Trabajo primer corte arquitectura de software

Correa Adriana,

Linares Jhojan

Pérez Dilanys,

Ingeniería de Sistemas,

Facultad de Ciencias Naturales e Ingeniería

Universidad de Bogotá Jorge Tadeo Lozano

Arquitectura de software

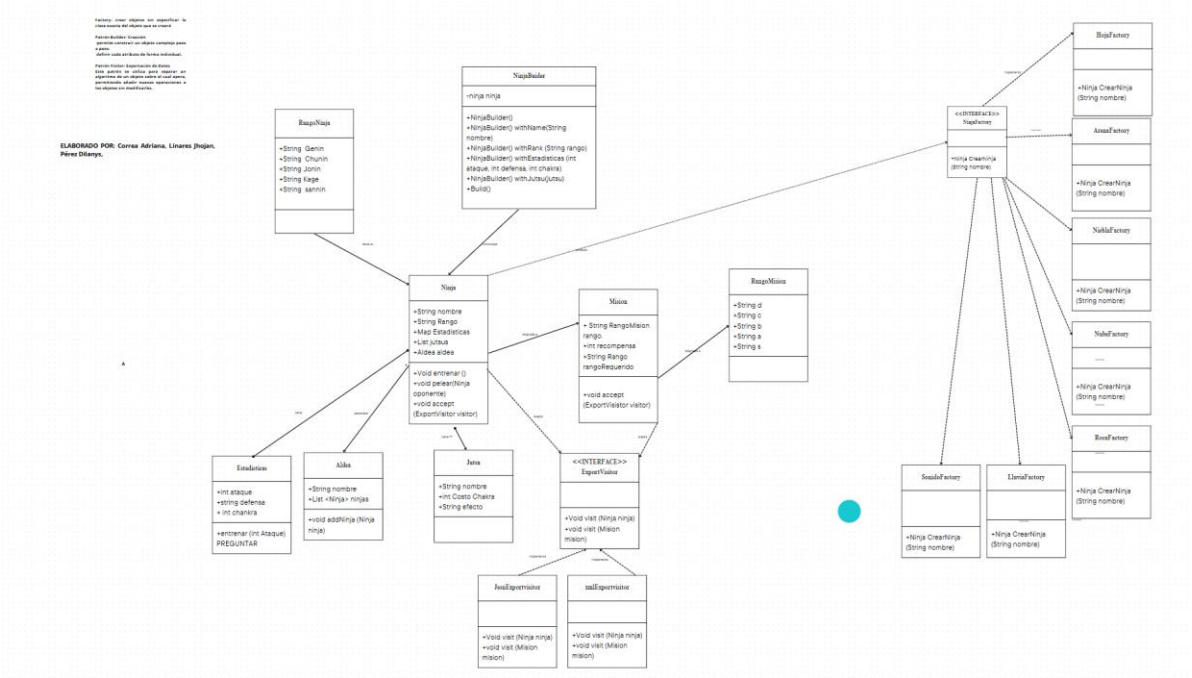
Felipe Esteban Hernández Baquero

2025-2S

1. **Diagrama UML:**

   ENLACE:

   https://www.canva.com/design/DAGwcgp8U18/jJJutjEDkaU0StdMIcILZA/edit?utm_content=DAGwcgp8U18&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton



2. **Resumen del código:**

   El sistema de gestión ninja fue desarrollado en Python, debido a la facilidad de la curva de aprendizaje y la capacidad del lenguaje para resolver el problema planteado. El proyecto consiste en un sistema interactivo que permite gestionar elementos del universo Naruto.

En este contexto, se consideran aldeas, ninjas y misiones como las entidades principales. Para garantizar eficiencia, modularidad y escalabilidad, se implementaron varios patrones de diseño.

3. **Estructura del código:**

Enumeraciones y clases de dominio:

Esta sección constituye el núcleo del sistema. Incluye las clases y enumeraciones que representan objetos reales del universo del anime.

