Introducción

Como cualquier lenguaje humano, Java proporciona una forma de expresar conceptos. Si tiene éxito, la expresión media será significativamente más sencilla y más flexible que las alternativas, a medida que los problemas crecen en tamaño y complejidad.

No podemos ver Java como una simple colección de características -algunas de las características no tienen sentido aisladas. Se puede usar la suma de partes sólo si se está pensando en ***diseño,*** y no simplemente en codificación. Y para entender Java así, hay que entender los problemas del lenguaje y de la programación en general. Este libro habla acerca de problemas de programación, por qué son problemas y el enfoque que Java sigue para solucionarlos. Por consiguiente, algunas características que explico en cada capítulo se basan en cómo yo veo que se ha solucionado algún problema en particular con el lenguaje. Así, espero conducir poco a poco al lector, hasta el punto en que Java se convierta en lengua casi materna.

Durante todo el tiempo, estaré tomando la actitud de que el lector construya un modelo mental que le permita desarrollar un entendimiento profundo del lenguaje; si se encuentra un puzzle se podrá alimentar de éste al modelo para tratar de deducir la respuesta.

Prerrequisitos

Este libro asume que se tiene algo de familiaridad con la programación: se entiende que un programa es una colección de sentencias, la idea de una subrutina/función/macro, sentencias de control como "ir' y bucles estilo "while", etc. Sin embargo, se podría haber aprendido esto en muchos sitios, como, por ejemplo, la programación con un lenguaje de macros o el trabajo con una herramienta como Perl. A medida que se programa hasta el punto en que uno se siente cómodo con las ideas básicas de programación, se podrá ir trabajando a través de este libro. Por supuesto, el libro será más fácil para los programadores de C y aún más para los de C++, pero tampoco hay por qué excluirse a sí mismo cuando se desconocen estos lenguajes (aunque en este caso es necesario tener la voluntad de trabajar duro; además, el CD multimedia que acompaña a este texto te permitirá conocer rápidamente los conceptos de la sintaxis de C necesarios para aprender Java). Presentaré los conceptos de la programación orientada a objetos (POO) y los mecanismos de control básicos de Java, para tener conocimiento de ellos, y los primeros ejercicios implicarán las secuencias de flujo de control básicas.

Aunque a menudo aparecerán referencias a aspectos de los lenguajes C y C++, no deben tomarse como comentarios profundos, sino que tratan de ayudar a los programadores a poner Java en perspectiva con esos lenguajes, de los que, después de todo, es de los que desciende Java. Intentaré hacer que estas referencias sean lo más simples posibles, y explicar cualquier cosa que crea que una persona que no haya programado nunca en C o C++ pueda desconocer.

Aprendiendo Java

Casi a la vez que mi primer libro ***Using*** C++ (Osborne/McGraw-Hill, 1989) apareció, empecé a enseñar ese lenguaje. Enseñar lenguajes de programación se ha convertido en mi profesión; he visto cabezas dudosas, caras en blanco y expresiones de puzzle en audiencias de todo el mundo desde 1989. A medida que empecé con formación ***in situ*** a grupos de gente más pequeños, descubrí algo en los ejercicios. Incluso aquéllos que sonreían tenían pegas con muchos aspectos. Al dirigir la sesión de C++ en la ***Software Development Conference*** durante muchos años (y después la sesión de Java), descubrí que tanto yo como otros oradores tendíamos a ofrecer a la audiencia, en general, muchos temas demasiado rápido. Por tanto, a través, tanto de la variedad del nivel de audiencia como de la forma de presentar el material, siempre se acababa perdiendo parte de la audiencia. Quizás es pedir demasiado, pero dado que soy uno de ésos que se resisten a las conferencias tradicionales (y en la mayoría de casos, creo que esta resistencia proviene del aburrimiento), quería intentar algo que permitiera tener a todo el mundo enganchado.

