a programar a:

- Niños (y niñas, evidentemente) mayores y adolescentes, que como actividad extraescolar o curricular quieren sacarle el máximo partido a su razonamiento lógico y prepararse para un mundo que nos exige ser capaces de afrontar y resolver problemas.
- Jóvenes estudiantes, ya estén estudiando programación ya sean de otras ramas del conocimiento: para casi cualquier profesión del futuro saber programar será un plus muy valorado.
- Adultos, profesionales o aficionados, programadores de otros lenguajes o con otras profesiones, reciclarse y seguir aprendiendo forma parte de nuestro día a día.
- Mayores, con más tiempo disponible y aún mucha curiosidad vital. Nunca es tarde para lanzarse a programar.

Para seguir los ejemplos y los ejercicios del libro necesitarás preferiblemente un ordenador con:

- JDK (kit de desarrollo Java), disponible en la web oracle.com. No es suficiente con la máquina virtual de Java (JVM) que suele estar ya instalada. Entre otras utilidades, incluye los ejecutables javac y java para compilar y ejecutar el código sin utilizar otras herramientas.
- Un IDE (entorno de desarrollo integrado), que es un editor de texto adaptado para programadores que integra muchas funcionalidades útiles, como la de compilar y ejecutar el código. No es imprescindible, pero, según van creciendo los proyectos, se agradece tenerlo. En este libro he utilizado Eclipse IDE, pero, si prefieres otro, también te sirve. Puedes descargar Eclipse en eclipse.org.

Si no puedes instalar estas herramientas, pero tienes acceso a internet, busca «java compiler online» y encontrarás páginas web que te permitirán probar tu código online, desde una tablet o un móvil. Si bien no son los dispositivos más adecuados para programar, sí te servirán para seguir aprendiendo.

Estructura del libro

Esta obra se divide en cuatro partes. Cada una de ellas trabaja tres o cuatro temas relacionados. El último capítulo es un proyecto práctico con el que afianzar los conocimientos adquiridos.

La primera parte nos introduce al mundo de la programación en general mediante la **programación estructurada**, es decir, los if, for, métodos y variables... ese tipo de cosas básicas. Si es la primera vez que vas a programar es imprescindible que empieces por aquí. Si ya sabes programar en algún otro lenguaje, te convendría al menos una lectura rápida para asociar tus conocimientos previos con sus equivalentes en Java. Si ya has hecho tus pinitos en Java, tampoco está de más que le eches un ojo, aunque quizá no sea lo más prioritario.

La segunda parte está dedicada a la **orientación a objetos** que es el paradigma de programación que realmente se utiliza en Java. Aprenderás desde cómo extraer los conceptos de un problema hasta convertirlos en una jerarquía de clases, aprendiendo a repartir funcionalidades, la reutilización de código mediante herencia, la sobrescritura y la sobrecarga de código.

En la tercera parte veremos **buenas prácticas** que quizá no son necesarias en prácticas académicas o proyectos personales, pero sí son muy importantes en proyectos profesionales. Por favor, dedícale especial energía al tema de las excepciones. Era el tópico en el que mis alumnos siempre querían profundizar más porque por lo general se trata muy superficialmente. Y sin entender bien las excepciones, no es posible manejarlas bien. Depuración, test unitarios y trazas de ejecución contribuyen a hacer el código mantenible.

Terminamos con la cuarta parte dedicada a los **datos**. En ella se explican tanto las estructuras de datos que se utilizan durante la ejecución de los programas (textos, números, colecciones...) como el acceso a las bases datos que almacenan de forma persistente la información, sin olvidar las expresiones lambda y los *streams*.

Dentro de los capítulos «de teoría», además de los ejemplos, encontrarás test para afianzar conocimientos y ejercicios para ponerlos en práctica. Intenta hacerlos antes de mirar las respuestas, pero no te saltes la lectura de la solución propuesta porque puede que te explique algo nuevo ahí mismo. Sin embargo, el capítulo final de cada parte es un proyecto en sí, a desarrollar tú y yo juntos. Esta vez, sin test ni ejercicios adicionales.

Convenios utilizados en este libro

Para facilitar la comprensión de este manual se han utilizado algunos formatos especiales:

- Los nombres de comandos, menús, opciones, cuadros de diálogo y otros elementos aparecen en letra «de palo seco» para distinguirlos fácilmente del resto del texto, por ejemplo, la instrucción System.out.println.
- Las combinaciones de teclas aparecen separadas por un guion y también en un tipo de letra diferente; por ejemplo, Ctrl-Mayús-O.
- Para indicar la secuencia para ejecutar una opción determinada, se ha decidido abreviar su escritura presentando la secuencia de menús u opciones en el orden el que deben

seleccionarse, separados por el signo «mayor que» (>). Por ejemplo, en lugar de indicar que seleccionemos la opción Perspectiva del menú Ventana y luego la opción Abrir Perspectiva, indicaremos directamente Ventana>Perspectiva>Abrir Perspectiva.

- Los fragmentos de código los verás como textos de ancho fijo, al igual que los resultados de la ejecución de los programas, el color del texto te dará la pista para saber de qué tipo se trata:

```
fragmento de código
resultados de ejecución
```

T0.0

E0.0

- Los test y ejercicios se encuentran al final del capítulo, pero en el momento en el que ya sabes lo imprescindible para resolverlos, encontrarás estos iconos referenciando el número de test (blanco sobre negro) o de ejercicio (negro sobre blanco) que ya puedes intentar.
- A lo largo del libro aparecen notas informativas separadas del texto principal que proporcionan información, tales como aclaraciones, advertencias, curiosidades o consejos:

NOTA:

Para facilitar o concretar información relacionada con el tema abordado. Incluyen recomendaciones que conviene tener en cuenta.

ADVERTENCIA

Para evitar posibles errores como consecuencia de una operación mal realizada.

TRUCO:

Consejos y artimañas para facilitar el trabajo o conseguir mejores resultados.

Información de soporte

Los ejemplos, ejercicios y proyectos desarrollados en este libro están disponibles en el repositorio git de acceso público https://github.com/DSRschool/Anaya-MI-Java.