RangoNinja y RangoMision (Enum): definen un conjunto fijo de valores (Genin, Chunin; D, C, B, A, S).

Estadisticas: almacena los atributos numéricos de un ninja (ataque, defensa, chakra).

Jutsu: representa una técnica con sus propios atributos (nombre, costo, efecto).

Aldea: contenedor que agrupa a los ninjas que le pertenecen.

Ninja: clase principal.

Mision: clase que define los detalles de una misión, incluyendo dificultad y recompensa.

Las clases de los elementos principales del sistema son: ninja, aldea, misión

Patrones de creación: buider (es un patrón de diseño creacional que permite construir objetos complejos paso a paso); factory (patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.)

Patrones de comportamiento. Aquí se utiliza el patrón de diseño visitor (patrón de diseño de comportamiento que te permite separar algoritmos de los objetos sobre los que operan.)

Patrón main: es la parte del código que permite la interacción del usuario con el sistema.

Estructura del código:

Enumeraciones y Clases de Dominio (enum y clases de dominio)

Esta sección es la parte central del sistema. Aquí se encuentran las clases y enumeraciones que representan a los objetos reales del universo dentro del anime.

RangoNinja y RangoMision (Enum): Definen un conjunto fijo de valores (por ejemplo, Genin, Chunin; D, C, B)

Estadisticas: Almacena los atributos numéricos de un ninja (ataque, defensa, chakra).

Jutsu: Representa una técnica con sus propios atributos (nombre, costo, efecto).

Aldea: Un contenedor que agrupa a los ninjas que le pertenecen.

Ninja: La clase principal.

Mision: Un objeto que define los detalles de una misión, incluyendo su dificultad y recompensa.

```
# enum y clases de dominio, lista a la izquierda
#usa libreria unum, porque son rcurrentes

class RangoNinja(Enum):
    GENIN = "Genin"
    CHUNIN = "Chunin"
    JONIN = "Jonin"
    KAGE = "Kage"
    SANNIN = "Sannin"


class RangoMision(Enum):
    D = "D"
    C = "C"
    B = "B"
    A = "A"
    S = "S"


class Estadisticas:
    def __init__(self, ataque: int, defensa: int, chakra: int):
        self.ataque = ataque
        self.defensa = defensa
        self.chakra = chakra

    def entrenar(self, inc_ataque=0, inc_defensa=0, inc_chakra=0):
        self.ataque += inc_ataque
        self.defensa += inc_defensa
        self.chakra += inc_chakra
```

Patrón Visitor:

Este patrón permite exportar datos a diferentes formatos sin necesidad de modificar las clases principales (Ninja, Mision).

Patrones Builder y Factory: Ambos se utilizan para simplificar la creación de objetos complejos.

NinjaBuilder (Builder): posibilita la construcción de un ninja paso a paso.

NinjaFactory (Factory): proporciona una interfaz para crear objetos, permitiendo que cada aldea defina sus propias fábricas (HojaFactory, *ArenaFactory*, etc.), las cuales generan ninjas con estadísticas y jutsus predefinidos.

```python
class NinjaBuilder:
    def __init__(self):
        self.nombre = None
        self.rango = RangoNinja.GENIN
        self.estadisticas = Estadisticas(10, 10, 50)
        self.jutsus: list[Jutsu] = []

    def with_nombre(self, nombre: str):
        self.nombre = nombre
        return self

    def with_rango(self, rango: RangoNinja):
        self.rango = rango
        return self

    def with_estadisticas(self, est: Estadisticas):
        self.estadisticas = est
        return self

    def with_jutsu(self, j: Jutsu):
        self.jutsus.append(j)
        return self
```

Lógica de Exportación:

