

ELEMENTOS BÁSICOS DEL LENGUAJE JAVA



NUEVO EN JAVA

Qué es java

```
1  class HelloWorld {  
2      public static void main(String[] args) {  
3          System.out.println("Hello, World!");  
4      }  
5  }  
  
Hello, World!
```





El fenómeno de la tecnología Java

Hablar de Java es muy común en el ámbito del desarrollo, pero ¿que es exáctamente java?. En este documentos hablaremos de qué es java y de porque se considera un lenguaje de programación y una plataforma.

Algunos temas a tratar:

- Acerca de la tecnología java.
- Que puede hacer java.
- Entre otras cosas interesantes.



El fenómeno de la tecnología Java

Java es un lenguaje de alto nivel que se puede caracterizar por todas las siguientes palabras de moda:

- Sencillo
- Orientado a objetos
- Distribuido
- MultiHilo
- Dinámico
- Multiplataforma
- Portatil
- Alto Rendimiento
- Robusto
- Seguro.



Sencillo

- Java se diseñó para ser parecido a C++ y así facilitar un rápido y fácil aprendizaje.
- Java posee una curva de aprendizaje muy rápida.
- Todos aquellos familiarizados con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características, como los punteros.
- Tiene un modelo de objetos más simple que otros lenguajes como C++.
- Elimina la mayor parte de los problemas de C++ como la manipulación directa de punteros o memoria, que suelen inducir a errores.
- Añade características muy útiles como el garbage collector (reciclador de memoria dinámica).



Orientado a objetos

- Java implementa el paradigma orientado a objetos desde el principio.
- Todo es un objeto. Cada elemento del problema debe ser modelizado como un objeto.
- Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantenerlo más simple
- Java trabaja con sus datos como objetos y con interfaces a esos objetos.
- Soporta las cuatro características propias del paradigma de la orientación a objetos: abstracción, encapsulación, herencia y polimorfismo.
- Las plantillas de objetos son llamadas, como en C++, clases y sus copias, instancias .

Java se ha construido con extensas capacidades de interconexión TCP/IP.

- Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp
- Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.
- Permiten abrir sockets, establecer y aceptar conexiones con servidores o clientes remotos



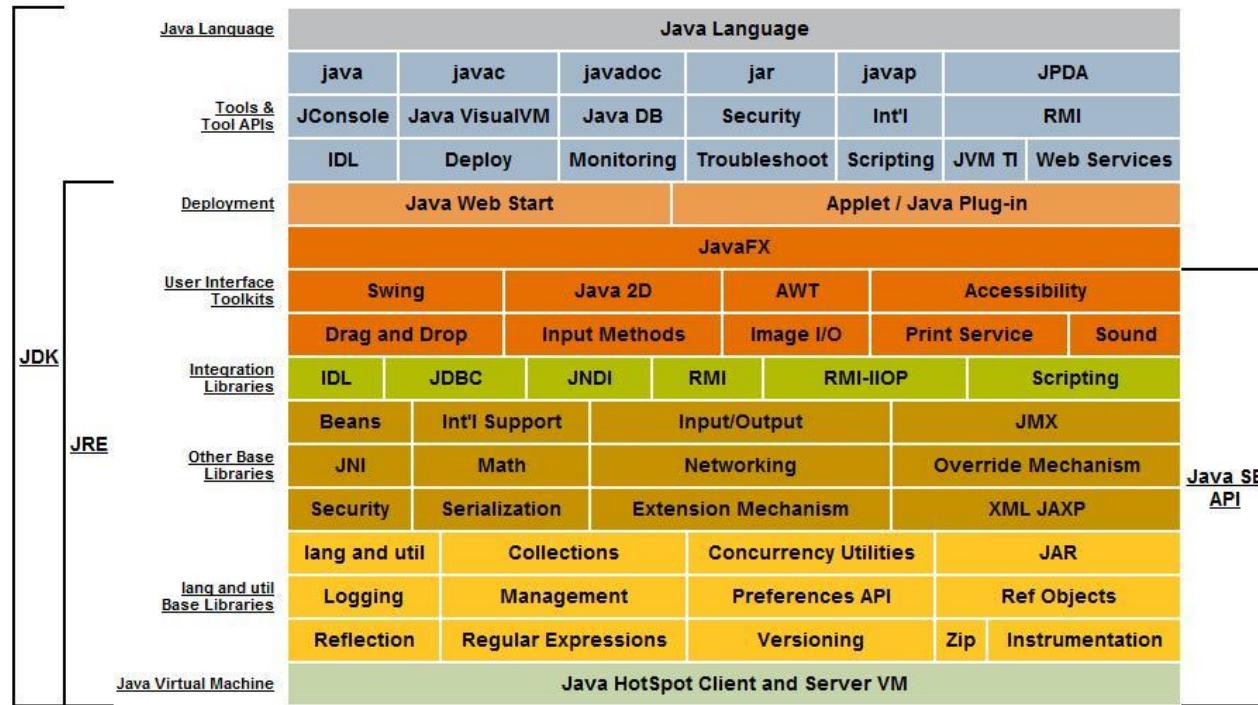
Java como plataforma

Una plataforma es el entorno de hardware o software en el que se ejecuta un programa. Ya hemos mencionado algunas de las plataformas más populares como Microsoft Windows, Linux, Solaris OS y Mac OS. La mayoría de las plataformas se pueden describir como una combinación del sistema operativo y el hardware subyacente. La plataforma Java se diferencia de la mayoría de las otras plataformas en que es una plataforma solo de software que se ejecuta sobre otras plataformas basadas en hardware.

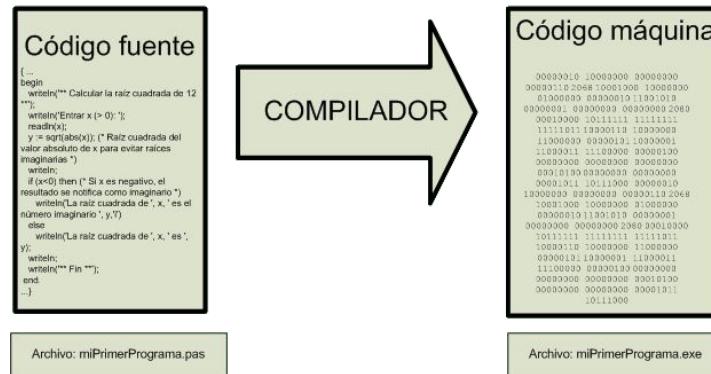
The Java™ Tutorials

Java como plataforma

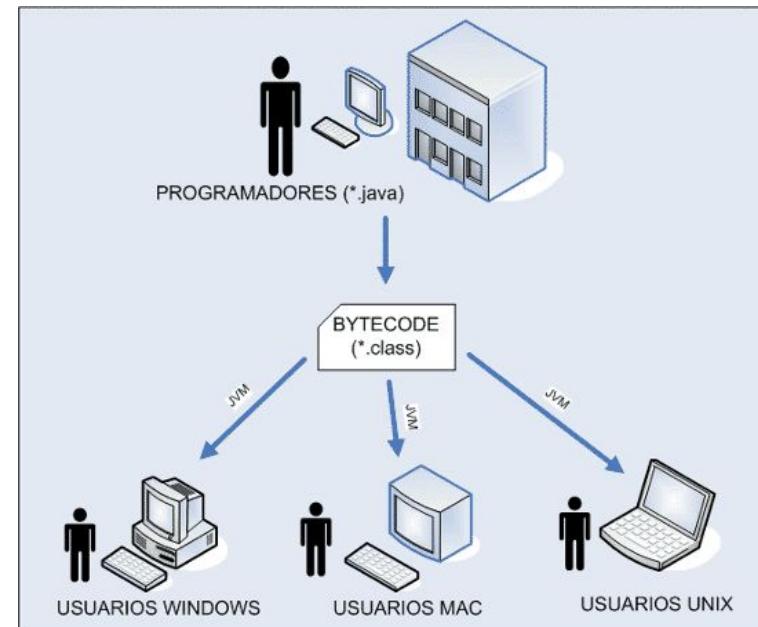
Description of Java Conceptual Diagram



Ya conoce la máquina virtual de Java; es la base de la plataforma Java y se adapta a varias plataformas basadas en hardware.



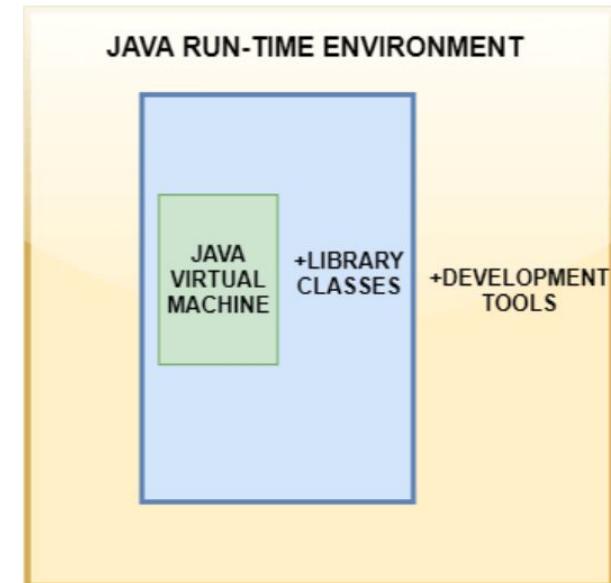
Java como plataforma



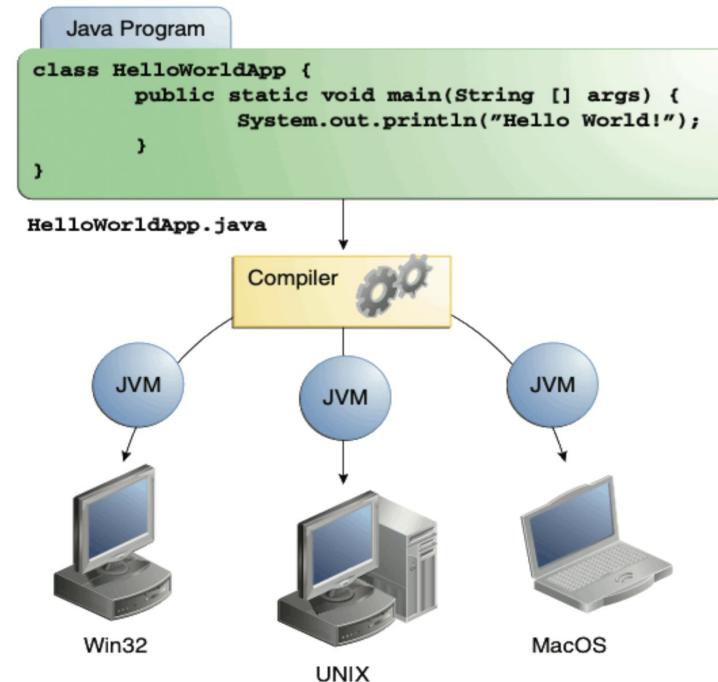


Java como plataforma

Java Run-time Environment (JRE) es parte del Java Development Kit (JDK). Es una distribución de software disponible gratuitamente que tiene una biblioteca de clases de Java, herramientas específicas y una JVM independiente. Es el entorno más común disponible en los dispositivos para ejecutar programas Java. El código fuente de Java se compila y se convierte en código de bytes de Java. Si desea ejecutar este código de bytes en cualquier plataforma, necesita JRE. El JRE carga clases, verifica el acceso a la memoria y recupera los recursos del sistema. JRE actúa como una capa en la parte superior del sistema operativo.



El fenómeno de la tecnología Java



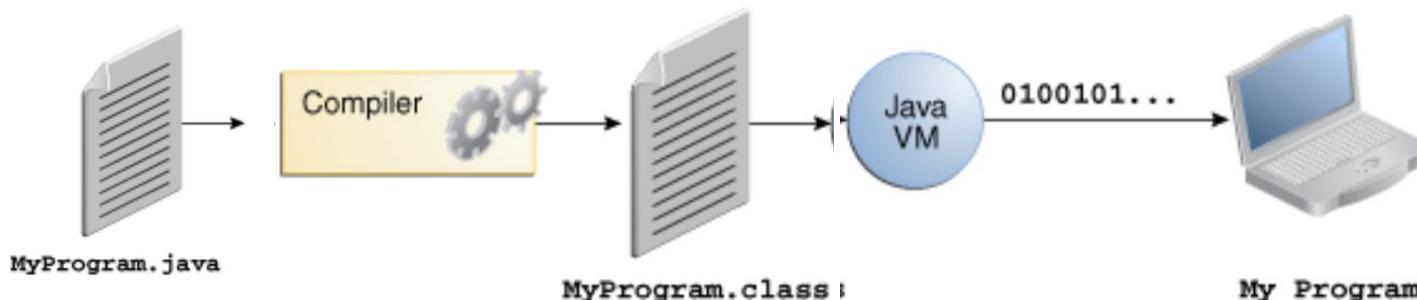
A través de Java VM, la misma aplicación es capaz de ejecutarse en múltiples plataformas.

¿Como funciona Java?

En el lenguaje de programación Java, todo el código fuente se escribe primero en archivos de texto sin formato que terminan con la extensión .java

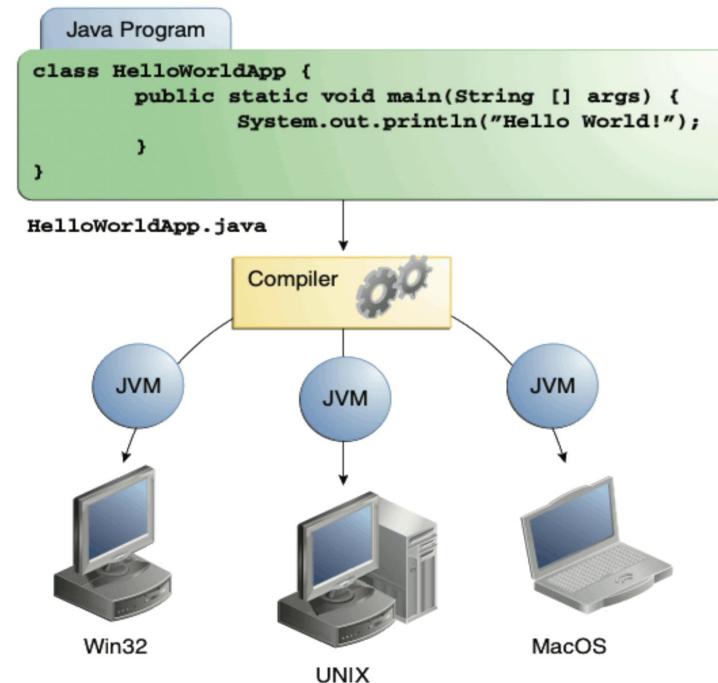
Esos archivos fuente luego son compilados en archivos .class por el compilador java. Un archivo .class no contiene código nativo de su procesador, en su lugar contiene bytecodes , el lenguaje de máquina de Java Virtual Machine.