Repositorio en git
bit.ly/gitMIJava

Puedes descargarte todo en un fichero zip, pero te recomiendo aprender las operaciones básicas de git para poder sacarle el máximo provecho, pues algunos de los ficheros pueden tener varias versiones, en función de cómo vamos trabajando sobre ellos. En esta figura se aprecian las dos versiones

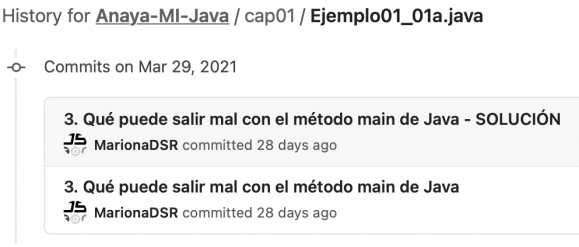


Figura C.1. Histórico de un fichero.

del primer ejemplo. Coge la primera (normalmente la de más abajo) y trabaja sobre ella para lograr llegar a la de más arriba, que ya contiene la solución.

Algunas nociones para empezar a utilizar el repositorio:

- La interfaz de gitHub puede cambiar sin previo aviso, pero además de ver el código en la propia web, mediante el botón Code es posible clonarlo parar tener tu propia versión del repositorio o descargártelo como zip.
- Viendo un fichero en la web, el botón History te permitirá conocer todos los *commits* hechos sobre ese fichero. Cuando cojas una versión en concreto, verás con el fondo blanco las líneas de código sin cambios respecto a las anteriores, con fondo rojo y un signo menos, las líneas que estaban antes y ya no están, y con fondo verde y un signo más, las nuevas. Si una línea ha sido modificada, aparecerá la versión antigua en rojo y la nueva en verde.
- El título de cada *commit* te dará una pista de en qué sección del libro se modifica ese fragmento de código.

Recomendaciones y buenas prácticas

Durante toda la obra te llevo de la mano para ir aprendiendo nuevos conceptos, te propongo ejemplos y ejercicios para afianzar esos conocimientos, pero ¡hay que ser valiente! ¡Suéltate de mi mano y avanza más! Si te muestro un ejemplo que calcula el máximo, intenta por tu cuenta calcular el mínimo. Si durante el ejercicio se te ocurren mejoras o retos, ¡hazlos! El espacio en el libro es limitado, tu capacidad de aprendizaje es infinita.

Puede parecer extraño, pero conviene que tengas papel y lápiz a mano. Poder tomar apuntes, hacer esquemas y diagramas ayuda, y mucho, a la hora de programar.

Tienes todos los ficheros disponibles para descargar, pero si de verdad quieres aprender a programar, escríbelos tú. Como los cuadernos de caligrafía de la escuela, escribir letra a letra los programas te llevará a asimilar lo que estás escribiendo. Olvídate de copiar y pegar hasta que no sepas perfectamente qué estás copiando y pegando.

Para generar buenos programas, debemos ordenar nuestra mente, pero también nuestros ficheros. Clasifícalos bien en carpetas, da nombres significativos tanto a carpetas como a los ficheros, de forma que te resulte fácil recuperar lo que estés buscando.

Contacta conmigo

Tus sugerencias, comentarios, correcciones, propuestas… son una fuente de crecimiento para mí y mis contenidos. Puedes escribirme a school@dsrroma.es. Procuraré contestar tan pronto como las circunstancias tengan a bien. También puedes seguirme en LinkedIn para estar al día de mis nuevas publicaciones.

También puedes seguirme en Twitter (@MarionaJava) o en LinkedIn (marionanadal) para estar al día de mis nuevas publicaciones.

Introducción

Has elegido aprender Java. Me parece muy buena idea. Yo también lo hice a finales del siglo pasado. Quizá algún amigo o conocido te diga que hubiera sido mejor escoger algún otro lenguaje más «de moda». Puede ser. Depende de tus objetivos, claro.

Java es un lenguaje ya veterano, pero piensa que sigue habiendo demanda de programadores COBOL, que es un lenguaje creado en 1959, que se ha utilizado en sistemas de gestión muy importantes (banca, seguros…) que siguen funcionando muy bien y continúan necesitando profesionales que los mantengan y evolucionen.

Java va por el mismo camino. En el prefacio de la segunda edición del libro *Java: Fundamentos de programación*, de Judy M. Bishop, de 1998, uno de los que utilicé para aprender yo este lenguaje que tanto me ha dado, decía: «…Java está revolucionando el mundo de la computación y está camino de convertirse en el lenguaje a enseñar a todos los estudiantes de programación del mundo entero…». Parece que Judy tenía razón.

Mi otro libro de referencia fue *Thinking in Java* de Bruce Eckel, también de 1998. Su prefacio decía: «…originalmente me acerqué a Java como simplemente "otro lenguaje de programación", que en muchos sentidos lo es. Pero a medida que pasaba el tiempo y lo estudiaba más profundamente, comencé a ver que la intención fundamental del lenguaje es diferente a todos los demás lenguajes que he visto».

Ya lo digo en la contraportada: «Quizá Java no sea el lenguaje aparentemente más sencillo para empezar, porque es estricto y fuerza a hacer las cosas bien. Aprender a programar en Java permite hacerlo con solidez y enfrentarse a otros lenguajes».

Mi objetivo principal con este curso es ayudarte como programador a generar código de calidad y fácil de mantener, y por desgracia... es necesario. Hay demasiado código de mala calidad desperdigado por el mundo. Y te vas a encontrar con él. Es una pena. Solo espero que, a partir de ahora, tú no generes más mal código y para ello necesitas una buena base. Así que también veremos cosas que se suelen hacer mal y que tú entenderás por qué están mal y por qué no deberías hacerlas.

No quiero enseñarte solo Java. Quiero que aprendas a programar bien y prepararte para formar parte de un equipo de desarrollo. Java es solo la herramienta.

¡Espero que disfrutes del camino!

Parte 1

Estructuras básicas



1

Mi primer programa

En este capítulo aprenderás a:

- Crear una clase Java.
- Identificar y generar el método main.
- Compilar una clase Java
- Ejecutar una clase Java.
- Mostrar textos al usuario.

Introducción

Java es un lenguaje orientado a objetos, pero antes nos centraremos en las estructuras básicas de programación (if, for…), comunes en casi cualquier lenguaje de programación, y olvidaremos las clases y los objetos hasta de aquí a unos capítulos, cuando ya hayamos asentado la **programación estructurada**.

Para hacer los ejercicios, utilizaremos un fichero distinto para cada uno, que será una clase Java diferente, con un método *main* (principal), que es el método que se ejecuta cuando ejecutamos una clase Java.

En el futuro, cuando hagas programas de verdad (pronto, no te preocupes), apenas usarás métodos *main*, pero ahora para aprender nos vendrá muy bien. No necesitaremos servidores ni otras complicaciones.

