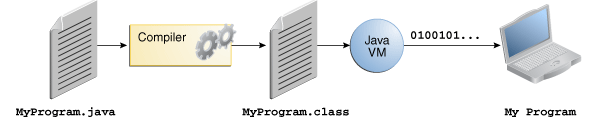
**El lenguaje de programación Java**

El lenguaje de programación Java es un lenguaje de alto nivel que posee las siguientes características:

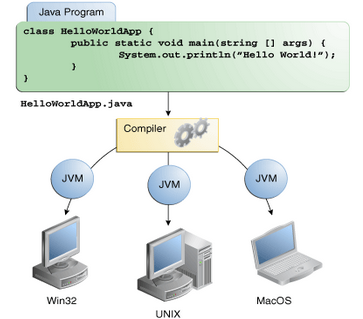
* Simple
* Orientado a Objetos
* Distribuido
* Multi-hilo
* Dinámico
* Portable
* De alto desempeño
* Robusto
* Seguro

En el lenguaje de programación Java, todo el código fuente es primero escrito en un archivo de texto plano con la extensión “**.java**”. Estos archivos fuente son luego compilados por el compilador ***javac*** a archivos “**.class**”.

Un archivo “.class” no contiene código que es nativo al procesador de la máquina donde es compilado, sino que contiene ***bytecodes*** (el lenguaje máquina de la Java Virtual Machine). El comando ***java*** ejecuta la aplicación con una instancia de la Java Virtual Machine.



Como la Java Virtual Machine está disponible en diferentes sistemas operativos, el mismo archivo “.class” es capaz de ser ejecutado en Microsoft Windows, Solaris, Linux o Mac OS.



**La plataforma Java**

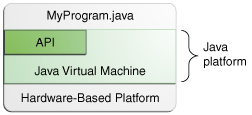
Una plataforma es el entorno de hardware o software en el que un programa se ejecuta. Ejemplos de plataformas populares son Microsoft Windows, Linux y Mac OS. La mayoría de las plataformas pueden ser descritas como una combinación del sistema operativo y el hardware sobre el que se ejecuta. La plataforma de Java difiere de la mayoría de las demás plataformas en que es una plataforma solo de software que se ejecuta sobre otras plataformas de hardware.

La plataforma de Java tiene dos componentes:

* La ***Java Virtual Machine***
* La ***Java Application Programming Interface*** (Java API)

Como vimos en el capítulo anterior, la Java Virtual Machine es la base de la plataforma Java y existen versiones para diferentes plataformas de hardware.

La API de Java es una colección grande de componentes de software (clases) que proveen muchas funcionalidades útiles. Está agrupada en librerías de clases e interfaces relacionadas entre sí, estas librerías son conocidas como ***packages***.



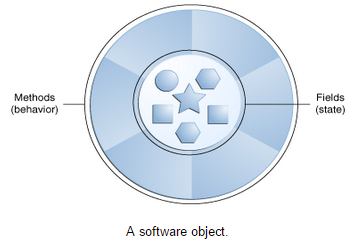
**Conceptos básicos de la Programación Orientada a Objetos**

**¿Qué es un Objeto?**

Los objetos son la clave para entender la tecnología orientada a objetos. Si miramos a nuestro alrededor, podemos encontrar muchísimos ejemplos de objetos del mundo real: un perro, un escritorio, un vaso, una televisión, una bicicleta.

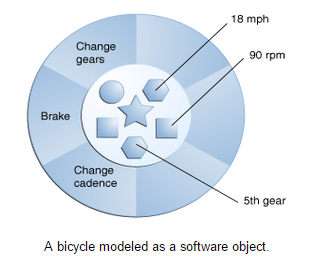
Los objetos del mundo real comparten dos características: Todos ellos tienen un estado y un comportamiento. Todos los perros tienen un estado (nombre, color, hambre) y un comportamiento (ladrar, mover la cola, comer). Identificar el estado y comportamiento de objetos del mundo real es una buena manera de comenzar a pensar en términos de la programación orientada a objetos.

