**Proyecto Final- Green Foot  
Programación Orientada a Objetos**

**Luigui Salazar**

**Lenguaje de Programación I**

***Profesor:* Eric Gustavo Coronel Castillo**

**Universidad Científica del Sur**

**2018**

**Contenido**

[1. Introducción](#_Toc494811069) 3

[2. Sobre Green Foot](#_Toc494811070) 3

[3. Instalacion del Programa](#_Toc494811071) 3

[4. Iniciando Green foot](#_Toc494811072) 5

[5. Clases y Objetos](#_Toc494811073) 5

[6. Parametros](#_Toc494811073) 5

[7. Sentencia IF/ELSE](#_Toc494811073) 8

[8. Bucle WHILE](#_Toc494811073) 9

[9. Array](#_Toc494811073) 10

[10. API de Greenfoot](#_Toc494811073) 11

[11. Exportando escenario](#_Toc494811073) 13

[12. Conclusion](#_Toc494811073) 13

[13. Bibliografia](#_Toc494811073) 14

# INTRODUCCION

La programación Orientada a objetos (POO) es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación.

Con la POO tenemos que aprender a pensar las cosas de una manera distinta, para escribir nuestros programas en términos de objetos, propiedades, métodos y otras cosas que veremos rápidamente para aclarar conceptos y dar una pequeña base que permita soltarnos un poco con este tipo de programación.

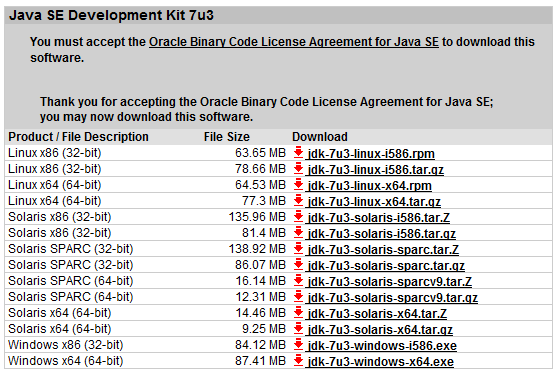
1. **SOBRE GREENFOOT**

[Greenfoot](http://www.greenfoot.org/)es un proyecto de software libre y gratuito,  desarrollado en la [Universidad de Kent](http://www.kent.ac.uk/)(UK),  enfocado en principio a jóvenes estudiantes que quieren iniciarse en el mundo de la programación. Existen otras herramientas con objetivos similares, como [Scratch](http://scratch.mit.edu/)o [Alice](http://www.alice.org/), pero Greenfoot tiene características más avanzadas, como la edición de código fuente en Java, lo que lo hace más potente e incluso interesante para personas que tienen ya conocimientos en programación. Greenfoot se distribuye gratuitamente con licencia [GNU GPL](http://es.wikipedia.org/wiki/GNU_GPL) y está disponible para Windows, Mac OS X y Linux.

1. **INSTALACION DEL PROGRAMA**

Para ejecutar Greenfoot es necesario tener instalado previamente el entorno de desarrollo Java (JDK),  que provee las herramientas para programar y ejecutar aplicaciones en este lenguaje. Podemos descargarlo de forma gratuita desde esta página: [http://www.oracle.com/technetwork/java/javase/downloads/jdk-7u3-download-1501626.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

Eligiendo la versión acorde a nuestro sistema operativo e instalándola según el proceso habitual de nuestro sistema.



Una vez instalado JDK, pasamos a la instalación del programa. Para ello nos dirigimos a la página <http://www.greenfoot.org/download>

Una vez allí, se nos presentan distintas versiones dependiendo del sistema operativo que usemos, e incluso podemos descargar una versión Java para cualquier SO o una que corra desde una memoria USB.

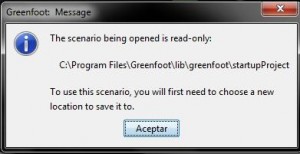


1. **INICIANDO GREENFOOT**

Una vez instalado, iniciamos Greenfoot. Sus aplicaciones se basan en **Escenarios**, que contienen todos los elementos necesarios para que dichos programas se visualicen y funcionen. Los escenarios incluyen un fondo gráfico, denominados Mundos (Worlds), y Actores (Actors ) que colocaremos en dichos mundos y que realizan acciones (ya lo veremos más adelante).

