3.2.2 CREANDO UN ESQUEMA CON MATISSE

En las secciones siguientes se trabajará sobre un ejemplo concreto para mostrar una aplicación de acceso a SGBD-OO. Sin embargo, para ello es necesario previamente utilizar el entorno de Matisse para crear el esquema básico de base de datos. A continuación se muestra cómo crear ese esquema.

El modelo de ejemplo que se muestra en la Figura 3.2 es con el que se trabajará en las siguientes secciones y representa una simplificación de una *Biblioteca*. El modelo contiene:

- Una clase *Autor*, que representa a todos los posibles autores de una obra literaria. Sus atributos son *nombre*, *apellidos* y *edad*. Además tiene un método *dameNombreyApellidos()* que devuelve la concatenación del *nombre* y la *edad* en una única cadena.
- Una clase *Obra*, que representa a los diferentes tipos de creaciones que puede hacer un autor. Tiene como atributos *título* y *páginas*.
- Una clase *Libro*, que hereda de *Obra* y añade un atributo más llamado *editorial* que representa a la editorial que publica el libro.
- Una clase *Artículo*, que también hereda de *Obra* y añade otro atributo llamado *revista* que representa a la revista que publica el artículo.

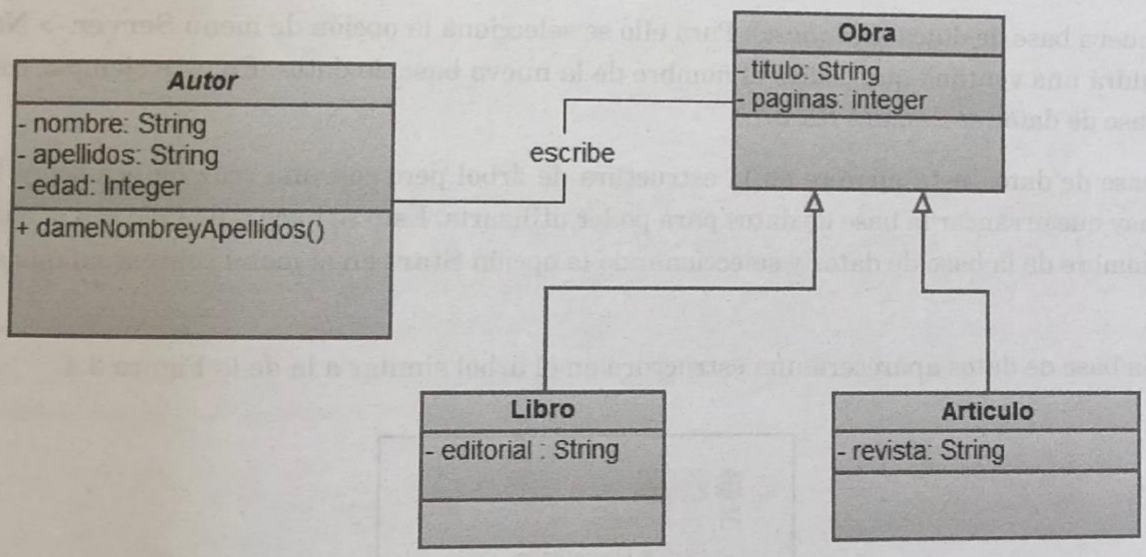

Figura 3.2. *Modelo de datos ejemplo - Biblioteca*

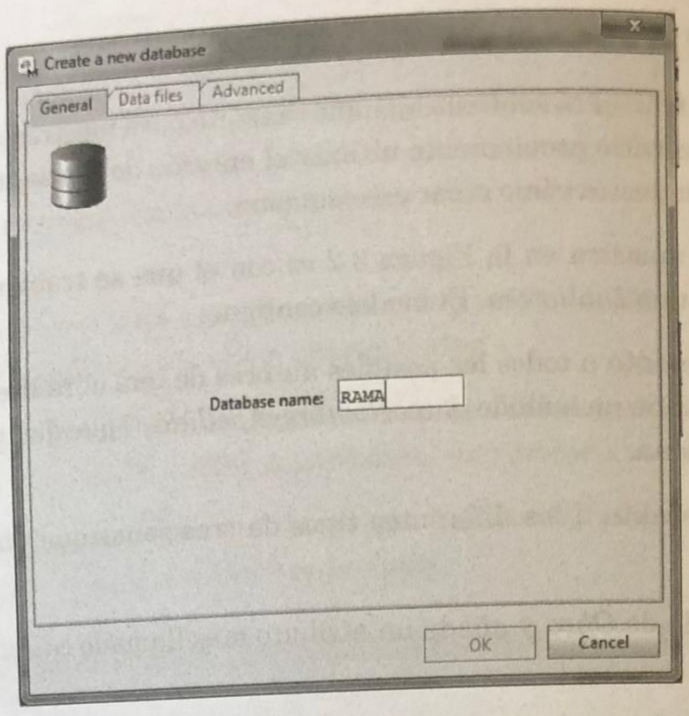

Figura 3.3. *Crear una BBDD*

Para crear este esquema en Matisse se deben seguir los siguientes pasos:

1 Crear una nueva base de datos (*Database*). Para ello se selecciona la opción de menú **Server -> New Database**. Entonces saldrá una ventana que pedirá el nombre de la nueva base de datos. En este ejemplo, como muestra la Figura 3.3, la base de datos se llamará *RAMA*.