Cuando programamos, es muy importante que escribamos todo correctamente. En un escrito que va a leer un humano (una carta, una redacción, un libro), quedan muy mal las erratas y las faltas de ortografía, pero aun así, el humano que lo lea, si conoce el idioma, lo entenderá.

«Buennos dias, qué Tal?». No está bien escrito, pero lo entiendes, ¿verdad? Pues un ordenador, no. No lo entendería. Por eso, tenemos que poner mil ojos en poner bien todas las letras y símbolos e ir con mucho cuidado con las mayúsculas y minúsculas.

Java es un lenguaje que distingue mayúsculas y minúsculas (otros lenguajes no lo hacen), y tú verás, ¡si no las respetas, no te funcionará nada! Aún recuerdo en la facultad ver a un profesor suspender un examen en cuestión de segundos porque estaba todo escrito en MAYÚSCULAS cuando el lenguaje usado tenía sus palabras clave en minúsculas (como en Java). Supongo que a ese estudiante le pareció superinjusto ese resultado, pero espero que aprendiera la importancia de este punto.

Lo dicho, fíjate bien, porque cada parte es muy importante: si te confundes en algún detalle, puede que no funcione.

Creando una clase en Java

Empezaremos con un programa sencillo, en el que mostraremos un mensaje de texto, con el que aprenderemos las palabras clave necesarias.

Vamos a crear nuestra primera clase en Java, esto es, un fichero normal de Java.

Crea un proyecto en tu entorno de desarrollo (IDE), para ir metiendo todos los ejercicios. Llámalo PracticasIniciacion (sin acentos, sin espacios y respetando mayúsculas y minúsculas). Aunque normalmente deberíamos separar los ficheros fuente y compilados, para este curso los dejaremos juntos.

Dentro de este proyecto, crea una nueva clase y llámala Ejemplo01_01. Presta atención a mayúsculas y minúsculas, no incluyas acentos, ni espacios, ni caracteres raros (como la eñe, la cedilla, otros símbolos…). Hoy en día es probable que funcione con caracteres especiales, pero también es muy probable que dé problemas al usarlo en otros entornos, pasarle el código a alguien… Es una mala práctica, así que mejor, nada de caracteres raros.

El entorno de desarrollo generará por ti un fichero llamado como la clase, con extensión punto java, en este caso Ejemplo01_01.java.

Si abrimos ese fichero, esa clase, veremos el siguiente contenido:

```
public class Ejemplo01_01 {


}
```

También puedes crear a mano el fichero, darle ese nombre y rellenarlo con ese texto. En ese caso, fíjate bien en que el nombre del fichero coincida exactamente con el nombre que le hemos dado a la clase. En este caso Ejemplo01_01.

Veamos qué significa cada parte del código que nos ha generado el entorno o de lo que escribimos:

- **public:** es una palabra clave (reservada en Java, es decir, que no puedes usar tú para nombrar métodos, variables, clases o lo que sea) que indica que lo que vas a crear es accesible «desde fuera». En todos los ficheros .java debe haber una y solo una clase pública.
- **class:** es otra palabra reservada, que dice que lo que vamos a declarar es una clase. Aunque de momento hemos dicho que pasamos de clases y objetos, en Java todo el código tiene que estar metido en clases, así que necesitaremos crear al menos una, sí o sí, para poder juguetear con el código.
- **Ejemplo01_01:** es el nombre que le queremos dar a nuestra clase. Podemos inventarnos el nombre que queramos, pero lo suyo es que tenga un sentido y solo con leerlo podamos imaginar para qué sirve. En este caso, es el primero de nuestros ejemplos. Por convenciones de Java y para facilitar la legibilidad de nuestro código, nos acostumbraremos desde ya a escribir el nombre de nuestras clases empezando con una letra mayúscula.
- **{ }:** las llaves. Lamento informarte, si no lo habías descubierto ya, que el teclado no está especialmente pensado para programadores… Usaremos muchos símbolos del teclado que requieren varias teclas para escribirlos. Las llaves de apertura y cierre, en el teclado español (en teclados de otros países o idiomas pueden estar en sitios distintos) suelen estar situadas a la derecha del teclado en la fila central, cerca de la tecla Intro, como se muestra en la figura 1.1. La llave de apertura está en la misma tecla que el acento agudo (´) y la de cierre, con la cedilla (ç). Para escribir las llaves, debes apretar a la vez la tecla Alt o Alt Gr, que están en la fila de abajo de tu teclado, cerca de la barra espaciadora, y la tecla del acento o de la cedilla. Si te cuesta, practica un poco en cualquier fichero, porque son dos teclas básicas que cualquier programador Java debe manejar con soltura.



Figura 1.1. Ubicación de las llaves en el teclado español.

Las llaves en Java se usan para marcar los distintos bloques del código, el ámbito de cada cosa. Las llaves que acabamos de poner para la clase nos servirán para indicar que todo lo que esté entre la llave de apertura y la de cierre forma parte de la clase. O, dicho de otra manera, que todo el código de la clase lo tenemos que meter entre la llave de apertura y la de cierre.

Como vamos a tener muchas llaves anidadas, porque se usan también para métodos, bloques, etc., es importante que tabulemos bien el código para que nos sea fácil de leer, entender y mantener.

Según la costumbre, la llave de apertura se pone al final de la línea en la que se declara la clase (o el método o el bloque o lo que sea en cada caso). Hay gente que prefiere ponerla al principio de la línea siguiente, pero cada vez se ve menos, así que te recomiendo que te acostumbres a escribirla al final de la primera línea.

T1.1 La llave de cierre de la clase irá sola en una línea y pegada al margen izquierdo, es decir, sin tabular.

T1.2 ¡Ya tenemos nuestra primera clase! Este fragmento de código lo repetiremos unas cuantas veces… eso sí, cambiándole el nombre de la clase en cada caso.

El método main en Java

El método `main` es como cualquier otro método Java (tiene las mismas partes), pero es un método un poco especial. Un método en Java es parecido a las funciones en otros lenguajes. Ya iremos aprendiendo más sobre métodos. De momento, este lo necesitamos para poder hacer cualquier cosa.

