Ana María Ángel Hernández

Sebastián Romero

Alejandro Sarmiento

**Documento Trabajo Primer Corte**

**Patrones de diseño implementados  
1. Factory**

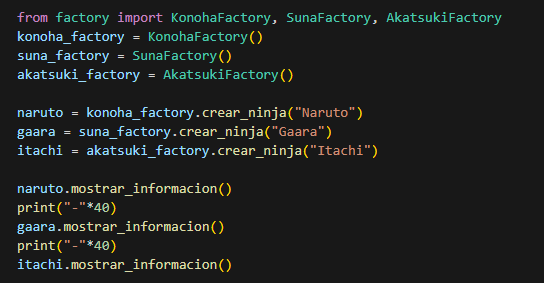
****

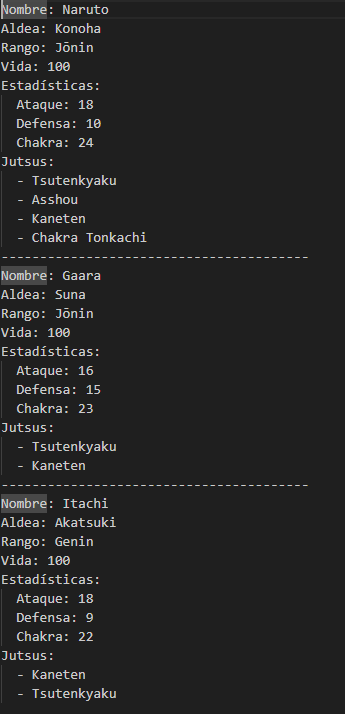
El sistema implementa el patrón Factory Method, cuyo objetivo es delegar la creación de objetos a fábricas concretas en lugar de instanciar los directamente en el código cliente.

* Las fábricas concretas utilizadas fueron:
  + KonohaFactory: crea ninjas pertenecientes a la aldea Konoha.
  + SunaFactory: crea ninjas de la aldea Suna.
  + AkatsukiFactory: crea ninjas de la aldea Akatsuki.

Cada fábrica asigna al ninja un rango aleatorio (Genin, Chūnin, Jōnin), dos jutsus aleatorios de la lista (LISTA\_JUTSUS) y estadísticas aleatorias con un rango establecido según la aldea.

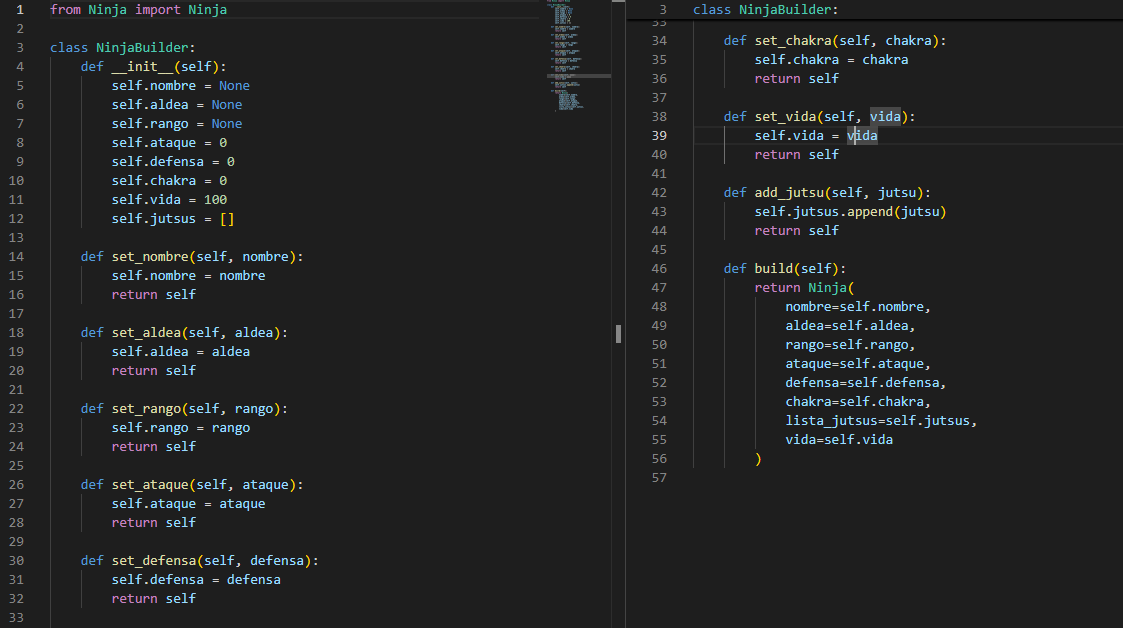
* Ejemplo de uso del Factory:



En el ejemplo se muestra como se utiliza el método factory para crear los siguientes ninjas (Naruto,Gaara,Itachi), luego muestra los datos en la consola con la función mostrar\_informacion.  
  