Toma un minuto ahora mismo para observar los objetos del mundo real que se encuentra a tu alrededor. Para cada objeto que veas, pregúntate: ¿Cuáles estados puede el objeto tener? Y ¿Cuáles comportamientos puede el objeto ejecutar? Podrás notar que los objetos del mundo real varían en complejidad; una lámpara puede tener solo 2 posibles estados (encendido y apagado) y 2 posibles comportamientos (encender y apagar), pero una radio puede tener estados adicionales (encendido, apagado, volumen actual, estación actual) y comportamientos (encender, apagar, bajar el volumen, buscar estación). Puedes ademar haber notado que los objetos del mundo real pueden, en efecto, contener otros objetos. Todas estas observaciones del mundo real pueden ser trasladadas al mundo de la programación orientada a objetos.



Los objetos de software son conceptualmente similares a los objetos del mundo real: también consisten de estado y comportamiento. Un objeto almacena su estado en ***propiedades*** (variables en otros lenguajes de programación) y exponen su comportamiento a través de ***métodos*** (funciones en otros lenguajes de programación). Los métodos operan en base al estado interno de un objeto y son el mecanismo primario para la comunicación de objeto a objeto. A la técnica de esconder el estado interno y requerir que todas las interacciones sean ejecutadas a través de los métodos de un objeto, se le conoce como ***encapsulamiento de datos***.

Consideremos, por ejemplo, una bicicleta:



Al definir el estado y proveer los métodos para cambiar ese estado, el objeto mantiene el control sobre cómo el mundo exterior puede usarlo. Por ejemplo, si la bicicleta tiene solamente 6 engranes, el método para cambiar engranes puede rechazar cualquier valor menor a 1 o mayor a 6.

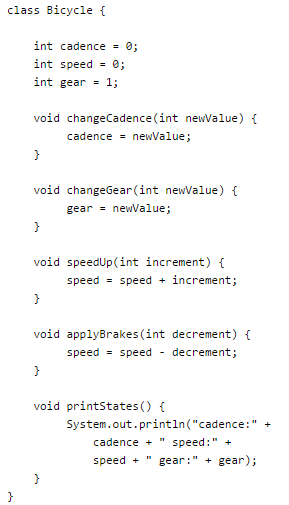
Construir código en objetos de software individuales provee un número de beneficios, incluyendo:

* **Modularidad**: El código fuente para un objeto puede ser escrito y mantenido independientemente del código fuente para otros objetos. Una vez creado, un objeto puede ser pasado fácilmente dentro del sistema.
* **Ocultamiento de información**: Al interactuar solamente con los métodos de un objeto, los detalles de su implementación interna permanecen ocultos del mundo exterior.
* **Reutilización de código**: Si un objeto ya existe (aún haya sido escrito por otro programador), puede ser usado en tú programa. Esto permite que especialistas creen/prueben objetos complejos y específicos a ciertas tareas, los cuales puedes utilizar confiando en que funcionaran en tu propio código.
* **Facilidad de depuración y remplazo**: Si un objeto en particular se torna problemático, puedes simplemente eliminarlo de tu aplicación y remplazarlo por un objeto diferente. Esto es análogo a los problemas mecánicos del mundo real, si una pieza se rompe, remplazas la pieza, no la máquina entera.

**¿Qué es una clase?**

En el mundo real, podemos encontrar muchos objetos todos del mismo tipo. Cada bicicleta fue construida del mismo conjunto de planos por lo que contienen los mismos componentes. En términos de programación orientada a objetos, decimos que una bicicleta es una instancia de la clase de objetos conocido como bicicletas. Una clase es un plano a partir del cual un objeto es creado.

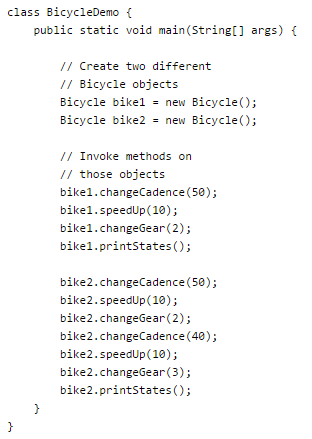
La siguiente clase Bcycle es una posible implementación de una bicicleta:



La sintaxis del lenguaje de programación Java puede parecer nueva para ti, pero el diseño de esta clase está basado en la discusión previa de los objetos bicicleta. Los campos cadence, speed y gear representa el estado del objeto, y los métodos changeGear, chageCadence y speedUp definen su interacción con el mundo exterior.

