| Lenguaje de programación | Descripción |
|---|---|
| Ruby on Rails | Ruby, que se creó a mediados de la década de 1990, es un lenguaje de programación orientado a objetos de código fuente abierto, con una sintaxis simple que es similar a Python. Ruby on Rails combina el lenguaje de secuencias de comandos Ruby con el marco de trabajo de aplicaciones Web Rails, desarrollado por 37Signals. Su libro, *Getting Real* (`gettingreal.37signals.com/toc.php`), es una lectura obligatoria para los desarrolladores Web. Muchos desarrolladores de Ruby on Rails han reportado ganancias de productividad superiores a las de otros lenguajes, al desarrollar aplicaciones Web que trabajan de manera intensiva con bases de datos. |

**Fig. 1.5** | Otros lenguajes de programación (parte 3 de 3).

## 1.8 Java

La contribución más importante a la fecha de la revolución del microprocesador es que hizo posible el desarrollo de las computadoras personales. Los microprocesadores han tenido un profundo impacto en los dispositivos electrónicos inteligentes para uso doméstico. Al reconocer esto, Sun Microsystems patrocinó en 1991 un proyecto interno de investigación corporativa dirigido por James Gosling, que dio como resultado un lenguaje de programación orientado a objetos y basado en C++, al que Sun llamó Java.

Un objetivo clave de Java es poder escribir programas que se ejecuten en una gran variedad de sistemas computacionales y dispositivos controlados por computadora. A esto se le conoce algunas veces como "escribir una vez, ejecutar en cualquier parte".

La popularidad del servicio Web explotó en 1993; en ese entonces Sun vio el potencial de usar Java para agregar *contenido dinámico*, como interactividad y animaciones, a las páginas Web. Java atrajo la atención de la comunidad de negocios debido al fenomenal interés en el servicio Web. En la actualidad, Java se utiliza para desarrollar aplicaciones empresariales a gran escala, para mejorar la funcionalidad de los servidores Web (las computadoras que proporcionan el contenido que vemos en nuestros navegadores Web), para proporcionar aplicaciones para los dispositivos de uso doméstico (como teléfonos celulares, teléfonos inteligentes, receptores de televisión por Internet y mucho más) y para muchos otros propósitos. Java también es el lenguaje clave para desarrollar aplicaciones para teléfonos inteligentes y tabletas de Android. En 2010, Oracle adquirió Sun Microsystems.

### *Bibliotecas de clases de Java*

Usted puede crear cada clase y método que necesite para formar sus programas de Java. Sin embargo, la mayoría de los programadores en Java aprovechan las ricas colecciones de clases y métodos existentes en las **bibliotecas de clases de Java**, que también se conocen como **API (Interfaces de programación de aplicaciones) de Java**.

**Tip de rendimiento 1.1**

*Si utiliza las clases y métodos de las API de Java en vez de escribir sus propias versiones, puede mejorar el rendimiento de sus programas, ya que estas clases y métodos están escritos de manera cuidadosa para funcionar con eficiencia. Esta técnica también reduce el tiempo de desarrollo de los programas.*

## 1.9 Un típico entorno de desarrollo en Java

Ahora explicaremos los pasos típicos utilizados para crear y ejecutar una aplicación en Java. Por lo general hay cinco fases: edición, compilación, carga, verificación y ejecución. Hablaremos sobre estos conceptos

en el contexto del Kit de desarrollo de Java (JDK) SE 8. Lea la sección *Antes de empezar de este libro para obtener información acerca de cómo descargar e instalar el JDK en Windows, Linux y OS X.*

*Fase 1: Creación de un programa*
La fase 1 consiste en editar un archivo con un *programa de edición*, conocido comúnmente como *editor* (figura 1.6). A través del editor, usted escribe un programa en Java (a lo cual se le conoce por lo general como **código fuente**), realiza las correcciones necesarias y guarda el programa en un dispositivo de almacenamiento secundario, como su disco duro. Los archivos de código fuente de Java reciben un nombre que termina con la **extensión `.java`**, lo que indica que éste contiene código fuente en Java.

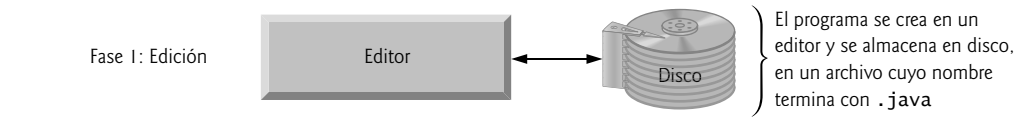


**Fig. 1.6** | Entorno de desarrollo típico de Java: fase de edición.

Dos de los editores muy utilizados en sistemas Linux son `vi` y `emacs`. Windows cuenta con el **`Bloc de Notas`**. OS X ofrece **`TextEdit`**. También hay muchos editores de freeware y shareware disponibles en línea, como Notepad++ (`notepad-plus-plus.org`), Edit-Plus (`www.editplus.com`), TextPad (`www.textpad.com`) y jEdit (`www.jedit.org`).

