

Introducción básica a Netbeans para desarrollo Java

NetBeans es un IDE muy completo para programar en varios lenguajes, aunque se especializa principalmente en Java. En un principio, puede que a personas que no estén familiarizadas con entornos de este tipo se le haga un poco grande, pero tampoco hay que asustarse, al principio utilizaremos sólo las herramientas básicas.

Vamos a hacer un pequeño ejercicio para entrar un poco en calor tanto con el lenguaje Java como con el IDE de NetBeans y el flujo de trabajo necesario para crear una aplicación. No esperes nada espectacular al principio, primero crearemos algunas aplicaciones de consola y ya diseñaremos algo un poco más visual más adelante.

Es importante recordar que, como dije en la [anterior entrega](#), tenéis que tener instalados el Java SDK y el IDE NetBeans para que podáis ir siguiendo todos los pasos.

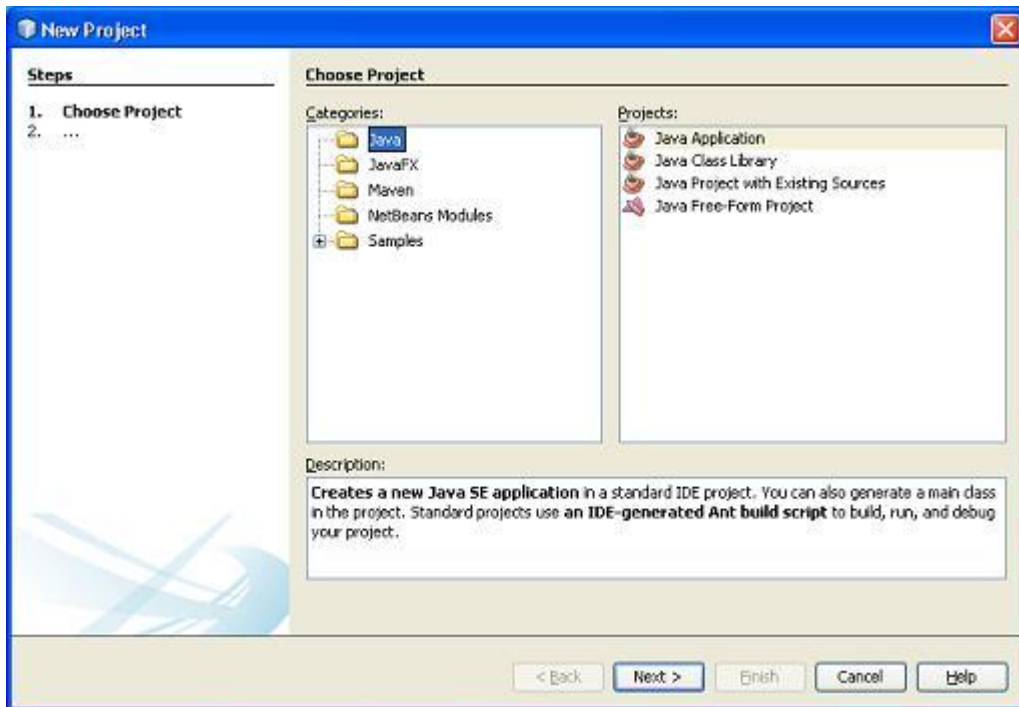
- *Dos simples notas, mucha de la información sobre desarrollo de software en Internet está escrita en inglés, la mayor parte de lenguajes de programación tienen sentencias en inglés y en general el idioma de la informática es el inglés. Si no sabes inglés te va a costar mucho más trabajo aprender a programar, sea con Java o con cualquier otro lenguaje. Aprender inglés es muy importante en el mundo de la informática, si aún no lo has hecho, te animo a que te pongas a ello.*
- *Cuando hago un tutorial, manual o cursillo, siempre intento que sea lo más práctico posible. No quiero decir con ello que la teoría no sea importante, que lo es, sino que los mejores estudiantes son los que estudian la teoría desde el punto de vista de su practicidad. En mi experiencia como formador, me he dado cuenta de que para realizar una práctica no hacen falta más que unas instrucciones precisas y unas pinceladas teóricas. Después de la práctica el alumno es capaz de obtener sus propias conclusiones, opinar con criterio o visualizar si podría haber hecho las cosas de otra manera. El estudiante ve la teoría desde un punto de vista diferente e incluso investiga por su cuenta cosas que se le antojan interesantes.*

Cómo crear una aplicación nueva

- Lo primero que tenemos que hacer es crear un proyecto. Para ello, abriremos el menú Archivo y después pulsaremos en Nuevo proyecto.



- Aparecerá una pantalla en la que tenemos que seleccionar qué tipo de proyecto queremos realizar. Seleccionaremos la categoría **Java** y en el panel de la derecha **Java Application**. Después pulsamos **Next**.