Finalmente se logra observar cómo muestra la información de todos los ninjas, nunca serán iguales, cada vez que se ejecute este main los valore cambiarán puesto que en el método factory se crean los ninjas con valores aleatorios

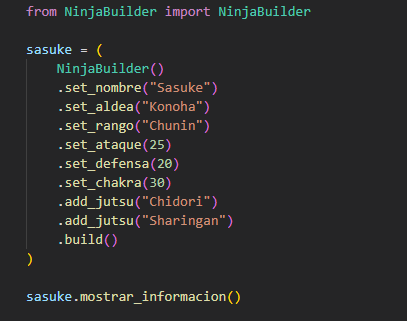
**Builder**

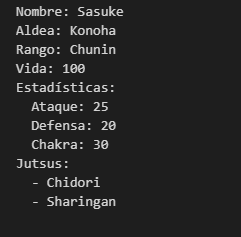


El patrón Builder permite construir ninjas paso a paso, definiendo sus atributos de manera controlada.

Esto evita la creación de objetos con demasiados parámetros en el constructor y ofrece flexibilidad al usuario.

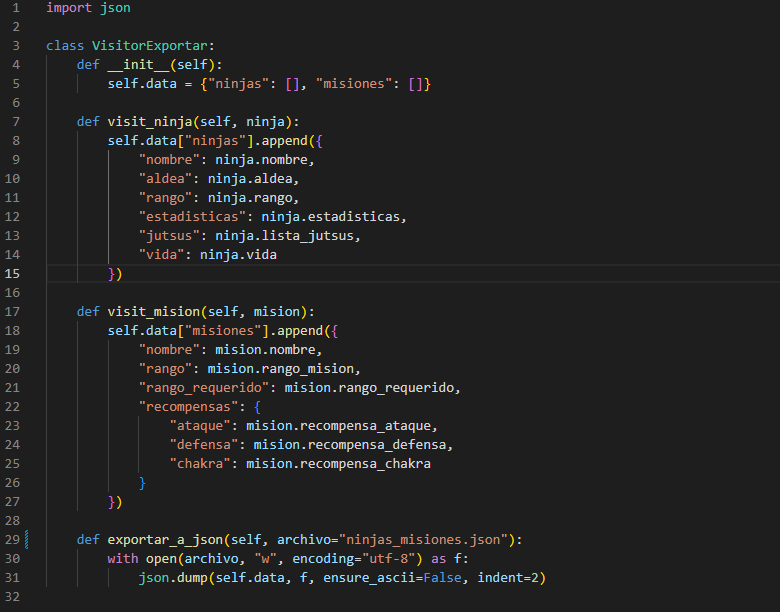
Ejemplo:



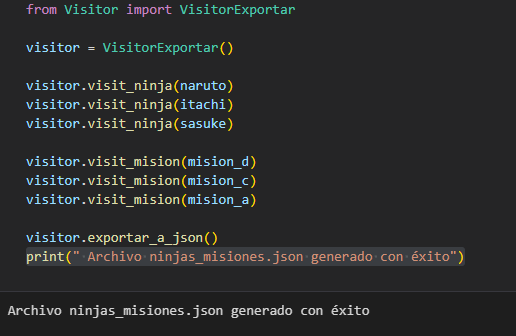
Resultado:  


