

Dragon Ball fanApp

Frank Aguilar Garzón

Frank.aguilar@pi.edu.co

Programación orientada a objetos

Politécnico Internacional, Sede sur (Autopista sur No. 67 - 71)Bogotá, Colombia

*Resumen*—Se desarrollará un sitio web teniendo en cuenta los cuatro pilares de la programación orientada a objetos; el polimorfismo, la abstracción, el encapsulamiento y la herencia, además de hacer uso de otros componentes como clases, objetos, métodos y atributos. El desarrollo de este proyecto está inspirado en la serie animada Dragon Ball del mangaka y diseñador de personajes japones Akira Toriyama, y está dirigido a sus espectadores con el objetivo de satisfacerlos con una plataforma exclusiva de la serie en la que se pueda acceder al contenido e información de manera sencilla e interactiva. El contenido del sitio web suministrará datos de los personajes, planetas, sagas, línea de producción, banda sonora y premios ganados de la serie el cual será creado, actualizado y controlado por medio de una base de datos.

Palabras clave—Polimorfismo, abstracción, encapsulación, herencia.

*Abstract*—A website will be developed taking into account the four pillars of object-oriented programming; polymorphism, abstraction, encapsulation, and inheritance, as well as making use of other components such as classes, objects, methods, and attributes. The development of this project is inspired by the animated series Dragon Ball by the Japanese mangaka and character designer Akira Toriyama, and is aimed at its viewers with the aim of satisfying them with an exclusive platform for the series where the content and content can be accessed. information in a simple and interactive way. The content of the website will provide data on the characters, planets, sagas, production line, soundtrack and awards won from the series which will be created, updated and controlled through databases.

Keywords—***Polymorphism, abstraction, encapsulation, inheritance.***

# Introducción

Este articulo presenta el desarrollo del proyecto Dragon Ball fanApp el cual es una plataforma web dirigida a los usuarios que desean tener conocimientos adicionales de la serie más allá de solo la historia. Dragon Ball es una serie animada japonesa del género acción, aventuras y fantasía transmitida por primera vez por el canal televisivo Fuji Television en Japón en 1989. Desde el año de su estreno, la serie gano una gran numero de fanes y espectadores8uuuuuuuuuu niños, jóvenes y adultos, que a medida que se desarrollaba la serie iban aumentando el interés por conocer un poco más sobre los personajes y sus fases, los planetas, los desarrolladores de la serie, fechas de lanzamiento de nuevas sagas o películas y hasta su banda sonora.

# Antes de empezar

## Para el desarrollo de este proyecto se utilizó el lenguaje de programación Java en una arquitectura tipo MVC (Modelo Vista Controlador) para lograr separar los datos, la interfaz de usuario y el controlador. También se hace uso de los cuatro pilares de la programación orientada a objetos que son el polimorfismo, la abstracción, el encapsulamiento y la herencia, además de la implementación de clases, objetos, métodos y atributos en los componentes creados, para así lograr la reutilización y segregación del código y garantizar que, en futuras intervenciones la arquitectura y el código del proyecto sean de fácil lectura y actualización. Para la creación, actualización y eliminación del contenido se implementa una base de datos. A continuación, vamos a definir algunos de los términos anteriormente mencionados para ayudar al entendimiento del lector.