Los **entornos de desarrollo integrados (IDE)** proporcionan herramientas que dan soporte al proceso de desarrollo del software, entre las que se incluyen editores, depuradores para localizar **errores lógicos** (errores que provocan que los programas se ejecuten en forma incorrecta) y más. Hay muchos IDE de Java populares, como

- Eclipse (`www.eclipse.org`)
- NetBeans (`www.netbeans.org`)
- IntelliJ IDEA (`www.jetbrains.com`)

En el sitio Web del libro en

```
www.deitel.com/books/jhtp10
```

proporcionamos videos Dive-Into® que le muestran cómo ejecutar las aplicaciones de Java de este libro y cómo desarrollar nuevas aplicaciones de Java con Eclipse, NetBeans e IntelliJ IDEA.

*Fase 2: Compilación de un programa en Java para convertirlo en códigos de bytes*
En la fase 2, el programador utiliza el comando **`javac`** (el **compilador de Java**) para **`compilar`** un programa (figura 1.7). Por ejemplo, para compilar un programa llamado `Bienvenido.java`, escriba

```
javac Bienvenido.java
```

en la ventana de comandos de su sistema (es decir, el **Símbolo del sistema** de Windows, o la aplicación **Terminal** en OS X) o en un shell de Linux (que también se conoce como **Terminal** en algunas versiones de Linux). Si el programa se compila, el compilador produce un archivo **`.class`** llamado `Bienvenido.class` que contiene la versión compilada del programa. Por lo general los IDE proveen un elemento de menú, como **Build** (Generar) o **Make** (Crear), que invoca el comando `javac` por usted. Si el compilador detecta errores, tendrá que ir a la fase 1 y corregirlos. En el capítulo 2 hablaremos más sobre los tipos de errores que el compilador puede detectar.

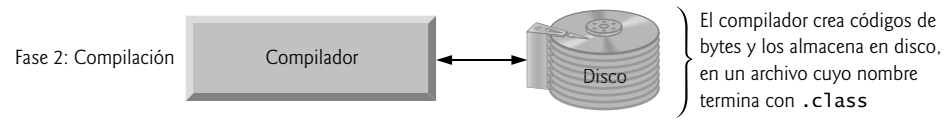


**Fig. 1.7** | Entorno de desarrollo típico de Java: fase de compilación.

El compilador de Java traduce el código fuente de Java en **códigos de bytes** que representan las tareas a ejecutar en la fase de ejecución (fase 5). La **Máquina Virtual de Java (JVM)**, que forma parte del JDK y es la base de la plataforma Java, ejecuta los códigos de bytes. Una **máquina virtual (VM)** es una aplicación de software que simula a una computadora, pero oculta el sistema operativo y el hardware subyacentes de los programas que interactúan con ésta. Si se implementa la misma VM en muchas plataformas computacionales, las aplicaciones escritas para ese tipo de VM se podrán utilizar en todas esas plataformas. La JVM es una de las máquinas virtuales más utilizadas en la actualidad. La plataforma .NET de Microsoft utiliza una arquitectura de máquina virtual similar.

A diferencia de las instrucciones de lenguaje máquina, que *dependen de la plataforma* (es decir, dependen del hardware de una computadora específica), los códigos de bytes son instrucciones *independientes de la plataforma*. Por lo tanto, los códigos de bytes de Java son **portables**; es decir, se pueden ejecutar los mismos códigos de bytes en cualquier plataforma que contenga una JVM que incluya la versión de Java en la que se compilaron los códigos de bytes sin necesidad de volver a compilar el código fuente. La JVM se invoca mediante el comando **`java`**. Por ejemplo, para ejecutar una aplicación en Java llamada `Bienvenido`, debe escribir el comando

```
java Bienvenido
```

en una ventana de comandos para invocar la JVM, que a su vez inicia los pasos necesarios para ejecutar la aplicación. Esto comienza la fase 3. Por lo general los IDE proporcionan un elemento de menú, como **`Run`** (**`Ejecutar`**), que invoca el comando `java` por usted.