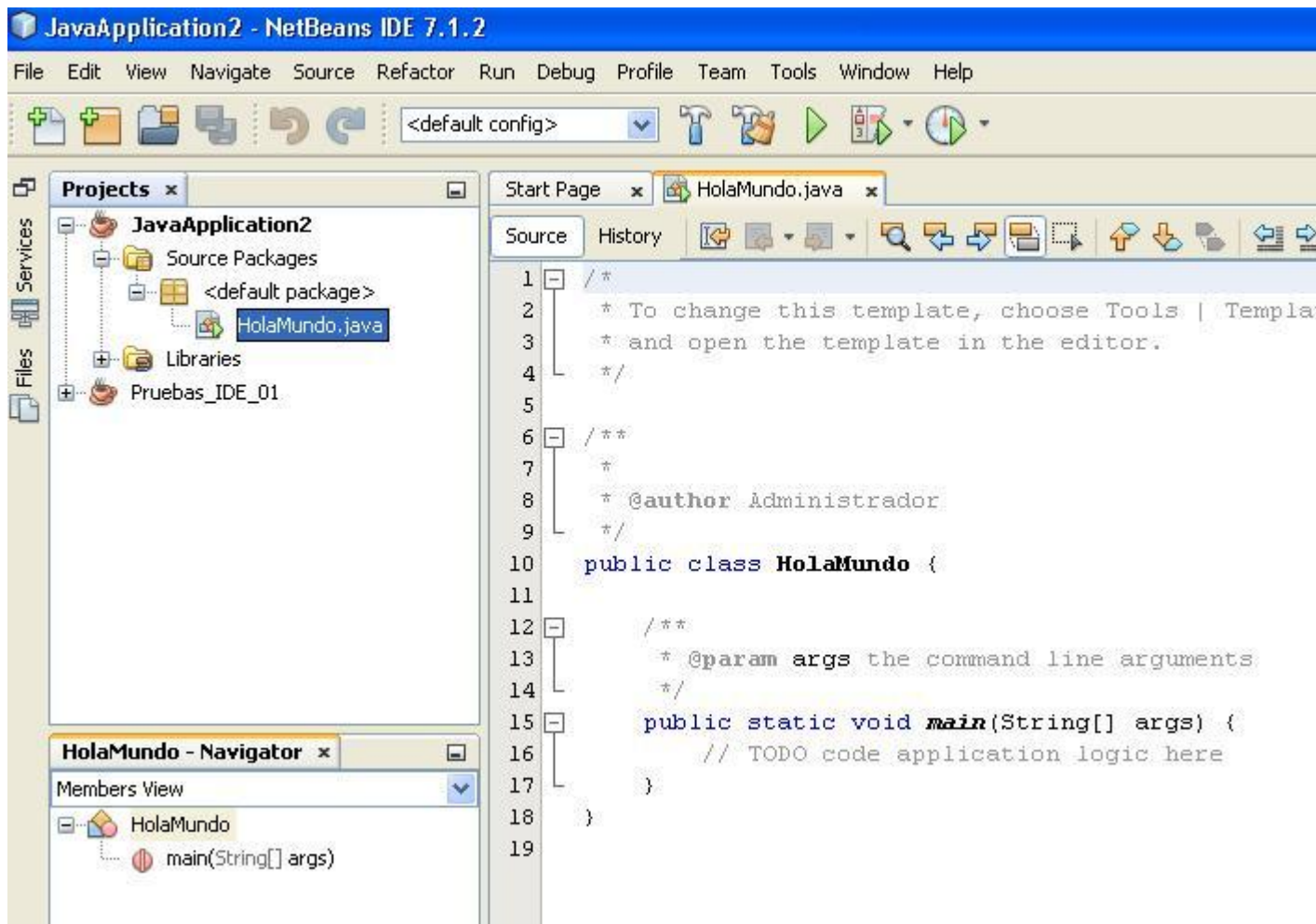


- En la siguiente pantalla se nos preguntará por las propiedades generales del proyecto: nombre, la carpeta donde lo queremos guardar, si queremos utilizar una carpeta específica para guardar las librerías, si queremos crear una clase principal y si queremos establecerla como proyecto principal.



- Las opciones de arriba las podéis configurar como queráis, dejad la casilla de las librerías desmarcada, y por último, como vemos en la fotografía, activaremos las dos casillas de la parte inferior. Esto es necesario para que el sistema sepa que lo que vamos a escribir será lo que tenga que ejecutar cuando finalicemos. En el cuadro de texto he escrito “HolaMundo”, que será la aplicación que hayamos creado. Por último pulsamos en **Finish**.

Una vez hecho esto aparecerá una ventana como esta:



Si no tienes experiencia previa con otros lenguajes de programación, todo esto te parezca en este momento un galimatías de compleja resolución. No hay problema, explicaremos un poco por encima qué son todos esos mensajes que aparecen en la pantalla para que al menos, sepas dónde estás de pie.

La ventana de proyectos



En la parte izquierda de la ventana, si no habéis estado toqueteando a lo loco, aparece un pequeño recuadro en el que veremos que aparece seleccionado un pequeño objeto llamado “HolaMundo.java”. Este es el programa que hemos creado, es un archivo con la extensión

.java que se ha guardado en la carpeta que seleccionados en la ventana de creación de proyectos.

Como vemos, hay una estructura de árbol en la que se encuentran otros elementos como “default package” o “Source Package”. Si queremos incluir un icono o una imagen, tendremos que añadirla al proyecto, todas las clases irán aquí, así como cualquier otro elemento que incluya nuestro programa.

Ventana de código

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5   package holamundo;
6
7   /**
8    *
9    * @author Ale
10   */
11   public class HolaMundo {
12
13       /**
14        * @param args the command line arguments
15        */
16       public static void main(String[] args) {
17           // TODO code application logic here
18       }
19   }
20
```

Todo lo que aparece en color gris claro son comentarios. Es muy importante que nos acostumbremos a realizar comentarios en nuestros programas. Esto tiene muchas utilidades como poder recordar mejor en un futuro qué es exactamente lo que hacía este código y también que si otra persona lo ve, pueda entender mejor cómo resuelve el problema nuestro programa.

Para hacer un comentario de varias líneas usaremos `/*` para iniciarlo y `*/` para finalizarlo. Si vamos a hacer un comentario de una sola línea podemos comenzar por `//`

En la línea 11 definimos la clase `holamundo`, el cuerpo de la clase queda encerrado entre `{` y `}`, cualquier cosa que coloquemos entre esas llaves será el código que pertenezca a la clase `holamundo`.

En la línea 16 se ha iniciado el método principal `main`. Cuando se ejecuta una aplicación, Java espera que haya un método `main`, ya que será lo primero que ejecute. Este método define el punto de entrada y salida de la aplicación.

En este momento el programa no tiene ninguna instrucción de modo que si lo ejecutamos, no ocurrirá nada. Vamos a cambiar esto.

Hola mundo!

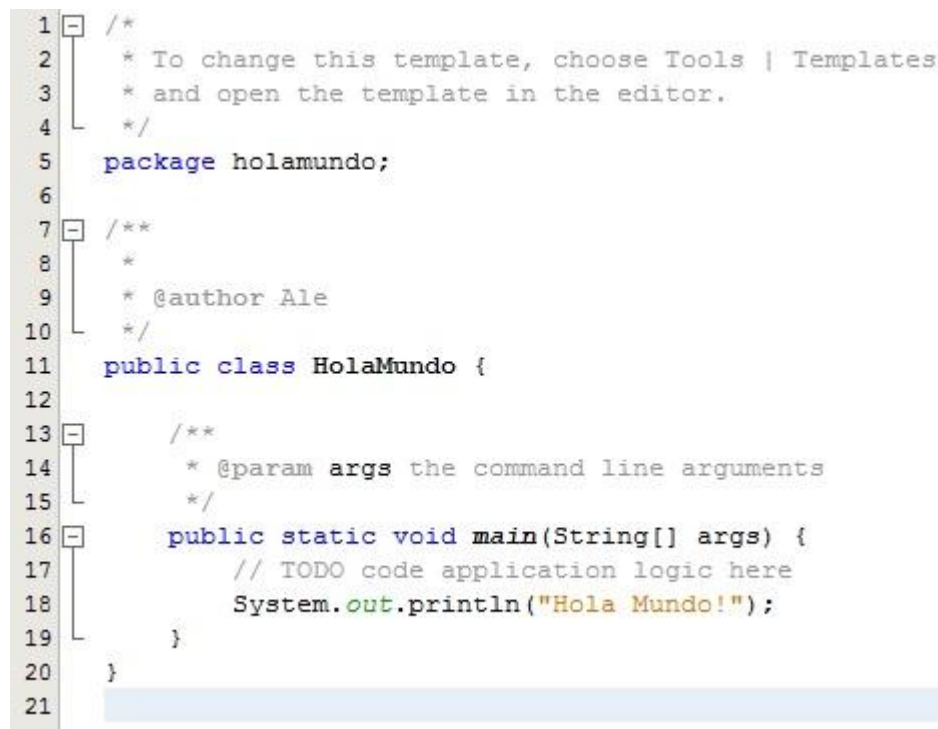
Cuando uno se inicia en un nuevo lenguaje de programación, el primer programa que hace siempre se llama “HolaMundo”. Es un programa muy simple que se utiliza para que la persona que está aprendiendo cómo funciona el sistema se familiarice con el código. La utilidad del programa no es otra que mostrar el mensaje “Hola mundo” en la pantalla.