El método que buscará Java para empezar la ejecución de una clase es `main`. Es decir, es el punto de entrada de nuestros programas. Para que Java reconozca este método, tenemos que escribirlo bien, respetando todas sus partes:

```
public static void main(String[] args) {

}
```

Veamos qué significa cada parte, palabra a palabra:

- **public:** como antes, es la misma palabra, es el mismo significado, pero ahora aplicado a un método en vez de a una clase. En un método cualquiera, poner el public significará que puedo llamarlo desde otras clases, pero eso ahora nos viene grande. En el caso concreto del main, es necesario que sea público, pues es nuestro punto de entrada. Antes de la palabra public, meteremos un tabulado (o 4 espacios) para separar la línea del margen izquierdo.
- **static:** esta es otra palabra reservada, y es muy pronto para explicarla, vamos a resumirlo en que significa que no necesitamos tener un objeto para usar este método, pero por ahora no te preocupes más por ella. Eso sí, es obligatoria en nuestro main.
- **void:** otra palabra reservada. En este caso, observa que va justo antes del nombre del método, está en la posición del tipo del retorno del método. void significa que el método en cuestión no devolverá ningún resultado. main siempre será void, nunca devolverá ningún resultado. Si escribes el método main devolviendo algo que no sea void, Java no lo reconocerá como main y no te funcionará.
- **main:** es el nombre del método. Como con el nombre de la clase, es posible ponerle el nombre que queramos a un método, excepto para este. El método main se tiene que llamar main. No hay otra. Observa que esta vez lo escribimos en minúsculas. Como hemos dicho antes, por legibilidad y convención, vamos a escribir siempre en minúsculas la inicial de los nombres de los métodos.
- **(:** paréntesis de apertura (en el teclado español, en la tecla del 8, Mayús-8). Los paréntesis envuelven, en la declaración de los métodos, los argumentos que estos van a recibir. Iremos aprendiendo más sobre los argumentos: en el main siempre serán String[] args.



Figura 1.2. Ubicación de los paréntesis en el teclado español.

- **String:** es el nombre de la clase que representa los textos en Java (fíjate en que se escribe con la S mayúscula, ¡es una clase!). Los argumentos del main son textos (ni números ni estructuras más complejas).
- [ ]**:** los corchetes se usan en Java (y en otros muchos lenguajes) para indicar que tenemos un *array* (una secuencia de elementos). En este caso, una secuencia de String… uno, ninguno, varios, muchos, aún no lo sabemos. En el teclado español, los corchetes están en las teclas de encima de las de las llaves, en el acento grave (`) el de apertura, y en el signo más (+) el de cierre.



Figura 1.3. Ubicación de los corchetes en el teclado español.

- **args:** nombre que le damos a ese argumento. En el main se le suele llamar args a la secuencia de textos que recibe de entrada, pero realmente podríamos ponerle otro nombre. Es el único fragmento de esta línea que es modificable, pero, como no hay ninguna necesidad de ello, mejor usar args y no complicar las cosas.
- **):** paréntesis de cierre. Ya hemos terminado con todos los argumentos. En el main solo tenemos uno, la secuencia de textos.
- **{ }:** de nuevo las llaves. Entre llave y llave meteremos todo el código del main. Como antes, la de apertura la pondremos al final de la línea. Y la de cierre, en una línea nueva, pero esta vez, le meteremos una tabulación (o 4 espacios), en vez de dejarla pegada al margen izquierdo. Tiene que quedar alineada con el inicio de nuestro método. Hoy en día la mayoría de IDE si están bien configurados saben colocar cada llave en su sitio, pero ahora mismo es importante que tú aprendas a colocarlas bien. Te ayudará a leer tu código.

Añadamos el código del método main dentro del código de nuestra clase:

```
public class Ejemplo01_01 {
   public static void main(String[] args) {

   }
}
```

Guarda los cambios (algunos IDE los guardan automáticamente). Si no se guardan los cambios, Java no los tendrá en cuenta y el comportamiento puede no ser el esperado.

Ya puedes intentar ejecutar la clase Ejemplo01_01. No hará nada, porque aún no hemos escrito nada de código, pero, si todo está bien, no debería fallar.

Aunque ahora parece la obra de El Escorial, porque lo estamos viendo con mucho detalle, pronto crearás estas líneas en apenas unos segundos.

Ejecutar el método main de Java desde la línea de comandos

Si usas un entorno de desarrollo preparado para Java, ejecutarás el método main dándole a un par de botones, pero, si quieres ser un buen programador, deberías tener ciertas nociones de cómo hacer las cosas por ti mismo y, sobre todo, entender cómo funcionan.

Para ejecutar una clase Java desde la línea de comandos tenemos que seguir dos pasos:

1. Compilar el código.
   El compilador javac es una herramienta que procesa los ficheros que hemos escrito nosotros (con letras y símbolos) y, si son correctos, genera otros ficheros que son legibles por el ordenador (en binario) y que serán los utilizados para ejecutar el programa.

   En el caso de Java, los ficheros que escribimos nosotros, el código fuente, deben ser nombrados *.java, siendo el nombre de cada clase. Los ficheros binarios generados serán nombrados *.class, siendo de nuevo el nombre de cada clase.

   Así pues, las instrucciones para este caso serían:

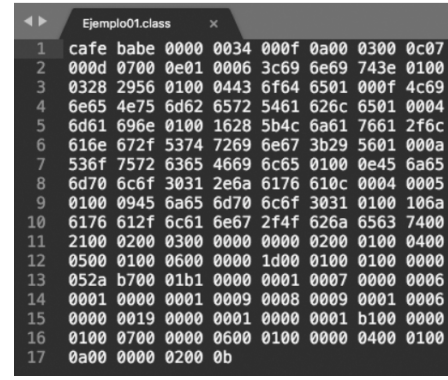


Figura 1.4. Contenido del fichero Ejemplo01_01.class.

```
> javac Ejemplo01_01.java
```

   Esta instrucción generará un fichero Ejemplo01_01.class en el mismo directorio en el que estamos.

   Si abrimos este fichero, veremos que, para nosotros, es ilegible.

2. Ejecutar el programa.

```
> java Ejemplo01_01
```

   Esta instrucción ejecutará nuestro programa, pero como nuestro programa aún no hace nada, pues no veremos ningún resultado. Pero tampoco deberíamos ver ningún error. Fíjate en que ahora no indicamos la extensión del fichero, sino el nombre de la clase.

```
> javac Ejemplo01_01.java
> ls -la
-rw-r--r--  1 mariona  staff  263  2 oct 18:27 Ejemplo01_01.class
-rw-r--r--@ 1 mariona  staff   80  2 oct 18:26 Ejemplo01_01.java
> java Ejemplo01_01
```

¿Qué puede salir mal con el método main de Java?