*Fase 3: Carga de un programa en memoria*

En la fase 3, la JVM coloca el programa en memoria para ejecutarlo; a esto se le conoce como **carga** (figura 1.8). El **cargador de clases** de la JVM toma los archivos `.class` que contienen los códigos de bytes del programa y los transfiere a la memoria principal. El cargador de clases también carga cualquiera de los archivos `.class` que su programa utilice, y que sean proporcionados por Java. Puede cargar los archivos `.class` desde un disco en su sistema o a través de una red (como la de su universidad local o la red de la empresa, o incluso desde Internet).

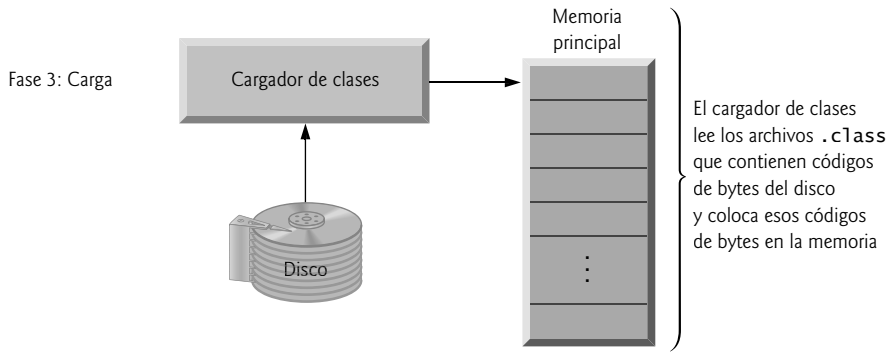


**Fig. 1.8** | Entorno de desarrollo típico de Java: fase de carga.

*Fase 4: Verificación del código de bytes*
En la fase 4, a medida que se cargan las clases, el **verificador de códigos de bytes** examina sus códigos de bytes para asegurar que sean válidos y que no violen las restricciones de seguridad de Java (figura 1.9). Java implementa una estrecha seguridad para asegurar que los programas en Java que llegan a través de la red no dañen sus archivos o su sistema (como podrían hacerlo los virus de computadora y los gusanos).

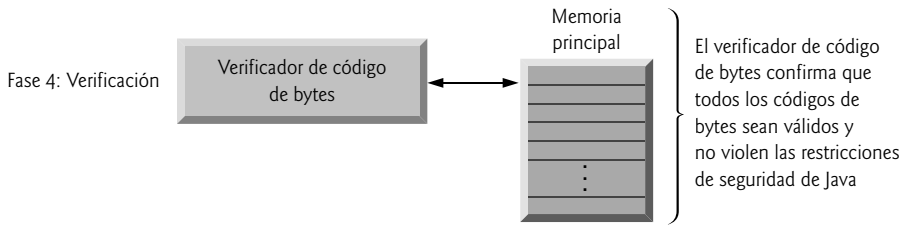


**Fig. 1.9** | Entorno de desarrollo típico de Java: fase de verificación.

*Fase 5: Ejecución*
En la fase 5, la JVM **ejecuta** los códigos de bytes del programa, realizando así las acciones especificadas por el mismo (figura 1.10). En las primeras versiones de Java, la JVM era tan sólo un *intérprete* de códigos de bytes de Java. Esto hacía que la mayoría de los programas se ejecutaran con lentitud, ya que la JVM tenía que interpretar y ejecutar un código de bytes a la vez. Algunas arquitecturas de computadoras modernas pueden ejecutar varias instrucciones en paralelo. Por lo general, las JVM actuales ejecutan códigos de bytes mediante una combinación de la interpretación y la denominada **compilación justo a tiempo (JIT)**. En este proceso, la JVM analiza los códigos de bytes a medida que se interpretan, en busca de *puntos activos* (partes de los códigos de bytes que se ejecutan con frecuencia). Para estas partes, un **compilador justo a tiempo (JIT)**, como el **compilador HotSpot™ de Java** de Oracle, traduce los códigos de bytes al lenguaje máquina correspondiente de la computadora. Cuando la JVM vuelve a encontrar estas partes compiladas, se ejecuta el código en lenguaje máquina, que es más rápido. Por ende, los programas en Java en realidad pasan por *dos* fases de compilación: una en la cual el código fuente se traduce a código de bytes (para tener portabilidad a través de las JVM en distintas plataformas computacionales) y otra en la que, durante la ejecución los *códigos de bytes* se traducen en *lenguaje máquina* para la computadora actual en la que se ejecuta el programa.

