Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2025

PROYECTO 1

NextGen Systems

202111134 Mario René Mérida Taracena

Resumen

El programa simula experimentos con proteínas utilizando Programación Orientada a Objetos (POO) en Python. Emplea clases como Nodo, ListaEnlazada, Experimento y SistemaExperimentos para gestionar y ejecutar experimentos. La lista enlazada almacena los experimentos, mientras que el sistema permite cargar datos desde archivos XML, crear experimentos manualmente y ejecutar simulaciones para analizar reacciones entre proteínas. El programa determina el efecto de un medicamento basado en el porcentaje de células inertes resultantes, ofreciendo opciones de ejecución paso a paso o directa. La estructura modular, el manejo de excepciones y la interacción intuitiva con el usuario destacan su robustez y flexibilidad, demostrando cómo la POO y las estructuras de datos personalizadas pueden aplicarse en simulaciones científicas.

Palabras clave

Encapsulación: Principio de la POO que oculta los detalles internos de una clase y expone solo lo necesario, protegiendo los datos.
Herencia: Mecanismo de la POO que permite crear nuevas clases basadas en clases existentes, reutilizando y extendiendo su funcionalidad.
Polimorfismo: Capacidad de objetos de diferentes clases para responder al mismo mensaje o método de manera específica a su tipo.
Abstracción: Proceso de simplificar sistemas complejos identificando patrones esenciales y omitiendo detalles irrelevantes.

Abstract

The program simulates protein experiments using Object-Oriented Programming (OOP) in Python. It utilizes classes such as Nodo, ListaEnlazada, Experimento, and SistemaExperimentos to manage and execute experiments. The linked list stores the experiments, while the system allows loading data from XML files, creating experiments manually, and running simulations to analyze protein interactions. The program determines the effect of a medication based on the percentage of inert cells resulting from the reactions, offering step-by-step or direct execution options. The modular structure, exception handling, and intuitive user interaction highlight its robustness and flexibility, demonstrating how OOP and custom data structures can be applied in scientific simulations.

Keywords

Encapsulation: Principle of OOP that hides the internal details of a class and exposes only what is necessary, protecting data.
Inheritance: Mechanism in OOP that allows creating new classes based on existing ones, reusing and extending their functionality.
Polymorphism: Ability of objects from different classes to respond to the same message or method in a way specific to their type.
Abstraction: Process of simplifying complex systems by identifying essential patterns and omitting irrelevant details.

Introducción

El programa desarrollado es una simulación de experimentos con proteínas, diseñado para analizar la interacción entre diferentes tipos de proteínas en una rejilla. Este sistema permite cargar experimentos desde un archivo XML, crear nuevos experimentos manualmente, y ejecutar simulaciones para determinar el efecto de un medicamento en función de las reacciones entre las proteínas. El programa está implementado en Python y utiliza conceptos avanzados de Programación Orientada a Objetos (POO), estructuras de datos personalizadas, y manipulación de archivos XML.

Desarrollo del Tema

Programación Orientada a Objetos (POO)

El programa está estructurado utilizando los principios de la POO, lo que permite una organización clara y modular del código. La Programación Orientada a Objetos (POO) es un paradigma de programación basado en el uso de objetos, los cuales encapsulan datos y comportamientos. Se centra en la modelización del mundo real mediante estructuras que interactúan entre sí.

Encapsulación

Permite ocultar los detalles internos de un objeto y solo exponer lo necesario a través de métodos públicos. Se logra mediante modificadores de acceso como private, protected y public.

Herencia

Facilita la reutilización de código al permitir que una clase derive (herede) características de otra. Se representa con la relación "es un" (Ejemplo: un Coche es un Vehículo).

Polimorfismo

Permite que un mismo método tenga diferentes comportamientos según el objeto que lo utilice. Puede implementarse mediante sobrecarga (mismo nombre, diferentes parámetros) o sobreescritura (redefinir métodos de una clase padre en la clase hija).

Abstracción

Consiste en definir solo las características esenciales de un objeto, omitiendo detalles innecesarios. Se implementa mediante clases abstractas o interfaces.

Elementos Clave en POO

- Clases: Son moldes o plantillas que definen atributos y métodos.
- Objetos: Instancias de una clase que almacenan valores específicos.
- Métodos: Funciones dentro de una clase que definen el comportamiento del objeto.
- Atributos: Variables dentro de una clase que almacenan datos.