Puede que hayas notado que la clase Bicycle no contienen un método main. Esto se debe a que no es una aplicación completa; solamente es el plano para las bicicletas que podrían ser usadas en una aplicación. La responsabilidad de crear y usar nuevos objetos Bicycle corresponde a otra clase en tu aplicación.

A continuación se muestra la clase BicyleDemo, la cual crea dos objetos Bicycle e invoca sus métodos.



**Variables**

Como vimos en los capítulos anteriores, un objeto almacena su estado en campos. El capítulo ¿Qué es un objeto? Introdujo el concepto de campos, sin embargo probablemente aun tengas algunas preguntas, como: ¿Cuáles son las reglas para nombrar los campos? Además de int, ¿Qué otros tipos de datos existen? ¿Los campos deben ser inicializados cuando se declaran? ¿Los campos asignan un valor default si no son explícitamente inicializados? Exploraremos las respuestas a estas preguntas en este capítulo, pero antes de que lo hagamos, hay algunas distinciones técnicas que debes conocer. En el lenguaje de programación Java, se utilizan los términos campo y variable, esto es una fuente común de discusiones entre nuevos desarrolladores porque pareciera que ambas se refieren a lo mismo.

El lenguaje de programación Java define los siguientes tipos de variables:

* **Variables de Instancia** (Campos no estáticos): Técnicamente hablando, los objetos almacenan sus estados en “campos no estáticos”, es decir, campos declarados sin la palabra clave *static*. A los campos no estáticos también se les conoce como variables de instancia, porque sus valores son únicos para cada instancia de una clase (en otras palabras, para cada objeto); la velocidadActual de una bicicleta es independiente de la velocidadActual de otra.
* **Variables de Clase** (Campos estáticos): Una variable de clase es todo campo declarado con el modificador *static*; esto le hace saber al compilador que solo existe exactamente una copia de esta variable, independientemente de cuantas veces la clase ha sido instanciada. Un campo que defina el número de engranes para un tipo particular de bicicleta puede ser marcado como *static*, pues conceptualmente el mismo número de engranes aplicaría a todas las instancias.
* **Variables Locales**: De manera similar a como un objeto almacena su estado en campos, un método comúnmente puede almacenar su estado temporal en variables locales. La sintaxis para declarar una variable local es similar a declarar un campo (por ejemplo, int count = 0;). No existe una palabra clave para designar una variable como local; esto se determina exclusivamente en base al lugar en donde la variable es declarada. Por esto, las variables locales son solo visibles para el método en el que fueron declaradas; no son accesibles por el resto de la clase.
* **Parámetros**: Recordemos que la firma del método main es public static void main(String[] args). Aquí, la variable args es el parámetro del método. Lo importante a recordar es que los parámetros siempre se clasifican como variables y no como campos.

**Nombrando variables**

Cada lenguaje de programación tiene su propio conjunto de reglas y convenciones del tipo de nombres que está permitido usar, y el lenguaje Java no es diferente. Las reglas y convenciones para nombrar tus variables pueden ser resumidas de la siguiente manera:

* Los nombres de variables son sensibles a mayúsculas y minúsculas. Un nombre de variable puede ser cualquier identificador válido (una secuencia ilimitada de letras y números, comenzando por una letra, el signo “$” o el guion bajo “\_”). La convención sin embargo, es comenzar siempre los nombres de variables con una letra, no el signo de “$” ni guion bajo. Adicionalmente, el carácter “$” por convención nunca es usado.
* Los caracteres siguientes pueden ser letras, números, “$” o “\_”.
* Si el nombre elegido consiste de solo una palabra, por convención se utiliza la palabra en minúsculas, si consiste de más de una, la primera letra de cada palabra subsecuente se utiliza en mayúsculas (camel case).

**Operadores**

Ahora que ya sabes cómo declarar variables, probablemente quieras saber cómo haber algo con ellas. Aprender los operadoras del lenguaje de programación Java es un buen inicio. Los operadores son símbolos especiales que ejecutan operaciones especificas en uno, dos o tres operandos y luego retornan un resultado.