2 Creada la base de datos, esta aparece en la estructura de árbol pero con una cruz blanca sobre fondo rojo. Esto indica que hay que arrancar la base de datos para poder utilizarla. Esto se hace pulsando con el botón derecho del ratón sobre el nombre de la base de datos y seleccionando la opción **Start** en el menú contextual que aparece.

3 Arrancada la base de datos aparecerá una estructura en el árbol similar a la de la Figura 3.4.

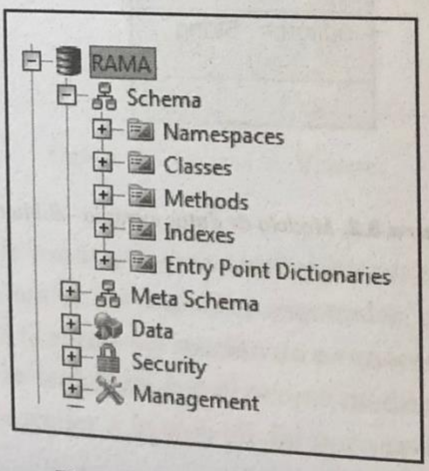

Figura 3.4. *Estructura Matisse*

- *Namespaces* representa el espacio de nombre en el cual se crearán las clases asociadas. Para una buena organización de la información es interesante definir un espacio de nombre que no cree ambigüedad sobre dónde están situados los elementos de la base de datos.[45]
- *Classes* albergará la estructura propia de tipos definidos. En el ejemplo será (*Autor*, *Libro*, *Revista* y *Obra*).
- *Methods* contendrá los métodos definidos en cada clase. Hay que recordar que en un SGBD-OO (Atkinson en la Sección 3.1) se deben poder almacenar objetos con sus atributos y sus métodos asociados. *Indexes* contiene los índices creados sobre los objetos (creados por el usuario) para optimizar las consultas.[46]

4 Crear una *Namespace* para la estructura *Biblioteca*. Para ello se pulsa con el botón derecho del ratón sobre **Namespaces** y se selecciona la opción **New Namespace**, que aparece en el menú contextual. En la parte derecha aparecerá una plantilla de ayuda para escribir la sentencia ODL *create namespace*. La Figura 3.5 muestra cómo crear el nuevo espacio de nombres (*biblioteca*).