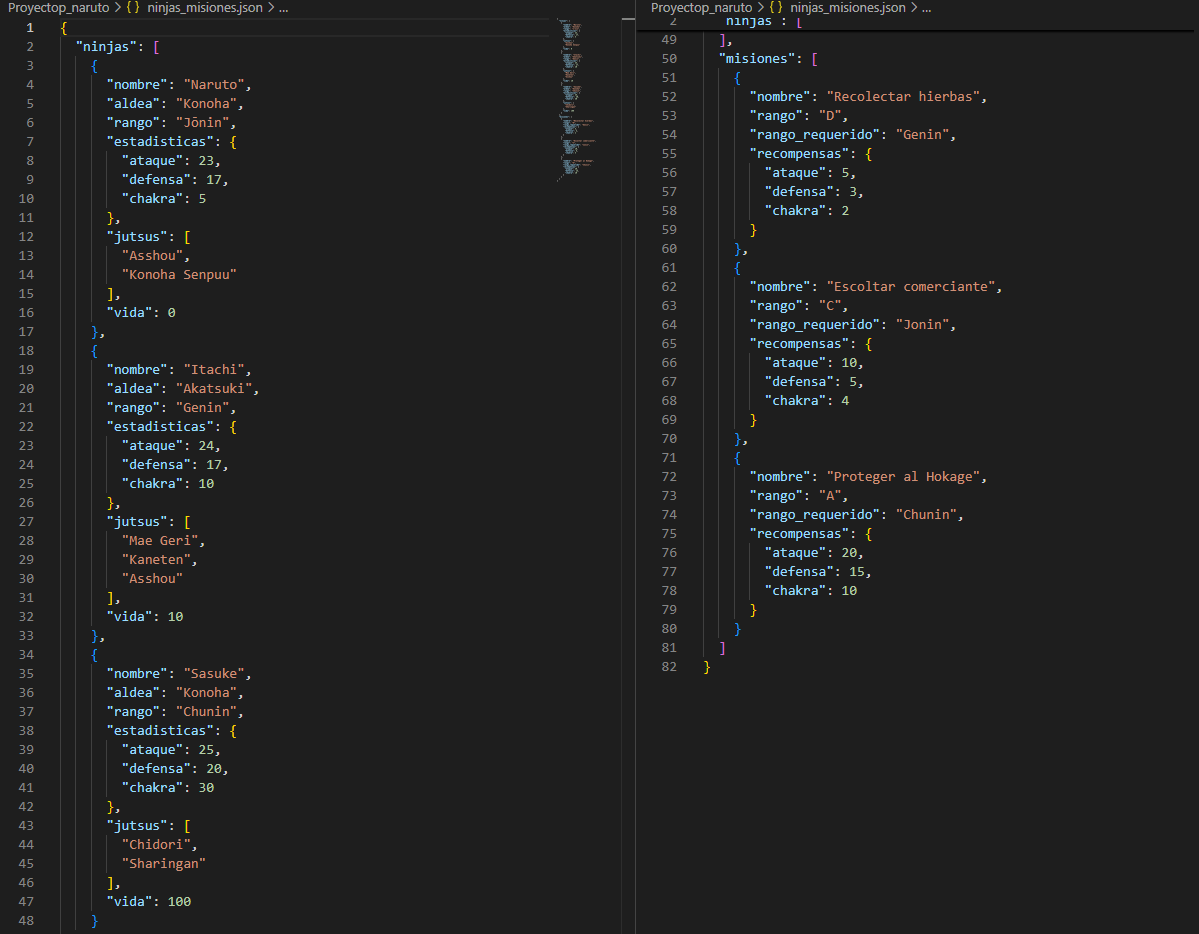
**Visitor**

El patrón Visitor se utilizó para implementar un sistema de exportación de datos.

La clase VisitorExportar recorre listas de ninjas y misiones, recopilando información en una estructura que puede exportarse a JSON.  