* Según Fredy Geek en su canal de YouTube, el polimorfismo es la habilidad de un objeto de realizar una acción de diferentes maneras, utilizando métodos iguales que se implementan de forma diferente en varias clases y que permite definir distintos comportamientos para un objeto dependiendo la manera en que se realice la implementación. Con base en esto, concluimos que cualquier objeto tiene la habilidad de ejecutar métodos y acciones de manera diferente a otros objetos creados a partir de la misma clase padre, con el fin de independizar el objeto y que su comportamiento se acoja a la manera en que se lleve a cabo su ejecución.
* Fredy Geek dice que la abstracción es un principio que busca descartar toda la información que no resulta relevante en un contexto en particular, enfatizando algunos de los detalles o propiedades de los objetos. La atracción destaca las principales características y funcionalidades que un objeto desempeña, también llamadas atributos y métodos. Basándose en la explicación de Fredy Geek, con la abstracción destacamos las principales características de una clase con el fin de poder diferenciarlas de las demás gracias a sus aspectos más relevantes para no extendernos en todos sus detalles o características menos importantes que nos confundan al momento de crear y clasificar una clase. Estas características y funcionalidades se comparten entre objetos que pertenezcan a la misma clase.
* El los videos de Fredy Geek, se explica la encapsulación como un procedimiento que consiste en almacenar y organizar en una clase las características y funcionalidades de los objetos, representándolas por medio de atributos y métodos como efecto secundario de la abstracción. El encapsulamiento garantiza la integridad de los datos que contiene un objeto y evita el acceso a datos por cualquier otro medio distinto al que nosotros especificamos en la clase. También habla de la existencia de modificadores de acceso, que son palabras clave que se usan para especificar el nivel de acceso que podemos tener hacia un elemento de una clase. De acuerdo con Fredy Geek, podemos concluir que la encapsulación es una manera de tratar los atributos y los métodos de los objetos con el fin de proteger y restringir el ingreso de datos falsos o erróneos a nuestros objetos, además de permitir especificar el tipo de acceso que tenemos sobre los elementos de una clase, esto con el fin de brindar seguridad y restringir el acceso de algunos atributos o métodos que consideremos puedan almacenar información importante que pueda verse alterada o modificada sin seguir cierta ruta de acceso.
* Dice Fredy Geek que la herencia es el mecanismo donde se crean una o varias clases a partir de una que ya existe, estas nuevas clases se denominan subclases y la clase a partir de la cual se crean las subclases se denomina clase primaria, clase padre o superclase. Las subclases contienen atributos y métodos obtenidos de la clase padre y se pueden definir nuevos atributos y nuevos métodos para estas subclases. A partir de la definición de herencia de Fredy Geek, decimos que cuando una clase es creada a partir de otra clase, esta clase nueva se denomina subclase o clase hija y contiene atributos o métodos obtenidos de la clase de la cual fue creada, también llamada super clase, clase padre o clase primaria; a esto se le llama herencia. La subclase además de heredar métodos y atributos de su clase padre, puede definir nuevos atributos, métodos y ser al mismo tiempo clase padre de otra clase aplicando la jerarquización, que es una de las ventajas de la herencia junto a la reutilización de código.

# Pseudocódigo

En el pseudocódigo se definen paso por paso de manera resumida como se ejecutará nuestro programa. Pseudocódigo, es una palabra compuesta que significa código falso, y se emplea como paso intermedio en el desarrollo de software para describir de manera informal el algoritmo y el código del programa, lo que significa, que está escrito en lenguaje común; como considera Roa Mackenzie, “cada quien lo podrá representar con la palabra que quiera", porque así, es como definimos las líneas de instrucciones, procedimientos y funciones para la realización del código. A continuación, se relaciona el pseudocódigo de la ejecución del programa para la plataforma Dragon Ball fanApp.

1. Ingresar a la plataforma
2. Cargar el contenido de la página principal
3. Seleccionar una opción del menú
4. Consultar la base de datos
5. Traer resultados encontrados y mostrarlos en pantalla

# Diagrama de flujo

El diagrama de flujo nos sirve para esquematizar gráficamente de inicio a fin el algoritmo que usa nuestro programa para ejecutar sus procesos y mostrar el paso a paso de su ejecución de manera lógica y ordenada, creando una estructura conformada de figuras geométricas y flechas que representan procesos, entrada y salida de datos, toma de decisiones, documentos, bases de datos, entre otros y conectan cada una de estas figuras para indicar la dirección del flujo, respectivamente. Como afirma la Wikipedia, El diagrama de flujo o flujograma o diagrama de actividades es la [representación gráfica](https://es.wikipedia.org/wiki/Gr%C3%A1fica) de un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) o proceso. Se utiliza en disciplinas como [programación](https://es.wikipedia.org/wiki/Programaci%C3%B3n), [economía](https://es.wikipedia.org/wiki/Econom%C3%ADa), [procesos industriales](https://es.wikipedia.org/wiki/Proceso_de_fabricaci%C3%B3n) y [psicología cognitiva](https://es.wikipedia.org/wiki/Psicolog%C3%ADa_cognitiva). A continuación, se relaciona el diagrama de flujo para la plataforma Dragon Ball fanApp.