Es importante que conozcas cuales son los operadores que tienen la mayor precedencia. Los operadores en la siguiente tabla están listados de acuerdo al orden de precedencia. Entre más cerca estén al inicio de la tabla, mayor es su precedencia. Los operadores con mayor precedencia son evaluados antes que los operadores con relativa menor precedencia. Los operadores que aparecen en la misma línea tienen la misma precedencia. Cuando los operadores de igual precedencia aparecen en la misma expresión, una regla existe para saber cuál es evaluado primero. Todos los operadores binarios a excepción de los operadores de asignación son evaluados de izquierda a derecha; los operadores de asignación son evaluados de derecha a izquierda.

|  |  |
| --- | --- |
| Sufijos | expr++ expr-- |
| Unarios | ++expr --expr +expr -expr ~ ! |
| Multiplicativos | \* / % |
| Aditivos | + - |
| Cambio | << >> >>> |
| Relacional | < > <= >= instanceof |
| Igualdad | == != |
| AND bit a bit | & |
| OR bit a bit exclusive | ^ |
| OR bit a bit inclusive | | |
| AND lógico | && |
| OR lógico | || |
| Ternario | ? : |
| Asignación | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

**Operadores de Asignación, Aritméticos y Unarios**

**El operador de asignación**

Uno de los operadores más comunes que encontraras es el operador de asignación “=”. Lo hemos visto en la clase Bicycle de los temas anteriores; se utiliza para asignar el valor de la derecha al operando de la izquierda:



Este operador también puede ser usado en objetos para asignar referencias a objetos.

**Operadores Aritméticos**

El lenguaje de programación Java provee operadores que realizan suma, resta, multiplicación y división. Probablemente los reconozcas pues también son utilizados en matemáticas básicas. El único símbolo que puede parecer nuevo es “%”, que divide un operando por otro y retorna el residuo como resultado.

|  |  |
| --- | --- |
| + | Operador aditivo (Utilizado también en concatenación de String) |
| - | Operador de sustracción |
| \* | Operador de multiplicación |
| / | Operador de división |
| % | Operador de residuo |

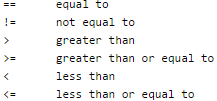
**Operadores unarios**

Los operadores unarios requieren solo un operando; realizan varias operaciones como incrementar/disminuir un valor por 1, negar una expresión o invertir el valor de un booleano.

|  |  |
| --- | --- |
| + | Indica un valor positivo, sin embargo los números son por default positivos |
| - | Niega una expresión |
| ++ | Operador de incremento, incrementa un valor por 1 |
| -- | Operador de disminución, disminuye un valor por 1 |
| ! | Invierte el valor de una expresión booleana. |

**Operadores de igualdad, relacionales y condicionales**

Los operadores de igualdad y relacionales determinan si un operando es mayor que, menor que, igual o no igual a otro operando. La mayoría de estos operadores probablemente te parezcan familiares. Ten en mente que debes siempre usar “==” y no “=” al comprobar si dos variables primitivas son iguales.



**Operadores Condicionales**

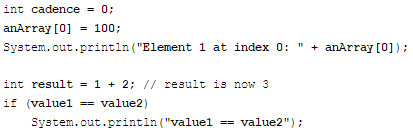
Los operadores && y || realizan operaciones AND y OR respectivamente en dos expresiones booleanas.

**Expresiones, sentencias y bloques**

Ahora que entiendes las variables y operadores, es tiempo de aprender acerca de expresiones, sentencias y bloques. Los operadores pueden ser utilizados en construir expresiones, que originan valores; las expresiones son componentes de las sentencias; las sentencias pueden ser agrupadas en bloques.

**Expresiones**

Una expresión es una construcción en base a la sintaxis del lenguaje que evalúa a un solo valor. Ya hemos visto algunos ejemplos de expresiones, ilustradas en negritas en el siguiente ejemplo:



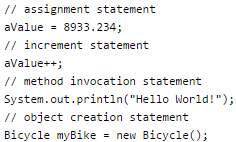
El tipo de dato retornado por una expresión depende de los elementos utilizados en la expresión. La expresión cadence = 0 retorna un int porque el operador de asignación retorna un valor del mismo tipo que el operador del lado izquierdo, en este caso, cadence es un int. Como puedes ver en las demás expresiones, una expresión puede retornar otros tipos de datos, como boolean o String.

