Módulo de software para el aprendizaje del proceso de construcción e inicialización de objetos en Java

Leónides Pérez Ortiz[a], Ulises Juárez Martínez[b], José Luis Sánchez Cervantes[c], Lisbeth Rodríguez Mazahua[d], Ma. Antonieta Abud Figueroa[e].

{[a] m21011179, [b] ulises.jm, [c] jose.sc, [d] lisbeth.rm2, [e] maria.af} @orizaba.tecnm.mx

Tecnológico Nacional de México / IT Orizaba. Maestría en Sistemas Computacionales. Oriente 9 No. 852, Col. Emiliano Zapata, C.P. 94320, Orizaba, Ver, México.

Resumen

Comprender el Proceso de Construcción e Inicialización de Objetos (PCIO) es una pieza clave para el aprendizaje de cualquier lenguaje de programación orientado a objetos. Una vez que los estudiantes y/o programadores identifican los pasos de este proceso, es más fácil codificar programas eficientes porque se trabaja de una manera más consciente de lo que se está programando, asimismo cuando se desconoce la construcción de un objeto se generan problemas de compresión del modelo de objetos [1]. Por lo anteriormente descrito, en el presente artículo se da a conocer un módulo de aprendizaje como apoyo para la comprensión del PCIO en Java. Este módulo utiliza la biblioteca *Blockly* de *Google* [2] para generar el código fuente necesario para la comprensión de los pasos del proceso de construcción a través del efecto de arrastrar y soltar bloques visuales. Las principales tecnologías en este proyecto fueron *React* y *Spring* como marcos de trabajo y *Java* como lenguaje de programación. El principal beneficio del uso de esta herramienta es facilitar la comprensión del proceso de construcción e inicialización de objetos en estudiantes del área de programación o programadores en general.

Palabras clave — aprendizaje, Java, objeto, proceso de construcción

Abstract

Understanding the Process of Construction and Initialization of Objects (PCIO) is a key to learning any object-oriented programming language. Once the students and/or programmers identify the steps of this process, it's easier to code efficient programs because they work in a more conscious way of what is being programmed, also when the construction of an object is unknown, problems of object model compression [1]*. Due to the above, this article discloses a learning module to support the understanding of the PCIO in Java. This module uses Google's Blockly library* [2] *to generate the necessary source code for understanding the steps of the building process through the drag and drop effect of visual blocks. The main technologies in this project were React and Spring as frameworks and Java as programming language. The main benefit of using this tool is to facilitate the understanding of the process of construction and initialization of objects in students of the programming area or programmers in general.*

Keywords*— learning, Java, object, building process.*

1. INTRODUCCIÓN

Java es una plataforma informática de lenguaje de programación creada por *Sun Microsystems* en 1995 [3]. Dentro del ámbito de programación, Java es un lenguaje Orientado a Objetos que trata a estos últimos como entidades que tienen estado (representa los datos o valores) y comportamiento (representa la funcionalidad). La identidad de un objeto se implementa a través de un identificador único, es decir, ese identificador no es visible para usuarios comunes, sin embargo, la Máquina Virtual de Java (del inglés *Java Virtual Machine*) [4] utiliza ese identificador para monitorizar un objeto de forma única. Pero ¿por qué es importante comprender las características de un objeto durante su proceso de construcción e inicialización en un lenguaje de programación como Java? Mensualmente el índice TIOBE (*The Importance of Being Earnest*) [5] muestra el ranking de los lenguajes de programación más utilizados, indica en qué lenguaje se codificó más durante el último mes; por otro lado, el índice PYPL (*PopularitY of Programming Language*) [6] ofrece un ranking basado en *Google Trends* que señala las tendencias de búsqueda de *Google* en relación con la cantidad de veces que se realizaron búsquedas de tutoriales clasificados por lenguaje de programación, la clasificación indica que, cuanto mayor número de búsquedas tenga un lenguaje determinado, más popular es respecto a otros. Según análisis de la compañía *Edix Digital Workers*® [7] a través de los índices anteriores determinó que Java, un lenguaje de programación con amplío ámbito de aplicación, se encuentra dentro de los primeros tres lenguajes de programación más utilizados en el año 2022. Pero más allá de comprender un lenguaje de programación con alta demanda como Java, este artículo pretende abordar la problemática de la falta de comprensión en el funcionamiento de los objetos y ofrecer una solución. Considerando lo ya mencionado, un módulo de aprendizaje para el estudio del PCIO, se considera una herramienta valiosa e importante como apoyo para la formación de estudiantes. La importancia de comprender correctamente el proceso de construcción radica en un desarrollo eficiente, consciente y mejoras en la resolución de problemas.