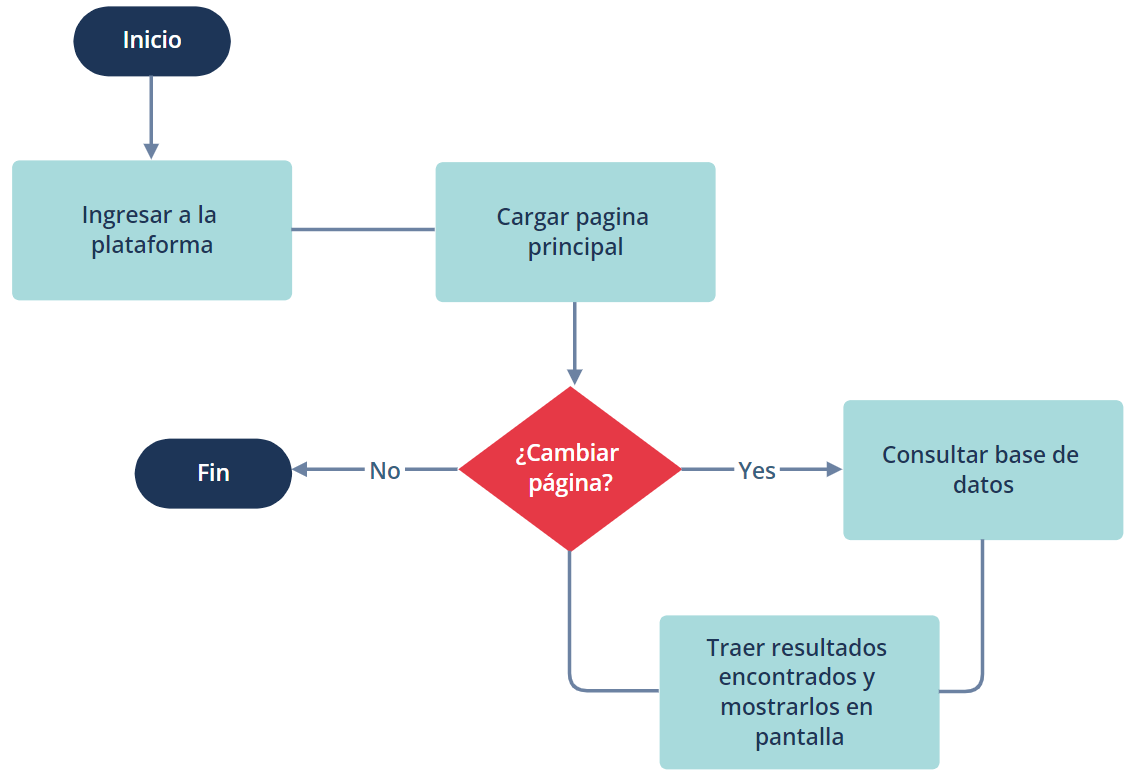


Fig 1. Diagrama de flujo

# Diagrama de arquitectura

Para este proyecto se usa una arquitectura MVC (Modelo Vista Controlador), que hace referencia a los tres componentes lógicos de un programa; la interfaz de usuario, los datos y la lógica de control. El modelo almacena los datos y la información que maneja el sistema, está listo para enviar datos al controlador y luego mostrarlos al usuario por medio de la interfaz gráfica. La vista se refiere a la interfaz gráfica del programa; con exactitud, es lo que el usuario ve en la pantalla de su ordenador y donde interactúa con los elementos gráficos de la interfaz para posteriormente obtener otra respuesta visual. El controlador ejecuta las funciones como intermediario entre la vista y el modelo para cumplir con las peticiones que haga el usuario a través de la interfaz gráfica. A continuación, se relaciona el diagrama de arquitectura para la plataforma Dragon Ball fanApp.