Ventajas de la POO

- Facilita la reutilización de código.
- Mejora la organización y mantenimiento del software.
- Permite modularidad y escalabilidad.
- Hace que el código sea más legible y estructurado.

A continuación, se describen los principales elementos de POO utilizados:

Clases y Objetos:

- Clase Nodo: Representa un nodo en una lista enlazada. Cada nodo contiene un valor y un puntero al siguiente nodo.

- Clase ListaEnlazada: Implementa una lista enlazada simple, permitiendo agregar y mostrar elementos.
- Clase Matriz: Representa una rejilla de proteínas utilizando una estructura personalizada basada en listas enlazadas. Permite almacenar y manipular una matriz de proteínas de manera eficiente.
- Clase Experimento: Representa un experimento con un nombre, una rejilla de proteínas (instancia de Matriz), y una lista enlazada de parejas de proteínas que reaccionan entre sí.
- Clase SistemaExperimentos: Es la clase principal que gestiona el catálogo de experimentos y proporciona métodos para cargar, crear, modificar y ejecutar experimentos.

Encapsulación:

Cada clase encapsula sus atributos y métodos, protegiendo los datos internos y exponiendo solo lo necesario. Por ejemplo, la clase ListaEnlazada encapsula la lógica de agregar y mostrar nodos, mientras que la clase Experimento encapsula los detalles de un experimento, incluyendo su rejilla de proteínas (manejada por la clase Matriz) y las parejas de proteínas (almacenadas en una lista enlazada).

Métodos y Atributos:

Los métodos de las clases permiten realizar operaciones específicas, como agregar un experimento al catálogo (agregar en ListaEnlazada), cargar un catálogo desde un archivo XML (cargar_catalogo en SistemaExperimentos), o ejecutar un experimento (ejecutar_experimento en SistemaExperimentos).

Los atributos de las clases almacenan los datos necesarios, como el nombre del experimento, la rejilla de proteínas (manejada por la clase Matriz), y las parejas de proteínas (almacenadas en una lista enlazada).

Herramientas Utilizadas

- ***Graphviz:*** Una herramienta de código abierto para la visualización de grafos. Se utiliza para crear diagramas a partir de descripciones textuales.
- ***Librería graphviz en Python:*** Una interfaz para Python que permite generar gráficos utilizando Graphviz de manera programática.

Relaciones entre Clases:

La clase SistemaExperimentos utiliza una instancia de ListaEnlazada para almacenar y gestionar los experimentos. Cada experimento es una instancia de la clase Experimento, que a su vez contiene una rejilla de proteínas (instancia de Matriz) y una lista enlazada de parejas de proteínas.

Estructuras de Datos Personalizadas

El programa utiliza estructuras de datos personalizadas para almacenar y manipular los datos de los experimentos. Estas estructuras son adecuadas para este caso, ya que permiten agregar y recorrer los experimentos de manera eficiente.

- Lista Enlazada: Es una estructura de datos dinámica que consiste en una secuencia de nodos, donde cada nodo contiene un valor y un puntero al siguiente nodo. En este programa, la lista enlazada se utiliza para almacenar los experimentos y las parejas de proteínas, permitiendo agregar nuevos elementos y recorrerlos para su ejecución o modificación.
- Matriz: Es una estructura personalizada que representa una rejilla de proteínas. Está implementada utilizando listas enlazadas para almacenar filas y columnas de proteínas. La clase Matriz proporciona métodos para acceder y modificar los valores de la rejilla.

Manipulación de Archivos XML

El programa permite cargar experimentos desde un archivo XML, lo que facilita la gestión de datos y la reutilización de experimentos. Para ello, se utiliza el módulo xml.etree.ElementTree de Python, que permite parsear y manipular archivos XML de manera sencilla.

- Carga de XML: El método cargar_catalogo en la clase SistemaExperimentos parsea el archivo XML y extrae los datos de cada experimento, creando instancias de la clase Experimento y agregándolas a la lista enlazada. La rejilla de proteínas se almacena en una instancia de la clase Matriz, y las parejas de proteínas se almacenan en una lista enlazada.

Ejecución de Experimentos

El programa permite ejecutar los experimentos de dos maneras: paso a paso o directamente para obtener el resultado final. Durante la ejecución, se simula la interacción entre las proteínas en la rejilla (manejada por la clase Matriz), y se determina el efecto del medicamento en función del porcentaje de células inertes resultantes.