Al final, la máquina virtual de java logra entender el archivo .class y ejecuta el programa.





El fenómeno de la tecnología Java



A través de Java VM, la misma aplicación es capaz de ejecutarse en múltiples plataformas.



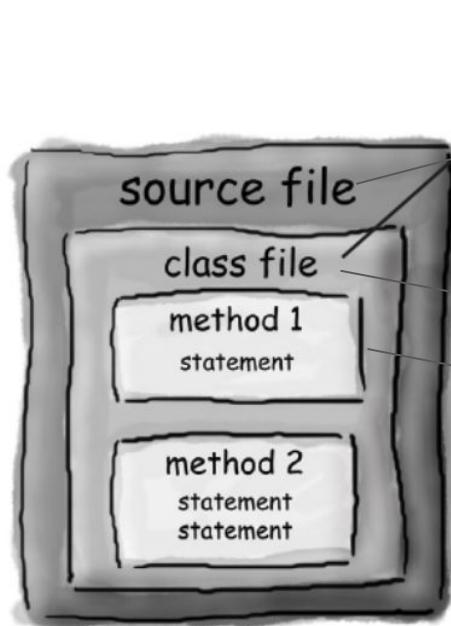
Java como plataforma

La API es una gran colección de componentes de software listos para usar que brindan muchas capacidades útiles. Se agrupa en bibliotecas de clases e interfaces relacionadas; estas bibliotecas se conocen como paquetes .

Package	Descripción del contenido
java.lang	Contiene las clases e interfaces más empleadas en la mayoría de los programas de Java. Es importado automáticamente por todos los programa Java: no se necesita sentencia <code>import</code> para utilizar lo declarado en este paquete.
java.io	Contiene clases que permiten las operaciones de entrada y salida de datos de un programa.
java.util	Contiene clases e interfaces de utilidades: operaciones con la fecha y la hora, generación de números aleatorios...
java.applet	Contiene todas las clases e interfaces necesarias para la construcción de <i>applets</i> de Java
java.net	Contiene clases que permite a un programa comunicarse a traves de redes (Internet o intranet)
java.text	Contiene clases e interfaces que permiten operaciones de números, fechas, caracteres y cadenas.
java.awt	Es el paquete <i>Abstract Windowing Toolkit</i> . Contiene muchas clases e interfaces necesarias para trabajar con la interfaz de usuario gráfica <i>clásica</i> .
java.beans	Contiene clases para facilitar a los programadores la generación de componentes de software reutilizables.

CÓMO LUCE UN PROGRAMA JAVA

Estructura java file



HolaMundo.java

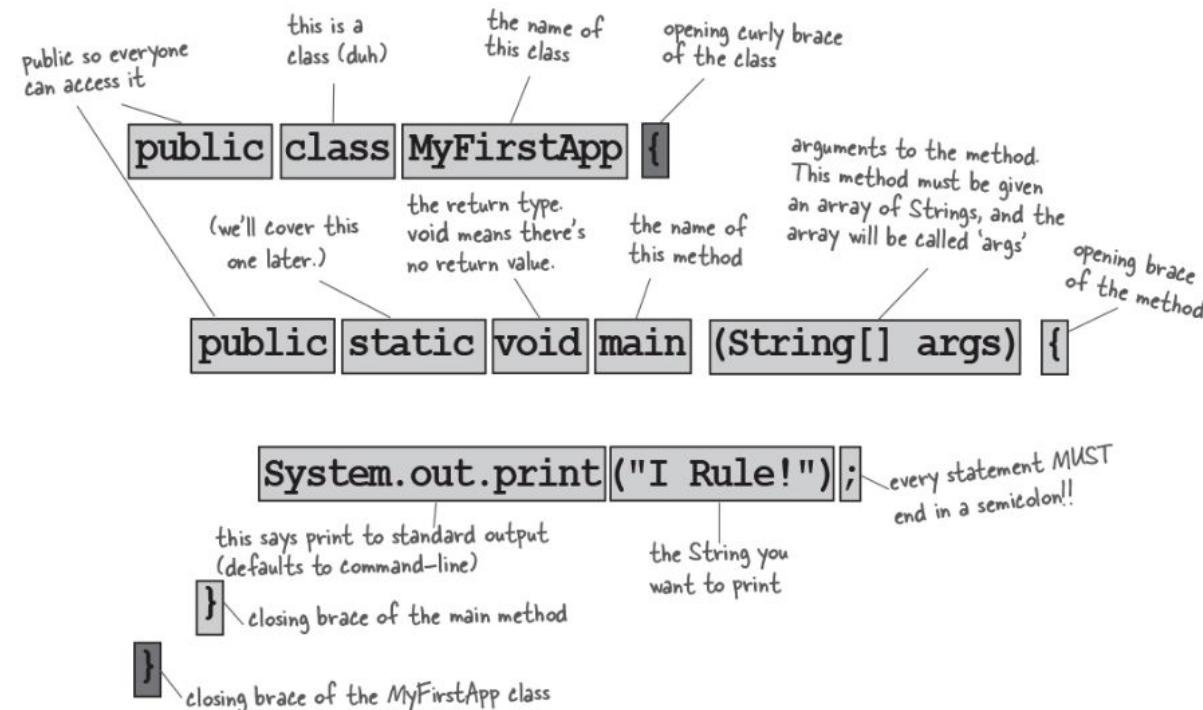
This is actually an image. The “class file” bit needs changing to say “class definition”

```
/*
 * Este programa escribe el texto "Hola Mundo" en la consola
 * utilizando el método System.out.println()
 */

public class HolaMundo {

    public static void main (String[] args) {
        System.out.println("Hola Mundo");
    }
}
```

Estructura java file





Estructura java file

Un archivo de código fuente (con la extensión .java) contiene la definición de clase. Esta clase representa una pieza de tu programa.

La clase comienza y termina con llaves.

```
public class Dog {
```

```
}
```

class

Estructura java file

Una clase tiene uno o más métodos. En la clase Dog, el método bark tiene las instrucciones para que el perro ladre.

©

```
public class Dog {  
    void bark() {  
    }  
}
```

method

Estructura java file

En el método van las instrucciones (o la lógica de negocio) que tenemos que programar.

```
public class Dog {  
    void bark() {  
        statement1;  
        statement2;  
    }  
}  
statements
```



Que ponemos en los métodos?

```
int x = 3;  
String name = "Dirk";  
x = x * 17;  
System.out.print("x is " + x);  
double d = Math.random();  
// this is a comment
```

Declaraciones, asignaciones, llamadas a métodos, etc.



Que ponemos en los métodos?

```
while (x > 12) {  
    x = x - 1;  
}
```

```
for (int i = 0; i < 10; i = i + 1) {  
    System.out.print("i is now " + i);  
}
```

Puedes hacer tus loops favoritos, como el for, el while, el do-while, etc.



Que ponemos en los métodos?

```
while (x > 12) {  
    x = x - 1;  
}
```

```
for (int i = 0; i < 10; i = i + 1) {  
    System.out.print("i is now " + i);  
}
```

Puedes hacer tus loops favoritos, como el for, el while, el do-while, etc.



Que ponemos en los métodos?

```
if (x == 10) {  
    System.out.print("x must be 10");  
} else {  
    System.out.print("x isn't 10");  
}  
  
if ((x < 3) && (name.equals("Dirk"))){  
    System.out.println("Gently");  
}  
  
System.out.print("this line runs no matter what");
```

Hacer muchas cosas basado en condiciones usando los if, los if-else, etc.



También tenemos algunas reglas

```
x = x + 1;
```

TODAS las sentencias terminan con punto y coma.



También tenemos algunas reglas

```
x = 22;  
// this line disturbs me
```

Puedes comentar colocando dos slash !



También tenemos algunas reglas

```
x = 22;  
// this line disturbs me
```

Puedes comentar colocando dos slash !

```
x = 3 ;
```

Los espacio no le importan



También tenemos algunas reglas

```
int weight;  
//type: int, name: weight
```

Las variables se declaran con su tipo de dato (no como javascript)

```
public void go() {  
    // amazing code here  
}
```

Las clases y los métodos comienzan y terminan con llaves.



VARIABLES

The Java™ Tutorials

Como aprendimos en la ppt anterior, los objetos almacenan sus estados (atributos) en campos. Como por ejemplo:

```
int cadence = 5;
```

Ya sabemos declarar los atributos, las variables de una clase, los nombres de clases y los nombres de métodos pero esto es solo el inicio ya que en java, existen reglas u/o convenciones que debemos seguir para generar código de calidad.



Tipos de variables

En el lenguaje de programación java se definen los siguientes tipos de variables:

- **Variables locales:** Al igual que las clases almacenan sus estados en campos, los métodos almacenar sus valores temporales en variables locales. Lo importante de las variables locales es que su valor sólo es válido dentro del método.

Variables locales

Variable local

sólo son accesibles desde el mismo método.

No es posible acceder a variables locales fuera de sus métodos.

```
1 package variablesinstancia;
2
3 public class DemostracionVariableInstancia {
4
5     public void muestraPuntaje() {
6         // variable de instancia
7         int puntaje = 987;
8         System.out.println("El puntaje es " + puntaje);
9     }
10
11    public void muestraPuntajeHistorico() {
12        // variable de instancia
13        int puntajeHistorico = 1000;
14        System.out.println("El puntaje historico es: " + puntajeHistorico);
15    }
16
17    public static void main(String... args) {
18        DemostracionVariableInstancia main = new DemostracionVariableInstancia();
19        main.muestraPuntaje();
20        main.muestraPuntajeHistorico();
21        System.out.println("puntajes resumen " + puntaje); // solo del objeto
22        System.out.println("puntajes resumen " + puntajeHistorico); // solo del objeto
23    }
24 }
```



Tipos de variables

En el lenguaje de programación java se definen los siguientes tipos de variables:

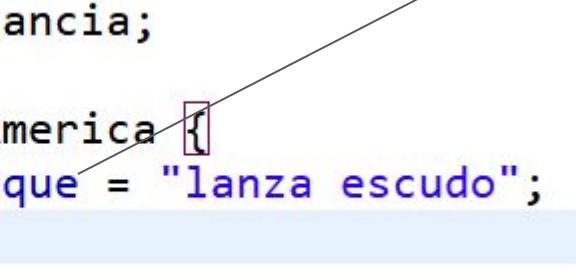
- **Variables de instancia (non-static)**: Técnicamente hablando los objetos almacenan sus estados en variables no estáticas. Estas son las que no van acompañadas de la palabra clave **static**. Las variables no estáticas también son conocidas como variables de instancia por que su valor es único para cada instancia de la clase. Por ejemplo la cantidad de cambios de una bicicleta_a es distinta a la velocidad de la bicicleta_b.

Variable de instancia

```
1 package variablesinstancia;  
2  
3 public class IronMan {  
4     public String ataque = "misiles aire-aire";  
5 }  
6
```



```
1 package variablesinstancia;  
2  
3 public class CapitanAmerica {  
4     public String ataque = "lanza escudo";  
5 }  
6
```





Variable de instancia

```
1 package variablesinstancia;  
2  
3 public class DemostracionVariablesDeInstancia {  
4     public static void main(String... args) {  
5         IronMan tony = new IronMan();  
6         CapitanAmerica roger = new CapitanAmerica();  
7         System.out.println("Iron-Man ataca con " + tony.ataque);  
8         System.out.println("Capitan America ataca con " + roger.ataque);  
9     }  
0 }
```

Una instancia es un objeto, y la variable es responsabilidad de el mismo (tony, roger)



Tipos de variables

En el lenguaje de programación java se definen los siguientes tipos de variables:

- **Variables de clase (static fields):** Una variable de clase es un campo declarado con la palabra reservada static. Como es de clase es accesible sin necesidad de crear un objeto.

Variable estatica

```
1 package variablesestaticas;  
2  
3 public class CapitanAmerica {  
4     public String ataque = "lanza escudo";  
5     public static String grito = "Soy el capitan america !"; // variable estatica  
6 }  
  
1 package variablesestaticas;  
2  
3 public class IronMan {  
4     public String ataque = "misiles aire-aire";  
5     public static String grito = "Soy el capitan america !"; // variable estatica  
6 }
```

Variable Estática

```
1 package variablesestaticas;  
2  
3 public class DemostracionVariablesDeInstancia {  
4     public static void main(String... args) {  
5         // NO NECESITO GENERAR UN OBJETO CON NEW  
6         // SOLAMENTE UTILIZO LA VARIABLE  
7         System.out.println(IronMan.grito);  
8         System.out.println(CapitanAmerica.grito);  
9     }  
10 }
```



OPERADORES



Operadores

Ahora que ha aprendido a declarar e inicializar variables, probablemente quiera saber cómo hacer algo con ellas. Aprender los operadores del lenguaje de programación Java es un buen lugar para comenzar. Los operadores son símbolos especiales que realizan operaciones específicas en uno, dos o tres operandos y luego devuelven un resultado.