**Sentencias**

Las sentencias son ligeramente equivalentes a las sentencias en los lenguajes naturales. Una sentencia forma una unidad completa de ejecución. Los siguientes tipos de expresiones pueden ser convertidas en una sentencia agregando un “;” para terminar la expresión.

* Expresiones de asignación
* Cualquier uso de ++ o –
* Invocación de métodos
* Expresiones de creación de objetos

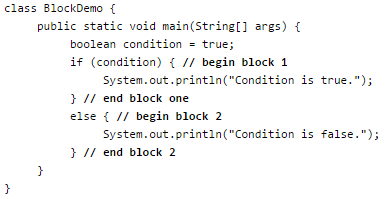
Dichas sentencias son llamadas sentencias de expresiones. A continuación tenemos algunos ejemplos.



Además de las sentencias de expresión, existen dos tipos más de sentencias: sentencias de declaración y sentencias de control de flujo. Una sentencia de declaración declara una variable, una sentencia de control de flujo regula el orden en que las sentencias son ejecutadas.

**Bloques**

Un bloque es un grupo de cero o más sentencias dentro de llaves. El siguiente ejemplo, BlockDemo, ilustra el uso de bloques:



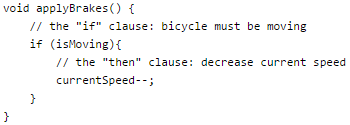
**Sentencias de Control de Flujo**

Las sentencias dentro de un archivo fuente generalmente son ejecutadas en orden de arriba hacia abajo según aparecen. Las sentencias de control de flujo, sin embargo, rompen este flujo de ejecución al emplear toma de decisiones, iteración y derivación, permitiendo a tu programa ejecutar condicionalmente bloques específicos de código. Está sección describe las sentencias de toma de decisión (if-then, if-then-else, siwtch), las sentencias de iteración (for, while, do-while) y las sentencias de derivación (break, continue, return) soportadas por el lenguaje de programación Java.

**Las sentencias if-then e if-then-else**

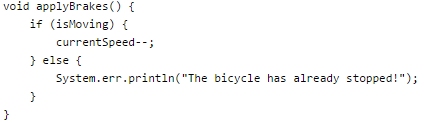
**La sentencia if-then**

La sentencia if-then es la más básica de todas las sentencias de control de flujo. Ordena a tu programa a ejecutar ciertas secciones del código solo si un condición en particular evalúa a true. Por ejemplo, la clase Bicycle puede permitir a los frenos que disminuyan la velocidad de las bicicletas solo si la bicicleta está en movimiento. Una posible implementación del método applyBrakes puede ser la siguiente:

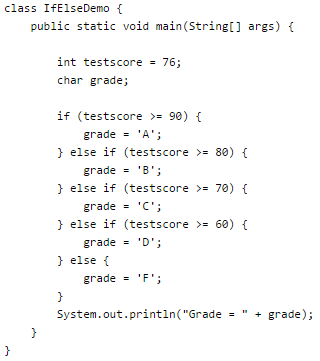


**La sentencia if-then-else**

La sentencia if-then-else provee un camino de ejecución secundario en caso de que una clausula “if” evalúe a false. Puedes usar if-then-else en el método applyBrakes para realizar alguna acción cuando la bicicleta no esté en movimiento. En este caso la acción es simplemente imprimir un mensaje de error diciendo que la bicicleta se encuentra detenida.



El siguiente programa, IfElseDemo, asigna una calificación en base al resultado de un examen.

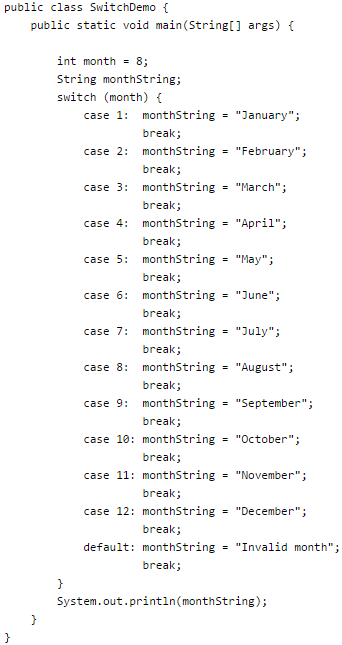


Puedes haber notado que el valor de testscore puede satisfacer más de una expresión, sin embargo, una vez que una condición es satisfecha, se ejecuta la sentencia adecuada y el resto de las condiciones no son evaluadas.