- Paso a Paso: Muestra el estado de la rejilla en cada paso de la simulación, permitiendo observar cómo las proteínas reaccionan y se convierten en inertes.
- Directamente: Ejecuta toda la simulación de una vez y muestra el resultado final, incluyendo el porcentaje de células inertes y el efecto del medicamento.

Información sobre la Programación del Proyecto

El proyecto está desarrollado en Python y utiliza varios conceptos y técnicas de programación para lograr su funcionalidad. A continuación, se detallan los aspectos más relevantes de la programación en sí:

1. *Estructura del Proyecto*

El programa está organizado en clases y métodos que siguen los principios de la Programación Orientada a Objetos (POO). Esto permite una estructura modular y facilita la extensión y mantenimiento del código.

Clases Principales:

- Nodo: Representa un nodo en una lista enlazada. Cada nodo contiene un valor y un puntero al siguiente nodo.
- ListaEnlazada: Implementa una lista enlazada para almacenar experimentos y parejas de proteínas. Permite agregar y recorrer elementos de manera eficiente.
- Matriz: Representa una rejilla de proteínas utilizando una estructura personalizada basada en listas enlazadas. Permite almacenar y manipular una matriz de proteínas de manera eficiente.
- Experimento: Almacena los datos de un experimento, incluyendo su nombre, una rejilla de proteínas (instancia de Matriz) y una lista enlazada de parejas de proteínas.
- SistemaExperimentos: Gestiona el catálogo de experimentos y proporciona la lógica para cargar, crear, modificar y ejecutar experimentos.

2. *Manejo de Datos*

El programa utiliza estructuras de datos personalizadas para almacenar y manipular los datos de los experimentos:

- Matriz: Para almacenar la rejilla de proteínas. Esta estructura está implementada utilizando

listas enlazadas para representar filas y columnas.
- Listas Enlazadas: Para gestionar el catálogo de experimentos y las parejas de proteínas. Estas estructuras permiten agregar y recorrer elementos de manera eficiente.

Además, se utiliza un archivo XML para cargar los experimentos, lo que permite una gestión externa de los datos.

3. Funcionalidades Clave

Carga de Experimentos desde XML:

El método cargar_catalogo utiliza el módulo xml.etree.ElementTree para parsear el archivo XML y extraer los datos de los experimentos. Los datos se almacenan en instancias de la clase Experimento, donde la rejilla de proteínas se maneja con la clase Matriz y las parejas de proteínas se almacenan en una lista enlazada.

Creación Manual de Experimentos:

El método crear_experimento_manual permite al usuario ingresar los datos del experimento (nombre, rejilla y parejas de proteínas) y lo agrega al catálogo. La rejilla de proteínas se almacena en una instancia de la clase Matriz, y las parejas de proteínas se almacenan en una lista enlazada.

Ejecución de Experimentos:

El programa simula las reacciones entre proteínas en la rejilla (manejada por la clase Matriz). Se verifica si dos proteínas adyacentes (horizontal o verticalmente) son una pareja reactiva. Las proteínas que reaccionan se convierten en "INERTE". El resultado del experimento se determina en función del porcentaje de células inertes:

- 30% - 60%: Medicamento exitoso.
- Menos del 30%: Medicamento no efectivo.
- Más del 60%: Medicamento fatal.

Modificación de Experimentos:

El método modificar_experimento permite cambiar el nombre, la rejilla (manejada por la clase Matriz) o las parejas de proteínas (almacenadas en una lista enlazada) de un experimento existente.

Resultados con Graphviz

La implementación de Graphviz en el sistema tiene como objetivo proporcionar una representación gráfica del estado inicial y final de la matriz de proteínas en un experimento. Esto permite visualizar de manera clara y efectiva cómo evoluciona la matriz durante la ejecución del experimento, facilitando la comprensión de los resultados y la toma de decisiones.

La herramienta Graphviz se utiliza para generar diagramas de grafos, lo que resulta ideal para representar la estructura de la matriz y las interacciones entre las proteínas.

Este método se encarga de generar dos gráficos:

- *Estado Inicial:* Representa la matriz de proteínas antes de ejecutar el experimento.
- *Estado Final:* Representa la matriz de proteínas después de ejecutar el experimento.