Durante algún tiempo, creé varias presentaciones diferentes en poco tiempo. Por consiguiente, acabé aprendiendo a base de experimentación e iteración (una técnica que también funciona bien en un diseño de un programa en Java). Eventualmente, desarrollé un curso usando todo lo que había aprendido de mi experiencia en la enseñanza -algo que me gustaría hacer durante bastante tiempo. Descompone el problema de aprendizaje en pasos discretos, fáciles de digerir, y en un seminario en máquina (la situación ideal de aprendizaje) hay ejercicios seguidos cada uno de pequeñas lecciones. Ahora doy cursos en seminarios públicos de Java, que pueden encontrarse en *http://www.BruceEckel.com*. (El seminario introductorio también está disponible como un CDROM. En el sitio web se puede encontrar más información al respecto.)

La respuesta que voy obteniendo de cada seminario me ayuda a cambiar y reenfocar el material hasta que creo que funciona bien como medio docente. Pero este libro no es simplemente un conjunto de notas de los seminarios -intenté empaquetar tanta información como pude en este conjunto de páginas, estructurándola de forma que cada tema te vaya conduciendo al siguiente. Más que otra cosa, el libro está diseñado para servir al lector solitario que se está enfrentando y dando golpes con un nuevo lenguaje de programación.

Objetivos

Como en mi libro anterior *Thinking in C++,* este libro pretende estar estructurado en torno al proceso de enseñanza de un lenguaje. En particular, mi motivación es crear algo que me proporcione una forma de enseñar el lenguaje en mis propios seminarios. Cuando pienso en un capítulo del libro, lo pienso en términos de lo que constituiría una buena lección en un seminario. Mi objetivo es lograr fragmentos que puedan enseñarse en un tiempo razonable, seguidos de ejercicios que sean fáciles de llevar a cabo en clase.

Mis objetivos en este libro son:

1. Presentar el material paso a paso de forma que se pueda digerir fácilmente cada concepto antes de avanzar.
2. Utilizar ejemplos que sean tan simples y cortos como se pueda. Esto evita en ocasiones acometer problemas del "mundo real", pero he descubierto que los principiantes suelen estar más contentos cuando pueden entender todos los detalles de un ejemplo que cuando se ven impresionados por el gran rango del problema que solucionan. Además, hay una limitación severa de cara a la cantidad de código que se puede absorber en una clase. Por ello, no dudaré en recibir críticas por usar "ejemplos de juguete", sino que estoy deseoso de aceptarlas en aras de lograr algo pedagógicamente útil.
3. Secuenciar cuidadosamente la presentación de características de forma que no se esté viendo algo que aún no se ha expuesto. Por supuesto, esto no es siempre posible; en esas situaciones se dan breves descripciones introductorias.
4. Dar lo que yo considero que es importante que se entienda del lenguaje, en lugar de todo lo que sé. Creo que hay una jerarquía de importancia de la información, y que hay hechos que el 95% de los programadores nunca necesitarán saber y que simplemente confunden a la gente y añaden su percepción de la complejidad del lenguaje. Por tomar un ejemplo de C, si se memoriza la tabla de precedencia de los operadores (algo que yo nunca hice) se puede escribir un código más inteligente. Pero si se piensa en ello, también confundirá la legibilidad y mantenibilidad de ese código. Por tanto, hay que olvidarse de la precedencia, y usar paréntesis cuando las cosas no estén claras.
5. Mantener cada sección lo suficientemente enfocada de forma que el tiempo de exposición - el tiempo entre periodos de ejercicios- sea pequeño. Esto no sólo mantiene más activas las mentes de la audiencia, que están en un seminario en máquina, sino que también transmite más sensación de avanzar.
6. Proporcionar una base sólida que permita entender los aspectos lo suficientemente bien como para avanzar a cursos y libros más difíciles.

Documentación en línea

El lenguaje Java y las bibliotecas de Sun Microsystems (de descarga gratuita) vienen con su documentación en forma electrónica, legible utilizando un navegador web, y casi toda implementación de Java de un tercero tiene éste u

otro sistema de documentación equivalente. Casi todos los libros publicados de Java, incorporan esta documentación. Por tanto, o ya se tiene, o se puede descargar, y a menos que sea necesario, este libro no repetirá esa documentación pues es más rápido encontrar las descripciones de las clases en el navegador web que buscarlas en un libro (y la documentación en línea estará probablemente más actualizada). Este libro proporcionará alguna descripción extra de las clases sólo cuando sea necesario para complementar la documentación, de forma que se pueda entender algún ejemplo particular.