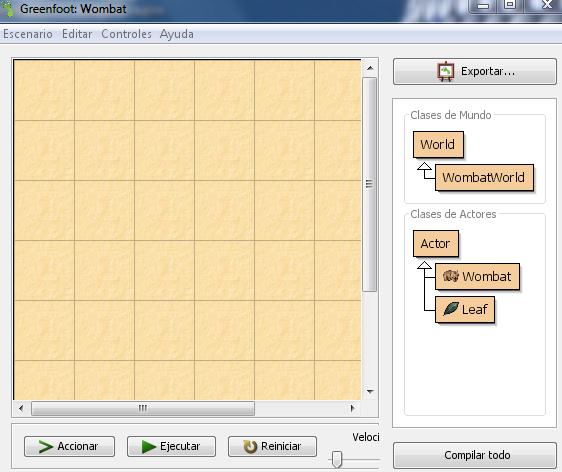
Hay muchos escenarios disponibles. El programa viene con algunos instalados pero también podemos acceder a escenarios creados por otros usuarios que los comparten libremente en la web de Greenfoot.

La primera vez que accedemos al programa nos saldrá un cuadro con opciones para empezar con un escenario ejemplo, elegir algún otro, crear uno nuevo o continuar sin cargar nada. Si elegimos un escenario, nos aparecerá otro mensaje indicándonos que debemos guardarlo en otra carpeta si queremos modificarlo:

1. **CLASES Y OBJETOS: JAVA Y POO CON GREENFOOT**

En primer lugar vemos que al abrir un escenario, por ejemplo el ‘Wombat’, y tras dar al botón de compilar en la parte inferior derecha no aparecen la siguiente pantalla, que será nuestra interfaz visual donde editaremos y ejecutaremos nuestros programas:



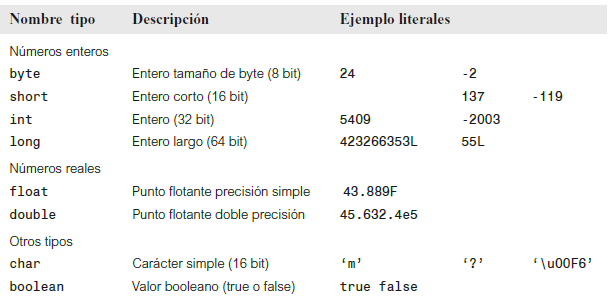
Se divide en tres partes principales:

1. El ***Mundo****,*que es el cuadrado con retícula de color arena que nos aparece en la izquierda. Es donde pondremos los objetos y donde se desarrollará el juego o simulación que programemos. Variará su apariencia en función del escenario que carguemos, ya que cada uno tiene un Mundo por defecto (en este caso es ‘*WombatWorld*“) y éste a su vez tiene una imagen de fondo que podemos cambiar fácilmente como veremos más adelante*.*
2. El Diagrama de Clases, en la parte derecha. Ahí nos aparecen tanto las clases de Mundo que tenemos disponibles y las clases de Actores (‘Wombat‘y ‘Leaf’) a partir de las cuales crearemos los objetos que incluiremos en el mundo elegido.
3. Los botones que aparecen en la parte inferior y que nos permiten ejecutar una serie de acciones que veremos a continuación.
4. **PARÁMETROS**

Los paréntesis después del nombre del método contienen la *lista de parámetros* (o *argumentos*)*.* Esto nos dice que el método requiere alguna información adicional para poder ejecutarse, y si la requiere, qué tipo de información.

Si sólo vemos un par de paréntesis sin nada más dentro (tal como hemos visto en todos los métodos hasta ahora), entonces el método tiene una lista vacía de parámetros. En otras palabras, no espera ningún parámetro —cuando invoquemos el método, simplemente se ejecutará. Si no hay nada dentro de los paréntesis, entonces el método espera uno o más parámetros —se debe proporcionar información adicional.

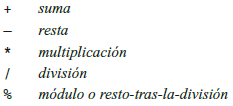
Las palabras int direction nos indican que este método espera un parámetro de tipo int, que especifica una dirección. Un parámetro es una información adicional que debemos proporcionar a un método para que se ejecute. Cada parámetro se define mediante dos palabras: primero, el tipo del parámetro (aquí: int) y luego el nombre, que nos da una idea de para qué se usa este parámetro. Si un método tiene un parámetro, necesitamos proporcionar esta información adicional cuando invoquemos el método.