4. Interacción con el Usuario

El programa utiliza una interfaz de línea de comandos (CLI) para interactuar con el usuario. El menú principal ofrece las siguientes opciones:

1. Inicializar el sistema.
2. Cargar un catálogo de experimentos desde un archivo XML.
3. Desarrollar un experimento (crear manualmente o cargar desde el catálogo).
4. Mostrar los datos del estudiante.
5. Salir del programa.

5. Manejo de Excepciones

El programa incluye manejo de excepciones para gestionar errores durante la ejecución, como:

- Errores al cargar el archivo XML.
- Entradas inválidas del usuario (por ejemplo, valores no numéricos donde se esperan números).

6. Eficiencia y Optimización

Lista Enlazada y Matriz: Aunque no son las estructuras más eficientes para todos los casos, son adecuadas para este proyecto debido a su simplicidad y facilidad de implementación. La clase Matriz maneja la rejilla de proteínas utilizando listas enlazadas, lo que permite una manipulación eficiente de los datos.

Algoritmo de Simulación: El algoritmo que simula las reacciones entre proteínas tiene una complejidad de $O(n^2)$ en el peor caso, donde n es el número de células en la rejilla. Esto es aceptable para rejillas pequeñas o medianas.

7. Extensibilidad

El programa está diseñado para ser extensible. Algunas posibles mejoras incluyen:

- Visualización Gráfica: Usar bibliotecas como matplotlib o tkinter para mostrar la rejilla de proteínas de manera gráfica.
- Optimización del Algoritmo: Implementar técnicas más eficientes para simular las reacciones, especialmente para rejillas grandes.
- Persistencia de Datos: Guardar los experimentos en una base de datos o archivos JSON/XML para su uso posterior.

8. Herramientas y Bibliotecas Utilizadas

Módulos de Python:

- xml.etree.ElementTree: Para parsear y manipular archivos XML.
- Graphviz: Para generar gráficos de los estados inicial y final de los experimentos.
- Funciones integradas de Python:
    - input: Para solicitar datos al usuario.
    - print: Para mostrar información en la consola.
    - int: Para convertir cadenas a números enteros.
    - range: Para iterar sobre un rango de números.
    - Exception: Para manejar errores.

Estructuras de Datos Personalizadas:

- Listas enlazadas y la clase Matriz implementadas desde cero.

Ejemplo de Flujo de Ejecución

1. El usuario carga un catálogo de experimentos desde un archivo XML.
2. Selecciona un experimento para ejecutar.
3. El programa simula las reacciones entre proteínas y muestra el resultado.
4. El usuario puede modificar el experimento o ejecutar otro.

Detalles Técnicos Adicionales

- Manejo de Excepciones: El programa incluye manejo de excepciones para gestionar errores durante la carga del archivo XML, lo que mejora la robustez del sistema.
- Interacción con el Usuario: La interfaz de línea de comandos es intuitiva y guía al usuario a través de las diferentes opciones del menú.
- Documentación: El código está documentado con comentarios que explican la funcionalidad de cada método y clase, facilitando su mantenimiento y extensión.

Conclusiones

Este programa es un ejemplo claro de cómo la Programación Orientada a Objetos (POO) y las estructuras de datos personalizadas permiten desarrollar sistemas complejos y modulares. Utiliza clases como *Nodo*, *ListaEnlazada*, *Experimento* y *SistemaExperimentos* para modelar entidades y procesos, demostrando la eficacia de la encapsulación y la reutilización de código. La manipulación de archivos XML facilita la carga y gestión de datos, mientras que la simulación de interacciones entre proteínas resalta la versatilidad de Python para aplicaciones científicas. El programa es altamente extensible, permitiendo mejoras como visualización gráfica, optimización de algoritmos o integración de bases de datos. Su diseño modular y enfoque en la interacción con el usuario lo convierten en una herramienta útil y adaptable, ideal tanto para entornos educativos como aplicaciones prácticas en investigación científica. Por último, la implementación de Graphviz en el sistema proporciona una herramienta poderosa para la visualización de resultados, mejorando la comprensión de los experimentos y facilitando el análisis de datos. Esta funcionalidad es un ejemplo de cómo las herramientas de visualización pueden integrarse en sistemas científicos para mejorar la usabilidad y la interpretación de los resultados.