Ejercicios

He descubierto que los ejercicios simples son excepcionalmente útiles para completar el entendimiento de los estudiantes durante un seminario, por lo que se encontrará un conjunto de ellos al final de cada capítulo.
La mayoría de ejercicios están diseñados para ser lo suficientemente sencillos como para poder ser resueltos en un tiempo razonable en una situación de clase mientras que observa el profesor, asegurándose de que todos los alumnos asimilen el material. Algunos ejercicios son más avanzados para evitar que los alumnos experimentados se aburran. La mayoría están diseñados para ser resueltos en poco tiempo y probar y pulir el conocimiento. Algunos suponen un reto, pero ninguno presenta excesivas dificultades. (Presumiblemente, cada uno podrá encontrarlos -o más probablemente te encontrarán ellos a ti.)

En el documento electrónico The Thinking in Java Annotated Solution Guide pueden encontrarse soluciones a ejercicios seleccionados, disponibles por una pequeña tasa en *http://www.BruceEckel.com*.

Código fuente

Todo el código fuente de este libro está disponible de modo gratuito sometido a copyright, distribuido como un paquete único, visitando el sitio web *http://www.BruceEckel.com*. Para asegurarse de obtener la versión más actual, éste es el lugar oficial para distribución del código y de la versión electrónica del libro. Se pueden encontrar versiones espejo del código y del libro en otros sitios (algunos de éstos están referenciados en *http://www.BruceEckel.com*), pero habría que comprobar el sitio oficial para asegurarse de obtener la edición más reciente. El código puede distribuirse en clases y en otras situaciones con fines educativos.

La meta principal del copyright es asegurar que el código fuente se cite adecuadamente, y prevenir que el código se vuelva a publicar en medios impresos sin permiso. (Mientras se cite la fuente, utilizando los ejemplos del libro, no habrá problema en la mayoría de los medios.)

En cada fichero de código fuente, se encontrará una referencia a la siguiente nota de copyright:

```
INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES,
INCLUDING LOST PROFITS, BUSINESS INTERRUPTION, LOSS OF
BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS, OR FOR
PERSONAL INJURIES, ARISING OUT OF THE USE OF THIS SOURCE
CODE AND ITS DOCUMENTATION, OR ARISING OUT OF THE INABILITY
TO USE ANY RESULTING PROGRAM, EVEN IF MINDVIEW, INC., OR
ITS PUBLISHER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE. MINDVIEW, INC. SPECIFICALLY DISCLAIMS ANY
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE. THE SOURCE CODE AND DOCUMENTATION PROVIDED
HEREUNDER IS ON AN "AS IS" BASIS, WITHOUT ANY ACCOMPANYING
SERVICES FROM MINDVIEW, INC., AND MINDVIEW, INC. HAS NO
OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES,
ENHANCEMENTS, OR MODIFICATIONS.

Please note that MindView, Inc. maintains a web site which
is the sole distribution point for electronic copies of the
Source Code, http://www.BruceEckel.com (and official mirror
sites), where it is freely available under the terms stated
above.

If you think you've found an error in the Source Code,
please submit a correction using the URL marked "feedback"
in the electronic version of the book, nearest the error
you've found.
```

El código puede usarse en proyectos y en clases (incluyendo materiales de presentación) mientras se mantenga la observación de ***copyright*** que aparece en cada archivo fuente.

Estándares de codificación

En el texto de este libro, los identificadores (nombres de funciones, variables y clases) están en **negrita.** La mayoría de palabras clave también están en negrita, excepto en aquellos casos en que las palabras se usan tanto que ponerlas en negrita podría volverse tedioso, como es el caso de la palabra "clase".

Para los ejemplos de este libro, uso un estilo de codificación bastante particular. Este estilo sigue al estilo que la propia Sun usa en prácticamente todo el código de sitio web (véase *http://java.sun.com/docs/codeconv/index.html),* y parece que esta soportado por la mayoría de entornos de desarrollo Java. Si ha leído el resto de mis trabajos, también verá que el estilo de codificación de Sun coincide con el mío -esto me alegra, aunque no tenía nada que hacer con él**.** El aspecto del estilo de formato es bueno para lograr horas de tenso debate, por lo que simplemente diré que no pretendo dictar un estilo correcto mediante mis ejemplos; tengo mi propia motivación para usar el estilo que uso. Java es un

lenguaje de programación de forma libre, se puede seguir usando cualquier estilo con el que uno esté a gusto.