Operadores Matemáticos

Se utilizan para realizar operaciones matemáticas básicas:

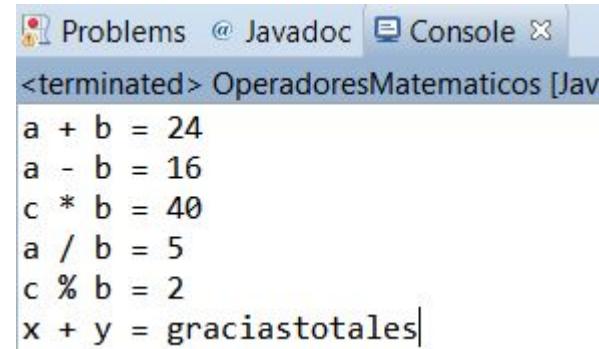
Figura 1.6 Operadores aritméticos

Operador	Significado	Ejemplo	Resultado
-	Resta	$a - b$	Resta de a y b
+	Suma	$a + b$	Suma de a y b
*	Multiplicación	$a * b$	Producto de a por b
/	División	a / b	Cociente de a entre b
%	Residuo	$a \% b$	Residuo de a entre b

```
1 package operadores;
2
3 public class OperadoresMatematicos {
4     public static void main(String... args) {
5         int a = 20;
6         int b = 4;
7         int c = 10;
8         String x = "gracias";
9         String y = "totales";
10        // operador +
11        System.out.println("a + b = " + (a + b));
12        System.out.println("a - b = " + (a - b));
13        // operador *
14        System.out.println("c * b = " + (c * b));
15        System.out.println("a / b = " + (a / b));
16        // operador %
17        System.out.println("c % b = " + (c % b));
18        // operador + si se usa con string los concatena
19        System.out.println("x + y = " + (x + y));
20    }
21 }
```

Operadores Matemáticos

Se utilizan para realizar operaciones matemáticas básicas:



```
Problems @ Javadoc Console
<terminated> OperadoresMatematicos [Java]
a + b = 24
a - b = 16
c * b = 40
a / b = 5
c % b = 2
x + y = gracias totales
```

Operadores Unarios

Los operadores unarios solo necesitan un operando y se usan para aumentar, disminuir y negar un valor.

Operator	Description
+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

Operadores Unarios

```
1 package operadores;
2
3 public class OperadoresUnarios {
4     public static void main(String... args) {
5         int result = +1; // el resultado es 1
6         System.out.println(result);
7
8         result--; // el resultado es ahora 0
9         System.out.println(result);
10
11         result++; // ahora es 1
12         System.out.println(result);
13
14         result = -result; // resultado ahora es -1
15         System.out.println(result);
16
17         boolean success = false; // success es falso
18         System.out.println(success);
19
20         System.out.println(!success); // ahora es verdadero
21
22     }
23 }
```

Console × Problems J De

<terminated> OperadoresUnarios

1

0

1

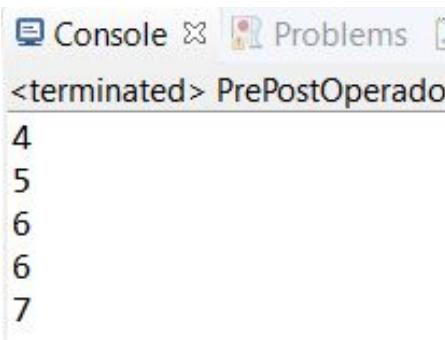
-1

false

true

Operadores Unarios

Los operadores de incremento y decremento pueden aplicarse antes (prefijo) o después (postfijo) del operando, variando el resultado.



The screenshot shows a Java development environment. On the left, a code editor displays a Java program with line numbers 1 through 14. On the right, a terminal window titled 'Console' shows the output of the program, which is the number 4 repeated four times. The code editor has syntax highlighting for Java keywords and punctuation.

```
1 package operadores;
2
3 public class PrePostOperadores {
4     public static void main(String[] args){
5         int i = 3;
6         i++;
7         System.out.println(i);
8         ++i;
9         System.out.println(i);
10        System.out.println(++i);
11        System.out.println(i++);
12        System.out.println(i);
13    }
14 }
```

```
4
5
6
7
8
9
10
11
12
13
14 }
```

→ Igualdad, Relacionales y condicionales

Los operadores de igualdad y relacionales determinan si un operando es mayor, menor que, igual o no igual a otro operando. La mayoría de estos operadores probablemente también les resulte familiar.

Tenga en cuenta que debe usar "==" , no "=". al probar si dos valores primitivos son iguales.

==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to



Igualdad, Relacionales y condicionales

1 es igual a 2

1 es distinto a 2

1 es mayor que 2

1 es menor que 2

1 es menor o igual que 2

Console Problems
<terminated> OperadoresDeComparacion

value1 != value2
value1 < value2
value1 <= value2

```
1 package operadores;
2
3 public class OperadoresDeComparacion {
4     public static void main(String[] args){
5         int value1 = 1;
6         int value2 = 2;
7         if(value1 == value2) {
8             System.out.println("value1 == value2");
9         }
10        if(value1 != value2) {
11            System.out.println("value1 != value2");
12        }
13        if(value1 > value2) {
14            System.out.println("value1 > value2");
15        }
16        if(value1 < value2) {
17            System.out.println("value1 < value2");
18        }
19        if(value1 <= value2) {
20            System.out.println("value1 <= value2");
21        }
22    }
23}
```

→ Igualdad, Relacionales y condicionales

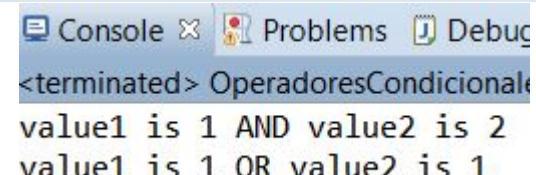
Los operadores `&&` y `||` realizan operaciones condicional-AND y condicional-OR en dos expresiones booleanas. Estos operadores exhiben un comportamiento de "cortocircuito", lo que significa que el segundo operando se evalúa sólo si es necesario.

`&&` Conditional-AND

`||` Conditional-OR

Igualdad, Relacionales y condicionales

```
1 package operadores;
2
3 public class OperadoresCondicionales1 {
4     public static void main(String[] args){
5         int value1 = 1;
6         int value2 = 2;
7         if((value1 == 1) && (value2 == 2)) {
8             System.out.println("value1 is 1 AND value2 is 2");
9         }
10        if((value1 == 1) || (value2 == 1)) {
11            System.out.println("value1 is 1 OR value2 is 1");
12        }
13    }
14 }
```



Console × Problems Debug
<terminated> OperadoresCondicionales1
value1 is 1 AND value2 is 2
value1 is 1 OR value2 is 1

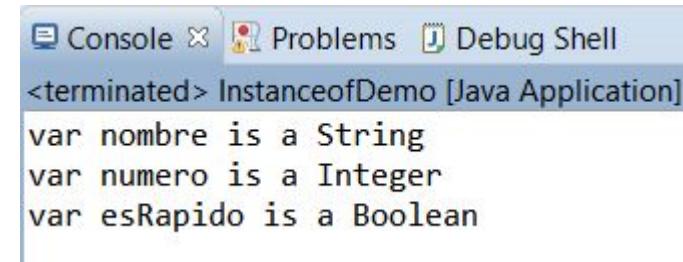
Operador instanceof

Este operador compara un objeto con un tipo especificado. Puede usarlo para probar si un objeto es una instancia de una clase, una instancia de una subclase o una instancia de una clase que implementa una interfaz particular.

The Java™ Tutorials

Operador instanseof

```
1 package operadores;
2
3 public class InstanceofDemo {
4     public static void main(String[] args) {
5
6         String nombre = "heroe";
7         Integer numero = 1234;
8         Boolean esRapido = true;
9
10        if ((nombre instanceof String)) {
11            System.out.println("var nombre is a String");
12        }
13
14        if (numero instanceof Integer) {
15            System.out.println("var numero is a Integer");
16        }
17
18        if (esRapido instanceof Boolean) {
19            System.out.println("var esRapido is a Boolean")
20        }
21
22    }
23 }
```



Terminamos los operadores





SENTENCIAS DE CONTROL

Sentencias de control

Las declaraciones dentro de sus archivos fuente generalmente se ejecutan de arriba a abajo, en el orden en que aparecen. Las declaraciones de flujo de control, sin embargo, dividen el flujo de ejecución al emplear la toma de decisiones, el bucle y la ramificación, lo que permite que su programa ejecuta condicionalmente bloques de código particulares. Esta sección describe las declaraciones de toma de decisiones (if-then, if-then-else, switch), las declaraciones de bucle (for, while, do-while) y las declaraciones de ramificación (break, continue, return) compatibles con Java lenguaje de programación.

Sentencia IF

La declaración **if** es la más básica de todas las declaraciones de flujo de control. Le dice a su programa que ejecute una determinada sección de código sólo si una prueba en particular se evalúa como verdadera:

```
if(condicion) {  
    // ejecuto la acción si condicion es verdadera  
}
```

Si la acción a realizar es de solo una línea, es posible omitir las llaves, tal como en el ejemplo siguiente:

```
if (esRapido instanceof Boolean) System.out.println("var esRapido is a Boolean");
```

Sentencia IF ejercicios



Ejercicio 1

Escribe un programa que pida por teclado un día de la semana y que diga qué asignatura toca a primera hora ese día.

ahora con SWITCH



Sentencia IF ejercicios



Sentencia IF-ELSE

La instrucción if-else proporciona una ruta de ejecución secundaria cuando una cláusula "if" se evalúa como falsa.

```
if (esRapido instanceof Boolean) {  
    System.out.println("var esRapido is a Boolean");  
}else {  
    System.out.println("var esRapido is not a Boolean");  
}
```

Es posible encadenar if-else-if en caso de ser necesario

Console x Pro
<terminated> IfElseDe
Grade = C

Sentencia IF-ELSE

```
1 package sentenciasdecontrol;  
2  
3 public class IfElseDemo {  
4     public static void main(String[] args) {  
5  
6         int testscore = 76;  
7         char grade;  
8  
9         if (testscore >= 90) {  
10             grade = 'A';  
11         } else if (testscore >= 80) {  
12             grade = 'B';  
13         } else if (testscore >= 70) {  
14             grade = 'C';  
15         } else if (testscore >= 60) {  
16             grade = 'D';  
17         } else {  
18             grade = 'F';  
19         }  
20         System.out.println("Grade = " + grade);  
21     }  
22 }
```



Sentencia SWITCH

A diferencia de las instrucciones if y if-else, la instrucción switch puede tener varias rutas de ejecución posibles. Un comutador funciona con los tipos de datos primitivos byte, short, char e int, la clase String y algunas clases especiales que envuelven ciertos tipos primitivos: Carácter, Byte, Corto e Entero (discutido en Números y cadenas).

22
23
24
25
26
27
28
29
30
31
32

```
int numero = 10;
switch(numero) {
    case 1:{
        System.out.println("el numero es 1");
        break;
    }
    default:{
        System.out.println("no esta el numero esperado");
    }
}
```

funciona con los tipos de datos primitivos byte, short, char, int y String

break; impide que la ejecución continúe evaluando.

default; solo si ninguna condición se cumple, se ejecuta.



Sentencia SWITCH

```
3 public class SwitchDemo2 {  
4     public static void main(String...args) {  
5         int month = 8;  
6         String monthString;  
7         switch (month) {  
8             case 1: monthString = "Enero";  
9                 break;  
10            case 2: monthString = "Febrero";  
11                break;  
12            case 3: monthString = "Marzo";  
13                break;  
14            case 4: monthString = "Abril";  
15                break;  
16            case 5: monthString = "Mayo";  
17                break;  
18            case 6: monthString = "Junio";  
19                break;  
20            case 7: monthString = "Julio";  
21                break;  
22            case 8: monthString = "Agosto";  
23                break;  
24            default: monthString = "Invalid month";  
25                break;  
26        }  
27        System.out.println(monthString);  
28    }  
29 }
```

Sentencia WHILE

La instrucción while ejecuta continuamente un bloque de instrucciones mientras una condición particular es verdadera. Su sintaxis se puede expresar como:

```
1 package sentenciasdecontrol;
2
3 public class WhileDemo {
4     public static void main(String[] args){
5         int count = 1;
6         while (count < 11) {
7             System.out.println("Contador es: " + count);
8             count++;
9         }
10    }
11 }
```

Sentencia DO-WHILE

También podemos utilizar la sentencia DO-WHILE. Su sintaxis se puede expresar como:

```
1 package sentenciasdecontrol;
2
3 public class DoWhileDemo {
4     public static void main(String[] args){
5         int count = 1;
6         do{
7             System.out.println("Contador es: " + count);
8             count++;
9         }while (count < 11);
10    }
11 }
```

Sentencia FOR

La instrucción for proporciona una forma compacta de iterar sobre un rango de valores. Los programadores a menudo se refieren a él como el "ciclo for" debido a la forma en que repite repetidamente hasta que se cumple una condición particular. La forma general de la declaración for se puede expresar de la siguiente manera:

```
1 package sentenciasdecontrol;  
2  
3 public class ForDemo {  
4     public static void main(String[] args){  
5         for (int i=0; i<11; i++) {  
6             System.out.println("Contador es: " + i);  
7         }  
8     }  
9 }
```

Sentencia FOR

```
1 package sentenciasdecontrol;  
2  
3 public class ForDemo {  
4     public static void main(String[] args){  
5         for (int i=0; i<11; i++) {  
6             System.out.println("Contador es: " + i);  
7             break;  
8         }  
9     }  
10 }
```

La sentencia break termina el ciclo.



CONCEPTOS POO.

Que es un objeto

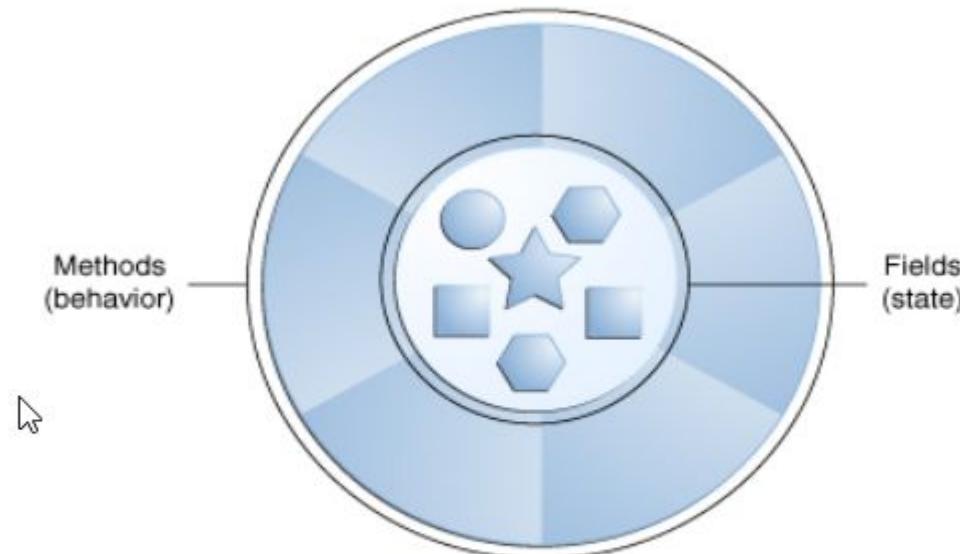




Que es un objeto

Tómese un minuto ahora mismo para observar los objetos del mundo real que se encuentran en su área inmediata. Para cada objeto que vea, hágase dos preguntas: "¿Cuáles son las características que identifican a cada objeto ?" y "¿Qué comportamiento posible puede realizar este objeto?". Asegúrese de anotar sus observaciones. Mientras lo hace, notará que los objetos del mundo real varían en complejidad; su lámpara de escritorio puede tener solo dos atributos (encendido y apagado) y dos comportamientos posibles (encender, apagar), pero su radio de escritorio puede tener estados atributos adicionales (encendido, apagado, volumen actual, estación actual) y comportamiento (encender , apagar, aumentar el volumen, disminuir el volumen, buscar, escanear y sintonizar). También puede notar que algunos objetos, a su vez, también contendrán otros objetos. Todas estas observaciones del mundo real se traducen en el mundo de la programación orientada a objetos.