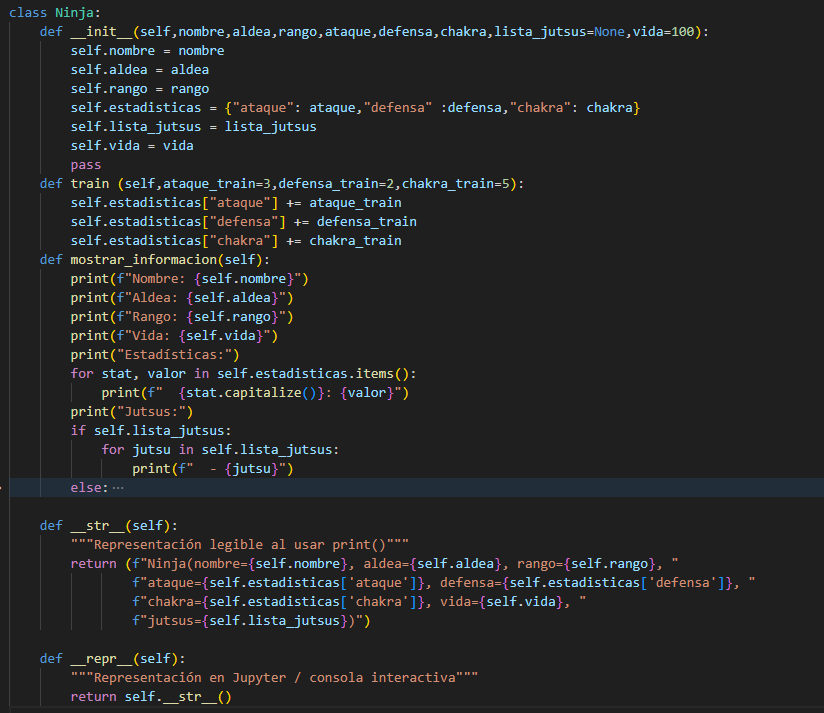
Ejemplo:



Esto permite generar informes de manera flexible sin modificar las clases Ninjas\_Mision.  


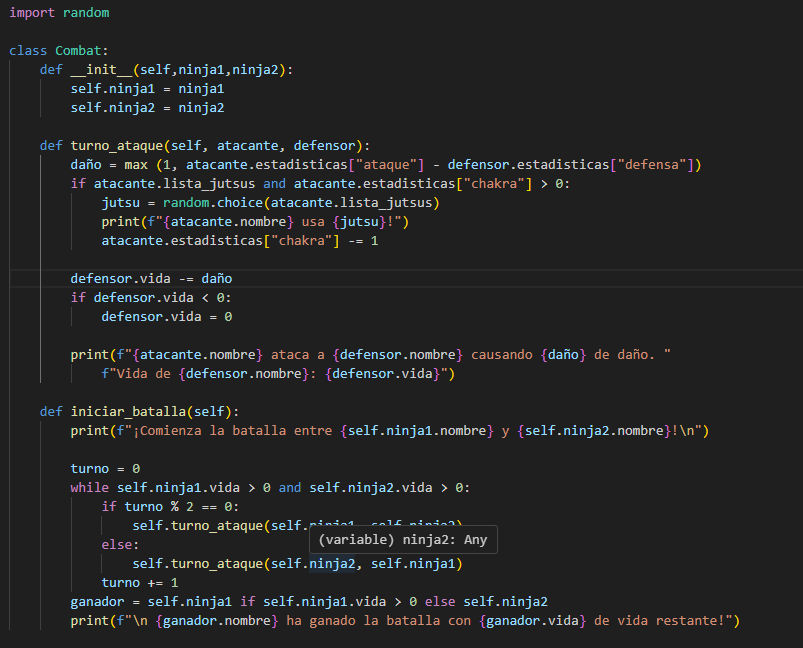
**Clases del sistema**

1. **Clase Ninja**

****

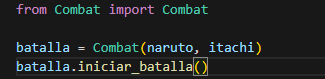
* + En la clase ninja se muestran las siguientes funciones (\_\_init**\_\_**,train, mostrar\_informacion,\_\_str\_\_ y \_\_repr\_\_).
  + La funcion init inicializa la clase ninja y espera los siguientes atributos:
    - Nombre
    - Aldea
    - Rango
    - Estadisticas = {ataque, defensa, chakra}(Recibe 3 números enteros los cuales son de ataque defensa y chakra que luego lo transforma en un diccionario llamado estadísticas)
    - Lista jutsus (Se espera que pueda recibir una lista)
    - Vida este valor se le asigna de forma predeterminada para poder simular combates
  + Las funciones como train y mostrar informacion son los metodos que tienen los ninjas por ejemplo:
    - train(): permite entrenar para mejorar estadísticas, el entrenamiento sube estadísticas de forma predeterminada.
    - mostrar\_informacion(): imprime los datos del ninja.

