Dragon Ball fan App

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 programa 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 programa está inspirado en la serie animada Dragon Ball del mangaka y diseñador de personajes japones Akira Toriyama. Este proyecto 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 suministrará datos de los personajes, planetas, sagas y línea de producción de la serie.

En una próxima versión se implementará una base de datos para controlar el ingreso y consulta de los registros y datos de la plataforma.

Palabras clave—Polimorfismo, abstracción, encapsulación, herencia.

Abstract— A program will be developed considering 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 program is inspired by the animated series Dragon Ball by Japanese mangaka and character designer Akira Toriyama. This project is aimed at its viewers with the aim of satisfying them with an exclusive platform for the series where content and information can be accessed in a simple and interactive way. The content will provide data on the characters, planets, sagas, and production line of the series.

In a future version, a database will be implemented to control the entry and consultation of the records and data of the platform.

Keywords—Polymorphism, abstraction, encapsulation, inheritance.

I. INTRODUCCIÓN

Este articulo presenta el programa Dragon Ball fan App, anexando imágenes y explicando su implementación para evidenciar su desarrollo. Este proyecto está dirigido a los usuarios que desean tener conocimientos adicionales de la serie Dragon Ball, como por ejemplo algunos datos interesantes que no fueron referidos en el transcurso de su 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 el año 1986. Desde su estreno, la serie ganó un gran número fanes y espectadores niños, jóvenes y adultos, que con el paso de los años y con el transcurrir de la serie, quisieron conocer más sobre los personajes, sus fases, sus poderes, los planetas, los productores de la serie y fechas de lanzamiento de nuevas sagas o películas.

II. Antes de empezar

A. 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 programa sean de fácil lectura y actualización.

En una próxima versión se implementará una base de datos para el ingreso y consulta de los registros y datos de la plataforma. A continuación, vamos a definir algunos de los términos anteriormente mencionados para ayudar al entendimiento del lector.

- Según Fredy Geek (2019) 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 (2018) 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 abstracción 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.

- En los videos de Fredy Geek (2018), 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. La encapsulación además permite especificar el tipo de acceso que tenemos sobre los elementos de una clase para brindar seguridad e impedir el paso a 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 (2019) 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 métodos para estas subclases. A partir de la definición de herencia de Fredy Geek, concluimos 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.

III. Pseudocódigo

En el pseudocódigo se definen paso por paso de manera resumida cómo 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 (1991), “cada uno 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.

En una próxima versión se implementará una base de datos para el ingreso y consulta de los registros y datos de la plataforma. A continuación, se relaciona el pseudocódigo de la ejecución del programa para la plataforma Dragon Ball fan App.

1. Cargar el contenido y menú principal.
2. Pedir al usuario ingresar un valor numérico correspondiente a una opción del menú principal.
3. Seleccionar una opción del menú principal.
4. Consultar los registros de la base de datos.
5. Imprimir el nombre de cada registro encontrado en la base de datos.
6. Pedir al usuario ingresar un valor numérico correspondiente a un registro.
7. Seleccionar un registro del menú.
8. Cargar el menú del registro seleccionado con sus respectivas funciones.
9. Seleccionar una función del menú.
10. Consultar la información del registro en la base de datos.
11. Imprimir la información encontrada y cargar nuevamente el menú del registro.
12. Si el usuario elige la opción salir, el programa se devuelve al menú principal.
13. Pedir al usuario ingresar un valor numérico correspondiente a una opción del menú principal.
14. Si el usuario elige la opción salir, el programa finalizará.

IV. 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 (2022), El diagrama de flujo o flujograma o diagrama de actividades es la representación gráfica de un algoritmo o proceso. Se utiliza en disciplinas como programación, economía, procesos industriales y psicología cognitiva.

En una próxima versión se implementará una base de datos para el ingreso y consulta de los registros y datos de la plataforma. A continuación, se relaciona el diagrama de flujo para la plataforma Dragon Ball fan App.