```
public static void main(String[] args) {  
    // TODO code application logic here  
}
```

Vamos a escribir lo siguiente justo después del comentario:

```
System.out.println("Hello World!");
```

Quedando nuestro programa así:



```
1  /*  
2  * To change this template, choose Tools | Templates  
3  * and open the template in the editor.  
4  */  
5  package holamundo;  
6  
7  /**  
8  *  
9  * @author Ale  
10 */  
11 public class HolaMundo {  
12  
13     /**  
14     * @param args the command line arguments  
15     */  
16     public static void main(String[] args) {  
17         // TODO code application logic here  
18         System.out.println("Hola Mundo!");  
19     }  
20 }  
21
```

La orden que hemos añadido lo que hace es escribir el mensaje “Hola Mundo!” en la pantalla. println es una herramienta que utilizaremos mucho. Hay varios aspectos importantes en esta simple línea de código y que vamos a tener que respetar cuando escribamos código:

- Hemos utilizado el método **println** del objeto **out** miembro de la clase **System** de la biblioteca de **Java** (vaya lío eh?).
- Después de println hemos abierto paréntesis, lo que haya dentro del paréntesis es lo que llamamos argumento que básicamente podemos decir que es el dato que necesita println para saber qué tiene que escribir.
- Los textos literales, como Hola Mundo!, tenemos que encerrarlos entre comillas
- La instrucción termina con ;

Guarda el proyecto desde el menú **File -> Save**.

Compilando y ejecutando

Para poder ejecutar nuestro programa tenemos que compilarlo; esto es, traducir el código que hemos escrito a código de bytes para que nuestro ordenador pueda entenderlo.

Podríamos hacerlo desde la línea de comandos con esta orden:


```
javac HolaMundo.java
```

que es el nombre que le dimos al archivo.

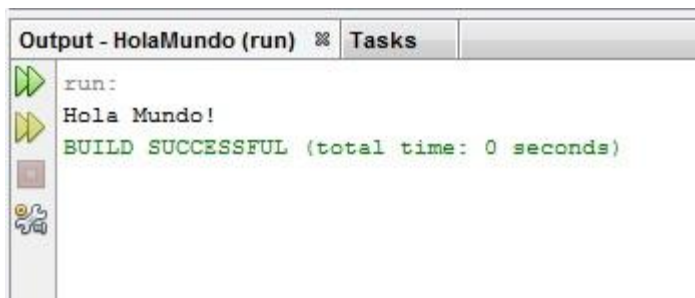
Pero como tenemos un IDE que se va a ocupar de esas cosas por nosotros, lo aprovecharemos. De hecho, ¿qué pensarías si te dijera que ya lo has compilado?

Pues sí, cuando guardas los archivos del proyecto, NetBeans se encarga de guardar el archivo y compilarlo. Así de sencillo, así que sólo nos queda ejecutarlo.

Para ejecutar el programa podemos:

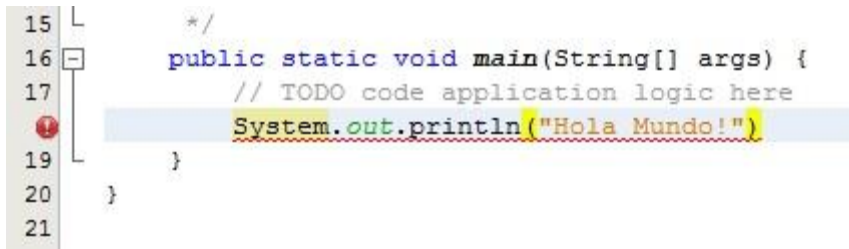
- Ir al menú Run -> Run Main Project
- Pulsar la tecla F6
- Pulsar el icono correspondiente de la barra de herramientas 

Después de pulsar cualquiera de las opciones anteriores, tendremos que fijarnos en el panel inferior, que es donde Java nos muestra el resultado obtenido:



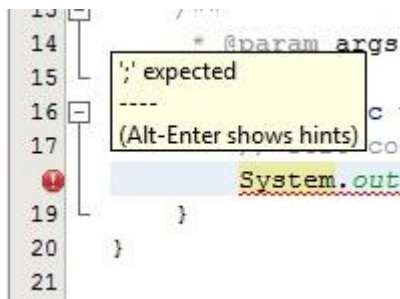
Efectivamente aparece nuestro mensaje y si todo va bien aparecerá un mensaje en verde indicando que no hay ningún error.

Si hubiera algún error de compilación, ejecución o de otro tipo, conviene fijarse en la ventana del editor. En la parte izquierda, donde aparecen los números de línea, NetBeans nos avisará si hay algún error de sintaxis. En la siguiente imagen he eliminado el ; del final de la instrucción y podemos ver claramente un símbolo de advertencia en rojo.



```
15  L      */
16  [-]    public static void main(String[] args) {
17  |      // TODO code application logic here
18  |      System.out.println('Hola Mundo!')
19  |    }
20  |  }
21  |
```

Si colocamos el cursor del ratón justo encima del símbolo de advertencia nos informará del error que ha encontrado:



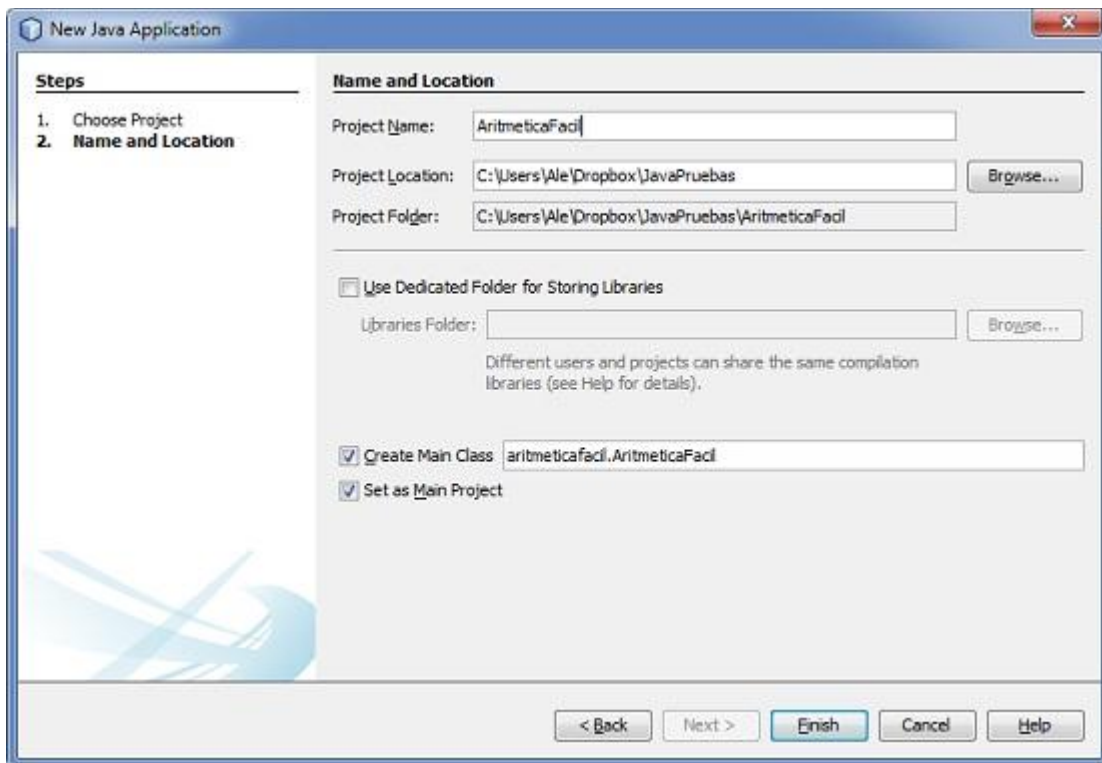
```
13  L      */
14  [-]    * @param args
15  |      ';' expected
16  |      ----
17  |      (Alt-Enter shows hints)
18  |      System.out.println('Hola Mundo!')
19  |    }
20  |  }
21  |
```

Como vemos en la imagen, nos informa de que se esperaba un ; y no lo ha encontrado.

El siguiente paso sería construir un fichero con extensión JAR que podamos distribuir a cualquier persona para que lo ejecute, ya lo veremos.

Vamos a escribir un par de programas básicos para irnos familiarizando con el lenguaje Java y el entorno de desarrollo Netbeans.

Vamos a realizar un pequeño programa que realice operaciones aritméticas. De modo que iniciaremos un proyecto nuevo que llamaremos AritmeticaFacil



Escribiremos todo el código dentro de main. Pero vayamos por pasos:

```
public static void main(String[] args) {  
    // Este código realiza operaciones matemáticas simples  
    int numero1, numero2, resultado;  
  
    numero1=30;  
    numero2=15;
```

Como veis he puesto un comentario debajo de main para explicar un poco qué es lo que hace el programa. Ahora puede parecer un poco tonto, pero a medida que los programas crecen, esto se hace más importante.