1. **Clase Combat**

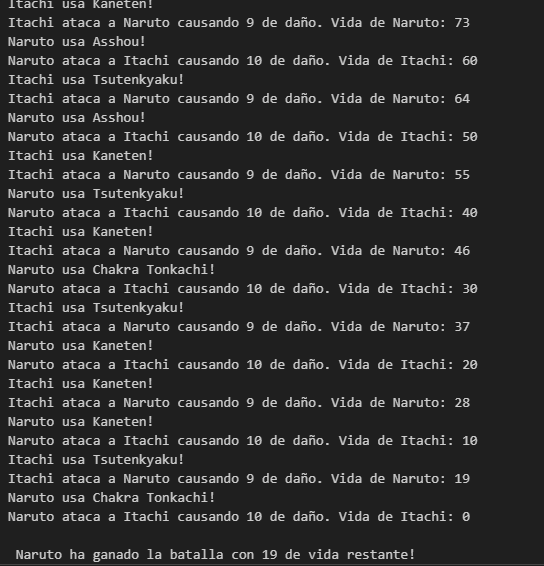
****

* + En esta clase se reciben 2 ninjas los cuales serán los que se utilicen en el combate
  + En la función init tenemos los siguientes atributos:
    - ninja1, ninja2
  + Las funciones como turno\_ataque e iniciar\_batalla son los métodos que tiene la clase combate por ejemplo:
    - turno\_ataque(atacante, defensor): ejecuta un ataque, considerando ataque, defensa, chakra y posibilidad de usar un jutsu.
    - iniciar\_batalla(): controla el flujo del combate hasta que un ninja queda sin vida.

Ejemplo de un combate entre dos ninjas

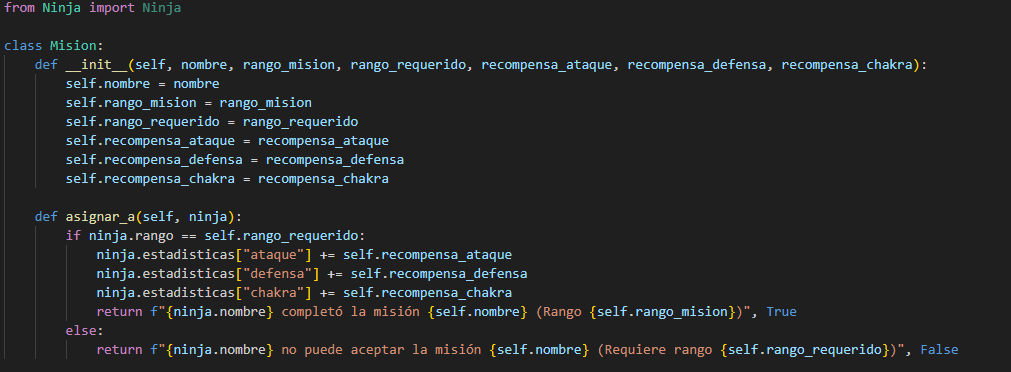


Para este ejemplo se usaron los ninjas creados anteriormente con los patrones de diseño factory



Finalmente la consola muestra turno por turno cuanta vida se van quitando mutuamente y cuando la vida de uno de los ninjas llega a 0 imprimer el que aun sigue con vida

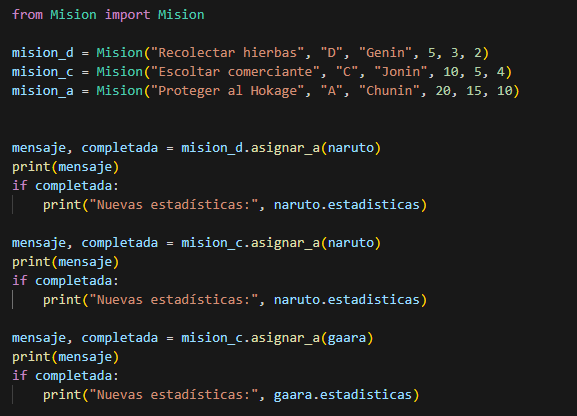
1. **Clase Mision**



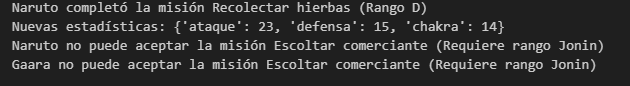
En esta clase se reciben toda la información de la misión