The Java™ Tutorials



A software object.

The Java™ Tutorials

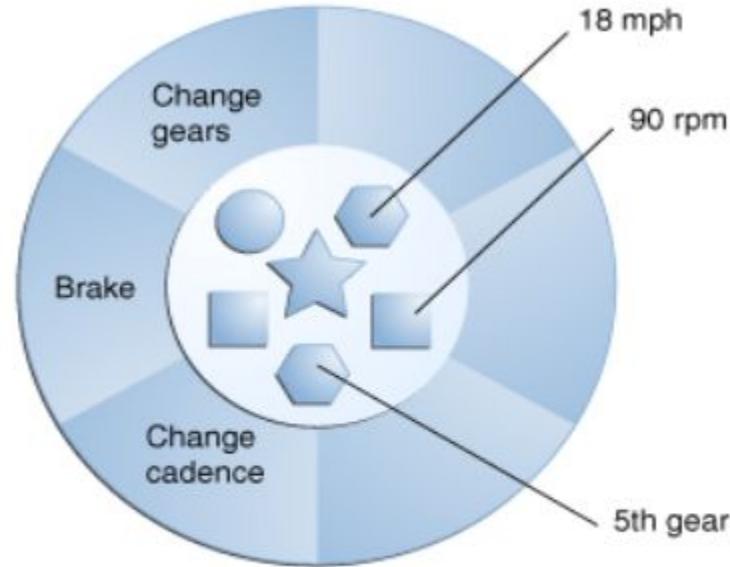


Los objetos de software son conceptualmente similares a los objetos del mundo real: también consisten en estado y comportamiento relacionado. Un objeto almacena su estado en atributos(variables en algunos lenguajes de programación) y expone su comportamiento a través de métodos (funciones en algunos lenguajes de programación).

Los métodos operan en el estado interno de un objeto y sirven como el mecanismo principal para la comunicación de objeto a objeto. Ocultar el estado interno y requerir que toda la interacción se realice a través de los métodos de un objeto se conoce como encapsulación de datos, un principio fundamental de la programación orientada a objetos.

Para llevar esto a la práctica, pensemos en una bicicleta.

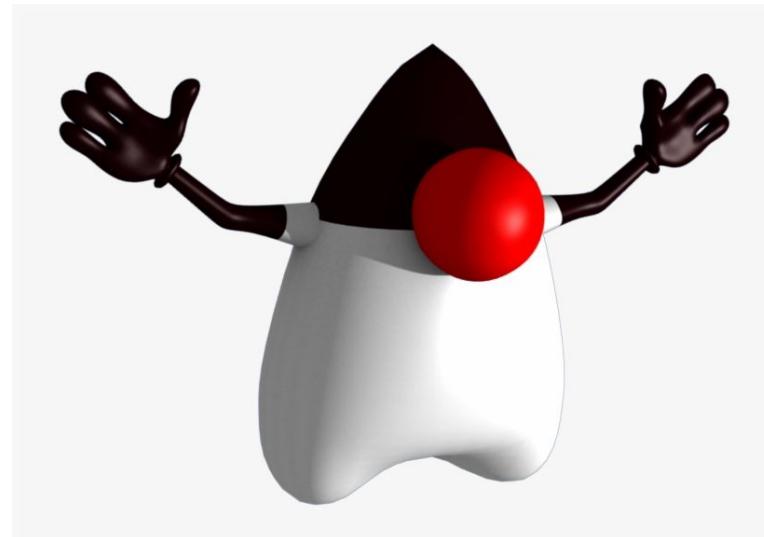
The Java™ Tutorials



A bicycle modeled as a software object.



Al atribuir los atributos (velocidad actual, cadencia actual del pedal y marcha actual) y proporcionar métodos para cambiar el valor de estos atributos, el objeto mantiene el control de cómo el mundo exterior puede usarlo. Por ejemplo, si la bicicleta solo tiene 6 marchas, un método para cambiar de marchas podría rechazar cualquier valor que sea menor que 1 o mayor que 6.



Que es una Clase

En el mundo real, a menudo encontrarás muchos objetos individuales, todos del mismo tipo. Puede haber miles de otras bicicletas en existencia, todas de la misma marca y modelo. Cada bicicleta se construyó con el mismo conjunto de planos y, por lo tanto, contiene los mismos componentes. En términos orientados a objetos, decimos que su bicicleta **es una instancia de la clase de objetos conocidos como bicicletas**. Una clase es el plano a partir del cual se crean los objetos individuales.

Con estos conceptos en mente, empecemos a crear bicicletas con JAVA...



The Java™ Tutorials



The Java™ Tutorials

```
1 package orientacionobjetos;
2
3 public class Bicycle {
4     int cadence = 0;
5     int speed = 0;
6     int gear = 1;
7
8     void changeCadence(int newValue) {
9         cadence = newValue;
10    }
11
12    void changeGear(int newValue) {
13        gear = newValue;
14    }
15
16    void speedUp(int increment) {
17        speed = speed + increment;
18    }
19
20    void applyBrakes(int decrement) {
21        speed = speed - decrement;
22    }
23
24    void printStates() {
25        System.out.println("cadence:" + cadence + " speed:" + speed + " gear:" + gear);
26    }
27 }
```

The Java™ Tutorials



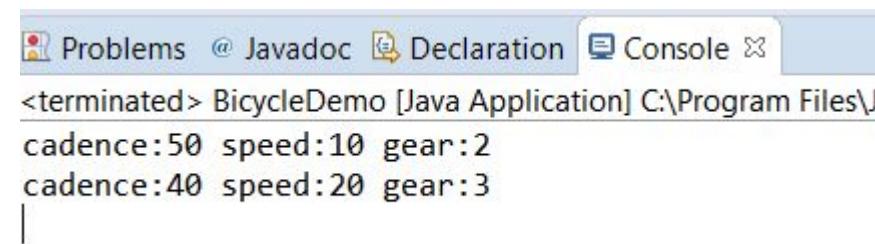
La sintaxis del lenguaje de programación Java le parecerá nueva, pero el diseño de esta clase se basa en la discusión previa de los objetos de bicicleta. Los campos cadence, speed y gear representan el estado del objeto, y los métodos (changeCadence, changeGear, speedUp, etc.) definen su interacción con el mundo exterior.

Es posible que haya notado que la clase de bicicletas no contiene un método principal. Eso es porque no es una aplicación completa; Es solo el modelo para las bicicletas que podrían usarse en una aplicación. La responsabilidad de crear y usar nuevos objetos Bicycle pertenece a otra clase en su aplicación.

Aquí hay una clase Bicycle Demo que crea dos objetos Bicycle diferentes e invoca sus métodos:

The Java™ Tutorials

```
1 package orientacionobjetos;
2
3 public class BicycleDemo {
4     public static void main(String[] args) {
5
6         // Create two different Bicycle objects
7         Bicycle bike1 = new Bicycle();
8         Bicycle bike2 = new Bicycle();
9
10        // Invoke methods on those objects
11        bike1.changeCadence(50);
12        bike1.speedUp(10);
13        bike1.changeGear(2);
14        bike1.printStates();
15
16        bike2.changeCadence(50);
17        bike2.speedUp(10);
18        bike2.changeGear(2);
19        bike2.changeCadence(40);
20        bike2.speedUp(10);
21        bike2.changeGear(3);
22        bike2.printStates();
23    }
24 }
```



```
Problems @ Javadoc Declaration Console
<terminated> BicycleDemo [Java Application] C:\Program Files\Java\BicycleDemo.java
cadence:50 speed:10 gear:2
cadence:40 speed:20 gear:3
```

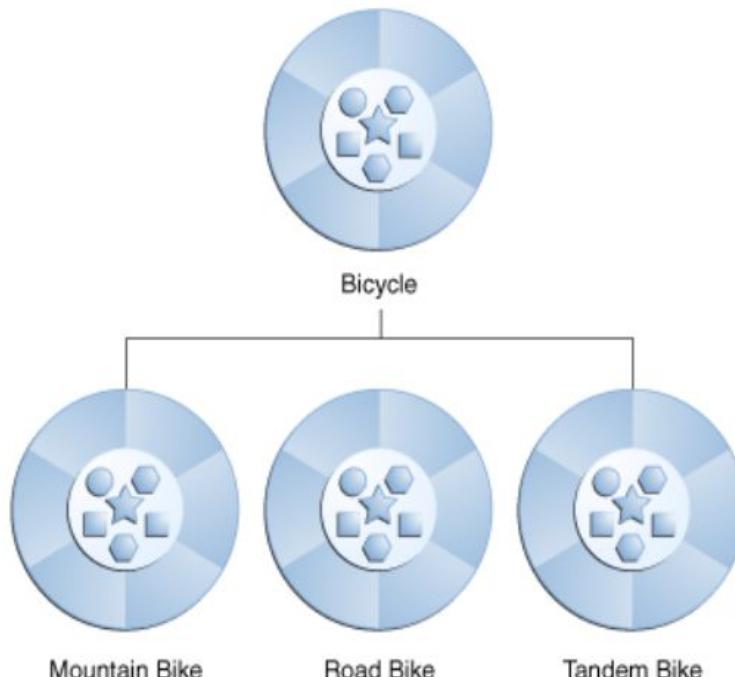


Que es la herencia

Los diferentes tipos de objetos a menudo tienen una cierta cantidad de atributos en común entre sí. Las bicicletas de montaña, las bicicletas de carretera y las bicicletas tandem, por ejemplo, comparten las características de las bicicletas (velocidad actual, cadencia actual de los pedales, equipo actual). Sin embargo, cada uno también define características adicionales que los hacen diferentes: las bicicletas tandem tienen dos asientos y dos juegos de manubrios; las bicicletas de carretera tienen manillares de caída; Algunas bicicletas de montaña tienen un anillo de cadena adicional, lo que les da una relación de transmisión más baja.



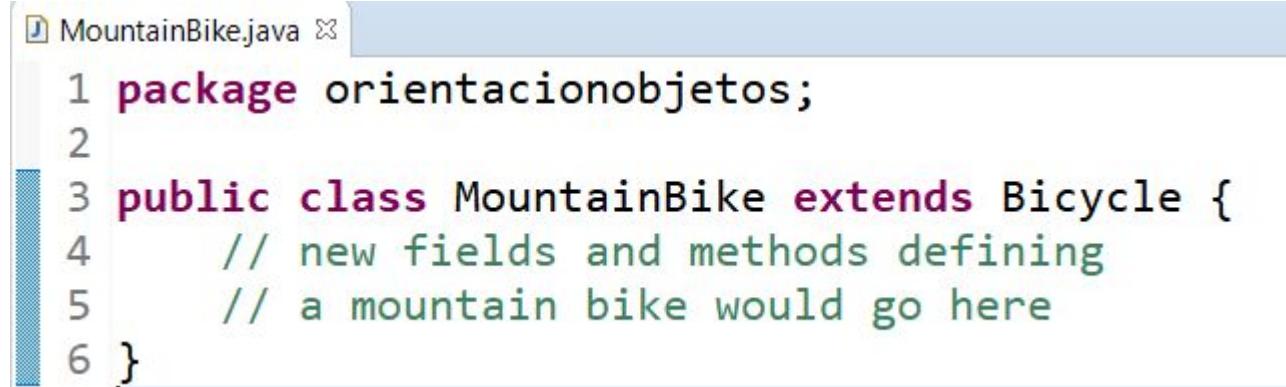
The Java™ Tutorials



A hierarchy of bicycle classes.

La sintaxis para crear una clase que herede otra es muy sencilla. Basta con acompañar el nombre de la clase con la palabra clave **extends**.

The Java™ Tutorials



A screenshot of a Java code editor showing a file named `MountainBike.java`. The code is as follows:

```
1 package orientacionobjetos;
2
3 public class MountainBike extends Bicycle {
4     // new fields and methods defining
5     // a mountain bike would go here
6 }
```

Esto le da a `MountainBike` los mismos campos y métodos que `Bicycle`, pero permite que su código se centre exclusivamente en las características que lo hacen único. Esto hace que el código de sus subclases sea fácil de leer. Sin embargo, debe tener cuidado de documentar adecuadamente el estado y el comportamiento que define cada superclase, ya que ese código no aparecerá en el archivo fuente de cada subclase.



Que es una interface

Como ya has aprendido, los objetos definen su interacción con el mundo exterior a través de los métodos que exponen. Los métodos forman la interfaz del objeto con el mundo exterior; Los botones en la parte frontal de su televisor, por ejemplo, son la interfaz entre usted y el cableado eléctrico en el otro lado de su carcasa de plástico. Presiona el botón de "encendido" para encender y apagar el televisor.

En su forma más común, una interfaz es un grupo de métodos relacionados con cuerpos vacíos. El comportamiento de una bicicleta, si se especifica como una interfaz, puede aparecer de la siguiente manera:

Que es una interface

```
1 package orientacionobjetos;
2
3 public interface BicycleInterface {
4 //  wheel revolutions per minute
5     void changeCadence(int newValue);
6
7     void changeGear(int newValue);
8
9     void speedUp(int increment);
10
11    void applyBrakes(int decrement);
12 }
```

Que es una interface

```
1 package orientacionobjetos;
2
3 public class Bicycle implements BicycleInterface{
4     int cadence = 0;
5     int speed = 0;
6     int gear = 1;
7
8     public void changeCadence(int newValue) {
9         cadence = newValue;
10    }
11
12    public void changeGear(int newValue) {
13        gear = newValue;
14    }
15
16    public void speedUp(int increment) {
17        speed = speed + increment;
18    }
19
20    public void applyBrakes(int decrement) {
21        speed = speed - decrement;
22    }
23
24    void printStates() {
25        System.out.println("cadence:" + cadence + " speed:" + speed + " gear:" + gear);
26    }
27 }
```



Que es una interface

La implementación de una interfaz permite que una clase se vuelva más formal sobre el comportamiento que promete proporcionar. Las interfaces forman un contrato entre la clase y el mundo exterior, y el compilador hace cumplir este contrato en el momento de la construcción. Si su clase afirma implementar una interfaz, todos los métodos definidos por esa interfaz deben aparecer en su código fuente antes de que la clase se compile correctamente.