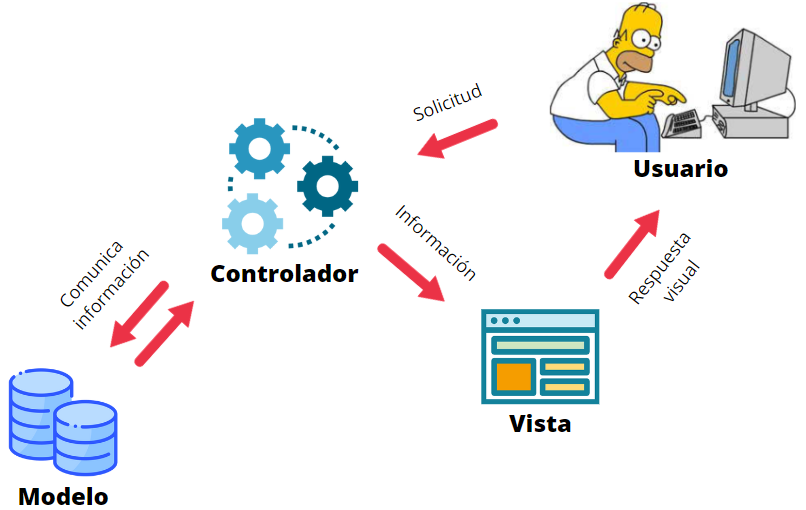


Fig 2. Diagrama de arquitectura MVC

# Diagrama de clases

Un diagrama de clases es un esquema de la composición de las clases de un programa, realizado a partir de tablas que contienen los atributos, los métodos y la relación entre ellas. Cada clase es representada mediante un rectángulo divido en tres partes de manera vertical, el nombre de la clase se escribe en la parte superior de la figura, luego van los atributos y el tipo de dato al que pertenece y por ultimo los métodos. La relación se define por medio de elementos que denotan el tipo de relación entre las clases, es decir, con una flecha indicamos que una clase hereda atributos y métodos de otra, una línea simple especifica que una clase está asociada a otra pero que no hay dependencia entre ellas, una línea con rombo blanco significa que una clase puede pertenecer a otra pero que no depende de ella y una línea con rombo negro señala que una clase pertenece a otra y que depende en su totalidad de ella; estas relaciones se conocen como, herencia, asociación, agregación y composición, respectivamente. A continuación, se relaciona el diagrama de clases para la plataforma Dragon Ball fanApp.

Nota: Profe, recuerde que debo cambiar el nombre de los atributos en esta imagen.