**La sentencia switch**

A diferencia de las sentencias if-then e if-then-else, la sentencia switch puede tener un gran número de posibles opciones para ejecutar. El switch funciona con los tipos de datos primitivos byte, short, char e int. También funciona con tipos de datos enumerados (los cuales conoceremos mas adelante), la clase String, y algunas clases especiales que funcionan como contenedores de ciertos primitivos: Character, Byte, Short e Integer.

El siguiente ejemplo, SwitchDemo, delcara un int con el nombre de month, cuyo valor representa un mes. El código muestra el nombre del mes, basado en el valor de month, usando la sentencia switch.



Al cuerpo de una sentencia switch se le conoce como un bloque switch. Una sentencia dentro del bloque switch puede ser etiquetada con uno o más etiquetas case o default. El switch evalúa su expresión, y luego ejecuta todas las sentencias que están a continuación de la etiqueta case que cumple la expresión.

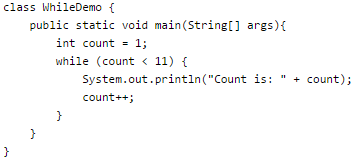
Otro punto de interés es la sentencia break. Cada break termina con el switch que la contiene. La sentencia break es necesaria porque sin ella todos los case serían ejecutados sin distinción, pues todas las sentencias encontradas después de un case que cumple la expresión de un switch son ejecutadas hasta que se encuentre un break.

**Las sentencias while y do-while**

La sentencia while ejecuta continuamente un bloque de sentencias mientras una condición sea true. Su sintaxis puede ser expresada de la siguiente manera:



La sentencia while evalua la expresión, la cual debe retornar un valor boolean. Si la expresión evalúa a true, se ejecutan las sentencias que están dentro del bloque while. La sentencia while continua entonces evaluando la expresión y ejecutando su bloque hasta que la expresión evalue a false. Usar la sentencia while para imprimir los números de 1 hasta 10 se ejemplifica en el siguiente programa WhileDemo:



Puedes implementar un ciclo infinito usando la sentencia while de la siguiente manera:



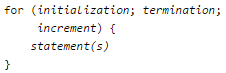
El lenguaje Java también provee una sentencia do-while, que puede ser expresada de la siguiente manera:



La diferencia entre la sentencia do-while y while es que do-while evalua su expresión al final del bloque en lugar de al principio, lo que significa que el bloque do es ejecutado cuando menos una vez.

**La sentencia for**

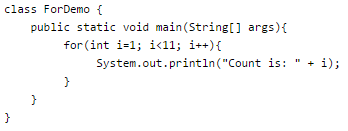
La sentencia for provee una manera compacta de iterar sobre un rango de valores. La forma general de una sentencia for puede ser expresada de la siguiente manera:



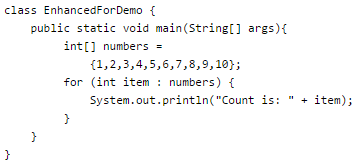
Al usar la sentencia for, ten en cuenta lo siguiente:

* La expresión initilization inicializa el bucle; se ejecuta una sola vez, al inicio del bucle.
* Cuando la expresión termination evalua a false, el bucle termina.
* La expresión incremente es invocada después de cada iteración del bucle, es aceptable para esta expresión que incremente o disminuya un valor.

El siguiente programa, ForDemo, utiliza la forma general del for para imprimir los números desde 1 hasta 10:



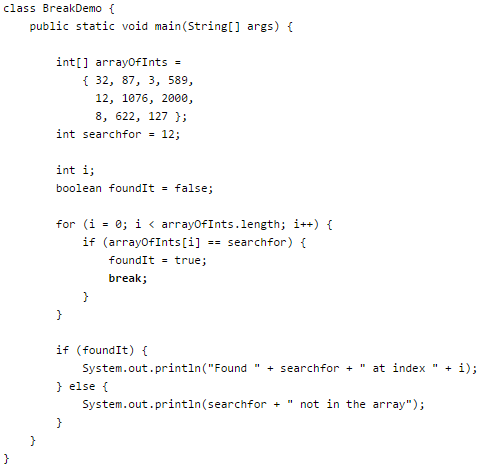
La sentencia for también tiene otra forma, diseñada para iterar sobre colecciones y arreglos. El siguiente programa, EnhancedForDemo, usa la forma mejorada del for para iterar sobre un array:



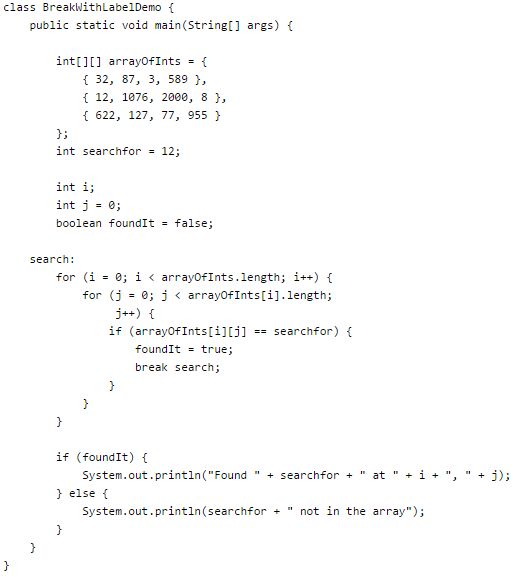
**Sentencias de derivación**

**La sentencia break**

La sentencia break tiene dos formas: etiquetada y no etiquetada. Ya hemos visto la forma no etiquetada en la sentencia switch. La forma no etiquetada también puede ser usada para terminar con la ejecución de un bucle for, do-while o while, como se muestra en el siguiente programa BreakDemo:

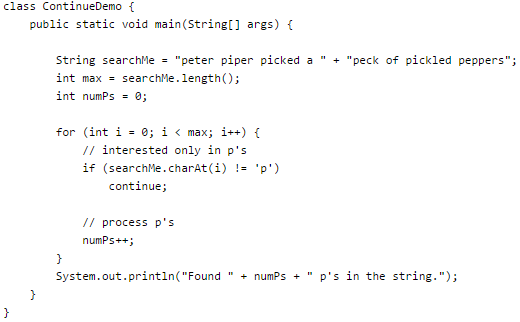


La forma no etiquetada del break termina el bucle más próximo a su declaración, sin embargo la forma etiquetada puede terminar un bucle exterior. El siguiente programa, BreakWithLabelDemo, es similar al ejemplo previo, pero usa bucles for anidados para buscar un valor en un arreglo de dos dimensiones. Cuando el valor es encontrado, el break etiquetado termina con el loop exterior (etiquetado con “search”):

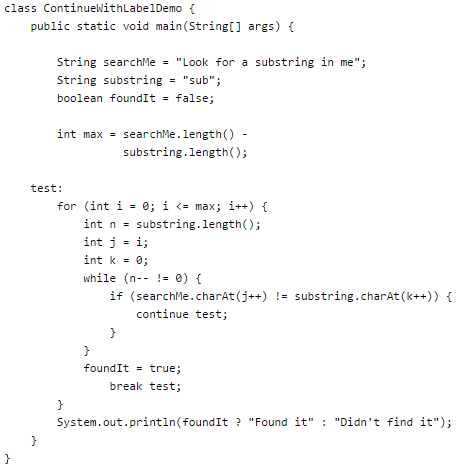


**La sentencia continue**

La sentencia continue salta la iteración actual de un bucle for, while o do-while. La forma no etiquetada salta hasta el final del bloque mas próximo a donde fue declarada. El siguiente programa, ContinueDemo, ejemplifica el uso de continue:



Un continue etiquetado salta al final de la iteración actual del bucle marcado con la etiqueta usada. El siguiente ejemplo, ContinueWithLabelDemo, utiliza bucles anidados para buscar una substring dentro de otra string. El programa de ejemplo usa la forma etiquetada de continue para saltar la iteración del bucle exterior:



**La sentencia return**

La última de las sentencias de derivación es la sentencia return. La sentencia return termina la ejecución del método actual y el control del flujo retorna a donde el método fue invocado. La sentencia return tiene dos formas, una donde se retorna un valor, y otra don de no. Para retornar un valor, simplemente coloca el valor (o la expresión que calcula el valor) después de la palabra clave return.