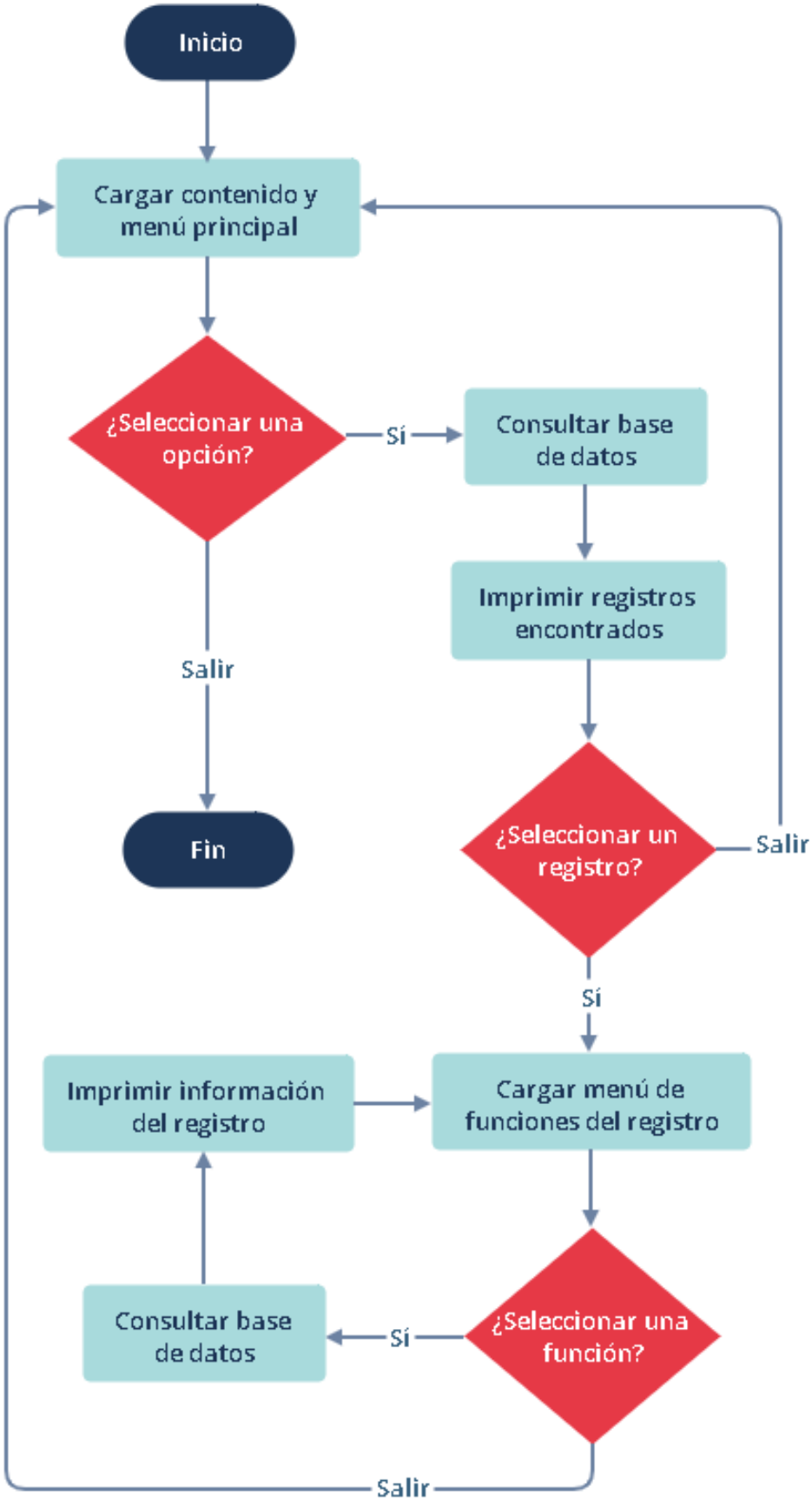

Fig 1. Diagrama de flujo

V. 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.

En una próxima versión se implementará una base de datos para el ingreso y consulta de los registros y datos de la plataforma. A continuación, se relaciona el diagrama de arquitectura para la plataforma Dragon Ball fan App.