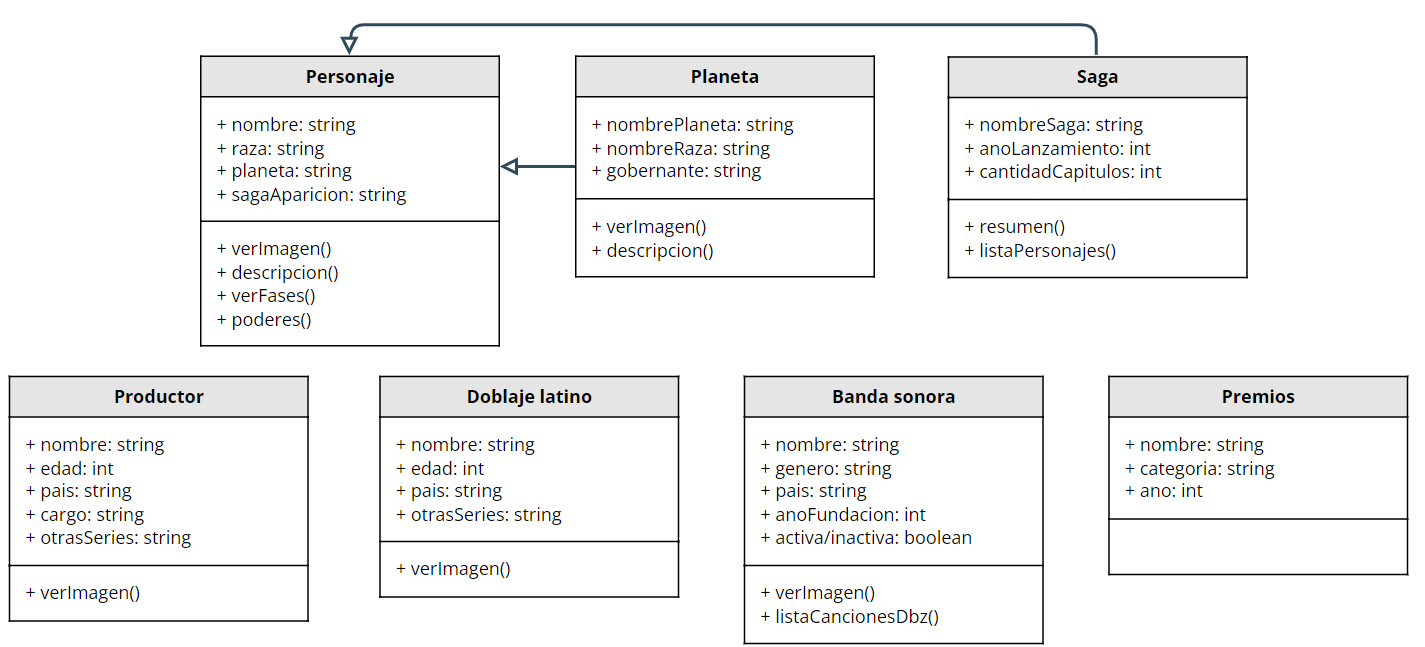


Fig 3. Diagrama de clases

# Diagrama de secuencia

El sitio web oficial de Lucidchart; la herramienta de diagramación, describe los diagramas de secuencia como una solución de modelado dinámico popular en UML porque se centran específicamente en *líneas de vida* o en los procesos y objetos que coexisten simultáneamente, y los mensajes intercambiados entre ellos para ejecutar una función antes de que la línea de vida termine. Tanto los desarrolladores de software como los profesionales de negocios usan estos diagramas para comprender los requisitos de un sistema nuevo o documentar un proceso existente. A los diagramas de secuencia en ocasiones se los conoce como diagramas de eventos o escenarios de eventos. A continuación, se relaciona el diagrama de secuencia para la plataforma Dragon Ball fanApp.