Este artículo está organizado de la siguiente forma: en el contenido de la sección 2 se describen los pasos del PCIO, la descripción de la solución al problema y el desarrollo del módulo de aprendizaje; en la sección 3 se encuentran las conclusiones y trabajo a futuro; en la sección 4 se incluyen los agradecimientos y finalmente las referencias utilizadas en la sección 5.

2. CONTENIDO

2.1 Pasos del PCIO

En la literatura, Bruce Eckel [8] reporta algunos fragmentos del proceso de construcción, él menciona que es difícil

diseñar correctamente los objetos como cualquier otra cosa, pero la intención es que pocos expertos diseñen los mejores objetos para que otros los consuman.
Las primeras líneas de ejecución serán aquellas que contengan datos y métodos de clase, esto se refiere a todo aquello que incluye la palabra reservada `static`, con esto se entiende que le pertenece a la clase todo aquello que es estático, porque no es necesario crear un objeto nombrado o anónimo para acceder a miembros y métodos estáticos.
En el siguiente ejemplo se crea una clase con una variable estática inicializada:

```
class StaticTest {
    static int i = 47;
}
```

Hay dos formas de invocar una variable estática, la primera es a través de un objeto nombrado, ejemplo `obj.i` o hacer referencia directa a través del nombre de la clase:

```
StaticTest.i++;
```

Aquí se observa la diferencia con un miembro no estático, los miembros no estáticos se consideran pertenecientes al objeto que se crea o miembros de instancia. Ejemplo, en una clase con un método de instancia:

```
class Test {
    void noStatic(){}
}
```

La invocación al método no estático es posible sólo a través de un objeto nombrado o anónimo:

```
Test t1 = new Test();
t1.noStatic();

new Test().noStatic();
```

En el orden de inicialización se ejecutará primero todo aquello que es estático siempre que no haya sido inicializado por la creación de objeto anterior, y después todo aquello que no es estático.

De forma general Bruce Eckel menciona seis pasos correspondientes al PCIO:

1. Aunque el constructor no tiene explícitamente la palabra reservada `static`, se considera un método estático a nivel clase. Entonces, la primera vez que se cree un objeto de una clase o la primera vez que se acceda a un método o campo estático de la clase, el intérprete de Java localizará el archivo `nombredeclase.class`, esto se hace a través de la variable de entorno *Classpath*.
2. Se ejecutan los inicializadores estáticos. Esto significa que la inicialización estática tiene lugar una sola vez cuando el objeto de clase se carga por primera vez.
3. Cuando se crea un objeto nuevo, el primer paso del proceso de construcción de este es asignar suficiente espacio de almacenamiento en memoria para alojarse.
4. El almacenamiento se borra a cero, este está configurado automáticamente para todos los primitivos de la clase con valores default (cero para números y el equivalente para valores booleanos y char) y las referencias a *null*.
5. Se ejecuta cualquier inicialización que ocurra en el punto de definición del campo.
6. Se ejecutan los constructores.

Aunque los pasos anteriores describen correctamente el proceso de construcción, no son suficientes para lograr una comprensión detallada dado que el autor no considera lo siguiente:

- El detalle de la carga de clase.
- Los campos e inicializadores se ejecutan en el orden que aparezcan en el código de forma descendente.
- El correcto llamado al super constructor cuando se utiliza la herencia entre clases.

Con base en la literatura revisada en conjunto con pruebas de ejecución de programas especialmente diseñados para observar el comportamiento en cada parte estructural de código, se refinó la descripción de los pasos que describen adecuadamente y con mayor claridad y comprensión el PCIO. A continuación, se presentan los pasos de dicho proceso de forma detallada:

A nivel clase:

1. Búsqueda de clases mediante *Classpath* (válido únicamente para versiones de Java <= 1.8).
2. Reserva de memoria para la clase.
3. Ejecución de campos estáticos con valores *default*.
4. Ejecución de campos estáticos con valores explícitos.
5. Ejecución de bloques estáticos (inicializadores de clase).