```
int numero1, numero2, resultado;
```

¿Qué significa esta línea? Pues es muy simple, **le estamos diciendo a Java que nos tiene que hacer tres huecos en la memoria**. Necesitamos tres huecos porque **vamos a almacenar tres números: numero1, numero2 y resultado**. La palabra int, viene de integer (entero). Esto quiere decir que el hueco que hemos reservado no será muy grande porque no nos va a hacer falta. En algunos libros os dirán que esto sirve para almacenar el rango de números $\{-2147483648; +2147483647\}$, yo simplemente me limitaré a decir que **el tipo de datos int nos va a servir para almacenar números no demasiado grandes y sin**

decimales. Para números grandes o con decimales utilizaremos otros tipos de datos y en vez de int utilizaremos otras palabras más “raras” como float o string que ya analizaremos.

Esta operación se denomina declaración de variables, en casi todos los lenguajes de programación deben declararse al principio de cada clase o método. Aunque en algunos se pueden ir declarando sobre la marcha, es considerado una muy mala costumbre de programación. Por ello, asumiremos que no se puede y siempre lo haremos al empezar.

Después nos encontramos con esto:

```
numero1=30;
numero2=15;
```

La operación que estamos haciendo aquí se llama asignación que básicamente consiste en decirle al ordenador que en número1 queremos almacenar el número 30 y en numero2 queremos almacenar el número 15.

Una vez realizada la asignación podemos utilizar numero1 y numero2 como si fueran números tal cual. Y aquí es donde empezaremos a realizar las diferentes operaciones.

```
//Ahora los sumamos
resultado=numero1+numero2;
//Y mostramos el resultado de sumarlos
System.out.println(numero1+ " + " + numero2+" = " + resultado);
```

Este código realiza la suma y la muestra por pantalla. Analicemos las líneas, omitiré las líneas de comentario por los motivos obvios.

```
resultado=numero1+numero2;
```

Esto ya nos lo conocemos, es la operación de asignación, solo que en este caso en lugar de darle nosotros el número, estamos indicando que lo que queremos almacenar en resultado es la suma de numero1 y numero2. Tan simple como eso.

Después hacemos que salga por pantalla:

```
System.out.println(numero1 + " + " + numero2 + " = " + resultado);
```

Esto también nos lo conocemos, pero supongo que haya varias preguntas como por ejemplo: si antes el signo + sumaba, ¿por qué lo utilizamos aquí?.

A veces sucede que un mismo operador utilizado en partes concretas del programa realiza una función diferente, aunque parecida. En este caso lo que está haciendo es concatenar. Concatenar consiste en unir fragmentos de texto para que luego aparezcan todos juntos. Fijémonos en un pequeño fragmento del argumento del método println:

```
numero1 + " + " + numero2
```

Si pusiéramos esto en el argumento del método, el resultado mostrado por pantalla sería el siguiente:

30 + 15

Primero escribe lo que hay guardado en la variable numero1 (fijáos que la escribimos sin las comillas), después, " + " es un literal, es decir, un texto que queremos que aparezca tal cual. Cuando queremos escribir un texto literal lo hacemos entre comillas, en este caso hay un espacio, el signo más y después otro espacio. Por último concatenamos la variable numero2 que contiene el número 15.

El resto del código ya podéis escribirlo, todos los bloques de código que tenemos que añadir son iguales, pero cambiaremos el operador + por – (resta), * (multiplicación) y / (división).

Resultado final

Este debe ser el resultado final. Mucho cuidado con las mayúsculas, minúsculas, punto y coma, comillas. Cualquier mínimo error hará que tengáis que depurar el programa. Podría ponerlo por escrito, pero mucha gente se perdería la oportunidad de escribir el código por su cuenta. *Escribir el código por tu cuenta te ayudará, porque cometerás errores, los corregirás y aprenderás, irás memorizando la sintaxis y otras muchas ventajas que no obtendrías si te dedicas al copia-pegar.*

```

public static void main(String[] args) {
    // Este código realiza operaciones matemáticas simples
    int numero1, numero2, resultado;

    numero1=30;
    numero2=15;

    //Ahora los sumamos
    resultado=numero1+numero2;
    //Y mostramos el resultado de sumarlos
    System.out.println(numero1 + " + " + numero2 + " = " + resultado);

    //Ahora la resta
    resultado=numero1-numero2;
    //Y mostramos el resultado de restarlos
    System.out.println(numero1 + " - " + numero2 + " = " + resultado);

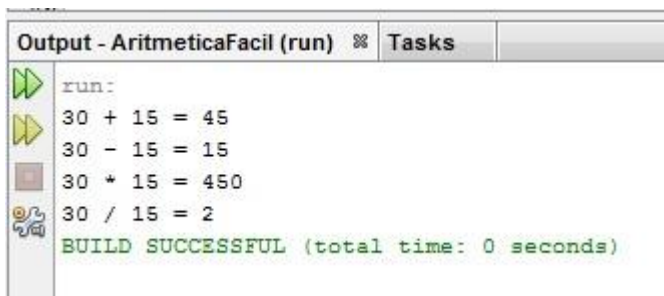
    //Ahora la multiplicación
    resultado=numero1*numero2;
    //Y mostramos el resultado de la multiplicación
    System.out.println(numero1 + " * " + numero2 + " = " + resultado);

    //Y por último, la división
    resultado=numero1/numero2;
    //Y mostramos el resultado
    System.out.println(numero1 + " / " + numero2 + " = " + resultado);