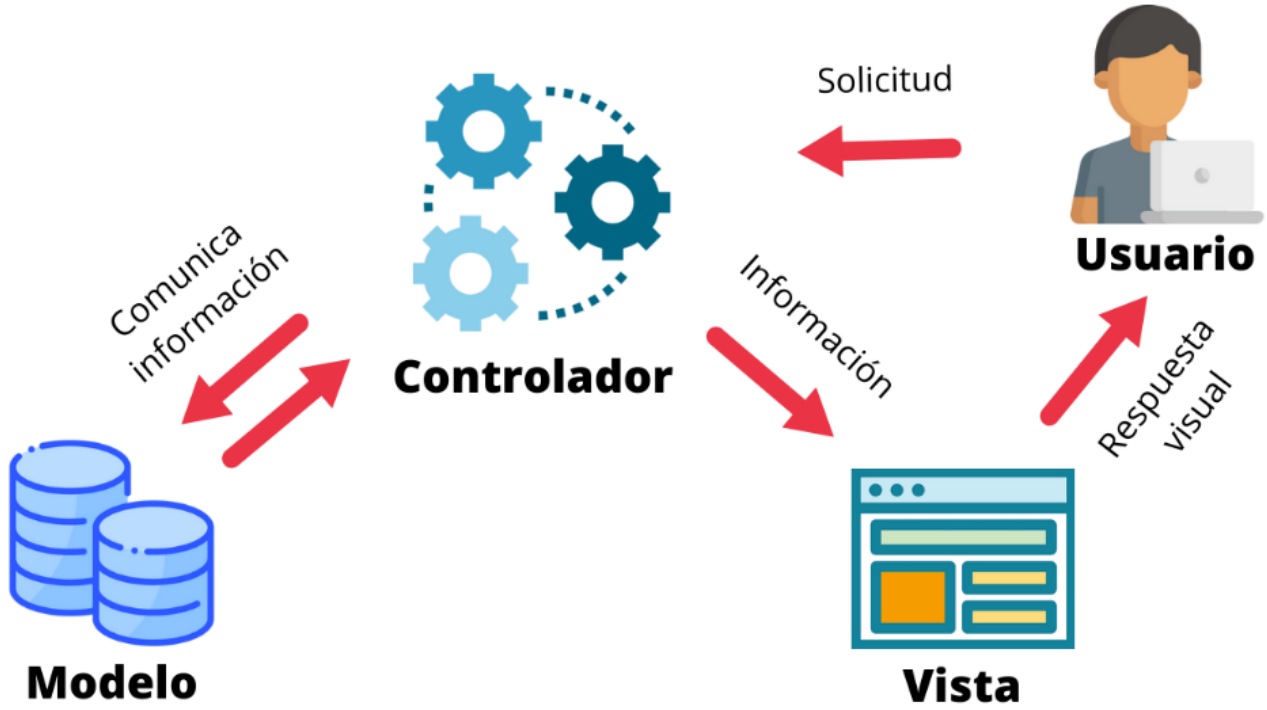

Fig 2. Diagrama de arquitectura MVC

VI. 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.

En una próxima versión se implementará una base de datos para el ingreso y consulta de los registros y datos de la plataforma. Las tablas de la base de datos tendrán asignados el nombre y tipo de dato que se muestran en el siguiente diagrama. A continuación, se relaciona el diagrama de clases para la plataforma Dragon Ball fan App.