A nivel objeto:

6. Reserva de memoria para el objeto.
7. Ejecución de campos de instancia con valores *default*.
8. Ejecución de campos de instancia con valores explícitos.
9. Ejecución de bloques de instancia (inicializadores).
10. Ejecución de los constructores.
    a. Siempre se ejecuta una instrucción de `super()`, ubicada en la primera línea, lo cual significa:
        i. Si en la superclase existe un constructor sin argumentos, se ejecuta de forma implícita (Java realiza este paso automáticamente).
        ii. Si en la superclase existe uno o más constructores con argumentos, se ejecuta de forma explícita con invocación al super constructor con argumentos establecidos.

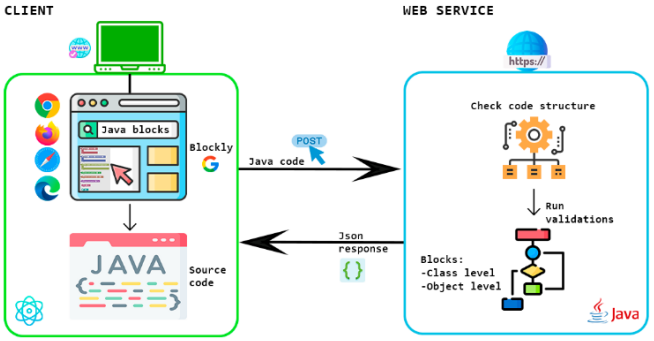
Los pasos 3 al 5 (a nivel clase) y 7 al 9 (a nivel objeto) son intercambiables según sea el orden de declaración en código.

2.2 Arquitectura del módulo de software

Una vez revisado y detallado el proceso de construcción se tiene como propósito principal ofrecer una herramienta de apoyo educativo para asegurar que los pasos del PCIO se siguen adecuadamente al momento de codificar o durante el proceso de aprendizaje de codificación.

Para resolver el problema planteado se propuso una arquitectura cliente-servidor (Fig. 1) como modelo de diseño del módulo de *software*.

Fig. 1. Arquitectura.



Fuente: Elaboración propia.

Cliente: se encuentra la Interfaz Gráfica de Usuario conocida también como GUI (del inglés *Graphical User Interface*) [9], aquí se genera el código fuente necesario para enviarse a validar. La creación de bloques de código se realiza a través de *Blockly*, una biblioteca de *Google*, 100% codificada en *JavaScript* para generar códigos de programación de forma visual a través de editores basados en bloques.
La implementación de *Blockly* se llevó a cabo a través de *React* [10] como marco de trabajo.

La selección de *Blockly* se debe a que es altamente personalizable para utilizarse con cualquier lenguaje de programación además de que se utiliza común y específicamente en proyectos con fines educativos.

Servidor: recibe el código fuente generado en el cliente y realiza validaciones de estructura de código, validaciones de salida de código de acuerdo con los pasos del proceso de construcción:

1. ¿Hay herencia en clase principal (clase pública)?
    a. ¿Hay herencia en la clase padre?
    b. Variables y métodos estáticos.
2. Variables y métodos estáticos de la clase principal.
3. ¿Se generan objetos nombrados o anónimos de otra clase?
4. Inicializadores.

Finalmente se realiza la ejecución del código a través de JVM para obtener la salida correspondiente y devolver al usuario una respuesta en formato *JSON* [11] con un resumen del código analizado.

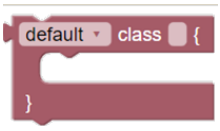
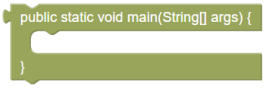
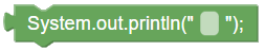
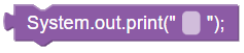
2.3 Desarrollo del módulo de aprendizaje

Parte fundamental del módulo de aprendizaje es el uso de bloques visuales para la generación de código fuente de acuerdo con el PCIO.

Para la correcta interacción del usuario con la aplicación se generaron bloques específicos y se personalizaron de acuerdo con lo que se pretende apoyar en cada paso del proceso de construcción, se contemplaron tres categorías: general, a nivel clase y a nivel objeto.