    // Ya veremos cómo se puede pedir los datos al usuario
}

```

Y el resultado al ejecutarlo:

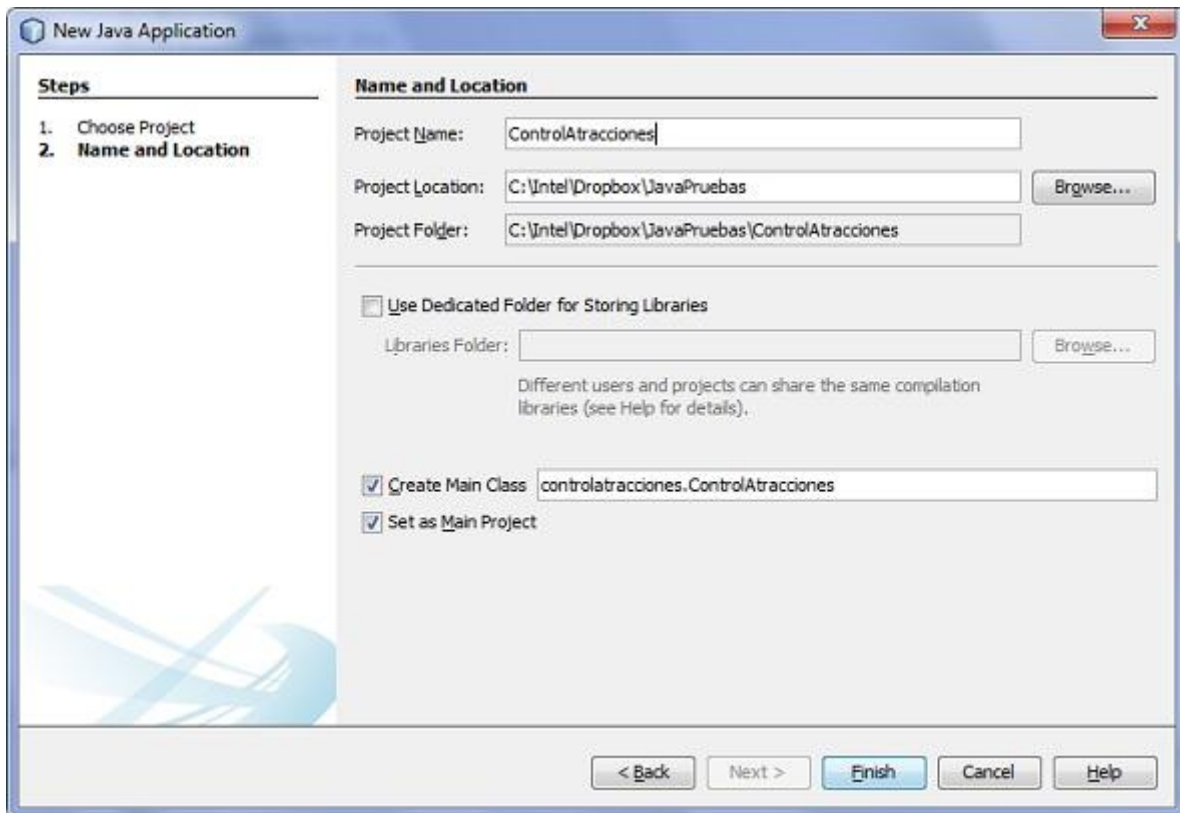


| Output - AritmeticaFacil (run) % | Tasks |
|--|-------|
| run: | |
| 30 + 15 = 45 | |
| 30 - 15 = 15 | |
| 30 * 15 = 450 | |
| 30 / 15 = 2 | |
| BUILD SUCCESSFUL (total time: 0 seconds) | |

Podéis probar a cambiar los valores de las variables para comprobar que efectivamente funciona, únicamente tened cuidado con los valores que asignáis porque si utilizáis decimales o números muy grandes, es posible que el programa genere errores.

Vamos a comenzar un nuevo proyecto. Vamos a suponer que tenemos un parque de atracciones, tenemos varias atracciones y queremos controlarlas desde nuestro ordenador. Cada atracción será un objeto y tendremos que crear métodos para controlarlas.

Crearemos un proyecto con las siguientes características:



Como utilizo varios equipos, algunos incluso con otros sistemas operativos, procuro crear mis proyectos siempre en una carpeta de intercambio mediante el programa Dropbox. De esta forma, cualquier modificación que haga en un equipo se ve reflejada automáticamente en los demás cuando los enciendo. Son algunas de las ventajas de trabajar en la nube. Dropbox es gratuito y os ofrece unos 2 Gb de almacenamiento, si utilizáis varios equipos no dudéis en instalarlo.

Definiendo las propiedades

Ya hemos comentado que cada atracción será un objeto, bien, pues cada atracción tiene unas características que nos interesa conocer. En nuestro caso vamos a controlar los siguientes parámetros:

- Nombre de la atracción
- Lugar donde está situada
- Encendida/Apagada
- Velocidad a la que está funcionando

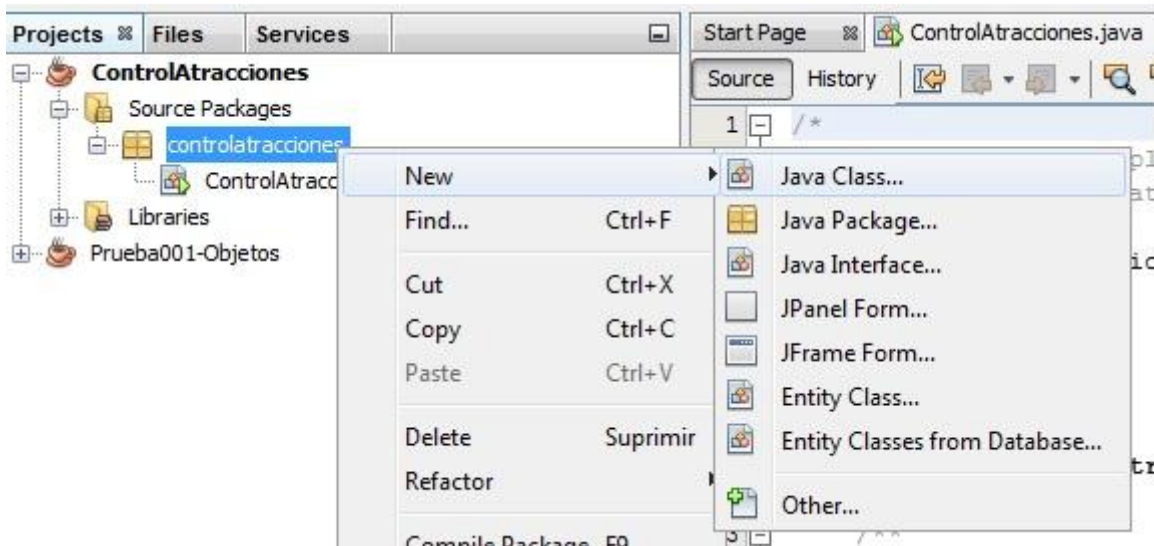
También debemos definir qué cosas podemos hacer con las atracciones, en nuestro caso:

- Aumentar la velocidad
- Disminuir la velocidad
- Apagar o encender
- Consultar estado (ver un informe que indique si está encendida o apagada y su velocidad)

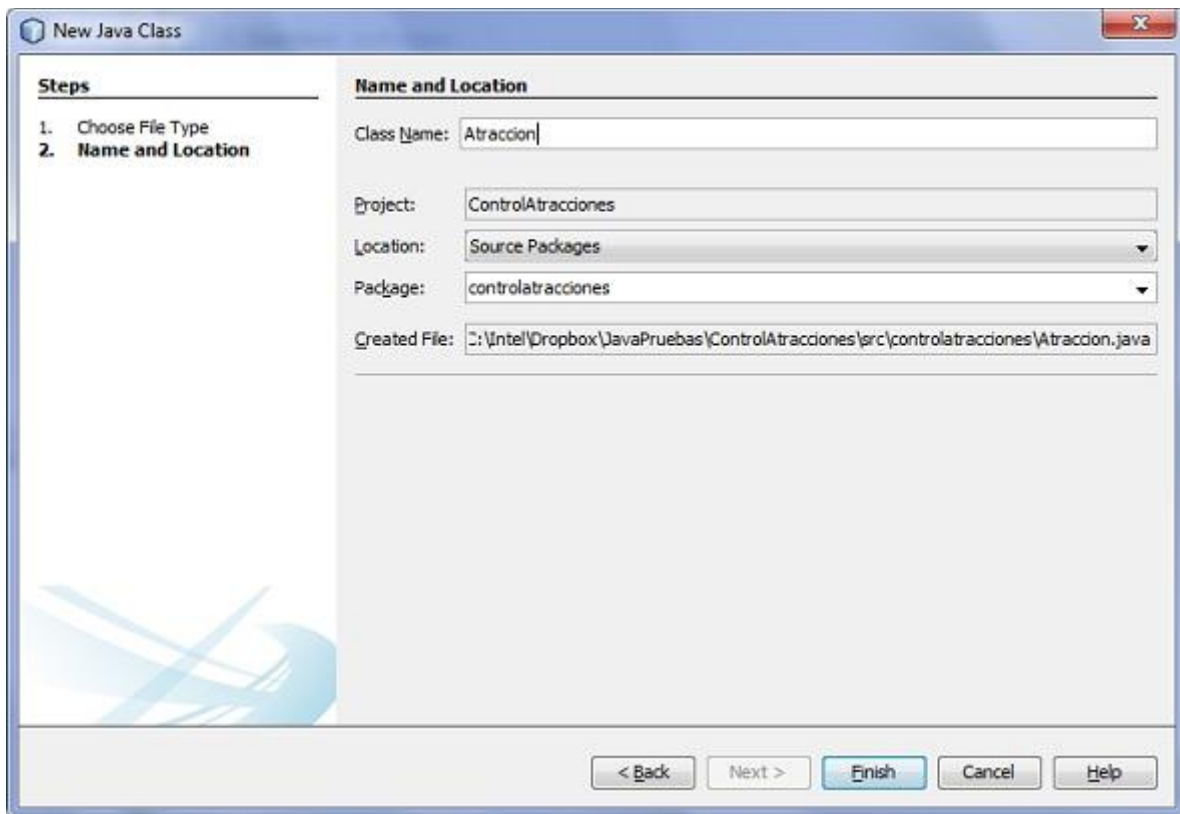
Todos estos puntos, es recomendable irlos definiendo en un papel y cuando tengáis claro qué queréis hacer, lo llevamos al ordenador. El papel y el lápiz son herramientas importantes para un programador.

Bien, como hemos decidido que tenemos una serie de problemas que resolver y que tenemos objetos que están implicados, tenemos que crear una clase nueva. En este proyecto tendremos dos: la clase principal desde donde indicaremos las instrucciones y la clase que vamos a crear ahora en la que definiremos todas estas propiedades y métodos que hemos escrito en el papel.

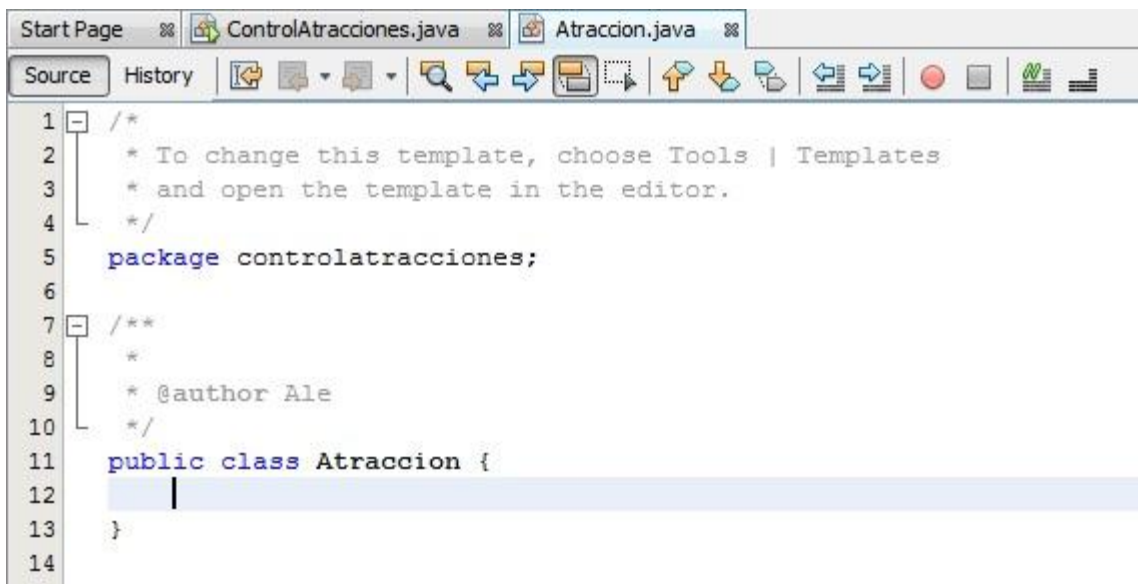
NetBeans nos lo pone muy fácil, pulsamos con el botón derecho en el paquete **controlatracciones**, seleccionamos **New** y después **Java Class...**



En la pantalla que aparece escribiremos el nombre Atraccion como nombre de clase:



Ahora tendremos dos archivos con extensión .java en el proyecto: Atraccion.java y ControlAtracciones.java. En este momento deberíais tener en la pantalla, el archivo Atraccion.java abierto.