Los programas de este libro son archivos incluidos por el procesador de textos, directamente sacados de archivos compilados. Por tanto, los archivos de código impresos en este libro deberían funcionar sin errores de compilador. Los errores que *deberían* causar mensajes de error en tiempo de compilación están comentados o marcados mediante **//!**, por lo que pueden ser descubiertos fácilmente, y probados utilizando medios automáticos. Los errores descubiertos de los que ya se haya informado al autor, aparecerán primero en el código fuente distribuido y posteriormente en actualizaciones del libro (que también aparecerán en el sitio web *http://www.BruceEckel.com*).

Versiones de Java

Generalmente confío en la implementación que Sun hace de Java como referencia para definir si un determinado comportamiento es o no correcto.

Con el tiempo, Sun ha lanzado tres versiones principales de Java: la 1.0, la 1.1 y la 2 (que se llama versión 2, incluso aunque las versiones del JDK de Sun siguen usando el esquema de numeración de 1.2, 1.3, 1.4, etc.). La versión 2 parece llevar finalmente a Java a la gloria, especialmente en lo que concierne a las herramientas de interfaces. Este libro se centra en, y está probado con, Java 2, aunque en ocasiones hago concesiones a las características anteriores de Java 2, de forma que el código pueda compilarse bajo Linux (vía el JDK de Linux que estaba disponible en el momento de escribir el libro).

Si se necesita aprender versiones anteriores del lenguaje no cubiertas en esta edición, la primera edición del libro puede descargarse gratuitamente de http://www.BruceEckel.corn, y también está en el CD adjunto a este libro.

Algo de lo que uno se dará cuenta es que, cuando menciono versiones anteriores del lenguaje, no uso los números de sub-revisión. En este libro me referiré sólo a Java 1.0, 1.1 y 2, para protegerme de errores tipográficos producidos por sub-revisiones posteriores de estos productos.

Errores

Sin que importe cuántos trucos utiliza un escritor para detectar errores, siempre hay alguno que se queda ahí y que algún lector encontrará.

Hay un formulario para remitir errores al principio de cada capítulo en la versión HTML del libro (y en el CD ROM unido al final de este libro, además de descargable de http://www.BruceEckel.corn) y también en el propio sitio web, en

la página correspondiente a este libro. Si se descubre algo que uno piense que puede ser un error, por favor, utilice el formulario para remitir el error junto con la corrección sugerida. Si es necesario, incluya el archivo de código fuente original y cualquier modificación que se sugiera. Su ayuda será apreciada.

Capítulos

Este libro se diseñó con una idea en la cabeza: la forma que tiene la gente de aprender Java. La realimentación de la audiencia de mis seminarios me ayudó a ver las partes difíciles que necesitaban aclaraciones. En las áreas en las que me volvía ambiguo e incluía varias características a la vez, descubrí -a través del proceso de presentar el material- que si se incluyen muchas características de golpe, hay que explicarlas todas, y esto suele conducir fácilmente a la confusión por parte del alumno. Como resultado, he tenido bastantes problemas para presentar las características agrupadas de tan pocas en pocas como me ha sido posible.

El objetivo, por tanto, es que cada capítulo enseñe una única característica, o un pequeño grupo de características asociadas, sin pasar a características adicionales. De esa forma se puede digerir cada fragmento en el contexto del conocimiento actual antes de continuar.

He aquí una breve descripción de los capítulos que contiene el libro, que corresponde a las conferencias y periodos de ejercicio en mis seminarios en máquina.

Capítulo 1: Introducción a los objetos

Este capítulo presenta un repaso de lo que es la programación orientada a objetos, incluyendo la respuesta a la cuestión básica "¿Qué es un objeto?", interfaz frente a implementación, abstracción y encapsulación, mensajes y funciones, herencia y composición, y la importancia del polimorfismo.