Al programar cometeremos errores. Es inevitable. Aprendamos cómo reconocerlos, entenderlos y resolverlos.

Si quieres seguir este ejemplo paso a paso, descarga el código de ejemplo y añádelo a la carpeta del proyecto (donde tienes Ejemplo01_01.java). Es código malo, pero es lo que necesitamos ahora.

El fichero y la clase no se llaman igual

Si tenemos un nombre de fichero (Ejemplo01_01a.java) que no coincide con el nombre de la clase (Ejemplo01_01).

Código fuente

```
// Ejemplo01_01a.java
public class Ejemplo01_01 {
   public static void main(String[] args) {

   }
}
```

Resultado de la compilación

```
> javac Ejemplo01_01a.java
Ejemplo01_01a.java:2: error: class Ejemplo01_01 is public, should be declared in
a file named Ejemplo01_01.java
public class Ejemplo01_01 {
       ^
1 error
```

Explicación

Como la clase y el fichero no se llaman igual, nos da un error. Mira cómo empieza diciéndonos:

- en qué fichero está el error :

```
Ejemplo01_01a.java
```

- el número de línea en el que falla:

```
:2:
```

- y qué problema ha habido:

```
error:
```

- seguido del mensaje:

```
class Ejemplo01_01 is public, should be declared in a file named Ejemplo01_01.
java
```

  que podríamos traducir como «la clase Ejemplo01_01 es pública. Debe ser declarada en un fichero que se llame Ejemplo01_01.java». ¿Puede ser más claro?

- después nos muestra el fragmento de código que está mal y nos lo señala con una flechita:

```
public class Ejemplo01_01 {
       ^
```

- terminando con el contador total de errores:

```
1 error
```

Solución

O cambiamos el nombre de la clase o el del fichero. En este caso, corregiremos el nombre de la clase:

```
// Ejemplo01_01a.java
public class Ejemplo01_01a {
   public static void main(String[] args) {

   }
}
```

Volvemos a compilar. Ahora ya no tenemos errores.

```
> javac Ejemplo01_01a.java
```

Hay alguna palabra clave mal escrita

Código fuente

```
public clas Ejemplo01_01b {
   public static void main(String[] args) {

   }
}
```

Resultado de la compilación

```
> javac Ejemplo01_01b.java
Ejemplo01_01b.java:1: error: class, interface, or enum expected
public clas Ejemplo01_01b {
       ^
Ejemplo01_01b.java:2: error: class, interface, or enum expected
    public static void main(String[] args) {
                  ^
2 errors
```

Explicación

En este caso tenemos dos errores. El primero nos marca un problema en clas, en la línea 1 y el segundo es en void en la línea 2.

El primer error es el que hemos provocado nosotros mismos, nos hemos dejado una s en class, y nos dice que «espera una clase, una interfaz o un enumerado». Clase, interfaz y enumerado son los elementos que Java nos permite crear y cuya declaración se hace en ese momento del fichero. Nosotros queremos crear una clase, aún es pronto para hablar de interfaces y enumerados.

El segundo error es un error fantasma: realmente no hay ningún otro problema en el código, pero como al principio está mal, el compilador, que es bastante tontorrón (o poco flexible), se pierde y puede ya no entender muchas otras cosas.

Solución

```
public class Ejemplo01_01b {
   public static void main(String[] args) {

   }
}
```

Añadimos la s que nos faltaba. Guardamos y volvemos a compilar. Ahora ya no tenemos errores.

```
> javac Ejemplo01_01b.java
```

Otro ejemplo del mismo tipo de error:

Código fuente

```
public class Ejemplo01_01c {
   public staic void main(String[] args) {

   }
}
```

Resultado de la compilación

```
> javac Ejemplo01_01c.java
Ejemplo01_01c.java:2: error: <identifier> expected
    public staic void main(String[] args) {
                ^
Ejemplo01_01c.java:2: error: invalid method declaration; return type required
    public staic void main(String[] args) {
                     ^
2 errors
```

Explicación

De nuevo tenemos dos errores, el primero en staic, donde nos falta una t, y nos está diciendo que espera un identificador. Como nos marca con la flechita la palabra staic, ya nos podemos fijar bien en ella y encontrar que nos falta la t.

Como antes, el segundo error no es real, porque el compilador ya se ha perdido y no entiende el resto de la declaración del main.

Solución

```
public class Ejemplo01_01c {
   public static void main(String[] args) {

   }
}
```

Añadimos la t que nos faltaba. Guardamos y volvemos a compilar. Ahora ya no tenemos errores.

```
> javac Ejemplo01_01c.java
```

Otros errores

T1.3

T1.4

En efecto, nos pueden salir muchísimos más errores si nos confundimos en algún otro punto de nuestro breve código. Ante un error, nunca pierdas la calma. Lee despacito, presta atención a lo que pone, porque normalmente el compilador nos dará buenas pistas del problema que haya encontrado.

Si te atascas en algún error, escríbeme, intentaré echarte una mano. Pega el código que falla y el mensaje de error que recibes, porque sin esa información sería imposible ayudarte.

Mostrando mensajes al usuario en Java (System.out.println)

Ya tenemos una clase y un método main, la estructura mínima básica para ejecutar algo en Java. Hagamos nuestro primer programa, que va a escribir «Bienvenido al curso de programación Java» por la «salida estándar». La salida estándar en un IDE es una ventana o pestaña que se suele llamar consola (o *console* si el entorno está en inglés). Si ejecutamos el código desde la línea de comandos, será la propia línea de comandos.

Para escribir por la salida estándar, simplemente tenemos que escribir esta línea:

```
System.out.println("bla bla bla");
```

Vamos a explicarla:

- **System.out.println:** complicado de explicar ahora mismo, de momento nos quedaremos con que es el método que tiene Java para escribir textos por la salida estándar.
- **( ):** los paréntesis engloban los parámetros que le vamos a pasar a ese método. Aquí, a diferencia de cuando hemos escrito la línea del main, no definimos qué nos podría pasar (tipo y nombre), sino que le pasamos valores reales. En este caso "bla bla bla".
- **"bla bla bla":** como acabamos de ver, es el valor real de lo que queremos imprimir por pantalla. Observa que va entre comillas dobles (encima del 2 en el teclado).



Figura 1.5. Ubicación de las comillas en el teclado español.