```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5   package controlatracciones;
6
7   /**
8    *
9    * @author Ale
10   */
11  public class Atraccion {
12      |
13  }
14
```

Debajo de **public class Atraccion {** será donde escribamos las propiedades y métodos. Comenzaremos con las propiedades:

```
public class Atraccion {  
  
    // Las propiedades de cada atracción  
    String nombre;  
    String lugar;  
    Boolean encendido;  
    int velocidad;  
}
```

Hasta ahora sólo habíamos utilizado el tipo int, que decíamos que permitía hacer un hueco en memoria para un número sencillo. Aquí introducimos otros dos tipos que son muy utilizados en múltiples lenguajes de programación, incluyendo Java: String y Boolean.

El tipo String sirve para almacenar cadenas de texto. No podemos almacenar el Quijote en una variable String, pero tiene suficientes caracteres para la mayor parte de los casos que se puedan presentar.

El tipo Boolean debe su nombre al inventor del Álgebra de Boole, George Boole. El álgebra de Boole, entre otras cosas, es lo que permite que los ordenadores sean posibles. Es una variable que sólo puede tener dos estados: 1 y 0. En ocasiones, ya que los lenguajes de programación intentan acercarse al lenguaje humano, pueden utilizarse las expresiones true o false (verdadero o falso) para representar 1 ó 0.

Definición de los métodos

Debajo de las propiedades vamos a comenzar a añadir los métodos que podremos aplicar al objeto. Una de las primeras cosas que tendremos que hacer cuando lleguemos al parque todas las mañanas será encender las atracciones, de modo que crearemos un método que lo haga.


```

11 public class Atraccion {
12
13     // Las propiedades de cada atracción
14     String nombre;
15     String lugar;
16     Boolean encendido;
17     int velocidad;
18
19     //Y ahora los métodos
20
21     void encender () {
22         if (encendido==true)
23             System.out.println("Esta atracción ya está funcionando.");
24         else
25         {
26             encendido=true;
27             System.out.println("La atracción " + nombre + " se ha encendido.");
28         }
29     }
30 }

```

Vayamos por partes porque hay varias cosas nuevas.