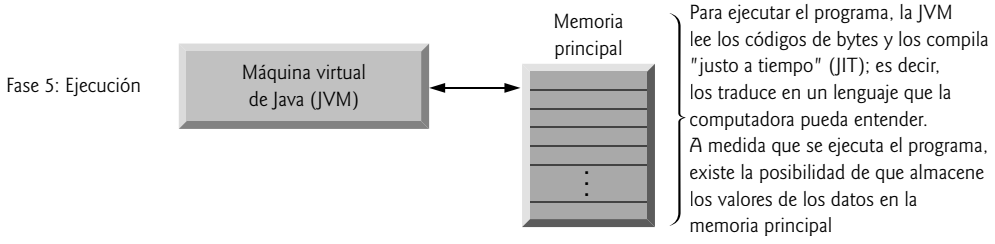


**Fig. 1.10** | Entorno de desarrollo típico de Java: fase de ejecución.

*Problemas que pueden ocurrir en tiempo de ejecución*
Es probable que los programas no funcionen la primera vez. Cada una de las fases anteriores puede fallar, debido a diversos errores que describiremos en este libro. Por ejemplo, un programa en ejecución podría

intentar una división entre cero (una operación ilegal para la aritmética con números enteros en Java). Esto haría que el programa de Java mostrara un mensaje de error. Si esto ocurre, tendría que regresar a la fase de edición, hacer las correcciones necesarias y proseguir de nuevo con las fases restantes, para determinar que las correcciones hayan resuelto el o los problemas [*nota:* la mayoría de los programas en Java reciben o producen datos. Cuando decimos que un programa muestra un mensaje, por lo general queremos decir que muestra ese mensaje en la pantalla de su computadora. Los mensajes y otros datos pueden enviarse a otros dispositivos, como los discos y las impresoras, o incluso a una red para transmitirlos a otras computadoras].

**Error común de programación 1.1**

*Los errores, como la división entre cero, ocurren a medida que se ejecuta un programa, de manera que a estos errores se les llama* ***errores en tiempo de ejecución****. Los* ***errores fatales en tiempo de ejecución*** *hacen que los programas terminen de inmediato, sin haber realizado bien su trabajo. Los* ***errores no fatales en tiempo de ejecución*** *permiten a los programas ejecutarse hasta terminar su trabajo, lo que a menudo produce resultados incorrectos.*

## 1.10 Prueba de una aplicación en Java

En esta sección ejecutará su primera aplicación en Java e interactuará con ella. La aplicación **Painter**, que creará en el transcurso de varios ejercicios, le permite arrastrar el ratón para "dibujar". Los elementos y la funcionalidad que podemos ver en esta aplicación son típicos de lo que aprenderá a programar en este libro. Mediante el uso de la interfaz gráficos de usuario (GUI) de **Painter**, usted puede controlar el color de dibujo, la forma a dibujar (línea, rectángulo u óvalo) y si la forma se debe llenar o no con un color. También puede deshacer la última forma que agregó al dibujo o borrarlo todo [*nota:* utilizamos fuentes para diferenciar las diversas características. Nuestra convención es enfatizar las características de la pantalla como los títulos y menús (por ejemplo, el menú **Archivo**) en una fuente **Helvetica sans-serif** en negritas, y enfatizar los elementos que no son de la pantalla, como los nombres de archivo, código del programa o los datos de entrada (como `NombrePrograma.java`) en una fuente `Lucida sans-serif`].

Los pasos en esta sección le muestran cómo ejecutar la aplicación **Painter** desde una ventana **Símbolo del sistema** (Windows), **Terminal** (OS X) o shell (Linux) de su sistema. A lo largo del libro nos referiremos a estas ventanas simplemente como *ventanas de comandos*. Realice los siguientes pasos para usar la aplicación **Painter** para dibujar una cara sonriente:

1. ***Revise su configuración.*** Lea la sección Antes de empezar este libro para confirmar que haya instalado Java de manera apropiada en su computadora, que haya copiado los ejemplos del libro en su disco duro y que sepa cómo abrir una ventana de comandos en su sistema.

2. ***Cambie al directorio de la aplicación completa.*** Abra una ventana de comandos y use el comando `cd` para cambiar al directorio (también conocido como *carpeta*) de la aplicación **Painter**. Vamos a suponer que los ejemplos del libro se encuentran en `C:\ejemplos` en Windows o en la carpeta `Documents/ejemplos` en Linux o en OS X. En Windows escriba `cd C:\ejemplos\cap01\painter` y después oprima *Intro*. En Linux u OS X escriba `cd ~/Documents/ejemplos/cap01/painter` y después oprima *Intro*.

3. ***Ejecute la aplicación Painter.*** Recuerde que el comando `java`, seguido del nombre del archivo `.class` de la aplicación (en este caso, `Painter`), ejecuta la aplicación. Escriba el comando `java Painter` y oprima *Intro* para ejecutar la aplicación. La figura 1.11 muestra la aplicación en ejecución en Windows, Linux y OS X, respectivamente; redujimos el tamaño de las ventanas para ahorrar espacio.