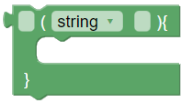
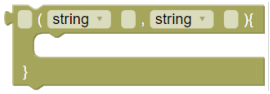
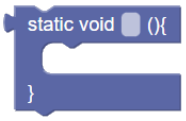
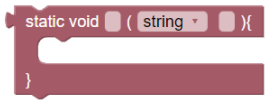
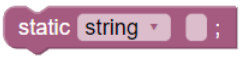
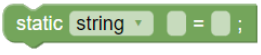
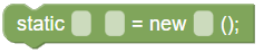
La definición de bloques se describe en las tablas 1, 2 y 3.

Tabla 1. Bloques de código generales.

Bloque	Forma visual	Descripción
Bloque de clase.		Genera el cuerpo de la clase con modificador de acceso y nombre personalizados.
Bloque de clase con herencia.		Genera el cuerpo de la clase con la opción para heredar de otra clase.
Bloque de método main.		Genera el cuerpo vacío del método de arranque de la aplicación.
Bloque de impresión en consola con salto de línea.		Agrega la línea de impresión con texto personalizado.
Bloque de impresión en consola sin salto de línea.		Agrega la línea de impresión con texto personalizado.
Bloque de invocación al super constructor.		Agrega la línea de super() con parámetros.
Bloque de objeto anónimo.		Agrega línea de objeto anónimo con nombre personalizado.

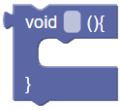
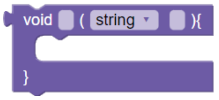
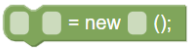
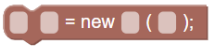
Fuente: elaboración propia.

Tabla 2. Bloques de código a nivel clase o estáticos.

Bloque	Forma visual	Descripción
Bloque de constructor sin parámetros.		Genera el cuerpo vacío de constructor.
Bloque de constructor con un parámetro.		Genera el cuerpo de constructor con la opción de enviar un parámetro tipado.
Bloque de constructor con dos parámetros.		Genera el cuerpo de constructor con la opción de enviar dos parámetros tipados.
Bloque de método estático sin parámetros.		Genera el cuerpo vacío de método estático con nombre personalizado.
Bloque de método estático con parámetro.		Genera el cuerpo vacío de método estático con nombre personalizado y un parámetro.
Bloque de inicializador de clase.		Genera el cuerpo de inicializador estático.
Bloque de variable de clase con valor *default*.		Agrega línea de variable con valor *default*.
Bloque de variable de clase con valor explícito.		Agrega línea de variable con valor explícito.
Bloque de variable de clase tipo referencia.		Agrega línea de variable inicializada.
Bloque de variable de clase tipo referencia con un parámetro.		Agrega línea de variable inicializada con un parámetro.

Fuente: elaboración propia.

Tabla 3. Bloques de código a nivel objeto.

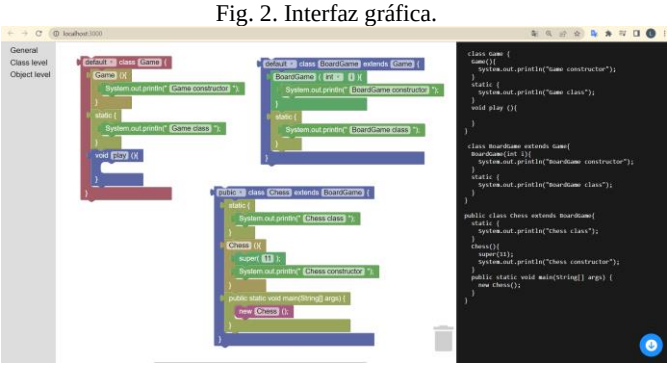
Bloque	Forma visual	Descripción
Bloque de inicializador de instancia.		Genera cuerpo vacío de inicializador.
Bloque de método sin valor de retorno.		Genera cuerpo vacío de método de instancia.
Bloque de método sin retorno con un parámetro personalizado.		Genera cuerpo vacío de método de instancia con un parámetro personalizado.
Bloque de variable de instancia tipo referencia.		Agrega línea de variable inicializada.
Bloque de variable de instancia tipo referencia con un parámetro.		Agrega línea de variable inicializada con un parámetro.

Fuente: elaboración propia.

Como se observa en la Tabla 3, la diferencia de algunos bloques con los bloques de la categoría a nivel clase radica únicamente en que no incluyen la palabra reservada `static`.