```
void encender()
```

Encender es el nombre del método, así de simple. Los paréntesis vacíos indica que el método no recibe ningún dato con el que trabajar porque no le hace falta. Más adelante veremos ejemplos en los que sí le hacen falta.

Void es una palabra especial y reservada del lenguaje java. Este método sólo ejecuta una acción, no trabaja con datos. Al igual que en el párrafo anterior comentábamos que el método no necesita ningún dato para funcionar, tampoco devuelve un dato al final. Si devolviera algún dato, tendríamos que sustituir void por int, string, boolean o el tipo de datos que correspondiera.

Después tenemos una estructura condicional: if

```
if (encendido==true)
```

Que literalmente se traduce por “Si encendido vale true...”. Lo que haya debajo de esta línea se ejecutará sólo en el caso de que la condición se cumpla.

Es importante reparar en la pequeña diferencia que supone el doble igual. Recordad en el ejemplo anterior cuando utilizábamos numero1=30; en este caso había un símbolo =, pero ahora hay dos. Cuando hay dos, lo que hace Java es comparar si lo que hay a la izquierda es igual que lo que hay a la derecha.

Un símbolo = -> Asignación

Dos símbolos == -> Comparación

Es una diferencia importante y en ocasiones lleva a errores simplemente por olvido. Hay que tener cuidado.

```
System.out.println("Esta atracción ya está funcionando.");
```

Esto ya sabemos lo que hace. Hemos comprobado si la atracción está encendida, en caso de que sea cierto, el programa nos dice que ya está funcionando.

```
else
{
    encendido=true;
    System.out.println("La atracción " + nombre + " se ha encendido.");
}
```

Esta es la segunda parte de la estructura condicional. Ya hemos visto lo que hace el programa en caso de que ya estuviera encendido, pero si no lo está, Java ejecutará el código que haya en ELSE (sino).

Es importante fijarse en otro detalle. En la parte del if no hemos utilizado las llaves para “encerrar” el código, aquí en cambio sí. Esto es porque debajo de if sólo había una instrucción pero else tiene dos. Al haber dos instrucciones tenemos que utilizar las llaves obligatoriamente.

Si hemos llegado al else (sino), resulta que es porque la atracción no estaba funcionando, de modo que el programa la enciende:

```
encendido=true;
```

Y muestra un mensaje que incluye su nombre indicando que se ha encendido:

```
System.out.println("La atracción " + nombre + " se ha encendido.");
```

Apagando las atracciones

Ahora que has visto cómo se enciende la atracción, intenta escribir el método que permita apagar la atracción. Se puede hacer de varias formas, no hay un método único, el código que he escrito yo es el siguiente. Puedes dedicarte a copiarlo, pero yo te aconsejo que intentes hacerlo por tu cuenta, si ves que no te sale míralo y estúdialo para tener una pista y después vuelve a intentarlo. Programar requiere pensar y es un ejercicio que requiere práctica.

```

21 void encender () {
22     if (encendido==true)
23         System.out.println("Esta atracción ya está funcionando.");
24     else
25     {
26         encendido=true;
27         System.out.println("La atracción " + nombre + " se ha encendido.");
28     }
29 }
30
31 void apagar () {
32     if (encendido==false)
33         System.out.println("Esta atracción ya está apagada.");
34     else
35     {
36         encendido=false;
37         System.out.println("La atracción " + nombre + " se ha apagado.");
38     }
39 }

```

Aun siendo un código todavía muy sencillo, es probable que haya mejores formas de resolverlo, pero ésta es funcional.

El cambio de velocidad

A estas alturas ya deberías ser capaz de saber, o al menos sospechar, cómo cambiar la velocidad de las atracciones. Yo he hecho el siguiente código. Como antes, no quiere decir que sea el único, ni el mejor.

Básicamente, primero dice la velocidad que tenía la atracción y su nombre, después aumenta o disminuye la velocidad y por último muestra en pantalla la velocidad actualizada.

```

41 void masrapido() {
42     System.out.println("Velocidad anterior de " + nombre + ": " + velocidad);
43     velocidad=velocidad+1;
44     System.out.println("Velocidad actual de " + nombre + ": " + velocidad);
45 }
46
47 void maslento() {
48     System.out.println("Velocidad anterior de " + nombre + ": " + velocidad);
49     velocidad=velocidad-1;
50     System.out.println("Velocidad actual de " + nombre + ": " + velocidad);
51 }

```

Ahora el método Estado

Por último, decíamos que necesitábamos un método que nos diera información sobre el estado de la atracción. Simplemente se trata de mostrar las variables por pantalla para que las vea el usuario.

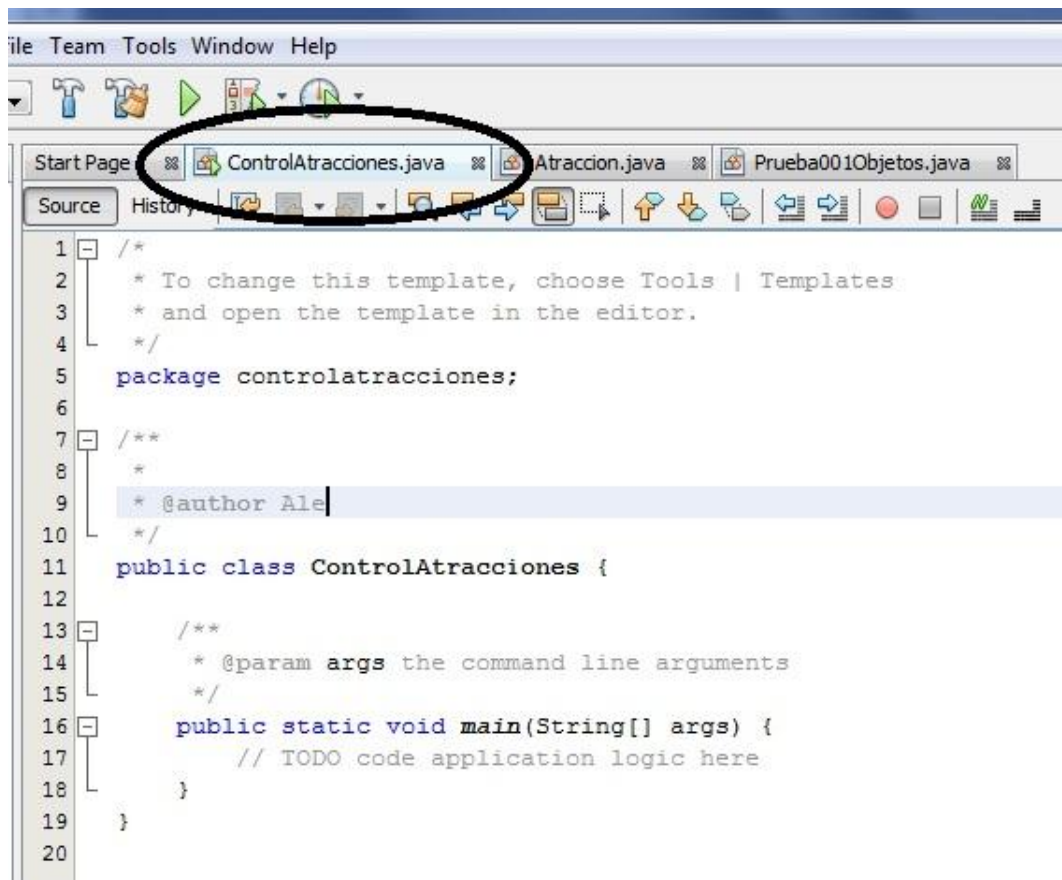
```
53 void estado() {  
54     System.out.println("Atracción: " + nombre);  
55     System.out.println("Situada en: " + lugar);  
56     if (encendido==true)  
57         System.out.println("Está encendida");  
58     else  
59         System.out.println("La atracción está apagada");  
60     System.out.println("Su velocidad es: " + velocidad);  
61 }  
62 }  
63
```

Creo que con lo que hemos visto hasta ahora, es suficiente para entender lo que este código hace.

Volvemos al control de atracciones

Ahora que ya hemos terminado de definir las propiedades y los métodos correspondientes, sólo queda crear nuestro programa principal.

Volvemos al archivo ControlAtracciones haciendo clic en la pestaña correspondiente de la parte superior del panel:



Ahora tenemos que crear tantos objetos como atracciones tengamos en el parque. Como la clase ya está creada, sólo tenemos que decirle a Java que queremos uno nuevo.

Escribiremos lo siguiente dentro del método main:

```
Atraccion ExprimeCerebros=new Atraccion();
Atraccion MachacaEstomagos=new Atraccion();
```

Estas dos sentencias han creado dos objetos de tipo Atraccion: el ExprimeCerebros y el MachacaEstomagos.

Ahora podemos pasar a definir las propiedades:

```
// Establecemos las propiedades
ExprimeCerebros.nombre="ExprimeCerebros";
ExprimeCerebros.lugar="Sector oeste, plaza 21";
ExprimeCerebros.encendido=false;
ExprimeCerebros.velocidad=5;

MachacaEstomagos.nombre="MachacaEstomagos";
MachacaEstomagos.lugar="Sector este, plaza 16";
MachacaEstomagos.encendido=false;
MachacaEstomagos.velocidad=3;
```

Y vamos a realizar las siguientes acciones:

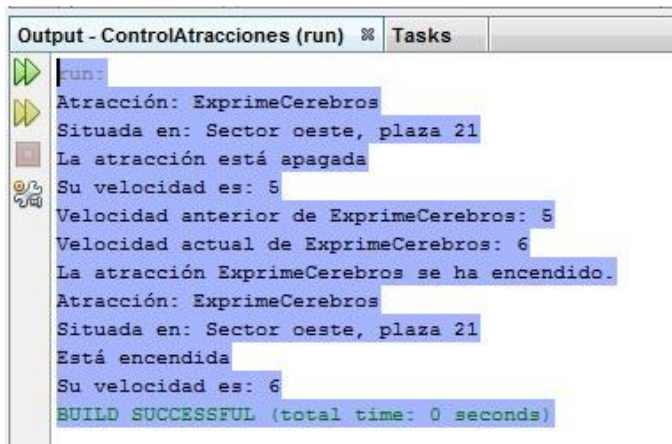
```
// Ahora mostramos la información del ExprimeCerebros
// Subiremos la velocidad, encenderemos la atracción
// y por último volveremos a mostrar su estado
ExprimeCerebros.estado();
ExprimeCerebros.masrapido();
ExprimeCerebros.encender();
ExprimeCerebros.estado();
```

Como veis, una vez que se ha creado la clase, añadir los objetos y trabajar con ellos es muy sencillo. De otra manera, cada vez que quisiéramos ejecutar, por ejemplo, estado(), tendríamos que haber escrito de nuevo todo el código anterior.

Todo junto, quedará algo parecido a esto:

```
16 public static void main(String[] args) {
17     // Creamos los objetos
18     Atraccion ExprimeCerebros=new Atraccion();
19     Atraccion MachacaEstomagos=new Atraccion();
20
21     // Establecemos las propiedades
22     ExprimeCerebros.nombre="ExprimeCerebros";
23     ExprimeCerebros.lugar="Sector oeste, plaza 21";
24     ExprimeCerebros.encendido=false;
25     ExprimeCerebros.velocidad=5;
26
27     MachacaEstomagos.nombre="MachacaEstomagos";
28     MachacaEstomagos.lugar="Sector este, plaza 16";
29     MachacaEstomagos.encendido=false;
30     MachacaEstomagos.velocidad=3;
31
32     // Ahora mostramos la información del ExprimeCerebros
33     // Subiremos la velocidad, encenderemos la atracción
34     // y por último volveremos a mostrar su estado
35     ExprimeCerebros.estado();
36     ExprimeCerebros.masrapido();
37     ExprimeCerebros.encender();
38     ExprimeCerebros.estado();
39
40     //Podemos hacer lo mismo con MachacaEstómagos
41     //El código es exactamente igual, sólo
42     //cambia el nombre del objeto
43 }
44 }
```

Y si ejecutamos el código la salida debería ser esta:



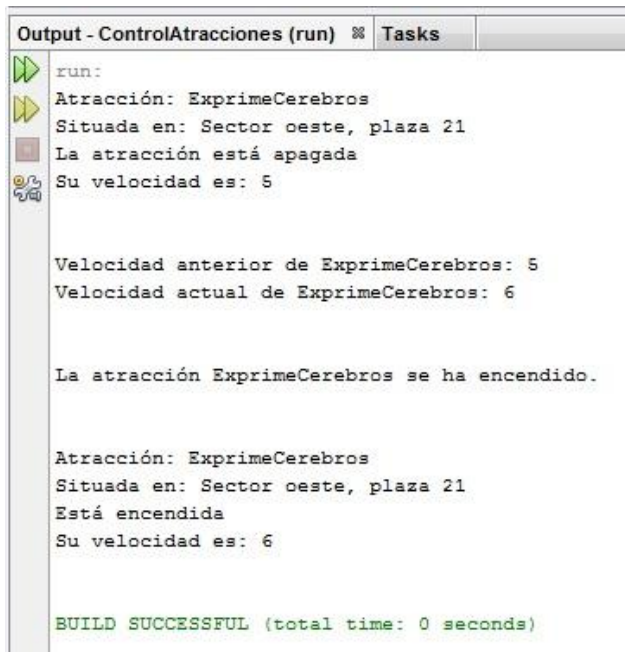
```
Output - ControlAtracciones (run) % Tasks
run:
Atracción: ExprimeCerebros
Situada en: Sector oeste, plaza 21
La atracción está apagada
Su velocidad es: 5
Velocidad anterior de ExprimeCerebros: 5
Velocidad actual de ExprimeCerebros: 6
La atracción ExprimeCerebros se ha encendido.
Atracción: ExprimeCerebros
Situada en: Sector oeste, plaza 21
Está encendida
Su velocidad es: 6
BUILD SUCCESSFUL (total time: 0 seconds)
```

La aplicación es funcional, pero bastante mejorable. Por ejemplo, toda la información aparece junta en el mismo sitio. Podríamos hacer que cada vez que termine un método, el programa haga un salto de línea para que la información no aparezca tan seguida.

Sólo tenemos que modificar los métodos de la clase Atracción para que esto ocurra. Añadiremos la siguiente línea al final de las instrucciones de cada método:

```
System.out.println("\n");
```

El código `\n` lo que hace es insertar un salto de línea. El resultado al ejecutar la aplicación ahora será este:



```
Output - ControlAtracciones (run) % Tasks
run:
Atracción: ExprimeCerebros
Situada en: Sector oeste, plaza 21
La atracción está apagada
Su velocidad es: 5

Velocidad anterior de ExprimeCerebros: 5
Velocidad actual de ExprimeCerebros: 6

La atracción ExprimeCerebros se ha encendido.

Atracción: ExprimeCerebros
Situada en: Sector oeste, plaza 21
Está encendida
Su velocidad es: 6

BUILD SUCCESSFUL (total time: 0 seconds)
```

De esta manera, cada una de las acciones queda aislada y se comprenden mejor los mensajes. Si esta información la guardáramos en un archivo, ya tendríamos lo que suele llamarse un “log” de la atracción.