También se obtendrá un repaso a los aspectos de la creación de objetos como los constructores, en los que residen los objetos, dónde ponerlos una vez creados, y el mágico recolector de basura que limpia los objetos cuando dejan de ser necesarios. Se presentarán otros aspectos, incluyendo el manejo de errores con excepciones, el multihilo para interfaces de usuario con buen grado de respuesta, y las redes e Internet. Se aprenderá qué convierte a Java en especial, por qué ha tenido tanto éxito, y también algo sobre análisis y diseño orientado a objetos.

Capítulo 2: Todo es un objeto

Este capítulo te lleva al punto donde tú puedas crear el primer programa en Java, por lo que debe dar un repaso a lo esencial, incluyendo el concepto de ***referencia*** a un objeto; cómo crear un objeto; una introducción de los tipos primitivos y arrays; el alcance y la forma en que destruye los objetos el recolector de basura; cómo en Java todo es un nuevo tipo de datos (clase) y cómo crear cada uno sus propias clases; funciones, argumentos y valores de retorno; visibilidad de nombres y el uso de componentes de otras bibliotecas; la palabra clave **static;** y los comentarios y documentación embebida.

Capítulo 3: Controlando el flujo de los programas

Este capítulo comienza con todos los operadores que provienen de C y C++. Además, se descubrirán los fallos de los operadores comunes, la conversión de tipos, la promoción y la precedencia. Después se presentan las operaciones básicas de control de flujo y selección existentes en casi todos los lenguajes de programación: la opción con **if**-**else**; los bucles con **while** y **for**; cómo salir de un bucle con **break** y **continue**, además de sus versiones etiquetadas en Java (que vienen a sustituir al "**goto perdido**" en Java); la selección con **switch**. Aunque gran parte de este material tiene puntos comunes con el código de C y C++, hay algunas diferencias. Además, todos los ejemplos estarán hechos completamente en Java por lo que el lector podrá estar más a gusto con la apariencia de Java.

Capítulo 4: Inicialización y limpieza

Este capítulo comienza presentando el constructor, que garantiza una inicialización adecuada. La definición de constructor conduce al concepto de sobrecarga de funciones (puesto que puede haber varios constructores). Éste viene seguido de una discusión del proceso de limpieza, que no siempre es tan simple como parece. Normalmente, simplemente se desecha un objeto cuando se ha acabado con él y el recolector de basura suele aparecer para liberar la memoria. Este apartado explora el recolector de basura y algunas de sus idiosincrasias. El capítulo concluye con un vistazo más cercano a cómo se inicializan las cosas: inicialización automática de miembros, especificación de inicialización de miembros, el orden de inicialización, la inicialización **static** y la inicialización de arrays.

Capítulo 5: Ocultando la implementación

Este capítulo cubre la forma de empaquetar junto el código, y por qué algunas partes de una biblioteca están expuestas a la vez que otras partes están ocultas. Comienza repasando las palabras clave **package** e **import**, que llevan a cabo empaquetado a nivel de archivo y permiten construir bibliotecas de clases. Después examina el tema de las rutas de directorios y nombres de fichero. El resto del capítulo echa un vistazo a las palabras clave **public**, **private** y

protected, el concepto de acceso "**friendly**", y qué significan los distintos niveles de control de acceso cuando se usan en los distintos conceptos.

Capítulo 6: Reutilizando clases

El concepto de herencia es estándar en casi todos los lenguajes de POO. Es una forma de tomar una clase existente y añadirla a su funcionalidad (además de cambiarla, que será tema del Capítulo 7). La herencia es a menudo una forma de reutilizar código dejando igual la "clase base", y simplemente parcheando los elementos aquí y allí hasta obtener lo deseado. Sin embargo, la herencia no es la única forma de construir clases nuevas a partir de las existentes. También se puede empotrar un objeto dentro de una clase nueva con la composición. En este capítulo, se aprenderán estas dos formas de reutilizar código en Java, y cómo aplicarlas.