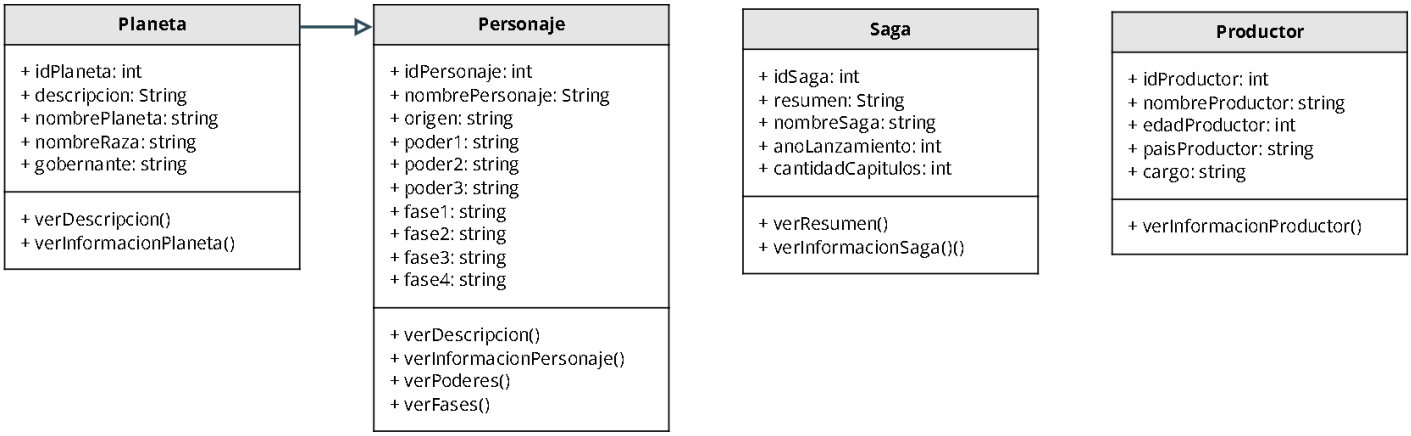

Fig 3. Diagrama de clases

VII. DIAGRAMA DE SECUENCIA

El sitio web oficial de Lucidchart (s.f.); 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.

En una próxima versión se implementará una base de datos para el ingreso y consulta de los registros y datos de la plataforma. A continuación, se relaciona el diagrama de secuencia para la plataforma Dragon Ball fan App.

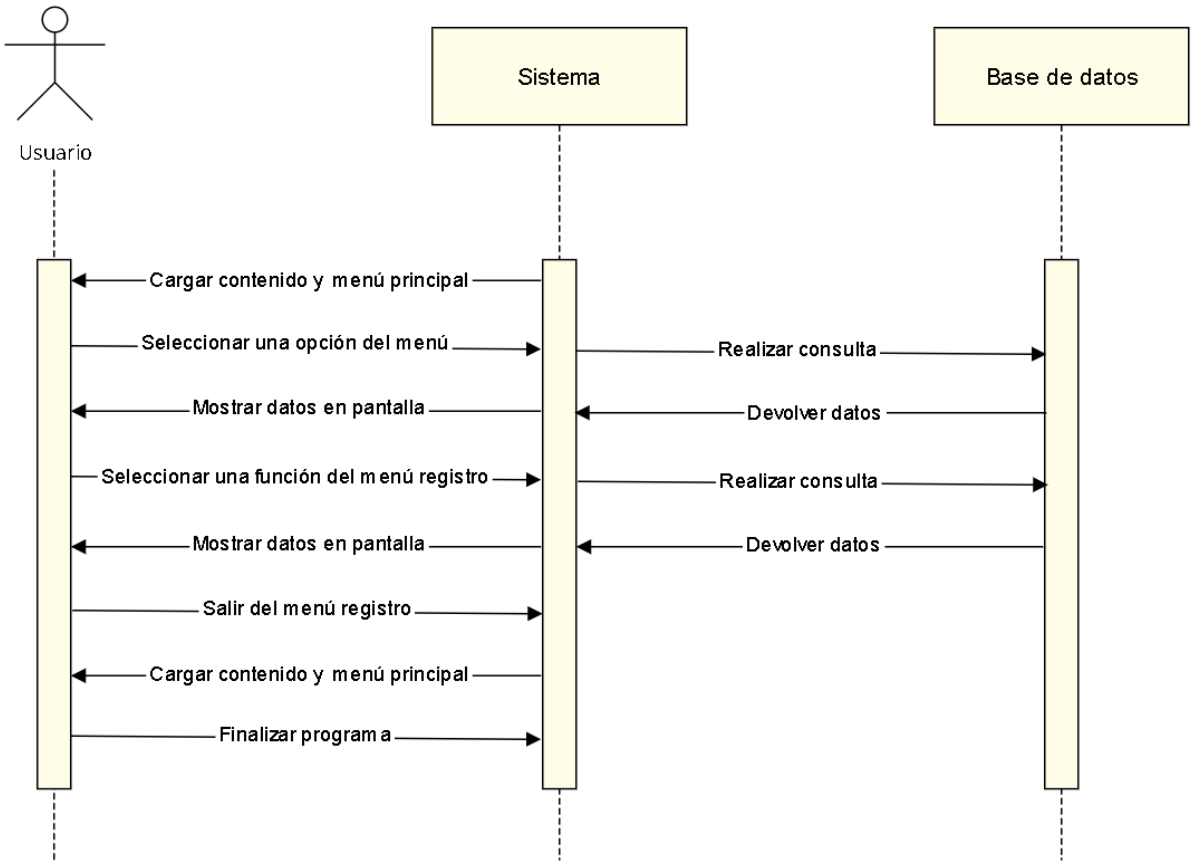

Fig 4. Diagrama de secuencia

VIII. 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 cuatro clases llamadas personaje, planeta, saga y productor, que contendrán los atributos enlistados en el siguiente diagrama.