En la Fig. 2 se muestra la interfaz gráfica para el usuario, aquí se crean los bloques entrelazados y el código fuente se genera automáticamente acorde a los elementos utilizados.

Fig. 2. Interfaz gráfica.



Fuente: elaboración propia.

Una vez generado el código fuente se realiza en el servicio web:

1. La validación para asegurar la generación de código Java sintácticamente correcto.
2. La validación del correcto cumplimiento de la estructura de clases, campos y métodos para posteriormente ejecutar una serie de validaciones acordes a los pasos del PCIO y obtener un resultado con las líneas ejecutadas paso a paso.
3. Devolver una respuesta de resumen de ejecución y salida del código.

3. DISCUCIÓN DE RESULTADOS

Se realizaron pruebas de generación de bloques visuales a partir de códigos fuente ya existentes, estos códigos se han analizado y revisado detenidamente a lo largo de varios cursos de nivel licenciatura y posgrado, asegurando que cumplen con los elementos necesarios para comprender el proceso de construcción e inicialización de objetos.

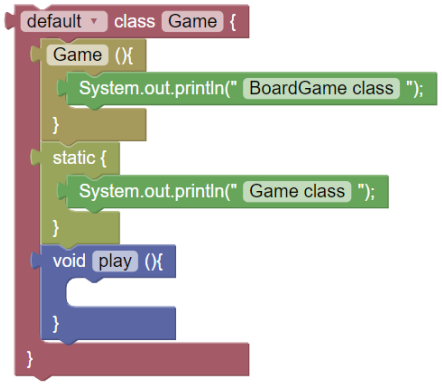
3.1 Ejemplo de código de pruebas

El siguiente código fue adaptado para probar *Blockly*, el ejemplo base se encuentra en la sección *Reusing Classes* [12] de Bruce Eckel:

```
class Game {
    Game() {
      out.println("Game constructor");
    }
    static {
      out.println("Game class");
    }
    void play(){}
}
class BoardGame extends Game {
    BoardGame(int i) {
        out.println("BoardGame constructor");
    }
    static {
      out.println("BoardGame class");
    }
}
public class Chess extends BoardGame {
    static {
      out.println("Chess class");
    }
    Chess() {
      super(11);
      out.println("Chess constructor");
    }
    public static void main(String[] args) {
         new Chess();
    }
}
```

En la Fig.3 se muestra la clase `Game` con todos los elementos que incluye el código anterior.

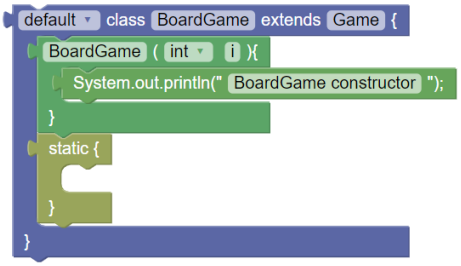
Fig. 3. Clase `Game`.



Fuente: Elaboración propia.

La Fig.4 muestra la clase `BoardGame` con todos los elementos que incluye además del uso de herencia hacia `Game`.

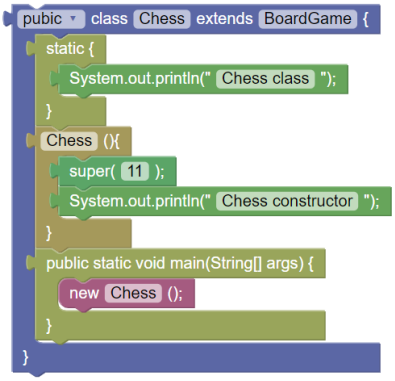
Fig. 4. Clase `BoardGame`.



Fuente: Elaboración propia.

En la Fig.5 se crea la clase `Chess` con todos sus elementos y hereda de `BoardGame`.

Fig.5. Clase `Chess`.



Fuente: Elaboración propia.

En la Fig.6 se muestra el código fuente que se genera a partir de la interfaz de *Blockly*, el cual coincide con el ejemplo de código de pruebas.

Fig.6. Código generado automáticamente

```
class Game {
 Game(){
   System.out.println("Game constructor");
 }
 static {
   System.out.println("Game class");
 }
 void play (){

 }
}

class BoardGame extends Game{
 BoardGame(int i){
   System.out.println("BoardGame constructor");
 }
 static {
   System.out.println("BoardGame class");
 }
}

public class Chess extends BoardGame{
 static {
   System.out.println("Chess class");
 }
 Chess(){
   super(11);
   System.out.println("Chess constructor");
 }
 public static void main(String[] args) {
   new Chess();
 }
}
```

Fuente: Resultado del ensamblado de bloques con *Blockly*.