API JAVA CLASE STRING



La clase STRING

Strings de Java son objetos, en contraposición a los tipos primitivos, que pueden ser utilizados para representar los caracteres y números. Esto significa que todas las instancias de String creadas dentro de un programa Java tienen acceso a los métodos descritos dentro de dicha clase. Los desarrolladores por lo tanto pueden llamar a los métodos String en sus instancias de clases String. A menudo son los primeros tipos de objetos utilizados por los desarrolladores que están en las primeras etapas del aprendizaje. Los métodos y propiedades de la clase String se describen en la Especificación Oficial de Lenguajes de Java.

Métodos de String

MÉTODO	DESCRIPCIÓN	PARÁMETRO	TIPO DE DATO DEVUELTO
charAt	Devuelve el carácter indicado por parámetro	Un parámetro int	char
compareTo	Sirve para comparar cadenas, devuelve un número según el resultado. Recuerda que no sigue el alfabeto español, lo compara según la tabla ASCII.	Un parámetro String, la cadena a comparar.	int - Si devuelve un número mayor que 0: la primera cadena es mayor que la segunda. - Si devuelve un 0: las cadenas son iguales. - Si devuelve un número menor que 0: la primera cadena es menor que la segunda
compareToIgnoreCase	Es igual que el anterior, pero ignorando mayúsculas o minúsculas.	Un parámetro String, la cadena a comparar	int - Si devuelve un número mayor que 0: la primera cadena es mayor que la segunda. - Si devuelve un 0: las cadenas son iguales. - Si devuelve un número menor que 0: la primera cadena es menor que la segunda
concat	Concatena dos cadenas, es como el operador +.	Un parámetro String, la cadena a concatenar	Un nuevo String con las cadenas concatenadas.
copyValueOf	Crea un nuevo String a partir de un array de char. Este método debe invocarse de manera estática, es decir, String.copyValueOf(array_char)	Un array de char	String
endsWith	Indica si la cadena acaba con el String pasado por parámetro.	String	boolean
equals	Indica si una cadena es igual que otra.	String	boolean
equalsIgnoreCase	Es igual que el anterior, pero ignorando mayúsculas o minúsculas.	String	boolean
getBytes	Devuelve un array de bytes con el código ASCII de los caracteres que forman el String.	Ningún parámetro	Un array de bytes



Métodos de String

indexOf	Devuelve la posición en la cadena pasada por parámetro desde el principio. -1 si no existe.	String o char	int
indexOf	Igual que el anterior, pero ademas le indicamos la posición desde donde empezamos a buscar.	String o char, el segundo parámetro es un int	int
lastIndexOf	Devuelve la posición en la cadena pasada por parámetro desde el final. -1 si no existe.	String o char	int
lastIndexOf	Igual que el anterior, pero ademas le indicamos la posición desde donde empezamos a buscar.	String o char, el segundo parámetro es un int	int
length	Devuelve la longitud de la cadena.	Ningún parámetro	int
matches	Indica si la cadena cumple con la expresión pasada como parámetro. Pincha aquí para tener mas detalles.	String	boolean
replace	Devuelve un String cambiando los caracteres que nosotros le indiquemos.	Dos parámetros char, el primero es el carácter que existe en el String y el segundo por el que queremos cambiar.	String

Activate Window

F10 to switch windows



Métodos de String

replaceAll	Devuelve un String intercambiando todas las coincidencias que se encuentren.	Dos parametros String, el primero son los caracteres que existe en el String y el segundo por el que queremos cambiar.	String
startsWith	Indica si la cadena empieza por una cadena pasada por parámetro.	String	boolean
substring	Trocea un String desde una posición a otra.	Dos parámetros int, indica desde donde empieza hasta donde acaba, este ultimo no se incluye.	String
toCharArray	Devuelve en un array de char, todos los caracteres de una String.	Ningún parámetro	Un array de char
toLowerCase	Convierte el String a minúsculas.	Ningún parámetro	String
toUpperCase	Convierte el String a mayúsculas.	Ningún parámetro	String
trim	Elimina los espacios del String.	Ningún parámetro	String
valueOf	Transforma una variable primitiva en un String. Para invocarse debe usarse con String. Por ejemplo, String.valueOf(variable)	Un parámetro, que puede ser un: <ul style="list-style-type: none">▪ boolean▪ char▪ double▪ int▪ float▪ long▪ Array de char▪ Referencia a un objeto	String

Métodos de String

Crear una variable String y usarla no significa que manejamos la clase String. Para lograr esto tenemos que comprender su funcionamiento.

Sabemos que la forma de crear un String es:

```
String greeting = "Hello world!";
```

En este caso, "Hello world!" es un literal de cadena: una serie de caracteres en su código que se encuentra entre comillas dobles. Cada vez que encuentra un literal de cadena en su código, el compilador crea un objeto de cadena con su valor, en este caso, ¡Hola, mundo!.

Métodos de String

Al igual que con cualquier otro objeto, puede crear objetos String utilizando la palabra clave 'new' y un constructor. La clase String tiene trece constructores que le permiten proporcionar el valor inicial de la cadena utilizando diferentes fuentes, como una matriz de caracteres:

```
1 package com.santotomas.apijava.string;
2
3 public class Ejemplo01 {
4     public static void main(String...args) {
5         char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
6         String helloString = new String(helloArray);
7         System.out.println(helloString);
8     }
9 }
0 |
```

Método Length

Los métodos utilizados para obtener información sobre un objeto se conocen como métodos de acceso. Un método de acceso que puede usar con cadenas es el método `length()`, que devuelve el número de caracteres contenidos en el objeto de cadena. Después de que se hayan ejecutado las siguientes dos líneas de código, `len` es igual a 17:

```
1 package com.santotomas.apijava.string;
2
3 public class Ejemplo2 {
4     public static void main(String...args) {
5         String palindrome = "Dot saw I was Tod";
6         int len = palindrome.length();
7         System.out.println("cant of char " + len);
8     }
9 }
```



Problems @ Javadoc Console ×
<terminated> Ejemplo2 [Java Application]
cant of char 17



Método Length

Un palíndromo es una palabra u oración que es simétrica: se escribe igual hacia adelante y hacia atrás, ignorando mayúsculas y minúsculas. Aquí hay un programa corto e ineficiente para revertir una cadena de palíndromo. Invoca el método de cadena `charAt (i)`, que devuelve el *i*-ésimo carácter en la cadena, contando desde 0.

Método Length

```
1 package com.santotomas.apijava.string;
2
3 public class Ejemplo3 {
4     public static void main(String[] args) {
5         String palindrome = "Dot saw I was Tod";
6         int len = palindrome.length();
7         char[] tempCharArray = new char[len];
8         char[] charArray = new char[len];
9
10        // put original string in an
11        // array of chars
12        for (int i = 0; i < len; i++) {
13            tempCharArray[i] =
14                palindrome.charAt(i);
15        }
16
17        // reverse array of chars
18        for (int j = 0; j < len; j++) {
19            charArray[j] =
20                tempCharArray[len - 1 - j];
21        }
22
23        String reversePalindrome =
24            new String(charArray);
25        System.out.println(reversePalindrome);
26    }
27 }
```



The screenshot shows a Java application window with tabs for 'Problems', '@ Javadoc', and 'Console'. The 'Console' tab is active, displaying the output of the application. The output text is: <terminated> Ejemplo3 [Java Application] C:\Program dot saw I was toD



Método Length

Para lograr la inversión de la cadena, el programa tuvo que convertir la cadena en una matriz de caracteres (primero para el bucle), invertir la matriz en una segunda matriz (segundo para el bucle) y luego volver a convertirla en una cadena. La clase String incluye un método, `getChars()`, para convertir una cadena, o una parte de una cadena, en una matriz de caracteres para que podamos reemplazar el primer bucle `for` en el programa anterior con:

```
// put original string in an
// array of chars
palindrome.getChars(0, len, tempCharArray, 0);
```

```
1 package com.santotomas.apijava.string;
2
3 public class Ejemplo4 {
4     public static void main(String[] args) {
5         String palindrome = "Dot saw I was Tod";
6         int len = palindrome.length();
7         char[] tempCharArray = new char[len];
8         char[] charArray = new char[len];
9
10        // put original string in an
11        // array of chars
12        palindrome.getChars(0, len, tempCharArray, 0);
13
14        // reverse array of chars
15        for (int j = 0; j < len; j++) {
16            charArray[j] =
17                tempCharArray[len - 1 - j];
18        }
19
20        String reversePalindrome =
21            new String(charArray);
22        System.out.println(reversePalindrome);
23    }
24 }
```

Método Length



Problems @ Javadoc Console

<terminated> Ejemplo3 [Java Application] C:\Program

dot saw I was toD

Concatenar String

La clase String incluye un método para concatenar cadenas

```
string1.concat(string2);
```

↙

Esto devuelve una nueva cadena que es string1 y string2 agregada al final. También puede usar el método concat () con literales de cadena, como en:

```
"My name is ".concat("Rumplestiltskin");
```



API JAVA CLASE MATH



La clase Math

La clase MATH contiene métodos que permiten ejecutar distintas operaciones matemáticas como números exponenciales, logaritmos y funciones trigonométricas. Si bien no entraremos en mucho detalle es importante darles un vistazo ya que son importantes para los usos matemáticos.



La clase Math

```
1 package com.santotomas.apijava.math;
2
3 /**
4  * Demostración de la clase Math, la cual provee de muchos metodos
5  * para generar calculos matematicos. Es muy facil utilizar estos metodos los cuales
6  * son todos estaticos. En este ejemplo se muestran los metodos para
7  * generar el numero PI, obtener potencias, redondear numeros y calcular la raiz cuadrada.
8  *
9  * @since 1.0
10 * @see java.Math
11 */
12 public class Ejemplo01 {
13     public static void main(String...args) {
14         //devuelve el numero PI
15         System.out.println("El numero PI:" + Math.PI);
16
17         // potencia 2 al cuadrado
18         System.out.println("Saca la potencia de 2 al cuadrado: " +Math.pow(2,2));
19
20         //redondea el numero
21         System.out.println("Redondedo de numeros: " +Math.round(2.34));
22
23         //raiz cuadrada
24         System.out.println("Raiz Cuadrada: " + Math.sqrt(16));
25
26     }
27 }
```



La clase Math

Una pequeña muestra de la cantidad de métodos disponibles. Para ver la lista completa hay que consultar el api java en la sección de la clase Math.

Method Summary	
All Methods	Static Methods
Modifier and Type	Method and Description
static double	abs(double a) Returns the absolute value of a double value.
static float	abs(float a) Returns the absolute value of a float value.
static int	abs(int a) Returns the absolute value of an int value.
static long	abs(long a) Returns the absolute value of a long value.
static double	acos(double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through <i>pi</i> .
static int	addExact(int x, int y) Returns the sum of its arguments, throwing an exception if the result overflows an int.
static long	addExact(long x, long y) Returns the sum of its arguments, throwing an exception if the result overflows a long.
static double	asin(double a) Returns the arc sine of a value; the returned angle is in the range - <i>pi/2</i> through <i>pi/2</i> .
static double	atan(double a) Returns the arc tangent of a value; the returned angle is in the range - <i>pi/2</i> through <i>pi/2</i> .
static double	atan2(double y, double x) Returns the angle <i>theta</i> from the conversion of rectangular coordinates (<i>x, y</i>) to polar coordinates (<i>r, theta</i>).
static double	cbrt(double a) Returns the cube root of a double value.



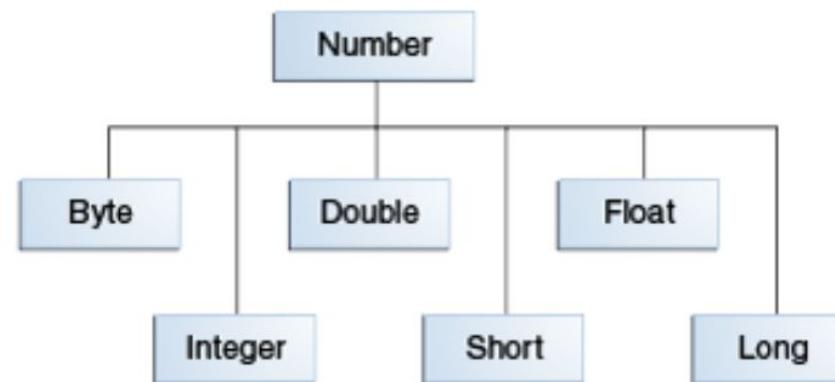
API JAVA CLASE INTEGER



La clase Integer

La clase Integer envuelve un tipo de dato primitivo de tipo int en una clase. Esto significa que tienes disponible métodos de la clase que puedes usar para dar ciertas funcionalidades y características a un número. Con la clase Integer es posible por ejemplo transformar un número entero a String o un String a un int.

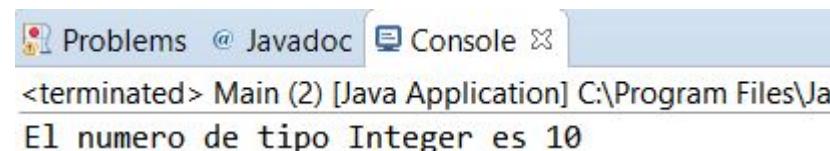
La clase Integer hereda de la clase Number.





La clase Integer-Constructores

```
/**  
 * La clase Integer cuenta con distintos constructores. El primero de todos  
 * esta compuesto por el Tipo de dato (Integer) y el numero que recibe como parametro.  
 * @param num correspondiente a algun numero.  
 */  
public void creacionInteger() {  
    Integer saldo = new Integer(10); // creación de un objeto de tipo Integer.  
    System.out.println("El numero de tipo Integer es " + saldo);  
}
```

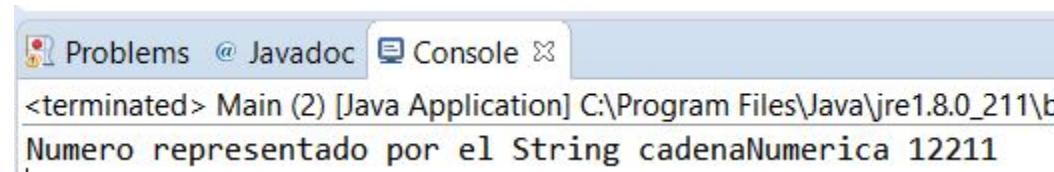


The screenshot shows a Java application window with two tabs: 'Problems' and 'Console'. The 'Console' tab is active, displaying the output of a program. The output text is: '<terminated> Main (2) [Java Application] C:\Program Files\Java\... El numero de tipo Integer es 10'.