Incluye las funciones que invocan a los visitantes para realizar la exportación. Las funciones exportar_json, exportar_xml, exportar_texto y exportar_excel toman las listas de ninjas y misiones como entrada.

```
def exportar_json(ninjas: list[Ninja], misiones: list[Mision], filename: str | None = None) -> str:
    visitor = JsonExportVisitor()
    data = {
        "ninjas": [n.accept(visitor) for n in ninjas],
        "misiones": [m.accept(visitor) for m in misiones]
    }
    texto = json.dumps(data, indent=2, ensure_ascii=False)

    if filename:
        # Añadir extensión .json si no la tiene
        if not filename.endswith('.json'):
            filename += '.json'

        # Obtener la ruta absoluta
        full_path = os.path.abspath(filename)

        with open(full_path, "w", encoding="utf-8") as f:
            f.write(texto)
        return f"Datos JSON exportados a: {full_path}"
    return texto
```

El Bucle Principal (main)

Es el corazón del sistema. Muestra un menú de opciones y, dependiendo de la selección del usuario, invoca la lógica correspondiente para crear, entrenar, pelear o exportar los datos.

En conclusión, el código está estructurado de forma organizada y emplea conceptos avanzados de programación, lo cual lo hace robusto y fácil de ampliar en el futuro.

```
print("\n=== MENÚ PRINCIPAL ===")
print("1. crear aldea")
print("2. crear ninja (builder o factory) y asignar a aldea")
print("3. crear misión")
print("4. entrenar ninja")
print("5. pelear entre dos ninjas")
print("6. exportar datos (Texto / JSON / XML / Excel)")
print("7. listar aldeas y ninjas")
print("0. Salir")
```

.

	A	B	C	D	E	F	G	H
1	Nombre	Rango	Aldea	Ataque	Defensa	Chakra	Jutsus	
2	Zoltham	Kage	ZoltamAlc	1000	1000	2000		
3								
4								
5								
6								
7								
8								

LIBRERIAS UTILIZADAS

Json: se utiliza para trabajar con datos en formato JSON (JavaScript Object Notation), se utilizo en el código para convertir los datos ingresados en este formato

Os: ayuda a tener la ruta completa de donde se exporta el archivo, asegura que el archivo se guarde correctamente sin importar dónde se esté ejecutando el programa.

Pandas: en este contexto sirve para organizar y descargar los datos en formato excel.

openpyxl: Es una librería para leer/escribir archivos de Excel. pandas la utiliza como su "motor" para escribir los DataFrames en el archivo de Excel (engine="openpyxl").

Abc: Esta librería se utiliza para definir clases abstractas base (ABC). En el sistema, se usa para crear la clase ExportVisitor y NinjaFactory, lo que obliga a las subclases a implementar ciertos métodos como visit_ninja, crear_ninja.

enum: Permite crear enumeraciones, que son conjuntos de nombres simbólicos (miembros) vinculados a valores.

BIBLIOGRAFIA

abc — Abstract Base Classes. (s/f). Python documentation. Recuperado el 26 de agosto de 2025, de https://docs.python.org/es/3.13/library/abc.html

enum — Soporte para enumeraciones — documentación de Python - 3.9.23. (s/f). Python.org. Recuperado el 26 de agosto de 2025, de https://docs.python.org/es/3.9/library/enum.html

enum — Support for enumerations. (s/f). Python Documentation. Recuperado el 26 de agosto de 2025, de https://docs.python.org/3/library/enum.html

Factory Method. (s/f). Refactoring.guru. Recuperado el 26 de agosto de 2025, de https://refactoring.guru/es/design-patterns/factory-method

JSON. (s/f). Json.org. Recuperado el 26 de agosto de 2025, de https://www.json.org/json-es.html

os — Miscellaneous operating system interfaces. (s/f). Python Documentation. Recuperado el 26 de agosto de 2025, de https://docs.python.org/3/library/os.html

pandas. (s/f). Pydata.org. Recuperado el 26 de agosto de 2025, de https://pandas.pydata.org

Python design patterns. (s/f). Translate.Goog. Recuperado el 26 de agosto de 2025, de https://python--patterns-guide.translate.goog/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc

Visitor. (s/f). Refactoring.guru. Recuperado el 26 de agosto de 2025, de https://refactoring.guru/es/design-patterns/visitor