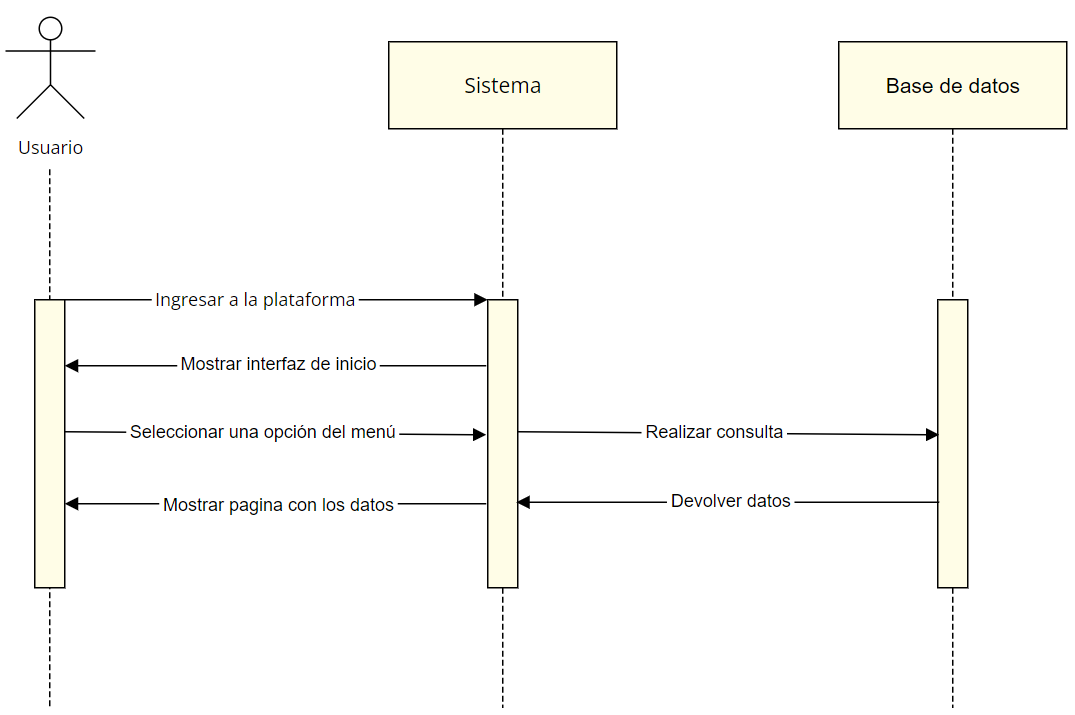


Fig 4. Diagrama de secuencia

# Diagrama de diccionario de datos

En el diagrama de diccionario de datos se definen en una lista el nombre del campo, el tamaño, el tipo de dato y una breve descripción de los parámetros que ocuparan los campos de las tablas en la base de datos. Para el desarrollo de este proyecto creamos siete clases llamadas personaje, planeta, saga, productor, doblaje latino, banda sonora y premios, que contendrán los atributos enlistados en el siguiente diagrama. A continuación, se relaciona el diagrama de diccionario de datos para la plataforma Dragon Ball fanApp.

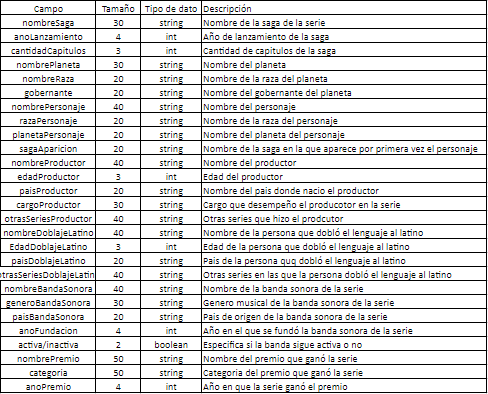


Fig 5. Diagrama de diccionario de datos

# Definición del proyecto

**DragonBallfanApp.java**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package dragon.ball.fanapp;

/\*\*

\*

\* @author frank

\*/

public class DragonBallFanApp {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

Planeta Tierra = new Planeta("Tierra", "Humana", "Ninguno");

Tierra.descripcionPlaneta();

Personaje Goku = new Personaje("Goku", "Sayayin", "Vegeta", "", "Dragon Ball");

Goku.descripcionPersonaje(); Este es un objeto y hace uso de polimorfismo porque se está haciendo el llamado de la clase Goku

}

}

**Planeta.java**

package dragon.ball.fanapp;

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author frank

\*/

public class Planeta { Esta es una clase

String nombrePlaneta; Aquí se hace uso de la abstracción al definir las propiedades más relevantes de la clase

String nombreRaza; Este es un atributo

String gobernante;

public Planeta(String nombrePlaneta, String nombreRaza, String gobernante){

this.nombrePlaneta = nombrePlaneta;

this.nombreRaza = nombreRaza;

this.gobernante = gobernante;

}

public String getnombrePlaneta(){ Aquí se hace uso del encapsulamiento al declarar el método como public

return nombrePlaneta;

}

public void setnombrePlaneta(String nombrePlaneta){

this.nombrePlaneta = nombrePlaneta;

}

public String getnombreRaza(){

return nombreRaza;