Capítulo 7: Polimorfismo

Cada uno por su cuenta, podría invertir varios meses para descubrir y entender el polimorfismo, claves en POO. A través de pequeños ejemplos simples, se verá cómo crear una familia de tipos con herencia y manipular objetos de esa familia a través de su clase base común. El polimorfismo de Java permite tratar los objetos de una misma familia de forma genérica, lo que significa que la mayoría del código no tiene por qué depender de un tipo de información específico. Esto hace que los programas sean extensibles, por lo que se facilita y simplifica la construcción de programas y el mantenimiento de código.

Capítulo 8: Interfaces y clases internas

Java proporciona una tercera forma de establecer una relación de reutilización a través de la ***interfaz,*** que es una abstracción pura del interfaz de un objeto. La **interfaz** es más que una simple clase abstracta llevada al extremo, puesto que te permite hacer variaciones de la "herencia múltiple" de C++, creando una clase sobre la que se puede hacer una conversión hacia arriba a más de una clase base.

A primera vista, las clases parecen un simple mecanismo de ocultación de código: se colocan clases dentro de otras clases. Se aprenderá, sin embargo, que la clase interna hace más que eso – conoce y puede comunicarse con la clase contenedora – y que el tipo de código que se puede escribir con clases internas es más elegante y limpio, aunque es un concepto nuevo para la mayoría de la gente y lleva tiempo llegar a estar cómodo utilizando el diseño clases internas.

Capítulo 9: Manejo de errores con excepciones

La filosofía básica de Java es que el código mal formado no se ejecutará. En la medida en que sea posible, el compilador detecta problemas, pero en ocasiones los problemas -debidos a errores del programador o a condiciones de error naturales que ocurren como parte de la ejecución normal del programa- pueden detectarse y ser gestionados sólo en tiempo de ejecución. Java tiene el ***manejo de excepciones*** para tratar todos los problemas que puedan surgir al ejecutar el programa. Este capítulo muestra cómo funcionan en Java las palabras clave **try**, **catch**, **throw**, **throws** y **finally**; cuándo se deberían lanzar excepciones y qué hacer al capturarlas. Además, se verán las excepciones estándar de Java, cómo crear las tuyas propias, qué ocurre con las excepciones en los constructores y cómo se ubican los gestores de excepciones.

Capítulo 10: Identificación de tipos en tiempo de ejecución

La identificación de tipos en tiempo de ejecución (RTTI) te permite averiguar el tipo exacto de un objeto cuando se tiene sólo una referencia al tipo base. Normalmente, se deseará ignorar intencionadamente el tipo exacto de un objeto y dejar que sea el mecanismo de asignación dinámico de Java (polimorfismo) el que implemente el comportamiento correcto para ese tipo. A menudo, esta información te permite llevar a cabo operaciones de casos especiales, más eficientemente. Este capítulo explica para qué existe la RTTI, cómo usarlo, y cómo librarse de él cuando sobra. Además, este capítulo presenta el mecanismo de reflectividad de Java.

Capítulo 11: Guardando tus objetos

Es un programa bastante simple que sólo tiene una cantidad fija de objetos de tiempo de vida conocido. En general, todos los programas irán creando objetos nuevos en distintos momentos, conocidos sólo cuando se está ejecutando el programa. Además, no se sabrá hasta tiempo de ejecución la cantidad o incluso el tipo exacto de objetos que se necesitan. Para solucionar el problema de programación general, es necesario crear cualquier número de objetos, en cualquier momento y en cualquier lugar. Este capítulo explora en profundidad la biblioteca de contenedores que proporciona Java 2 para almacenar objetos mientras se está trabajando con ellos: los simples arrays y contenedores más sofisticados (estructuras de datos) como **ArrayList** y **HashMap**.

Capítulo 12: El sistema de E/S de Java

Teóricamente, se puede dividir cualquier programa en tres partes: entrada,

proceso y salida. Esto implica que la E/S (entrada/salida) es una parte importante de la ecuación. En este capítulo se aprenderá las distintas clases que proporciona Java para leer y escribir ficheros, bloques de memoria y la consola. También se mostrará la distinción entre E/S "antigua" y "nueva". Además, este capítulo examina el proceso de tomar un objeto, pasarlo a una secuencia de bytes (de forma que pueda ser ubicado en el disco o enviado a través de una red) y reconstruirlo, lo que realiza automáticamente la serialización de objetos de Java. Además, se examinan las bibliotecas de compresión de Java, que se usan en el formato de archivos de Java CJAR).