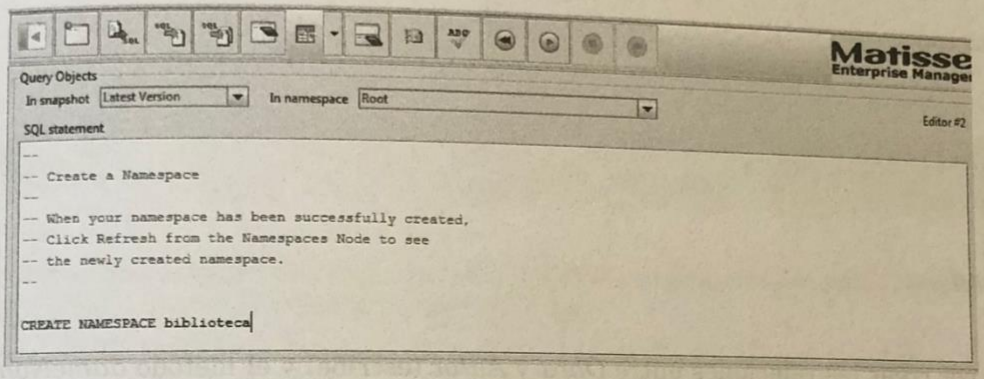

Figura 3.5. *Namespace Biblioteca*

5 Creado el espacio de nombres, lo siguiente es crear la estructura definida en la Figura 3.2. Para ello se pulsa en **Classes** y se selecciona la opción de menú **New Class**. Al igual que con los *namespaces*, en la parte derecha saldrá una plantilla con la sintaxis para crear una nueva clase. Esta plantilla utiliza un lenguaje propio de Matisse pero que luego puede ser exportado a ODL (ODMG). La Figura 3.6 muestra el código necesario para la creación de la clase *Autor* y se puede observar en ella cómo es necesario seleccionar de la lista (*In namespace*) el *namespace biblioteca* para que la clase se cree en ese espacio de nombres y no en la raíz (*root*).

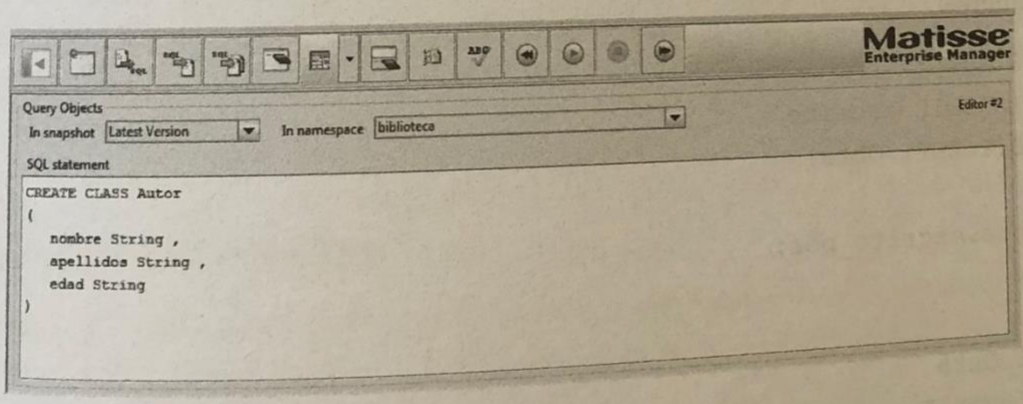

Figura 3.6. *Clases - Biblioteca*

45 Aquí, el concepto de *Namespace* es sinónimo del utilizado en Java para los paquetes (*package*).

46 Los índices en SGBD-OO son necesarios, al igual que ocurre en los SGBD relacionales.

Las otras clases del ejemplo se crean de la misma manera con el siguiente código. Debemos observar que en este código no se especifica la relación entre *Obra* y *Autor* ni se define el método *dameNombreyApellidos()*, estos elementos se definirán en el siguiente paso.

```
CREATE CLASS Obra
(
 titulo string ,
 paginas Integer
)

CREATE CLASS Libro
 INHERIT Obra
(
 editorial String
)

CREATE CLASS Articulo
 INHERIT Obra
(
 revista String
)
```

6 Por último, falta por crear las relaciones entre *Obra* y *Autor* (escribe) y el método *dameNombreyApellidos()* que devuelve ambos atributos de *Autor* concatenados. Para ello se usa el mismo procedimiento que para crear las clases.

- En el árbol se selecciona la clase a la que se quiere crear una nueva relación, por ejemplo *Autor*, y se pulsa con el botón derecho del ratón. En el menú contextual se selecciona **Alter Class -> Add Relationship**. Entonces saldrá una plantilla en la parte derecha con la sintaxis para añadir una nueva relación. El siguiente código es el necesario para crear primero una relación entre *Autor* y *Obra* (*escribe*) y segundo la inversa entre *Obra* y *Autor* (*escrita_por*). Esto optimizará la recuperación de objetos. Es importante recordar que se debe poner el *namespaces* en *Biblioteca* para que el sistema sepa que las clases que se quieren relacionar están bajo ese espacio de nombres.

```
ALTER CLASS Autor
 ADD RELATIONSHIP escribe
 RELATIONSHIP SET( Obra)

 INVERSE Obra.escrito_por;


ALTER CLASS Obra
 ADD RELATIONSHIP escrito_por
 RELATIONSHIP SET( Autor)

 INVERSE Autor.escribe;
```