Cabe destacar que esta herramienta también involucra el concepto de composición dentro de los elementos importantes para comprender el PCIO, la estructura composicional se verá reflejada en líneas de código como:

```
Y y = new Y();
```

Además del uso de herencia que se observa en el ejemplo de código de pruebas mediante la palabra reservada `extends`.

4. CONCLUSIONES

El proceso de creación de objetos va más allá de hacer una referencia de tipo, sin embargo, e independientemente del lenguaje de programación, en la literatura consultada no se reportó ningún trabajo directamente vinculado con el proceso de construcción e inicialización de objetos.

Lo anteriormente descrito se considera un área de oportunidad para el presente trabajo. Al considerar las características que brinda Java para trabajar con objetos y su proceso de construcción, se obtuvo un prototipo funcional del módulo de aprendizaje como apoyo para comprensión PCIO.

De esta forma se busca una amplia comprensión del lenguaje Java, específicamente en la construcción de objetos. Durante el desarrollo de sistemas, es necesario hacerlo de forma consiente a lo que se está programando, es por esto por lo que la correcta comprensión del PCIO es tema relevante para el área de programación.

Gracias a las características que ofrece *Blockly* para el diseño de bloques personalizados se generaron solamente los bloques necesarios para la comprensión del PCIO. Finalmente, este proyecto aborda el proceso de construcción de forma total como un complemento en la enseñanza.

4.1 Trabajo a futuro

Dentro de los planes a futuro se tiene: llevar a cabo pruebas del módulo de software en grupos con estudiantes de licenciatura y posgrado; así también es altamente deseable ejecutar pruebas con programadores interesados en alcanzar una certificación Java, por ejemplo.

De esta forma se pretende evaluar la herramienta para recibir la realimentación necesaria, realizar mejoras y liberar un módulo multiplataforma.

5. AGRADECIMIENTOS

Este trabajo fue patrocinado por el Consejo Nacional de Ciencia y Tecnología (CONACYT) y el Tecnológico Nacional de México (TecNM).

6. REFERENCIAS

[1] G. Booch, «The Object Model,» de *Object-Oriented Analysis and Design with Applications*, Boston, Pearson Education, Inc., 2007, pp. 29-74.

[2] Google, «Try Blockly,» [En línea]. Available: https://developers.google.com/blockly/. [Último acceso: Mayo 2022].

[3] Oracle, «¿Qué es la tecnología Java y por qué la necesito?,» [En línea]. Available: https://www.java.com/es/download/help/whatis_java.html.

[4] B. J. Evans, J. Gough y C. Newland, «Overview of the JVM,» de *Optimizing Java*, Sebastopol, O'Reilly Media, Inc., 2018, pp. 73-76.

[5] TIOBE Software BV, «TIOBE Index for January 2023,» [En línea]. Available: https://www.tiobe.com/tiobe-index/. [Último acceso: Enero 2023].

[6] Github, «PYPL PopularitY of Programming Language,» [En línea]. Available: https://pypl.github.io/PYPL.html. [Último acceso: Enero 2023].

[7] edix, «Los lenguajes de programación más usados,» 26 Julio 2022. [En línea]. Available: https://www.edix.com/es/instituto/lenguajes-de-programacion/. [Último acceso: Enero 2023].

[8] B. Eckel, «Constructor initialization,» de *Thinking in Java*, Cuarta ed., Upper Saddle River, Pearson Education, Inc., 2006, pp. 127-133.

[9] W. O. Galitz, «Introduction of the Graphical User Interface,» de *The Essential Guide to User Interface Design*, Segunda ed., Cánada, John Wiley & Sons, Inc., 2002, pp. 7-8.

[10] «React,» [En línea]. Available: https://es.reactjs.org/. [Último acceso: Enero 2023].

[11] «Introducción a JSON,» [En línea]. Available: https://www.json.org/json-es.html. [Último acceso: Agosto 2022].

[12] B. Eckel, «Constructors with arguments,» de *Thinking in Java*, Upper Saddle River, Pearson Education, Inc., 2006, pp. 170-171.