- **;:** el punto y coma… ¡otro signo más! Situado en el teclado español en la misma tecla que la coma (,), escrito usando mayúsculas y coma, el punto y coma lo usaremos al final de cada sentencia en Java. Si no lo ponemos, el compilador (el programa que lee y procesa nuestros programas) no entenderá el código y nos dará errores.



Figura 1.6. Ubicación del punto y coma en el teclado español.

Entonces, si queremos que nuestro programa escriba el texto que hemos dicho, tendremos que añadir la línea que acabamos de ver **dentro** del método main de nuestro código:

```
public class Ejemplo01_01 {
   public static void main(String[] args) {
      System.out.println("Bienvenido al curso de programación Java");
   }
}
```

T1.5

Fíjate en que la línea que acabamos de añadir lleva ahora dos tabulaciones (u ocho espacios) por delante. De forma que el código de dentro de cada bloque quede anidado, y así sean más fáciles de localizar los bloques.

E1.1

Ejecutemos nuestro segundo programa, que es el primero que hace algo.

E1.2

```
> javac Ejemplo01_01.java
> java Ejemplo01_01
Bienvenido al curso de programación Java
```

Test y ejercicios

Test 1.1. ¿Cuál es la forma correcta de declarar una clase?

a) `class public NombreDeLaClase() { }`
b) `public class NombreDeLaClase { }`
c) `class public NombreDeLaClase { }`
d) `public class NombreDeLaClase() { }`

Test 1.2. ¿Cómo debe llamarse la clase que escribamos en el fichero MiClase.java?

a) MiClase.class
b) mi-clase
c) MiClase
d) Como quiera

Test 1.3. ¿Cuál es la forma correcta de declarar el método main?

a) `static public main void(String[ ] args) { }`
b) `static public main void(String[ ] args) { }`
c) `public static void main(String[ ] args) { }`
d) `public static main(String[ ] args): void { }`

Test 1.4. ¿Cuál es la única palabra de public static void main(String[] args) que se puede modificar?

a) `args`
b) `main`
c) `static`
d) `void`

Test 1.5. ¿Cuál es la forma correcta de escribir una línea por la salida estándar?

a) `system.out.println()`
b) `System.Out.Println()`
c) `System.out.println()`
d) `System_out_println()`

Ejercicio 1.1. Mostrando un mensaje de una línea

Escribe un programa que imprima por pantalla la frase «`Estoy aprendiendo a programar bien en Java`».

Ejercicio 1.2. Mostrando un mensaje de varias líneas

Escribe un programa que imprima por pantalla estas dos frases: «`La vida es tan buena maestra,`» y «`que si no aprendes la lección te la repite.`».

Soluciones

Test 1.1. ¿Cuál es la forma correcta de declarar una clase?

b) **public class NombreDeLaClase { }:** El orden de las palabras y el uso de los símbolos es el que tiene que ser.

Test 1.2. ¿Cómo debe llamarse la clase que escribamos en el fichero MiClase.java?

c) **MiClase:** Efectivamente, debe coincidir con el nombre del fichero, sin la extensión.

Test 1.3. ¿Cuál es la forma correcta de declarar el método main?

c) **public static void main(String[ ] args) { }:** ¡Exacto! Apréndetelo bien, porque este es el único orden válido.

Test 1.4. ¿Cuál es la única palabra de public static void main(String[] args) que se puede modificar?

a) **args:** Aunque no ganamos nada cambiando el nombre de los argumentos, efectivamente, podríamos poner otro.

Test 1.5. ¿Cuál es la forma correcta de escribir una línea por la salida estándar?

c) **System.out.println():** Entenderás el porqué más adelante, pero si te impacienta: System es una clase, out un atributo y println un método, por eso System empieza en mayúscula y el resto no.

Ejercicio 1.1. Mostrando un mensaje de una línea

- Primero declararemos la clase:

```
public class Ejercicio01_01 {
}
```

ADVERTENCIA:

Usar el guion bajo o de subrayado en el nombre de la clase no es una buena idea, porque no cumple las convenciones de nombrado de Java (que aprenderemos más adelante); sin embargo, llamar a la clase «Ejercicio0101» dificulta la legibilidad.

- El segundo paso será declarar el método main, dentro de la clase.

```
public class Ejercicio01_01 {
   public static void main(String[] args) {
   }
}
```

No olvides respetar las tabulaciones.

- Tercer y último paso: incluir la sentencia para imprimir por pantalla el mensaje que nos han pedido:

```
public class Ejercicio01_01 {
   public static void main(String[] args) {
      System.out.println("Estoy aprendiendo a programar bien en Java");
   }
}
```

Cuidado no te confundas con las mayúsculas y minúsculas.

Guardamos el fichero, vamos a la línea de comandos y:

- compilamos

```
> javac Ejercicio01_01.java
```

- ejecutamos

```
> java Ejercicio01_01
Estoy aprendiendo a programar bien en Java
```

¡Ahora sí obtenemos un resultado! Justo el mensaje que queríamos enseñarle al usuario.

Ejercicio 1.2. Mostrando un mensaje de varias líneas

- Primero declararemos la clase:

```
public class Ejercicio01_02 {
}
```

- El segundo paso será declarar el método main, dentro de la clase.

```
public class Ejercicio01_02 {
   public static void main(String[] args) {
   }
}
```

- Tercer y último paso: incluir las sentencias para imprimir por pantalla los mensajes que nos han pedido:

```
public class Ejercicio01_02 {
   public static void main(String[] args) {
      System.out.println("La vida es tan buena maestra,");
      System.out.println("que si no aprendes la lección te la repite.");
   }
}
```

Para escribir dos frases, nos basta con llamar dos veces al System.out.println, una tras otra, dentro del método main, cada vez con una de las dos frases.

Guardamos el fichero, vamos a la línea de comandos y:

- compilamos

```
> javac Ejercicio01_02.java
```

- ejecutamos

```
> java Ejercicio01_02
La vida es tan buena maestra,
que si no aprendes la lección te la repite.
```

¡Esta vez veremos dos líneas!

Podemos también ejecutarlo desde nuestro IDE. En la figura 1.7 se muestra en Eclipse.