- De la misma manera se añade un método, sin embargo en este caso se selecciona la clase *Autor* y pulsando en el botón derecho se selecciona la opción de menú **Alter Class -> Add Method**. El código para añadir un método a *Autor*[47] que concatene el *nombre* y el *apellido* será:

```
CREATE METHOD dameNombreyApellidos ()
RETURNS String
FOR Autor
--
-- Describe your method here
--
BEGIN
 return CONCAT (nombre, apellidos);
END;
```

ACTIVIDADES 3.4

- Instalar en local el sistema gestor Matisse. Una vez instalado, y siguiendo los pasos descritos en esta sección, crear una base de datos según el modelo de la Figura 3.2.

3.3 INTERFAZ DE PROGRAMACIÓN DE APLICACIONES DE LA BASE DE DATOS

Una vez el esquema de la base de datos se ha creado, el siguiente paso es preparar el sistema para que pueda ser accedido desde código Java. Como se ha comentado, ODMG no define un lenguaje de manipulación de objetos (OML) y Matisse tampoco, por lo que la única manera de añadir, eliminar, modificar y consultar objetos en el esquema creado en la sección anterior es usando código fuente. En esta sección se tratará esta forma de acceso mediante código fuente. Aunque Matisse ofrece alternativas para varios lenguajes (como por ejemplo Eiffel, C y Microsoft .NET), en los ejemplos mostrados se usará Java.

3.3.1 PREPARANDO EL CÓDIGO JAVA

Para poder entender el proceso para acceder a los objetos almacenados en las bases de datos OO hay que tener siempre presente que lo que se busca es tener la sensación de *trabajar solo con objetos*, sin tener que pensar en si

47 Los métodos de las clases son como procedimientos almacenados (típicos de los SGBDR). Por ello, su ejecución siempre se hará en el servidor de bases de datos.

están almacenados en una base de datos o están creados en memoria. Es decir, se busca un acceso a objetos totalmente transparentes.

Por lo tanto, cuando se trabaja con SGBD-OO hay que olvidarse de buscar algo similar a las típicas API de acceso a datos de sistemas relacionales como ODBC o JDBC. Lo que se tiene que buscar es una manera de que el SGBD-OO *genere en un lenguaje de programación* (Java en este caso) *las clases que componen el esquema de base de datos*. Esta es la clave del problema, lo que beneficia el trabajo con SGBD-OO desde lenguajes de programación OO y lo que lo diferencia a su vez del acceso a datos en sistema relacionales.

La librería de clases creada por el SGBD-OO en el lenguaje de programación seleccionado (en nuestro caso Java) puede ser integrada en un proyecto en el cual, usando la librería adecuada, se puede abordar la persistencia de un objeto con solo invocar a un método del mismo. El mecanismo por el cual el contenido de un objeto se almacena en la base de datos es totalmente transparente al programador. Solo tiene que pedir que se haga persistente y el objeto se hará, solo hay que pedir que se recupere y el objeto aparecerá en memoria como si de cualquier otro objeto se tratase.

A continuación se creará con Matisse la estructura de clases en Java que representa el esquema *Biblioteca* de la base de datos RAMA.

1 Seleccionar la opción de menú **Schema -> Generate Code**. En el siguiente menú hay que seleccionar RAMA para que su esquema se convierta en clases Java. La Figura 3.7 muestra la ventana de generación:

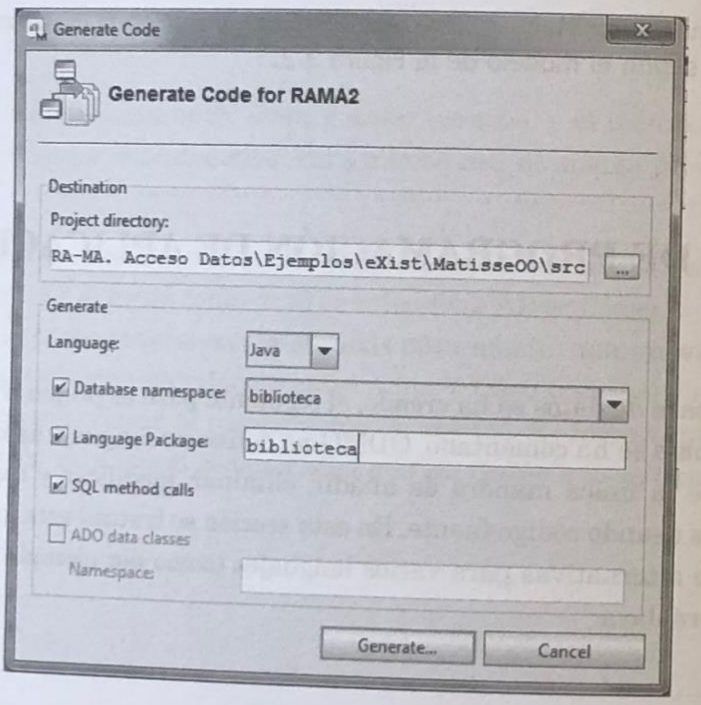

Figura 3.7. *Crear código Java*

Las opciones de generación son las siguientes:

- *Project directory* (directorio de proyecto): se utiliza para indicar dónde está ubicada la carpeta del proyecto Java en el que se integrará el código Java generado. Esto es útil para que las clases creadas se coloquen en la carpeta deseada y dentro del paquete seleccionado y así no tener que hacer importaciones posteriores.
- *Lenguage* (lenguaje destino): indica el lenguaje en el que se quiere generar el código, en este caso Java.

- *Database namespace*: se utiliza para indicar de qué espacio de nombres almacenados en la base de datos se quieren sacar las clases de las que se generará el código. En este caso será del *namespace Biblioteca*.
- *Language Package*: sirve para indicar el paquete Java en el que se quieren agrupar las clases que se generarán. En el ejemplo de la Figura 3.7 todas las clases se colocarán en un paquete llamado *biblioteca*.
- *SQL method call*: indica si se desea hacer llamadas a SQL[48] para invocar a los métodos definidos en la base de datos. En el ejemplo *Biblioteca* se creó un método en la clase *Autor* llamado *dameNombreyApellidos()*. Seleccionando la opción *SQL method call*, Matisse genera el código necesario para que cuando el usuario quiera invocar este método de la clase *Autor* se llame directamente a su código ODL almacenado en la base de datos. De alguna manera, los métodos creados en la base de datos *son como procedimientos almacenados* (típicos de los SGBDR), que siempre deben ser ejecutados en el servidor de base de datos por razón de optimización. Por esto es por lo que el código generado en el método no se traduce a Java en la generación de las clases Java, sino que solo se pone el código para conectar a la base de datos y ejecutarlo desde el propio Matisse. En las siguientes secciones se concretará este código y su acceso desde Java.

2 Una vez generadas las clases, todas se ubicarán en el paquete seleccionado dentro del directorio del proyecto seleccionado. Esas clases ya estarán listas para ser incorporadas a un proyecto y poder realizar operaciones de modificación y consulta.

La Figura 3.8 muestra un proyecto Java (hecho en NetBeans IDE 7.1.2) en el que se muestra un paquete biblioteca (*matisseoo.biblioteca*) que contiene las clases Java creadas desde Matisse.

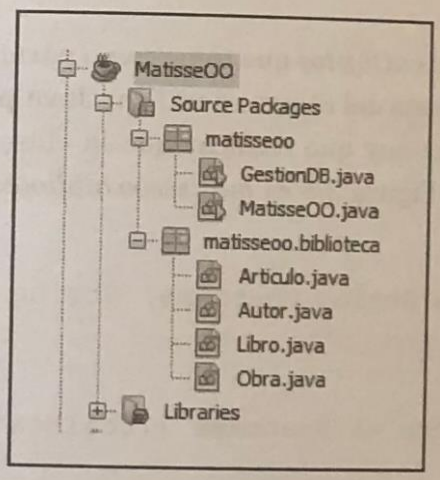

Figura 3.8. *Estructura de clases en Java*

3 Para que el proyecto reconozca las clases y métodos creados por Matisse es necesario incluir en el proyecto la librería *matisse.jar*. En el caso de que la instalación de Matisse se haga en la unidad C:\, bajo sistema Windows, la ruta donde se localiza la librería sería: *C:\Products\Matisse\lib*.