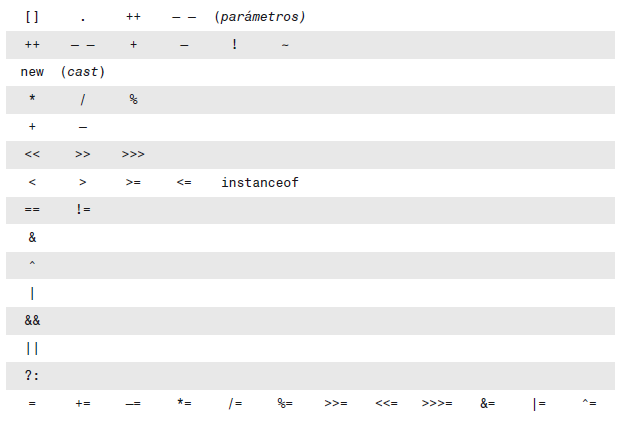


1. **OPERADORES**

Greenfoot tiene un gran número de operadores disponibles tanto para expresiones aritméticas como lógicas. La tabla muestra todo lo que está clasificado como un operador, incluyendo cosas como la conversión de tipos (casting) y el paso de parámetros. La mayoría de los operadores son operadores binarios (tomando un operando izquierdo y derecho) u operadores unarios (tomando un único operador).

Las principales operaciones aritméticas binarias son:





1. **SENTENCIA IF/ELSE**

Vamos a ver con más detalle la sentencia if. Como acabamos de ver, una sentencia if se puede escribir con la forma:

**If** (condición)

{

*Sentencias;*

}

**Else**

{

*Sentencias;*

}

Esta sentencia if contiene dos bloques (pares de llaves rodeando una lista de sentencias): la cláusula if y la cláusula-else (en este orden).

Cuando se ejecuta esta sentencia if, primero se evalúa la condición. Si la condición es true, entonces la cláusula if se ejecutará, y la ejecución continúa tras la cláusula else. Si la condición es falsa, no se ejecuta la cláusula if, sino que se ejecuta la cláusula else. Por tanto, uno de los dos bloques de sentencias se ejecuta siempre, pero nunca ambos.

La parte else con el segundo bloque es opcional —si se omite queda la versión más corta de la sentencia if que hemos visto antes.

1. **BUCLE WHILE**

Los lenguajes de programación ofrecen un constructor específico para hacer una tarea similar muchas veces: un bucle. Un bucle es un constructor de un lenguaje de programación que nos permite expresar órdenes como: Haz esta sentencia 20 veces» o «Llama a estos dos métodos 3 millones de veces» de forma fácil y concisa (sin escribir 3 millones de líneas de código). Java tiene varios tipos de bucles. Vamos a ver ahora el llamado bucle while.

Un bucle while sigue la siguiente forma:

**While** (condición)

{

*Sentencia*;

*Sentencia*;

...

}

La palabra clave while de Java está seguida de una condición entre paréntesis y de un bloque (un par de llaves) conteniendo una o más sentencias. Estas sentencias serán ejecutadas una y otra vez, hasta que la condición sea cierta (true).

Un patrón muy común de un bucle es ejecutar unas sentencias un número dado de veces. Para hacer esto, usamos una variable del bucle como contador. Es una práctica común nombrar a las variables de bucle i, así que lo haremos así. Aquí sigue un ejemplo que ejecuta el cuerpo del bucle while 100 veces:

**int** i = 0;

**While** (i < 100)

{

*Sentencia*;

*Sentencia*;

...

i = i + 1;

}

1. **ARRAYS**

Un array es un objeto que puede guardar muchas variables, y por tanto, puede almacenar muchos valores. Podemos ver esto en un diagrama. Supón que tenemos una variable llamada «nombre» de tipo String. Y que asignamos el String «Pepe» a esta variable:

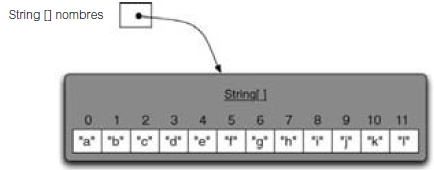
String nombre;

name = “Pepe”;



Este caso es muy simple. La variable es un contenedor que puede guardar un valor. El valor se almacena en la variable.