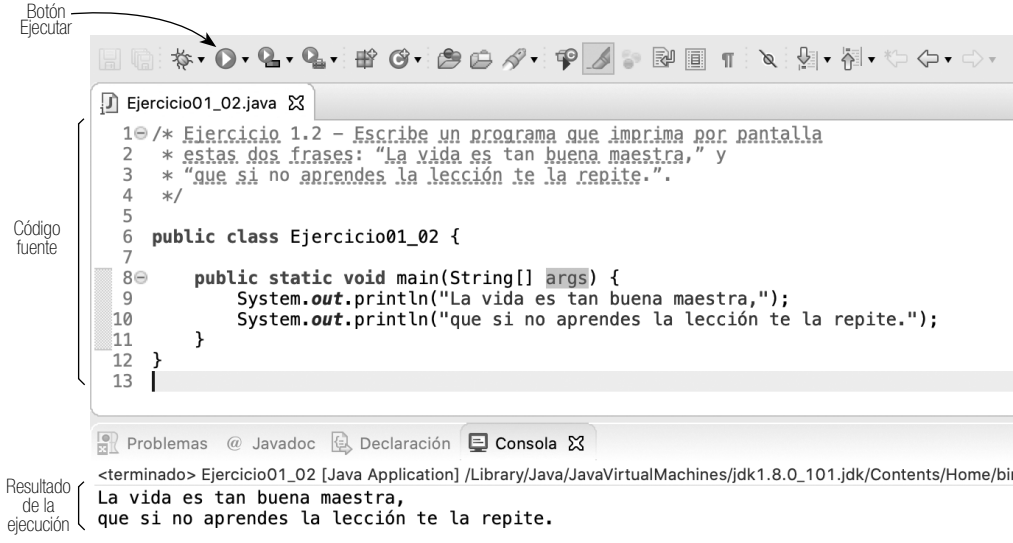


Figura 1.7. Ejecución de un programa desde Eclipse.



2

Argumentos, variables, métodos y operadores

En este capítulo aprenderás a:

- Identificar qué son los argumentos y cómo pasárselos a tus programas.
- Reconocer las variables y de qué tipos pueden ser.
- Declarar y cómo llamar a métodos.
- Utilizar los operadores de Java.

Introducción

Argumentos, variables, métodos y operadores son cuatro conceptos fundamentales de la programación.

Los **operadores** nos permiten realizar operaciones lógicas o matemáticas sobre las **variables** en las que almacenamos datos. Estas cuentas las haremos dentro de **métodos** que pueden ser llamados desde distintos puntos de los programas, pasándoles **argumentos** para adaptarlos a cada caso.

Aunque lo que veremos en este capítulo está orientado al lenguaje de programación Java, es aplicable a casi cualquier otro lenguaje de alto nivel, aunque puede haber diferencias en la sintaxis, es decir, en la forma de escribir las cosas.

Los conceptos como tal son universales. Algunos detalles pueden ser propios de Java.

Argumentos

Los argumentos del main y las secuencias

Cuando declaramos nuestro primer método `main`, dijimos que recibía como argumentos una secuencia de textos (`String[] args`). Vamos a jugar ahora un poco con estos argumentos.

Los argumentos de un método posibilitan que el método se comporte de manera distinta en función del valor de los mismos. De hecho, hasta ahora, le hemos pasado argumentos distintos al `System.out.println` cada vez que lo hemos llamado y, en función del texto que le hayamos pasado cada vez, eso nos ha impreso por pantalla.

En el cole, en su día nos enseñaron a sumar y a poner el símbolo + entre dos números. Y si la señorita escribía 1 + 2 en la pizarra, coreábamos juntos un «¡tres!». Y si escribía 2 + 3, decíamos «¡cinco!». Pues bien, ahí tenemos un método, la suma, que recibe argumentos (1 y 2 en el primer caso, 2 y 3 en el segundo) y cuyo resultado depende de esos dos argumentos. Pues esa misma idea se aplica a los métodos cuando programamos.

Volvemos a los argumentos del `main`. Son una secuencia de `Strings`. En la programación Java de hoy en día apenas se usan los *arrays* (o secuencias), porque tenemos otras estructuras más flexibles y potentes, las colecciones, pero para aprender, las secuencias nos vendrán muy bien y en el `main` siempre las tendremos. Aunque en la programación real, apenas se usa el `main` o el `System.out.println`.

Pero no te preocupes, no estás perdiendo el tiempo. De adultos apenas gateamos, pero antes de andar, cuando fuimos bebés, gatear nos resultó muy útil. Pues aquí es lo mismo. Estamos aprendiendo a gatear en Java y a dar nuestros primeros pasos, para que cuando «seamos mayores» podamos correr, saltar y hacer lo que haga falta para resolver los problemas a los que hagamos frente.

Volvamos a la secuencia de argumentos del main. Para acceder a los argumentos, tenemos que usar la palabra que hayamos utilizado para nombrarlos. En este caso es args, pero recuerda que podríamos haber puesto cualquier otra.

Para acceder a cada uno de los elementos de la secuencia, tenemos que usar ese nombre, args, junto a la posición del elemento deseado entre corchetes: args[0].

Ten en cuenta que en Java empezamos a contar los elementos de las secuencias, listas y del resto de colecciones desde cero. El primer elemento estará en la posición 0 y, por ejemplo, el último elemento de una secuencia de tres elementos estará en la posición 2 (args[2]).

Para saber cuántos elementos tiene una secuencia, es posible consultar el atributo length, haciendo args.length. Por tanto, para acceder al último elemento de una secuencia, haremos args[args.length - 1]. Es muy importante el -1, porque empezamos a contar desde cero y, si no pusiéramos el -1, ¡nos saldríamos del *array*!

Hagamos nuestro primer ejemplo de uso de los argumentos. Vamos a imprimir por pantalla el texto «El último elemento que me has pasado es …» y el valor del último argumento recibido.

- Creamos la clase.
- Añadimos el método main.
- Escribimos la sentencia System.out.println.

TRUCO:

Eclipse y los entornos de desarrollo en general son muy majos y nos ayudan… Si escribimos sysout así todo junto y en minúsculas, y le damos a Ctrl + espacio, ¡sorpresa!, ¡nuestro IDE escribe el resto!

- Añadimos el texto entrecomillado.
- Concatenamos el valor requerido usando el signo más (+), el de sumar, de toda la vida. Lo emplearemos aquí para «sumar» textos o, dicho más profesionalmente, para «**concatenar** Strings».

```
public class Ejemplo02_01 {
   public static void main(String[] args) {
      System.out.println("El último elemento que me has pasado es "
         + args[args.length-1]);
   }
}
```

Fíjate bien en que al final del String literal (con el texto "El último elemento que me has pasado es ") y antes de las comillas que lo cierran he puesto un espacio. Esto es para que la palabra "es" no se quede pegada al valor que saquemos del último elemento. Cuando concatenas Strings debes tener cuidado con los espacios y otros separadores, para cerciorarte de que el resultado sea legible.