}

public void setnombreRaza(String nombreRaza){

this.nombreRaza = nombreRaza;

}

public String getgobernante(){

return gobernante;

}

public void setgobernante(String gobernante){

this.gobernante = gobernante;

}

public void descripcionPlaneta(){ Este es un método

System.out.println("DESCRIPCIÓN DEL PLANETA:" + "\nNombre: " + getnombrePlaneta() + "\nRaza: " + getnombreRaza() + "\nGobernante: " + getgobernante());

}

}

**Personaje.java**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package dragon.ball.fanapp;

/\*\*

\*

\* @author frank

\*/

public class Personaje extends Planeta { Aquí se hace uso de la herencia al hacer uso de la palabra reservada extends

String nombrePersonaje;

String sagaAparicion;

public Personaje(String nombrePersonaje, String nombreRaza, String nombrePlaneta, String gobernante, String sagaAparicion){

super(nombreRaza, nombrePlaneta, gobernante);

this.nombrePersonaje = nombrePersonaje;

this.sagaAparicion = sagaAparicion;

}

public String getnombrePersonaje(){

return nombrePersonaje;

}

public void setnombrePersonaje(String nombrePersonaje){

this.nombrePersonaje = nombrePersonaje;

}

public String getsagaAparicion(){

return sagaAparicion;

}

public void setsagaAparicion(String sagaAparicion){

this.sagaAparicion = sagaAparicion;

}

public void descripcionPersonaje(){

System.out.println("DESCRIPCIÓN DEL PERSONAJE:" + "\nNombre: " + getnombrePersonaje() + "\nRaza: " + getnombreRaza() + "\nPlaneta: " + getnombrePlaneta() + "\nSaga en la que aparece: " + getsagaAparicion());

}

}

# IMPLEMENTACION DEL PROYECTO (AVANCE)

1. Agregar el paso a paso de como descargar JDK y NetBeans
2. Para la implementación del proyecto, hacemos uso del JDK (Java Develompment Kit) para interpretar el lenguaje Java y sus herramientas de desarrollo y el IDE de desarrollo NetBeans versión 8.2.
3. En el video de YouTube “Instalación del JDK y NetBeans 8.2 para iniciar en java” del canal de Charly Cimino, se encuentran los links de descarga del JDK 8 y NetBeans 8:2. <https://www.youtube.com/watch?v=ywqyv3m_sng>

##### Referencias

1. Fredy Geek. (2019b, abril 30). *¿Qué es el Polimorfismo? - Programación Orientada a Objetos* [Vídeo]. YouTube. Recuperado 6 de septiembre de 2022, de <https://www.youtube.com/watch?v=tjjecfz9Cvk>
2. Fredy Geek. (2018b, marzo 30). *¿Qué es la Abstracción? - Programación Orientada a Objetos* [Vídeo]. YouTube. Recuperado 6 de septiembre de 2022, de <https://www.youtube.com/watch?v=oBO01Cx_YwQ>
3. Fredy Geek. (2018b, abril 4). *¿Qué es el Encapsulamiento? - Programación Orientada a Objetos* [Vídeo]. YouTube. Recuperado 6 de septiembre de 2022, de <https://www.youtube.com/watch?v=gR0EssHrl24>
4. Fredy Geek. (2019a, febrero 13). *¿Qué es la Herencia? - Programación Orientada a Objetos* [Vídeo]. YouTube. Recuperado 6 de septiembre de 2022, de <https://www.youtube.com/watch?v=9NynVRpZzv4>
5. Mackenzie. (1991). *Curso básico de programación*. McGraw-Hill Education.
6. colaboradores de Wikipedia. (2022, 30 julio). *Diagrama de flujo*. Wikipedia, la enciclopedia libre. Recuperado 7 de septiembre de 2022, de <https://es.wikipedia.org/wiki/Diagrama_de_flujo>
7. *Tutorial de diagrama de secuencia UML*. (s. f.). Lucidchart. <https://www.lucidchart.com/pages/es/diagrama-de-secuencia>