En una próxima versión se implementará una base de datos para el ingreso y consulta de los registros y datos de la plataforma. Los campos en las tablas de la base de datos tendrán asignados el nombre, tamaño y tipo de dato que se muestran en el siguiente diagrama. A continuación, se relaciona el diagrama de diccionario de datos para la plataforma Dragon Ball fan App.

Campo	Tamaño	Tipo de dato	Descipción
idPlaneta	3	int	Identificador único del planeta
descripcion	500	String	Descripcion del planeta
nombrePlaneta	30	String	Nombre del planeta
nombreRaza	20	Sring	Nombre de la raza del planeta
gobernante	20	String	Nombre del gobernante del planeta
idPersonaje	3	int	Identificador único del personaje
nombrePersonaje	30	String	Nombre del personaje
origen	30	String	Origen del personaje
poder1	30	String	Primer poder del personaje
poder2	30	String	Segundo poder del personaje
poder3	30	String	Tercer poder del personaje
fase1	30	String	Primera fase de tranformación del personaje
fase2	30	String	Segunda fase de transformación del personaje
fase3	30	String	Tercera fase de transformación del personaje
fase4	30	String	Cuarta fase de transformación del personaje
idSaga	3	int	Identificador único de la saga
resumen	700	String	Resumen de la saga
nombreSaga	50	String	Nombre de la saga
anoLanzamiento	4	int	Año de lanzamiento de la saga
cantidadCapitulos	3	int	Cantidad de capitulos de la saga
idProductor	3	int	Identificador único del productor
nombreProductor	30	String	Nombre del productor
edadProductor	3	int	Edad del productor
paisProductor	30	String	País de origen del productor
cargo	50	String	Cargo del productor en la serie

Fig 5. Diagrama de diccionario de datos

IX. Implementación del proyecto

El programa Dragon Ball fan App ha sido desarrollado en el lenguaje de programación Java, para ponerlo en funcionamiento es necesario descargar e instalar algunos programas y archivos que se indican en la siguiente lista. Para la correcta ejecución del programa es necesario seguir los siguientes pasos en el mismo orden en que han sido enumerados.

1. Descargar e instalar el JDK (Java Development Kit) versión 8 para interpretar el lenguaje Java.
   1.1. Enlace de descarga para sistema operativo de 32 bits (https://drive.google.com/file/d/1G0cGAt74LLvX_5Z1xuhLzKGvtnfnExm2/view).

1.2. Enlace de descarga para sistema operativo de 64 bits (https://drive.google.com/file/d/1w4Khoz6hl7SvulYOFj-4W4UiwEWBBnTG/view).

2. Descargar e instalar el IDE (Integrated Development Environment) NetBeans versión 8.2.
   2.1. Enlace de descarga para sistema operativo de 32 bits (https://drive.google.com/file/d/1tuozk0dYsVuRs4P-bYvpPznsivxy49NJ/view).
   2.2. Enlace de descarga para sistema operativo de 64 bits (https://filehippo.com/es/download_netbeans/8.2/).
3. En la barra de direcciones del navegador, ingresar la URL (https://github.com/FrankGitMaster/DragonBallfanApp/archive/refs/heads/main.zip) y pulsar la tecla Enter para descargar el código fuente del programa en una carpeta comprimida en zip.
4. Extraer los archivos de la carpeta comprimida en zip.
5. Iniciar el programa NetBeans IDE 8.2.
6. En el menú superior, hacer clic en File, Open Project y buscar la ruta del proyecto nombrado “Dragon Ball fanApp” y seleccionarlo.
7. En el menú superior, hacer clic en Window y Projects.
8. En la ventana Projects, oprimir clic derecho sobre el proyecto Dragon Ball fanApp, luego dar clic en Run.
9. El programa se ejecutará.