Capítulo 13: Concurrencia

Java proporciona una utilidad preconstruida para el soporte de múltiples subtareas concurrentes denominadas hilos, que se ejecutan en un único programa. (A menos que se disponga de múltiples procesadores en la máquina, los múltiples hilos sólo son aparentes.) Aunque éstas pueden usarse en todas partes, los hilos son más lucidos cuando se intenta crear una interfaz de usuario con alto grado de respuesta, de forma que, por ejemplo, no se evita que un usuario pueda presionar un botón o introducir datos mientras se está llevando a cabo algún procesamiento. Este capítulo echa un vistazo a la sintaxis y la semántica del multihilo en Java.

Capítulo 14: Creación de ventanas y applets

Java viene con la biblioteca IGU Swing, que es un conjunto de clases que manejan las ventanas de forma portable. Estos programas con ventanas pueden o bien ser applets o bien aplicaciones independientes. Este capítulo es una introducción a Swing y a la creación de applets de World Wide Web. Se presenta la importante tecnología de los "JavaBeans", fundamental para la creación de herramientas de construcción de programas de Desarrollo Rápido de Aplicaciones **(RAD)**.

Capítulo 15: Descubriendo Problemas

Los mecanismos de comprobación de lenguaje nos pueden tomar sólo en lo que va de nuestra búsqueda para desarrollar un programa que trabaja correctamente. Este capítulo presenta herramientas para solucionar los problemas que el compilador no soluciona. Una de los pasos más grandes adelante es la incorporación de prueba de unidades automatizada. Para este libro, un sistema personalizado de prueba fue desarrollado para asegurar la exactitud de la salida de programa, pero el sistema de pruebas del **JUnit** del estándar de defacto es también introducido. La construcción automática es implementada con la herramienta del estándar de la fuente abierta **Ant**, y para el trabajo de equipo, los fundamentos de CVS son explicados. Para información de problema durante la

corrida, este capítulo introduce al mecanismo de aserción Java (mostrado aquí usado con Diseño por contrato), la API de registro, depuradores, los perfiladores, y hasta doclets (el cual puede ayudar a descubrir problemas en código fuente).

Capítulo 16: Computación distribuida

Todas las características y bibliotecas de Java aparecen realmente cuando se empieza a escribir programas que funcionen en red. Este capítulo explora la comunicación a través de redes e Internet, y las clases que proporciona Java para facilitar esta labor. Presenta los tan importantes conceptos de Servlets y JSP (para programación en el lado servidor), junto con Java DataBase Connectivity (JDBC) y el Remote Method Invocation (RMI). Finalmente, hay una introducción a las nuevas tecnologías de JINI, JavaSpaces, y Enterprise JavaBeans (EJBS).

Capítulo 17: Patrones de Diseño

Este capítulo introduce los acercamientos de patrones muy importantes y todavía poco tradicionales para diseño de programas. Un ejemplo del proceso de evolución del diseño es estudiado, comenzando con una solución inicial y moviéndose a través de la lógica y el proceso de desarrollar la solución para los diseños más correctos. Verás una forma en la que un diseño puede materializarse con el paso del tiempo.

Capítulo 18: Proyectos

Este capítulo incluye un conjunto de proyectos que se construyen en el material presentado en este libro, o de otra manera se adecuaría en los capítulos anteriores. Estos proyectos son significativamente más complicados que los ejemplos en el resto de libro, y a menudo demuestran usos y técnicas nuevas de librerías de clase.

Hay temas que no parecen ajustarte dentro del núcleo del libro, y todavía me encuentro con que los discuto durante los seminarios.