Apéndice

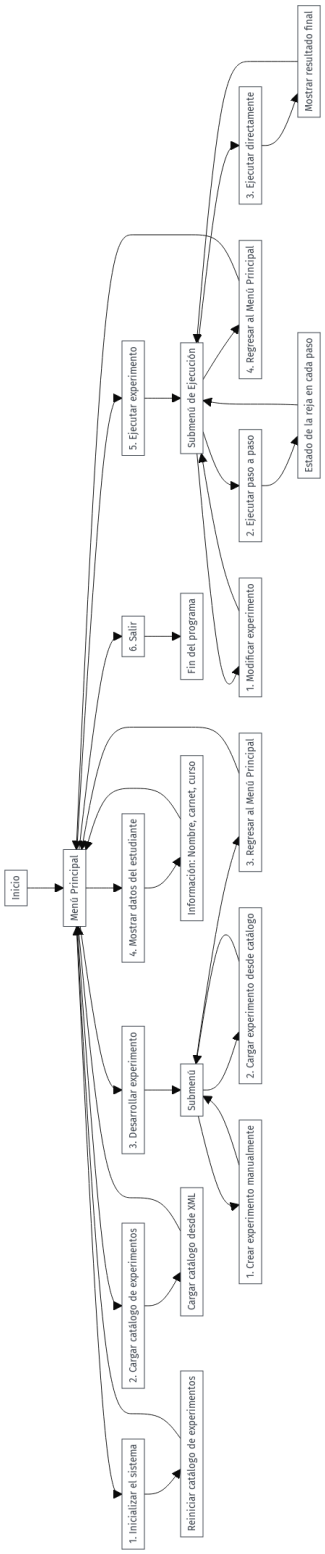

Figura 1. Diagrama de Actividades

Fuente: Elaboración propia (2025)

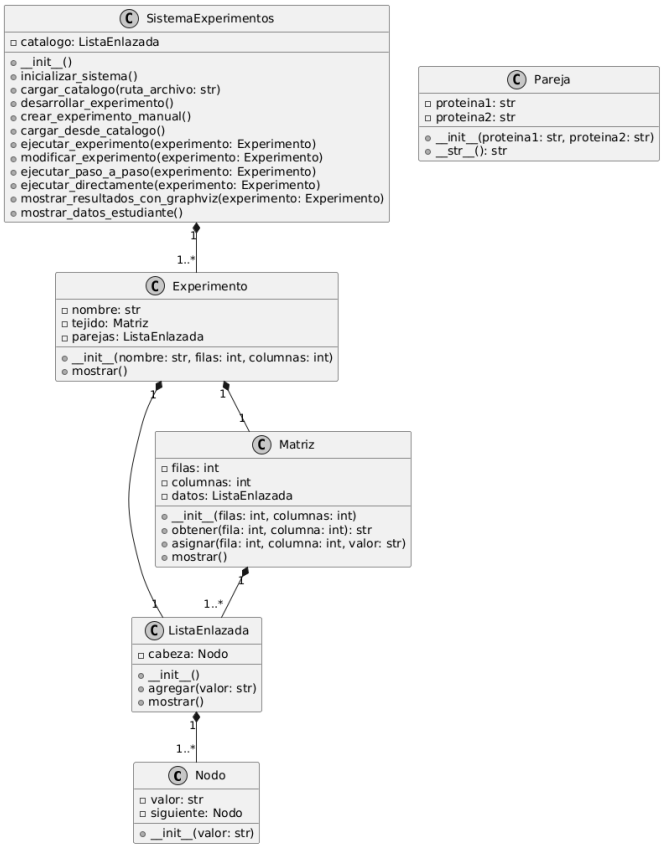

Figura 2. Diagrama de Clases

Fuente: Elaboración propia (2025)

Referencias Bibliográficas

Deitel, P., & Deitel, H. (2020). *Java: How to Program (Early Objects) (11th ed.)*. Pearson.

Eckel, B. (2006). *Thinking in Java (4th ed.)*. Prentice Hall.

Bloch, J. (2018). *Effective Java (3rd ed.)*. Addison-Wesley.

Sierra, K., & Bates, B. (2019). *Head First Java (3rd ed.)*. O'Reilly Media.

Horstmann, C. S. (2019). *Core Java Volume I–Fundamentals (11th ed.)*. Prentice Hall.