La clase Integer-Constructores

```
/**  
 * Este constructor recibe un String, el cual transformara a su representación  
 * numerica. En caso de error lanza una exception NumberFormatException  
 * @param  String: cadena correspondiente a algun numero.  
 */  
public void creacionIntegerDesdeUnString(String numeroCadena) {  
    Integer saldo = new Integer(numeroCadena);  
    System.out.println("Numero representado por el String cadenaNumerica " + saldo);  
}
```

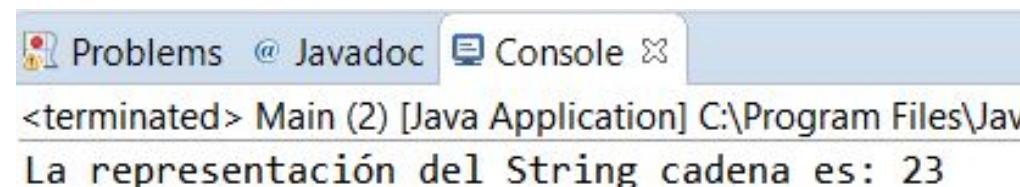


The screenshot shows a Java application window with three tabs: 'Problems', '@ Javadoc', and 'Console'. The 'Console' tab is active, displaying the output of a program. The output text is: '<terminated> Main (2) [Java Application] C:\Program Files\Java\jre1.8.0_211\b' followed by 'Numero representado por el String cadenaNumerica 12211'.



La clase Integer-Métodos

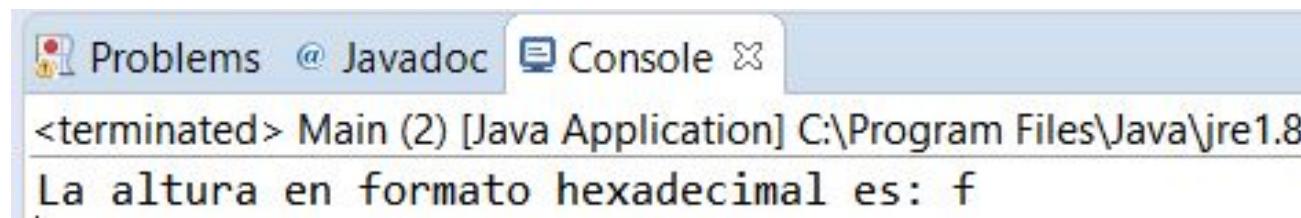
```
/**  
 * El metodo toString devuelve un numero en formato String.  
 */  
public void pruebaToString(int numero){  
    Integer edad = new Integer(numero);  
    String cadena = edad.toString();  
    System.out.println("La representación del String cadena es: " + cadena);  
}
```



The screenshot shows a Java application running in an IDE. The title bar of the window says "Problems @ Javadoc" and "Console". The console tab is active, showing the output of the application. The output text is: "<terminated> Main (2) [Java Application] C:\Program Files\Java La representación del String cadena es: 23".

La clase Integer-Método toHexString

```
/**  
 * Retorna un String en formato hexadecimal a partir de un integer  
 * @param numero  
 */  
public void pruebaTransformaHexadecimal(int numero){  
    Integer altura = new Integer(numero);  
    String hexadecimal = Integer.toHexString(altura);  
    System.out.println("La altura en formato hexadecimal es: " + hexadecimal);  
}
```



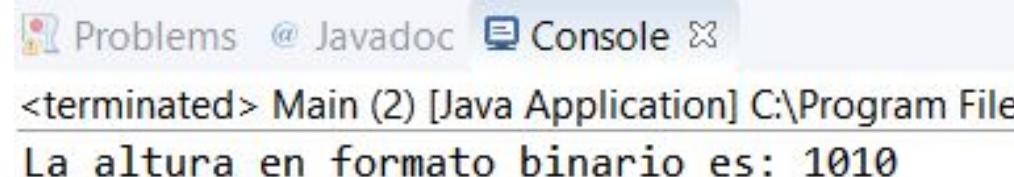
The screenshot shows a Java application window with three tabs: 'Problems', '@ Javadoc', and 'Console'. The 'Console' tab is active, displaying the output of the program. The output text is: '<terminated> Main (2) [Java Application] C:\Program Files\Java\jre1.8. La altura en formato hexadecimal es: f'. The text 'La altura en formato hexadecimal es: f' is highlighted in blue, indicating it is the output of the program's System.out.println statement.

La clase Integer-Método toOctalString

```
/**  
 * Retorna un String en formato octal a partir de un integer  
 * @param numero  
 */  
public void pruebaTransformaOctal(int numero){  
    Integer altura = new Integer(numero);  
    String octal = Integer.toOctalString(altura);  
    System.out.println("La altura en formato octal es: " + octal);  
}  
}
```

La clase Integer-Método toBinaryString

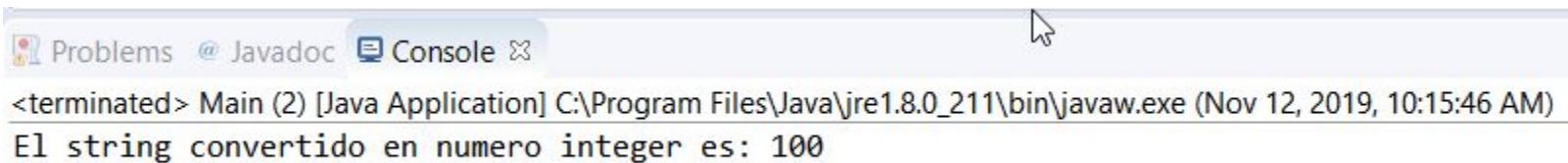
```
/*
 * Retorna un String correspondiente al valor numerico en formato binario
 * @param numero
 */
public void pruebaTransformaBinario(int numero){
    Integer altura = new Integer(numero);
    String binario = Integer.toBinaryString(altura);
    System.out.println("La altura en formato binario es: " + binario);
}
```



The screenshot shows a Java application window with three tabs: 'Problems', '@ Javadoc', and 'Console'. The 'Console' tab is active, displaying the output of the application. The output text is: '<terminated> Main (2) [Java Application] C:\Program File' followed by 'La altura en formato binario es: 1010'.

La clase Integer-Método valueOf()

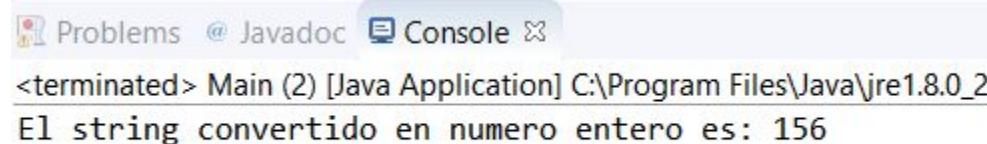
```
/**  
 * devuelve el objeto entero inicializado con el valor proporcionado  
 * @param numero  
 */  
public void pruebaValueOf(String numero){  
    System.out.println("El string convertido en numero integer es: " + Integer.valueOf(numero));  
}
```



The screenshot shows a Java application running in an IDE. The console tab is active, displaying the output of the application. The output text is: <terminated> Main (2) [Java Application] C:\Program Files\Java\jre1.8.0_211\bin\javaw.exe (Nov 12, 2019, 10:15:46 AM) El string convertido en numero integer es: 100

La clase Integer-Método parseInt()

```
/**  
 * devuelve el objeto entero inicializado con el valor String proporcionado  
 * @param numero  
 */  
public void pruebaParseInt(String numero){  
    int numeroEntero = Integer.parseInt(numero);  
    System.out.println("El string convertido en numero entero es: " + numeroEntero);  
}
```



Problems @ Javadoc Console ✎

<terminated> Main (2) [Java Application] C:\Program Files\Java\jre1.8.0_2
El string convertido en numero entero es: 156



ARREGLOS



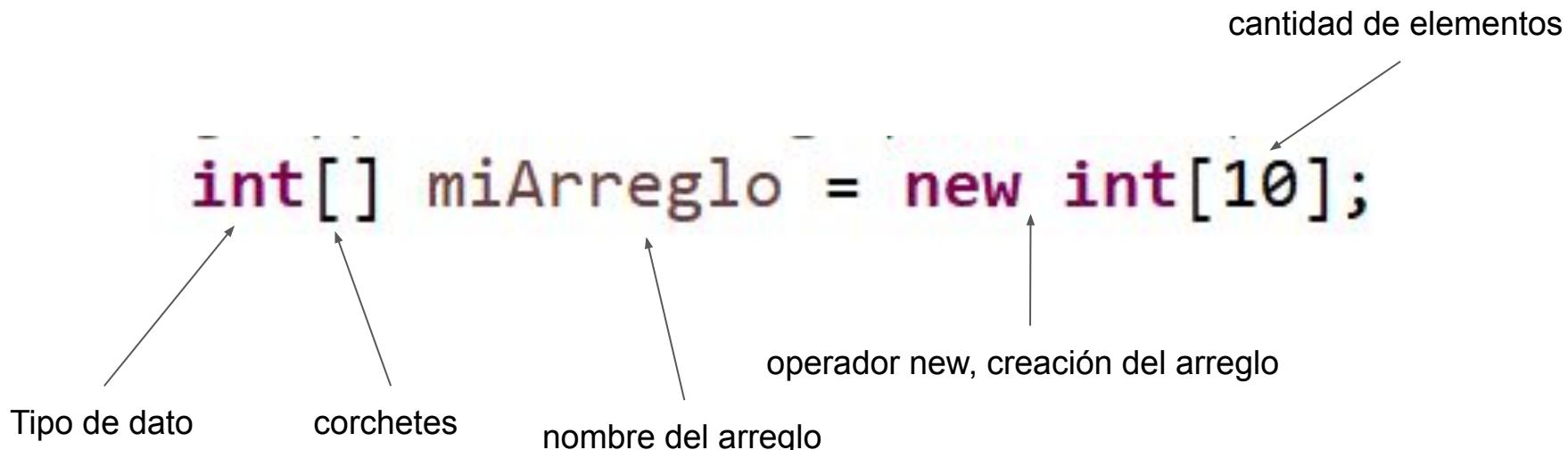
Que es un arreglo



Un arreglo es básicamente una estructura de datos que contiene valores de un mismo tipo. En ellos podemos almacenar valores que tienen una relación entre sí. Por ejemplo podemos pensar en los meses del año, o en los numeros de telefono, o en cualquier otro conjunto de valores que tengan la misma naturaleza.

Sintaxis

La sintaxis para declarar un arreglo es la siguiente:



The diagram shows the Java code for declaring an array: `int[] miArreglo = new int[10];`. Annotations with arrows explain the components:

- Tipo de dato** (Type): Points to the `int` in `int[]`.
- corchetes** (brackets): Points to the `[]` in `int[]`.
- nombre del arreglo** (array name): Points to `miArreglo`.
- operador new, creación del arreglo** (new operator, array creation): Points to `new int[10]`.
- cantidad de elementos** (number of elements): Points to the `10` in `new int[10]`.

Tipo de dato: Corresponde al tipo de elemento que se guardará en el arreglo. Pueden ser todos los tipos primitivos como int, char, byte, long, etc.

Corchetes: Símbolo que le indica al compilador que se creará un arreglo de tipo int. De hecho el símbolo que identifica a los arreglos son los corchetes.

Nombre del arreglo: Este es el nombre que le das a tu arreglo, no confundir con alguna palabra reservada, este nombre puede ser el que quieras, tanto como arregloSueldos como arregloPapitas.

Operador new: Con él crear el arreglo, al igual que creas un objeto de alguna clase.

Cantidad de elementos: Cuando creas un arreglo puedes asignar de antemano el tamaño total de elementos que podrá contener.

¿Ya pero que es un arreglo en la vida real?

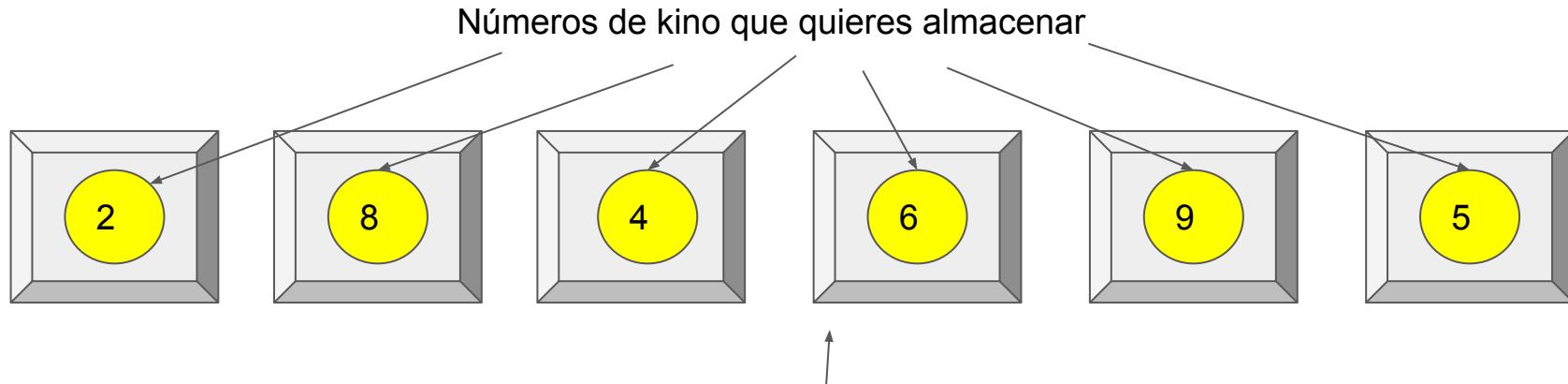
Bien, pensemos que queremos guardar los números que elegí en mi KINO en un arreglo. Todos conocen el KINO cierto?



```
//vamos a guardar solo los 6 primeros digitos del carton.  
int[] numerosKino = new int[6];
```