48 Desafortunadamente, Matisse utiliza SQL para referirse al lenguaje que usa para interactuar con las bases de datos OO que almacena. Sin embargo, no hay que confundirlo con SQL de ANSI ya que no sigue la misma sintaxis.

3.3.2 AÑADIENDO OBJETOS

Una vez creadas las clases, la manera de añadir objetos a la base de datos es sencilla, solo hay que crear e objetos y luego llamar para almacenarlos. Evidentemente, para saber dónde se tienen que almacenar los ob necesario previamente establecer una conexión con la base de datos mediante su dirección (*localhost* si tra local) y su nombre (*RAMA* en el ejemplo). El objeto necesario para la persistencia es:[49]

- *MtDatabase*:[50] Este objeto gestiona el acceso a Matisse. Sus métodos más destacados son:
  - *MtDatabase()*: constructor de la clase. Crea la conexión a la base de datos indicada por la direcci servidor Matisse, el nombre de la base de datos y el espacio de nombres al que se quiere acceder de la base de datos.
  - *Open()*: abre la base de datos definida en el constructor.
  - *startTransaction()*: crea una transacción para que lo que se haga hasta el final de la transacción sea at y si algo falla a media ejecución de la transacción se deshagan todos los cambios.
  - *Commit()*: da por finalizada la transacción y materializa los cambios hechos en los objetos en la ba datos.
  - *Close()*: cierra la base de datos.
- *MtException*: atiende las excepciones que se produzcan.

El siguiente código muestra el método *creaObjetos* que tiene como parámetros el servidor Matisse (*hostname* nombre de la base de datos (*dbname*). El resto del código es utilizar Java para crear objetos sin pensar en su pe persistencia en un SGBD-OO. Únicamente hay que recalcar que la clase que contenga este código debe imp el paquete *biblioteca* (en el ejemplo de la Figura 3.8 es *matisseoo.biblioteca*) que es el que contiene las clases generadas de Matisse.

```
public static void creaObjetos(String hostname, String dbname)
{
  try {
     //Abre la base de datos con el Hostname (localhost), dbname (RAMA) y el namesp
     //"biblioteca".
     MtDatabase db = new MtDatabase(hostname, dbname, new MtPackageObjectFactory(
"biblioteca"));

     //Abre la base de datos y empieza una transacción.
     db.open();
     db.startTransaction();

     // Crea un objeto Autor
```

49 Para más información sobre las clase propias para el acceso a Matisse desde Java véase *http://www.matisse.com/pdf/developer/*
pg.pdf

50 Esta clase está definida en la librería *matisse.jar*

```
    Autor a1 = new Autor(db);
    a1.setNombre("Haruki");
    a1.setApellidos("Murakami");
    a1.setEdad("53");

    // Crea un objeto Libro
    Libro l1 = new Libro(db);
    l1.setTitulo("Baila Baila Baila");
    l1.setEditorial("TusQuests");
    l1.setPaginas(512);

    // Crea otro objeto Libro
    Libro l2 = new Libro(db);
    l2.setTitulo("Tokio Blues");
    l2.setEditorial("TusQuests");
    l2.setPaginas(498);

    //Crea un array de Obras para guardar los libros y hacer las relaciones
    Obra o1[]= new Obra[2];
    o1[0]=l1;
    o1[1]=l2;
   //Guarda las relaciones del autor con los libros que ha escrito.
    a1.setEscribe(o1);
   //Ejecuta un commit para materializar las peticiones.
    db.commit();
   //Cierra la base de datos.
    db.close();
  } catch (MtException mte) {
   System.out.println("MtException : " + mte.getMessage());
  }
}
```

Una vez ejecutado este código Java, en Matisse se puede ver que los objetos se han ejecutado correctamente. Para o se pulsa en el árbol en la clase de la que se quiere ver los objetos y en el menú se selecciona **View data**. Entonces arecerán los valores de los objetos creados. Por ejemplo, los objetos que el código ha almacenado de tipo *Obra* arecerían en Matisse en el listado de la Figura 3.9.

Page limit: 200 Page: 1 Total count: 2 Offset: 1

OID	titulo	paginas	escrito_por
0x10aa Libro	Baila Baila Baila	512	0x10a9
0x10ab Libro	Tokio Blues	498	0x10a9

Figura 3.9. *Objetos de la clase Obra desde Matisse*