¿Cómo ejecutar el main con argumentos de texto?

Intentemos compilar y ejecutar este código como hicimos en el capítulo anterior:

```
> javac Ejemplo02_01.java
> java Ejemplo02_01
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at Ejemplo02_01.main(Ejemplo02_01.java:3)
```

¿Qué ha pasado aquí? Al ejecutar el primer ejemplo, no necesitábamos argumentos, pero esta vez ya sí. Y como no le estamos pasando ningún argumento y luego intentamos acceder a uno de ellos, nos da una excepción. Observa que no nos dio ningún problema de compilación, porque la sintaxis de nuestro código es correcta. Nos da una excepción que es el mecanismo que usa Java para avisarnos de que ha habido algún problema durante la ejecución del código. En el capítulo 10 aprenderás a tratar y resolver excepciones, pero, de momento, lo resolveremos sin meternos en ellas.

Nuestro problema viene de no pasarle argumentos… ¡pasémoselos!

```
> java Ejemplo02_01 hola
El último elemento que me has pasado es hola
```

Como puedes ver, ahora, ya no nos da errores. Le hemos pasado un argumento (hola). Como solo tenemos uno, ese mismo será el último.

Probamos ahora con más argumentos, esta vez tres: hola, qué, tal. Presta atención a que no les ponemos ni comas ni comillas.

```
> java Ejemplo02_01 hola qué tal
El último elemento que me has pasado es tal
```

Si quisiéramos que «hola qué tal» fuera un único argumento, con ponerlo entre comillas tendríamos suficiente:

```
> java Ejemplo02_01 "hola qué tal"
El último elemento que me has pasado es hola qué tal
```

Ejecución con argumentos en un IDE

En cada entorno de desarrollo los pasos para pasarle los argumentos pueden ser ligeramente distintos, veamos cómo hacerlo en Eclipse.

1. En vez de clicar directamente sobre la flecha verde para ejecutar el código, vamos a desplegar, con un clic en la flechita negra hacia abajo (paso 1 de la figura 2.1), y escogeremos la opción Configuraciones de Ejecución o Run configurations (según en qué idioma tengas configurado el Eclipse) (paso 2 de la figura 2.1).
2. Seleccionaremos la pestaña Argumentos o Arguments (paso 3 de la figura 2.2) y en el primer cuadro de texto (paso 4 de la figura 2.2) meteremos nuestros argumentos. Como hemos hecho por línea de comandos, sin comas ni comillas, a no ser que, como hemos visto antes, queramos pasar un argumento con espacios, que entonces sí meteremos las comillas.
3. Clicamos sobre el botón Ejecutar (paso 5 de la figura 2.2).

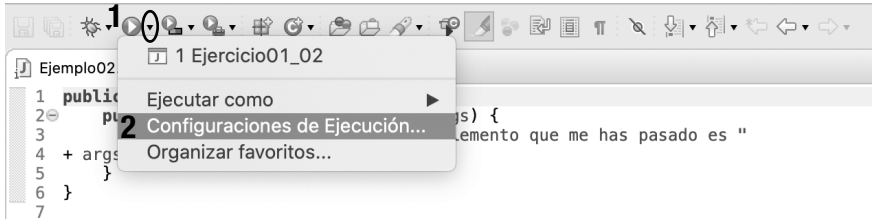


Figura 2.1. Lanzar configuraciones de ejecución en Eclipse.

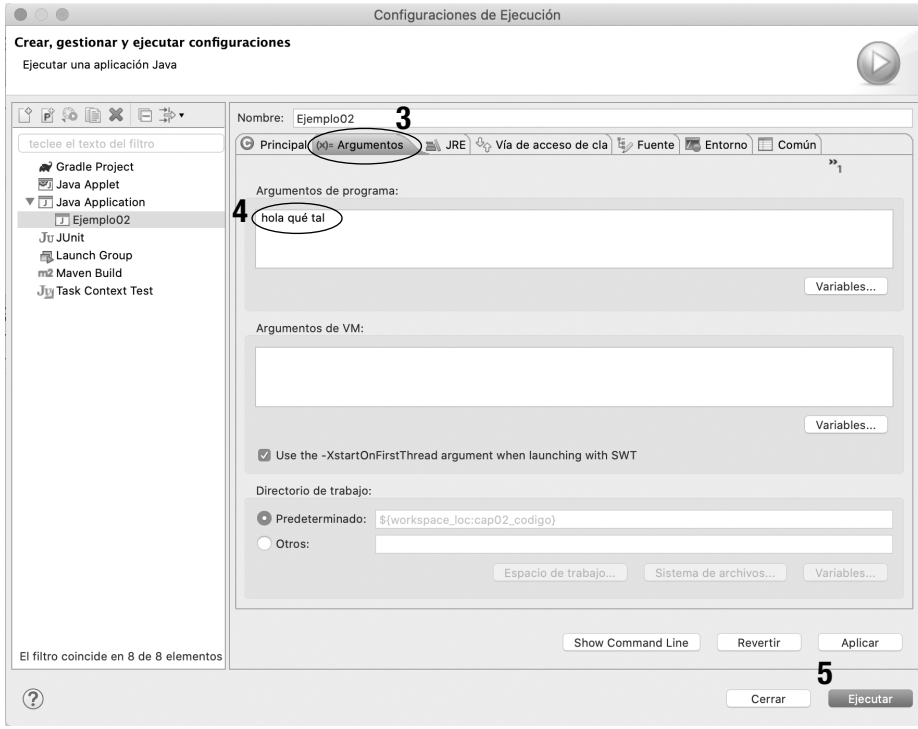


Figura 2.2. Establecer los argumentos de ejecución en Eclipse.

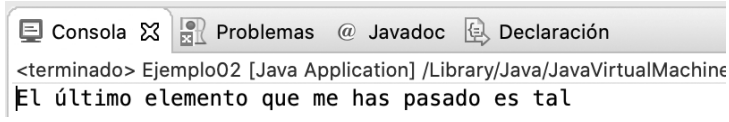


Figura 2.3. Resultado de ejecución con argumentos en Eclipse.

A partir de ahora, mientras no lo modifiques, cada vez que ejecutes esta clase, tendrás los argumentos establecidos.

T2.1
E2.1
E2.2