Tras bambalinas esto es lo que ocurre

```
//vamos a guardar solo los 6 primeros dígitos del cartón.  
int[] númerosKino = new int[6];
```



Arreglo numerosKino, con 6 casillas en donde guardar elementos

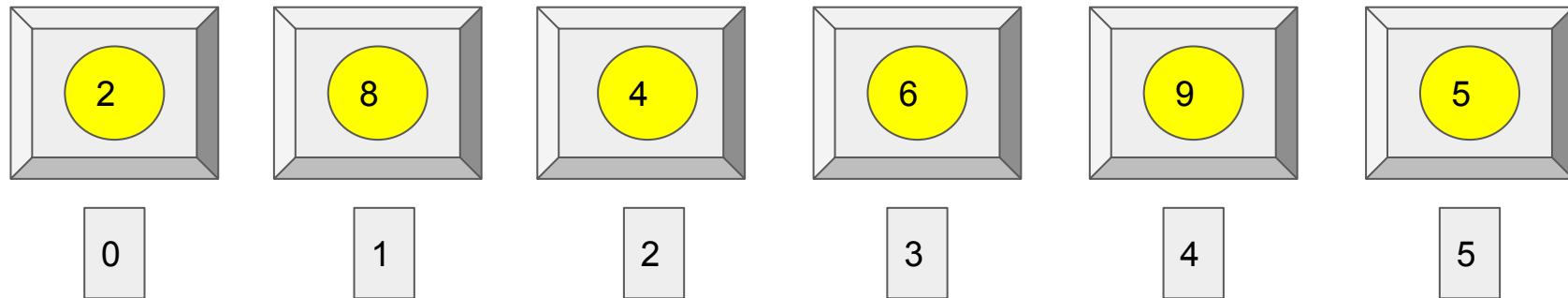
Si llevamos lo anterior a código:

```
/*
 * Metodo que crea un arreglo de 6 posiciones para guardar los numeros
 * del kino
 */
public void creacionArreglo(){
    //vamos a guardar solo los 6 primeros digitos del carton.
    int[] numerosKino = new int[6];
    //en cada casilla del arreglo guardamos un numero del kino
    numerosKino[0]=1;
    numerosKino[1]=2;
    numerosKino[2]=3;
    numerosKino[3]=4;
    numerosKino[4]=5;
    numerosKino[5]=6;
}
```



¿Qué pasa con el indice 3?

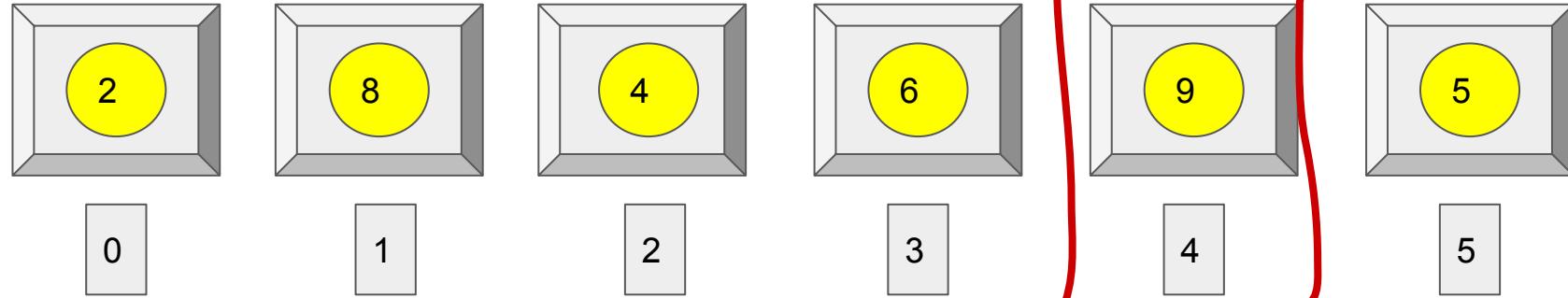
Si llevamos lo anterior a código:



Cada casilla de un arreglo cuenta con una posición que parte desde el cero. Por ejemplo el número 2 está en la casilla número 0, el número 8 está en la casilla número 1, etc.

Sintaxis

Si llevamos lo anterior a código:

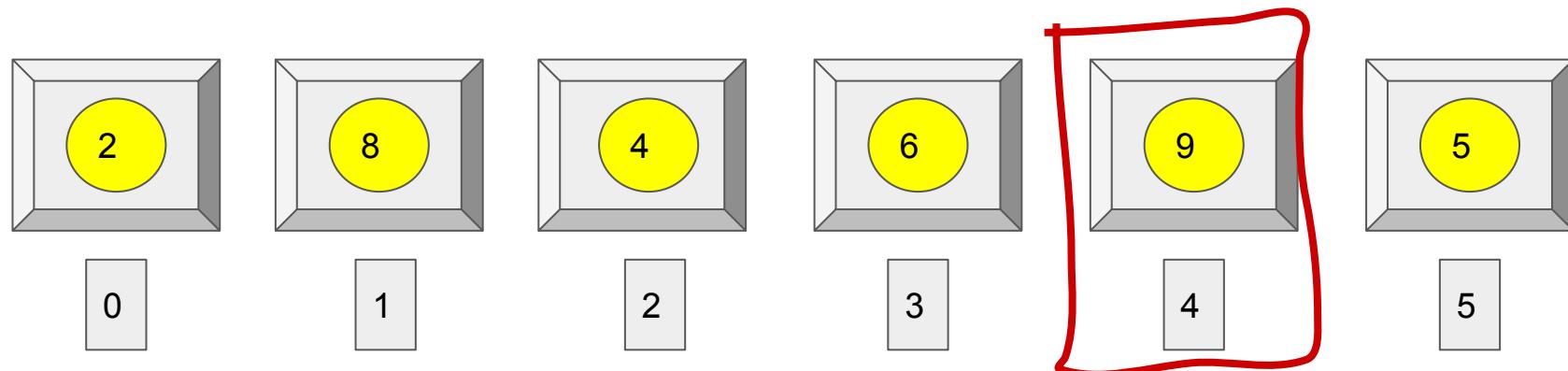


Para acceder a la casilla número 4 por ejemplo, debemos hacerlo mediante la sintaxis:

`numerosKino[4]`

Sintaxis

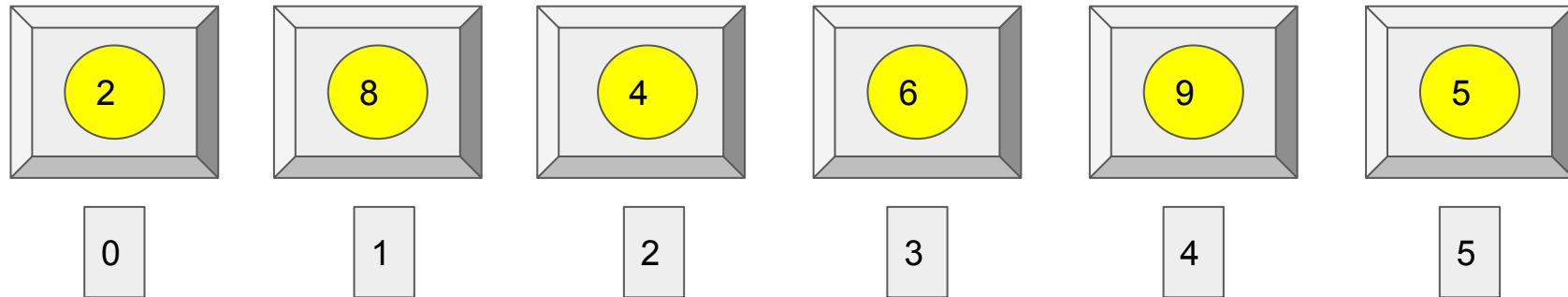
Si llevamos lo anterior a código:



Con esta acción estamos accediendo a la posición 4 del arreglo, y ahora podemos asignar un valor a tal casilla.

```
numerosKino[4]=9; // guardamos el N° 9 en esta posición
```

Perfecto, entonces podemos crear un arreglo y guardar valores a él mediante su posición.



Ahora debemos mostrar los elementos del arreglo. Como podrás pensar, si accedemos a la posición de la casilla para asignar un valor, podemos hacer lo mismo para mostrar el valor que está contenido en el.

```
/*
 * Metodo que crea un arreglo de 6 posiciones para guardar los numeros
 * del kino
 */
public void creacionArreglo(){
    //vamos a guardar solo los 6 primeros digitos del carton.
    int[] numerosKino = new int[6];
    //en cada casilla del arreglo guardamos un numero del kino
    numerosKino[0]=2;
    numerosKino[1]=8;
    numerosKino[2]=4;
    numerosKino[3]=6;
    numerosKino[4]=9;
    numerosKino[5]=5;

    //muestro los elementos que guarde en mi arreglo.
    System.out.println("Numero de kino: " + numerosKino[0]);
    System.out.println("Numero de kino: " + numerosKino[1]);
    System.out.println("Numero de kino: " + numerosKino[2]);
    System.out.println("Numero de kino: " + numerosKino[3]);
    System.out.println("Numero de kino: " + numerosKino[4]);
    System.out.println("Numero de kino: " + numerosKino[5]);
}

}
```

Sintaxis

Funciona bien, pero imagina que en vez de 6 elementos tienes 1000 !, vas a hacer mil `sistem.out.print??`

```
/*
 * Metodo que crea un arreglo de 6 posiciones para guardar los numeros
 * del kino
 */
public void creacionArreglo(){
    //vamos a guardar solo los 6 primeros digitos del carton.
    int[] numerosKino = new int[6];
    //en cada casilla del arreglo guardamos un numero del kino
    numerosKino[0]=2;
    numerosKino[1]=8;
    numerosKino[2]=4;
    numerosKino[3]=6;
    numerosKino[4]=9;
    numerosKino[5]=5;

    //muestro los elementos que guarde en mi arreglo.
    for (int i = 0; i < numerosKino.length; i++) {
        System.out.println("Numero casilla:" + i + " contiene el valor: " + numerosKino[i]);
    }
}
```

Para lograr imprimir 1 o n elementos de un arreglo, utilizamos un ciclo FOR, el cual recorre cada posición.

Sintaxis

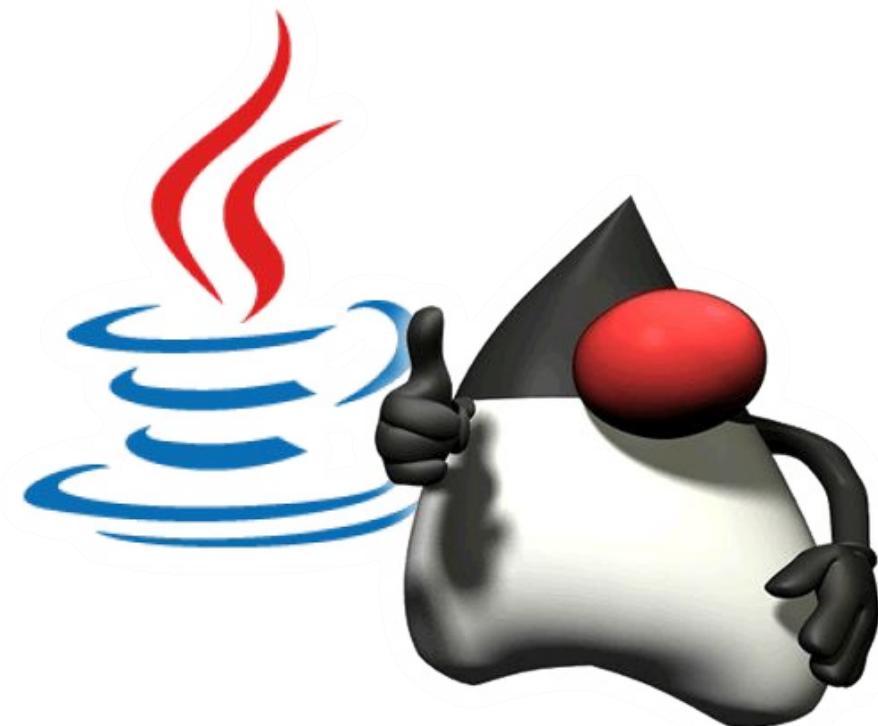
Los arreglos también cuentan con el método `.length` que te indica cual es la longitud total del mismo.

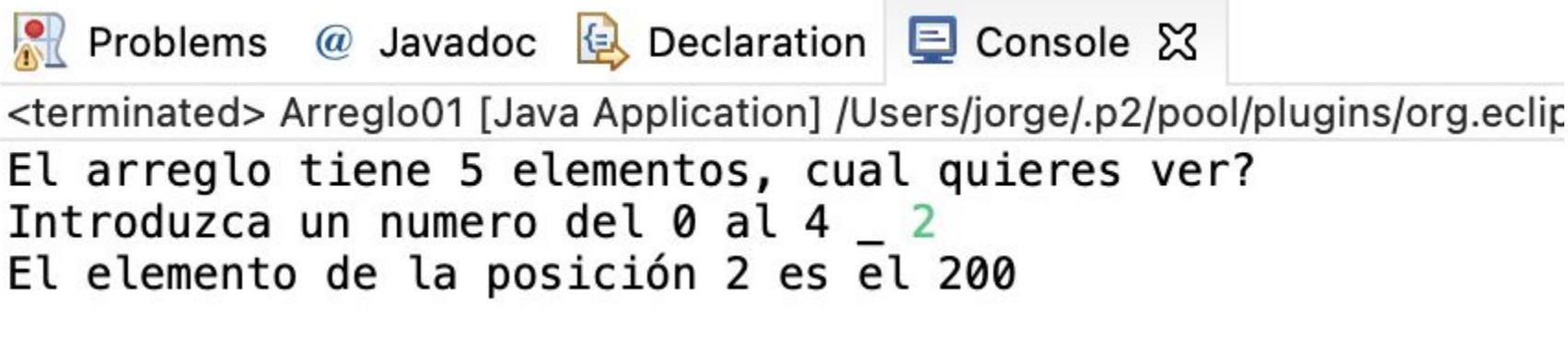
También es posible llenar el arreglo en el momento en que se crea.

```
/*
 * Metodo que crea un arreglo con inicialización en el mismo momento
 * en el que se crea.
 */
public void creacionArregloExplicito() {
    int[] lista = {1,2,3,4,5,6,7,8,9,10};
    for (int i = 0; i < lista.length; i++) {
        System.out.println("Número casilla:" + i + " contiene el valor: " + lista[i]);
    }
}
```

Crea un programa java tenga 5 valores fijos en un arreglo. Pídele al usuario qué posición quiere ver y muestrale el valor.