En el caso de un array, podemos obtener objetos separados —el objeto array— que guarda muchas variables. Podemos entonces almacenar una referencia a un objeto array en nuestra propia variable



El código Java para crear esta situación es:



En la declaración de la variable, el par de corchetes ([]) indica que el tipo de la variable es un array. La palabra delante de las corchetes indica el tipo de elemento del array, esto es, el tipo que cada entrada debería tener. Por tanto String[] representa un array de Strings, mientras que int[] representa un array de enteros (integers).

La expresión:



Crea el objeto array y lo rellena con los Strings de la «a» a la «l». Este objeto array se asigna entonces a nuestra variable nombres. Podemos ver en el diagrama que, cuando se asigna un objeto array a una variable, la variable contiene entonces un puntero a ese objeto.

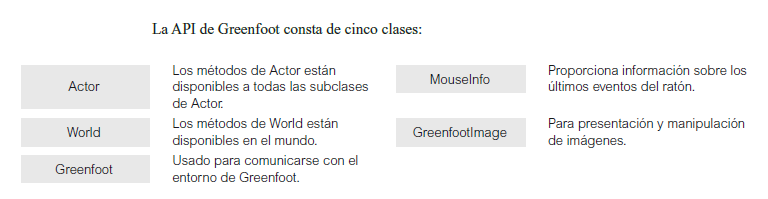
Una vez que tenemos nuestra variable array rellena, podemos acceder a los elementos individuales del array usando un *índice* —un número con la posición en el objeto array.

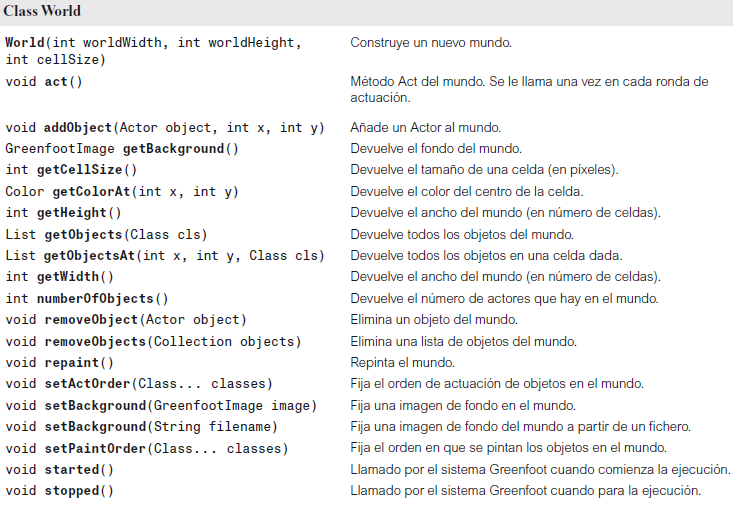
Accedemos a los elementos del array añadiendo el índice entre corchetes al nombre del array. Por ejemplo:

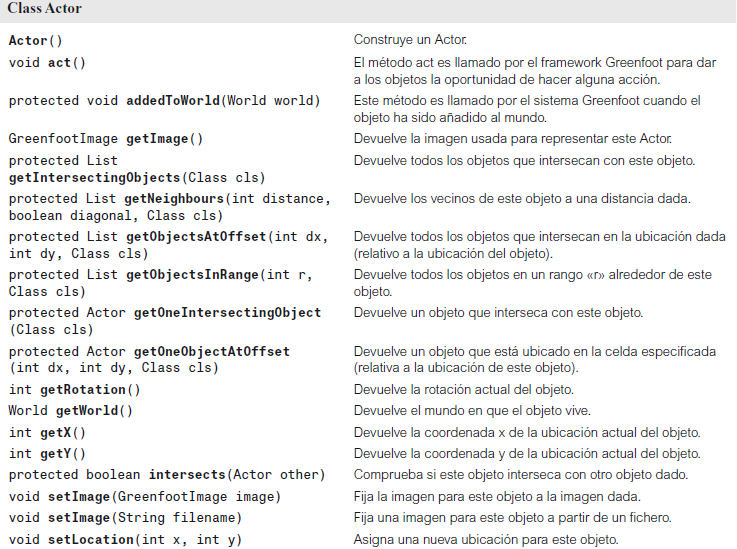


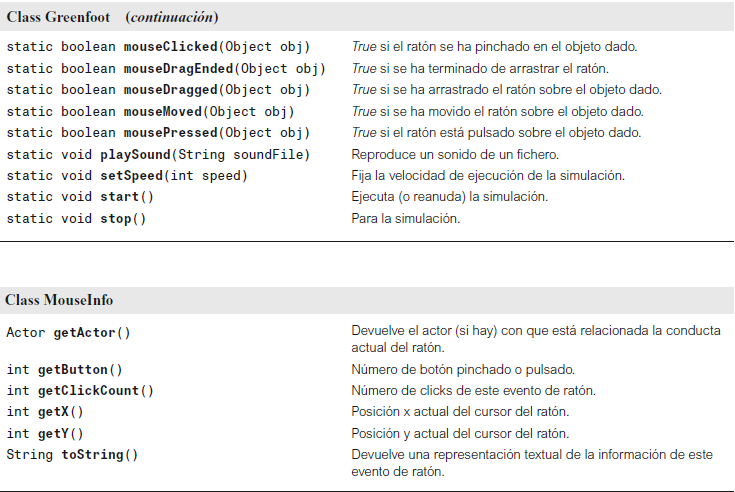
Accede al elemento del array con índice 3 —el String «d».

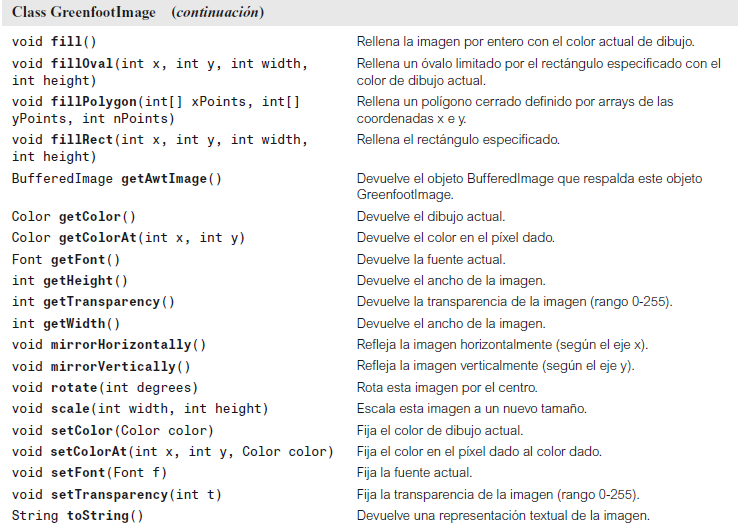
1. **API DE GREEN FOOT**











1. **EXPORTANDO EL ESCENARIO**

Cuando has terminado de escribir un escenario —puede ser un juego o una simulación— puedes querer que otros usuarios lo usen. Esos otros usuarios deberían tener la oportunidad de lanzar (y relanzar) el juego, pero no necesitan acceder al diagrama de clases o al código fuente. No deberían modificar el juego, sino que implemente deberían poder usarlo.

En Greenfoot, esto se hace exportando el escenario. Puedes exportar tu escenario seleccionando Exportar del menú Escenario. Te mostrará un diálogo que te deja escoger entre tres opciones: Aplicación, Página Web, y Publicar.

1. **CONCLUSION**

Greenfoot es una herramienta pertinente para implementar estrategias creativas para el aprendizaje de conceptos de la POO, ya que ofrece una valiosa oportunidad a los estudiantes para que apliquen y aprendan conceptos de programación de una forma interactiva y divertida.

1. **BIBLIOGRAFIA**

<https://www.researchgate.net/publication/303401503_Greenfoot_Un_IDE_para_Aprender_Programacion_Orientada_a_Objetos_con_el_Lenguaje_JavaR>

<http://endhowx.info/linux/7128-aprender-a-escribir-programas-en-java-con.html>

<https://mesa.ucsc.edu/mesa-day/2015-16-rules/greenfoot-resources.pdf>

<https://www.greenfoot.org/door>

<https://es.wikipedia.org/wiki/Greenfoot>