X. Definición del proyecto

El programa Dragon Ball fan App ha sido desarrollado con base en los cuatro pilares de la programación orientada a objetos y la implementación de clases, objetos, atributos y métodos. Para el programa, se crearon cuatro clases definidas como Saga, Planeta, Personaje y Productor, cada una con sus respectivos objetos, atributos y métodos e implementados bajo los estándares de la programación orientada a objetos. A continuación, se anexan capturas de pantalla del código fuente del programa Dragon Ball fan App en el entorno de desarrollo NetBeans 8.2 para ayudar a la interpretación del lector.

Implementación del polimorfismo (línea 54) y objetos: Se crea el objeto goku, se definen los atributos de la clase Personaje y los heredados de su clase padre Planeta. Se aplica el polimorfismo porque no todos los atributos heredados se usan, en concreto, se realiza la implementación de manera diferente. Se puede evidenciar también la creación de cuatro objetos para la clase Personaje (línea 53 hasta la 59) y cinco objetos para la clase Productor (línea 68 hasta la 72).

```
51          // Clase Personaje
52          // - Objetos
53          Personaje goku = new Personaje(0, 1, "Su nombre real y de nacimiento es Kakaroto y es uno de los pocos sayayines que loc
54                  ,"Kakaroto / Son Goku", "Vegeta", "Sayayin", "", "Dragon Ball", "Genkidama", "Kamehameha"
55                  , "Teletransportación", "Súper Sayayin", "Súper Sayayin fase 2", "Súper Sayayin fase 3"
56                  , "Súper Sayayin fase 4");
57          Personaje gohan = new Personaje(0, 2, "El nombre fue dado en honor a su bisabuelo adoptivo. Cuando era un recién nacido,
58          Personaje vegeta = new Personaje(0, 3, "Vegeta IV reconocido como el Príncipe Vegeta y conocido mayormente como Vegeta e
59          Personaje pikoro = new Personaje(0, 4, "Pikoro o Pikoro Jr. es un namekiano que surgió tras ser creado en los últimos mc
60          // - Lista de objetos
61          List<Personaje> listaPersonajes = new ArrayList<>();
62          listaPersonajes.add(goku);
63          listaPersonajes.add(gohan);
64          listaPersonajes.add(vegeta);
65          listaPersonajes.add(pikoro);
66          // Clase Productor
67          // - Objetos
68          Productor akiraToriyama = new Productor(1, "Akira Toriyama", "67 años", "Japón", "Autor original");
69          Productor kozomorishita = new Productor(2, "Kozo Morishita", "74 años", "Japón", "Productor ejecutivo");
70          Productor kenjiShimizu = new Productor(3, "Kenji Shimizu", "61 años", "Japón", "Productor");
71          Productor takaoKoyama = new Productor(4, "Takao Koyama", "74 años", "Japón", "Composición de la serie");
72          Productor minoruMaeda = new Productor(5, "Minoru Maeda", "68 años", "Japón", "Diseño de personajes");
73          // - Lista de objetos
74          List<Productor> listaProductores = new ArrayList<>();
75          listaProductores.add(akiraToriyama);
76          listaProductores.add(kozomorishita);
77          listaProductores.add(kenjiShimizu);
78          listaProductores.add(takaoKoyama);
79          listaProductores.add(minoruMaeda);
80
81          //PROGRAMA
```

Fig 6. Implementación del polimorfismo

Implementación de la abstracción (línea 14 hasta la 18) y atributos: Se hace uso de la abstracción al definir las características más relevantes para la clase Planeta. Para esta clase se creó el atributo de tipo int idPlaneta (línea 14) como identificador único y descripcion, nombrePlaneta, nombreRaza y gobernate (línea 15 hasta la 18) como atributos de tipo String para almacenar campos de texto.

```
public class Planeta {

    int idPlaneta;
    String descripcion;
    String nombrePlaneta;
    String nombreRaza;
    String gobernante;

    public Planeta(int idPlaneta, String descripcion, String nombrePlaneta, String nombreRaza, String gobernante) {
        this.idPlaneta = idPlaneta;
        this.descripcion = descripcion;
        this.nombrePlaneta = nombrePlaneta;
        this.nombreRaza = nombreRaza;
        this.gobernante = gobernante;
    }

    public int getidPlaneta(){
        return idPlaneta;
    }

    public void setidPlaneta(int idPlaneta){
        this.idPlaneta = idPlaneta;
    }

    public String getdescripcion() {
        return descripcion;
    }

    public void setdescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