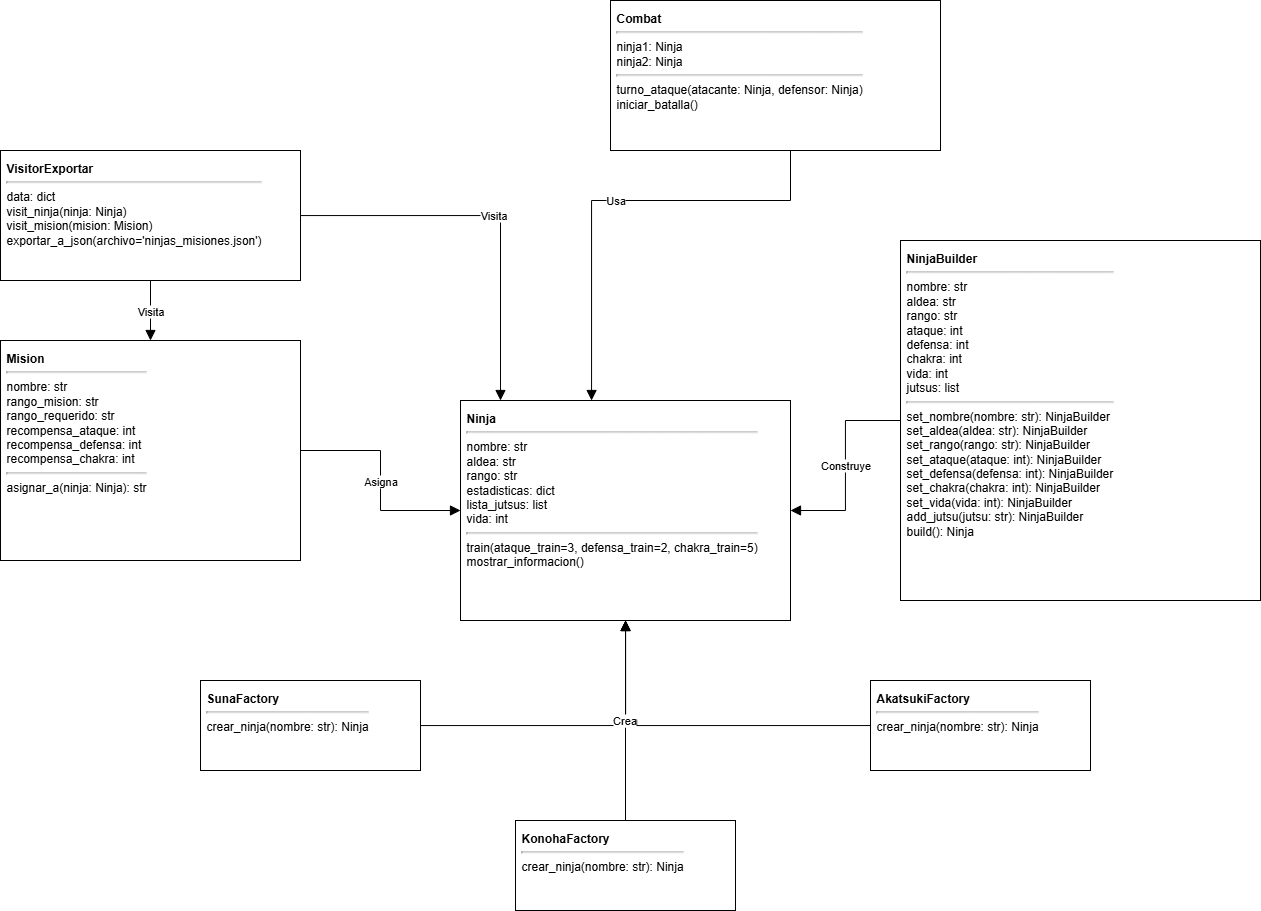
* En la funcion init espera recibir los siguientes atributos:
  + Nombre
  + Rango\_mision
  + Rango\_requerido
  + Recompensa\_ataque
  + Recompensa\_defensa
  + Recompensa\_chakra
* Esta clase contaria con un unico metodo llamado asignar\_as:
  + Lo que hace es que valida si un ninja puede realizar la misión y aplica recompensas.

Ejemplo de uso:



Se crean las misiones con la clase Misión ,como la clase mision retorna un booleano este se utiliza para saber si el rango del ninja es suficiente realizar la misión y de así serlo imprime las nuevas estadísticas del ninja com se muestra en la consola. en caso de no tener el rango suficiente retornara para que rango esta hecha la misión y no imprimirá las estadísticas.



**Diagrama UML**  
****