Capítulo 19: Análisis y Diseño

El paradigma orientado a objetos es una forma de pensar nueva y diferente acerca de programar, y muchas personas tienen problemas al principio sabiendo cómo acercarse un proyecto OOP. Una vez que entiendes el concepto de un objeto, y como aprendes a pensar más en un estilo orientado a objetos, puedes comenzar a crear "buenos" diseños que se aprovechan de todos los beneficios que OOP tiene que ofrecer. Este capítulo introduce las ideas de análisis, diseño, y algunas formas para acercarse los problemas de desarrollar buenos programas orientados a objetos en una cantidad razonable de tiempo. Los temas incluyen

diagramas *Unified Modeling Language* (UML) y metodología asociada, casos de uso, cartas de *Class-Responsibility-Collaboration* (código de redundancia cíclica), desarrollo iterativo, Programación Extrema (XP), formas a desarrollar y desarrollar código reusable, y estrategias para la transición para la programación orientada a objetos.

Apéndice A: Paso y retorno de objetos

Puesto que la única forma de hablar con los objetos en Java es mediante referencias, los conceptos de paso de objetos a una función y de devolución de un objeto de una función tienen algunas consecuencias interesantes. Este apéndice explica lo que es necesario saber para gestionar objetos cuando se está entrando y saliendo de funciones, y también muestra la clase **String,** que usa un enfoque distinto al problema.

Apéndice B: La Interfaz Nativa de Java (JNI)

Un programa Java totalmente portable tiene importantes pegas: la velocidad y la incapacidad para acceder a servicios específicos de la plataforma. Cuando se conoce la plataforma sobre la que está ejecutando, es posible incrementar dramáticamente la velocidad de ciertas operaciones construyéndolas como métodos nativos, que son funciones escritas en otro lenguaje de programación (actualmente, sólo están soportados C/C++). Este apéndice da una introducción más que satisfactoria que debería ser capaz de crear ejemplos simples que sirvan de interfaz con código no Java.

Apéndice C: Guías de programación Java

Este apéndice contiene sugerencias para guiarle durante la realización del diseño de programas de bajo nivel y la escritura de código.

Apéndice D: Comparando C++ y Java

Si eres programador de C++, ya tienes la idea básica de programación orientada a objetos, y la sintaxis de Java sin duda se verá muy familiar para ti. Esto tiene sentido porque Java fue derivada de C++. Sin embargo, hay un número sorprendente de diferencias entre C++ y Java. Estas diferencias están dirigidas a ser mejoras significativas, y si entiendes las diferencias verás el por qué Java es un lenguaje de programación tan beneficioso. Este apéndice te conduce por las características importantes que hacen a Java distinto de C++.

Apéndice E: Rendimiento

Esto te permitirá encontrar cuellos de botella y mejorar la velocidad en tu programa Java.

Apéndice F: Un poco sobre el Recolector de Basura

Este apéndice describe la operación y los acercamientos que se usan para implementar el recolector de basura.

Apéndice G: Suplementos

Las descripciones de material de aprendizaje adicional disponible de *MindView*:

1. El CD-ROM que está en la parte de atrás de este libro, que contiene los Fundamentos para el seminario Java en CD, para prepararte para este libro.
2. El CD-ROM Hands On Java, Edición 3, disponible en www.MindView.net. Un seminario en CD que está basado en el material en este libro.
3. The Thinking In Java Seminar. El MindView, Inc., principal seminario introductorio basado en el material en este libro. El horario y las páginas de inscripción pueden ser encontrados en www.MindView.net.
4. Thinking in Enterprise Java, un libro que cubre temas Java más adelantados apropiados para la programación de la empresa. Disponible en www.MindView.net.
5. The J2EE Seminar. Te inicia en el desarrollo práctico de aplicaciones del mundo real, habilitadas por medio de la Internet, distribuidas con Java. Vea www.MindView.net.
6. Designing Objects && Systems Seminar. El análisis orientado a objetos, el diseño, y las técnicas de implementación. Vea www.MindView.net.
7. Thinking in Patterns (con Java), que cubre temas Java más avanzados en patrones de diseño y técnicas de resoluciones de problemas. Disponible en www.MindView.net.
8. Thinking In Patterns Seminar. Un seminario en vivo basado en el libro de arriba. El horario y las páginas de inscripción pueden ser encontrados en www.MindView.net.
9. Design Consulting And Reviews. La asistencia para ayudar a conservar tu proyecto en buena forma.

Apéndice H: Recursos

Una lista de algunos libros sobre Java que he encontrado particularmente útil.