```

Fig 7. Implementación de la abstracción

Implementación del encapsulamiento (línea 119) y métodos: El encapsulamiento se puede evidenciar cuando se crea el método verDescripcionPersonaje y se declara público, es decir que es posible acceder a este método desde cualquier ubicación del programa. Podemos ver que para la clase Personaje han sido creados tres métodos más (línea 123 hasta la 133) que de igual manera son encapsulados pública.

```
    public void setfase3(String fase3) {
        this.fase3 = fase3;
    }

    public String getfase4() {
        return fase4;
    }

    public void setfase4(String fase4) {
        this.fase4 = fase4;
    }

    public void verDescripcionPersonaje() {
        System.out.println("\nDESCRIPCIÓN DE " + nombrePersonaje.toUpperCase() + ":\n" + getdescripcion());
    }

    public void verInformacionPersonaje() {
        System.out.println("\nINFORMACIÓN DE " + nombrePersonaje.toUpperCase() + ":\n" + "Nombre: " + getnombrePersonaje() + "\r
    }

    public void verPoderes() {
        System.out.println("\nPODERES DE " + nombrePersonaje.toUpperCase() + ":\n" + getpoder1() + "\n" + getpoder2() + "\n" + g
    }

    public void verFases() {
        System.out.println("\nFASES DE " + nombrePersonaje.toUpperCase() + ":\n" + getfase1() + "\n" + getfase2() + "\n" + getfa
    }

}
```

Fig 8. Implementación del encapsulamiento

Implementación de la herencia (línea 13 y 27) y clases: Aquí creamos la clase Personaje y se observa que se hace uso de la palabra reservada extends, esto significa que la clase Personaje heredará los métodos y atributos de la clase Planeta. En el super se agregan todos los atributos que hereda la clase Personaje de la clase Planeta.

```
12
13  public class Personaje extends Planeta {
14
15      int idPersonaje;
16      String nombrePersonaje;
17      String origen;
18      String poder1;
19      String poder2;
20      String poder3;
21      String fase1;
22      String fase2;
23      String fase3;
24      String fase4;
25
26      public Personaje(int idPlaneta, int idPersonaje, String descripcion, String nombrePersonaje, String nombrePlaneta, String no
27          super(idPlaneta, descripcion, nombrePlaneta, nombreRaza, gobernante);
28          this.idPersonaje = idPersonaje;
29          this.nombrePersonaje = nombrePersonaje;
30          this.origen = origen;
31          this.poder1 = poder1;
32          this.poder2 = poder2;
33          this.poder3 = poder3;
34          this.fase1 = fase1;
35          this.fase2 = fase2;
36          this.fase3 = fase3;
37          this.fase4 = fase4;
38      }
39
40      public int getidPersonaje() {
41          return idPersonaje;
42      }
```

Fig 9. Implementación de la herencia

XI. Conclusiones

Para el desarrollo de software es bastante practico el modelo que propone la programación orientada a objetos porque aseguramos que el código sea reutilizable y organizado, además de garantizar el fácil entendimiento para futuras intervenciones o actualizaciones. Con la programación orientada a objetos aprendemos a manipular de manera autónoma los elementos del programa, a comparación de la programación lineal, en la que muchos elementos dependen de otros, lo cual dificulta el entendimiento y la estructuración del programa y el manejo de los elementos.

Como Java es un lenguaje de programación orientado a objetos, se logró implementar bastante bien para el desarrollo del programa Dragon Ball fan App. Aunque la curva de aprendizaje de Java es bastante lenta si la comparamos con otros lenguajes de programación, el conocimiento adquirido tras el desarrollo de este programa es mucho y muy satisfactorio, ya que Java es un lenguaje de programación muy popular en el mundo y por lo tanto las oportunidades de trabajo aumentan al aprenderlo.

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