Ejercicios





The screenshot shows a Java application running in an IDE. The 'Console' tab is active, displaying the following output:

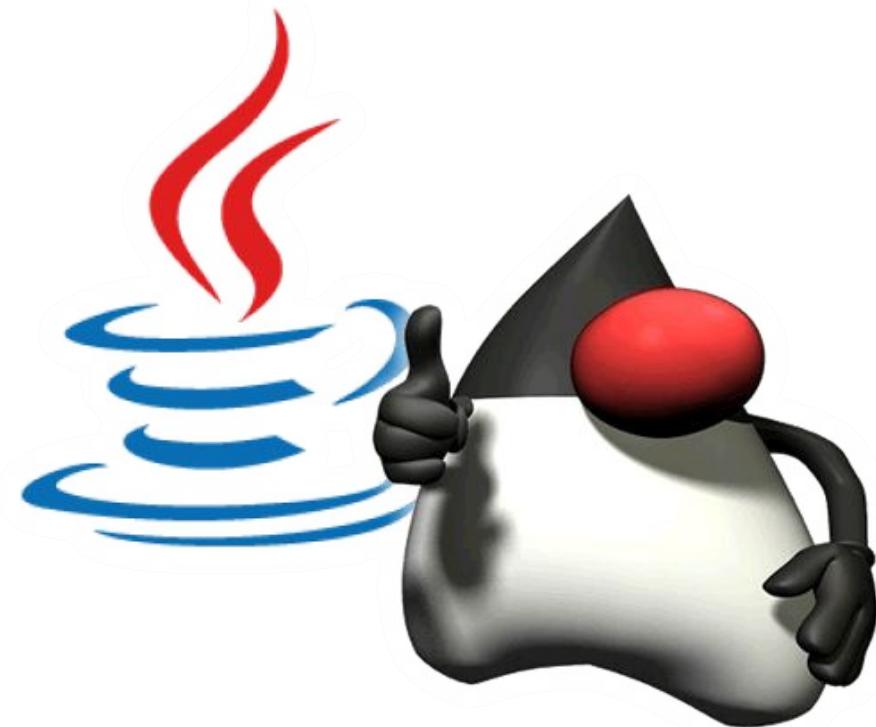
```
<terminated> Arreglo01 [Java Application] /Users/jorge/.p2/pool/plugins/org.eclipse.jdt.core_3.12.0.v20180320-1200
El arreglo tiene 5 elementos, cualquieres ver?
Introduzca un numero del 0 al 4 _ 2
El elemento de la posición 2 es el 200
```

```
import java.util.Scanner;

public class Arreglo01 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int[] x = new int[5];
        x[0] = 8;
        x[1] = 33;
        x[2] = 200;
        x[3] = 150;
        x[4] = 11;
        System.out.printf("El arreglo tiene %d elementos, cual quieres ver? \n",x.length);
        System.out.print("Introduzca un numero del 0 al 4 _ ");
        int indice = scan.nextInt();
        System.out.printf("El elemento de la posición %d es el %d",indice,x[indice]);
        scan.close();
    }
}
```

Crea un programa java que le pida al usuario 10 números. Cada número debe ser almacenado en un arreglo y luego debes mostrar cada valor.

Ejercicios





Sintaxis

Cada elemento del array se puede utilizar exactamente igual que cualquier otra variable, es decir, se le puede asignar un valor o se puede usar dentro de una expresión. En el siguiente ejemplo se muestran varias operaciones en las que los operandos son elementos del array num. Para recorrer todos los elementos de un array se suele utilizar un bucle for junto con un índice que va desde 0 hasta el tamaño del array menos 1.

```
public class Arreglo02 {  
  
    public static void main(String[] args) {  
  
        int[] num = new int[10];  
        num[0] = 8;  
        num[1] = 33;  
        num[2] = 200;  
        num[3] = 150;  
        num[4] = 11;  
        num[5] = 88;  
        num[6] = num[2] * 10;  
        num[7] = num[2] / 10;  
        num[8] = num[0] + num[1] + num[2];  
        num[9] = num[8];  
        System.out.println("El array num contiene los siguientes elementos:");  
        for (int i = 0; i < 10; i++) {  
            System.out.println(num[i]);  
        }  
    }  
}
```

En Java, a diferencia de otros lenguajes como Ruby o PHP, todos los elementos de un array deben ser del mismo tipo; por ejemplo, no puede haber un entero en la posición 2 y una cadena de caracteres en la posición 7 del mismo array. En el siguiente ejemplo se muestra un array de caracteres.

```
public class Arreglo03 {  
    public static void main(String[] args) {  
        char[] caracter = new char[6];  
        caracter[0] = 'R';  
        caracter[1] = '%';  
        caracter[2] = '&';  
        caracter[3] = '+';  
        caracter[4] = 'A';  
        caracter[5] = '2';  
        System.out.println("El array caracter contiene los siguientes elementos:");  
        for (int i = 0; i < 6; i++) {  
            System.out.println(caracter[i]);  
        }  
    }  
}
```

Ahora almacenaremos valores de tipo double:

```
public class Arreglo04 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        double[]nota=new double[4];
        System.out.println("Para calcular la nota media necesito saber la ");
        System.out.println("nota de cada uno de tus exámenes.");
        for(int i=0;i<4;i++){
            System.out.print("Nota del examen nº "+(i+1)+": ");
            nota[i]=Double.parseDouble(scan.nextLine());
        }
        System.out.println("Tus notas son: ");
        double suma=0;
        for(int i=0;i<4;i++){
            System.out.print(nota[i] + " ");
            suma+=nota[i];
        }
        System.out.println("\nLa media es "+suma/4);
    }
}
```

p2/pool/plugins/org
r la



Recorrer un arreglo con foreach

En ejemplos anteriores logramos recorrer una arreglo mediante un ciclo for común y corriente, pero existe otra forma de hacer lo mismo con una sentencia más corta y precisa, hablamos del FOREACH cuya traducción sería “por cada”.

Comienza como un ciclo
for normal

```
for (int i : array) {  
}
```

Tipo de dato de cada
elemento individual del
arreglo que se quiere
recorrer

Arreglo que deseamos
recorrer.

Recorrer un arreglo con foreach

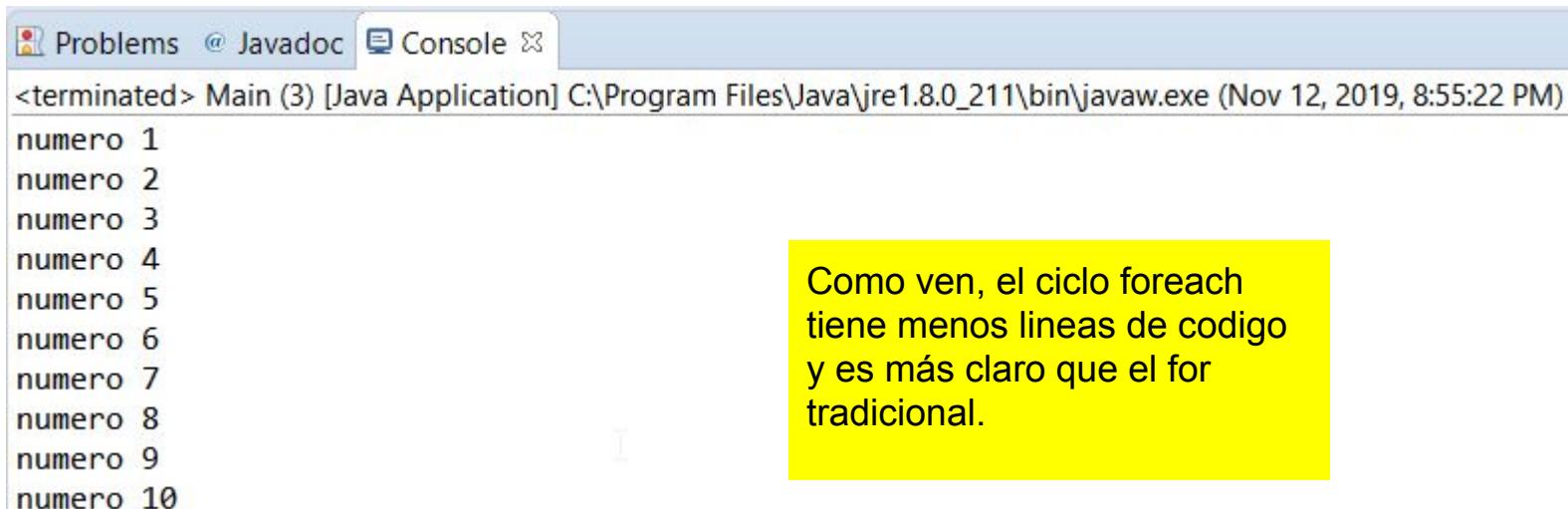
Comienza como un ciclo
for normal

```
/**  
 * Recorremos un arreglo utilizando un ciclo FOR-EACH  
 */  
public void recorrerForEach() {  
    int[] lista = {1,2,3,4,5,6,7,8,9,10};  
    for (int i : lista) {  
        System.out.println("numero " + i);  
    }  
}
```

Tipo de dato de cada
elemento individual del
arreglo que se quiere
recorrer

Arreglo que deseamos
recorrer.

Recorrer un arreglo con foreach



Problems @ Javadoc Console 

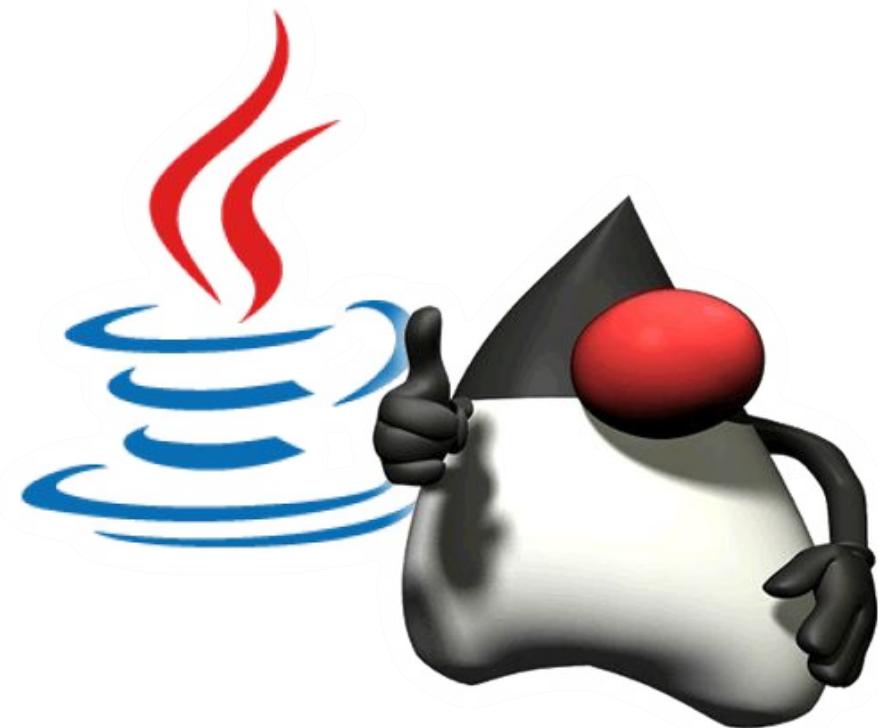
<terminated> Main (3) [Java Application] C:\Program Files\Java\jre1.8.0_211\bin\javaw.exe (Nov 12, 2019, 8:55:22 PM)

```
numero 1
numero 2
numero 3
numero 4
numero 5
numero 6
numero 7
numero 8
numero 9
numero 10
```

Como ven, el ciclo foreach tiene menos lineas de codigo y es más claro que el for tradicional.

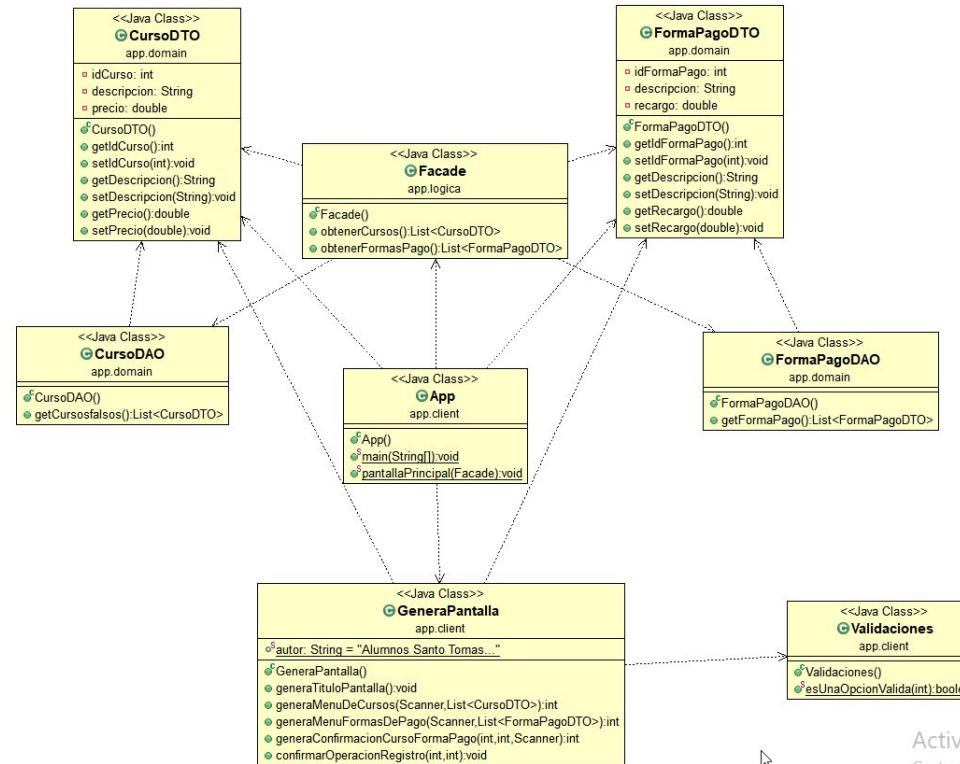
Crea un programa java que tenga un arreglo con 5 nombres de países de américa latina, y mostrarlos por pantalla utilizando un ciclo for-each

Ejercicios



FIN

The Java™ Tutorials



Activate
Go to Settings