Patrones de diseño - Aldea Ninja

Integrantes:

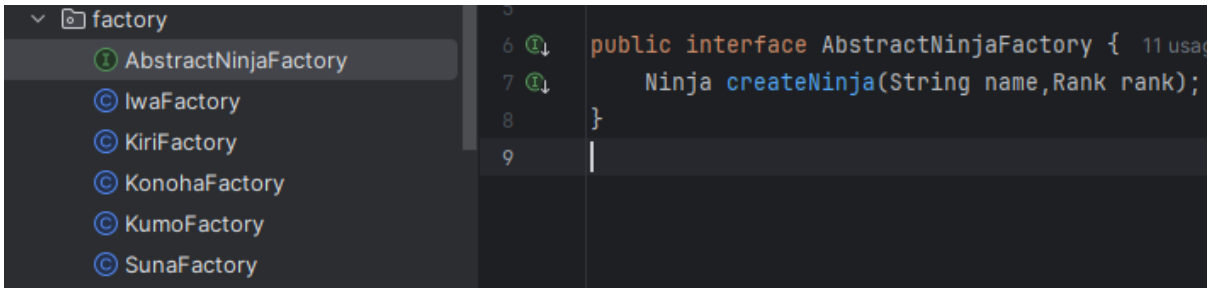
- José Miguel Becerra
- Juan Pablo Vargas Balli
- Laura Sofía Castañeda

¿Como se utilizó cada patrón?

Factory

Este patrón se implementó para crear ninjas por cada aldea específica, cada fábrica corresponde a una aldea, en total 5 representando las 5 aldeas del mundo de Naruto: Konoha, Suma, Kumo, Iwa y Kiri.

La clase base es ***'AbstractNinjaFactory'***, *es la interfaz común* que se encarga de definir el contrato, es decir, lo que debe cumplir cada subclase, en este caso cada fabrica debe poseer un método para crear ninjas. Este patrón permite añadir nuevas aldeas y personalizar características asociadas a cada aldea, sin necesidad de modificar la función *main*.



Ejecución:

1. Se crea la fábrica KonohaFactory, se asignan estadísticas y Jutsus.

```
public class KonohaFactory implements AbstractNinjaFactory{ 2 usages
    @Override 5 usages  PabloB324
    public Ninja createNinja(String name,Rank rank) {
        return new Ninjabuilder()
                .setName(name)
                .setRank(rank)
                .setVillage(Village.KONOHA)
                .setStats(new Stats( attack: 7, defense: 7, chakra: 12))
                .addJutsu(new Jutsu( name: "Rasengan",  chakraCost: 8))
                .addJutsu(new Jutsu( name: "Kage Bushin", chakraCost: 5))
                .build();
    }
}
```

2. Se inicializan las fábricas concretas con su respectiva interfaz, luego se utilizan para crear nuevos ninjas.

```java
public class Main {  PabloB324
    public static void main(String[] args) {  PabloB324
        // FACTORIES
        AbstractNinjaFactory konoha = new KonohaFactory();
        AbstractNinjaFactory suna   = new SunaFactory();
        AbstractNinjaFactory kiri   = new KiriFactory();
        AbstractNinjaFactory iwa    = new IwaFactory();
        AbstractNinjaFactory kumo   = new KumoFactory();

        // NINJAS
        Ninja naruto    = konoha.createNinja( name: "Naruto",    Rank.GENIN);
        Ninja gaara     = suna.createNinja( name: "Gaara",     Rank.CHUNIN);
        Ninja zabuza    = kiri.createNinja( name: "Zabuza",    Rank.JONIN);
        Ninja deidara   = iwa.createNinja( name: "Deidara",    Rank.JONIN);
        Ninja killerBee = kumo.createNinja( name: "Killer Bee", Rank.JONIN);
```

3. **Terminal-salida:** Los ninjas adoptan las características de sus aldeas y a partir de allí, se enfrentan a otros ninjas.

```
⚔️ Combate entre Naruto y Gaara
 - Poder de Naruto: 23
 - Poder de Gaara: 21
🏆 Ganador: Naruto
```

Builder

Se utilizó para crear ninjas de manera personalizada, asignando las características que se deseen. Cada ninja posee nombre, rango, aldea, estadísticas y Jutsus.

```java
public class NinjaBuilder {
    private String name;
    private Rank rank =Rank.GENIN;
    private Village village = Village.KONOHA;
    private Stats stats = new Stats( attack: 5, defense: 5, chakra: 10);
    private final List<Jutsu> jutsus = new ArrayList<>();
    public NinjaBuilder setName(String name){
        this.name = name;
        return this;
    }
    public NinjaBuilder setRank(Rank rank){
        this.rank = Objects.requireNonNull(rank);
        return this;
    }
    public NinjaBuilder setVillage(Village village){
        this.village = Objects.requireNonNull(village);
        return this;
    }
    public NinjaBuilder setStats(Stats stats){
        this.stats = Objects.requireNonNull(stats);
        return this;
    }
    public NinjaBuilder addJutsu(Jutsu jutsu){
        this.jutsus.add(Objects.requireNonNull(jutsu));
        return this;
```

Se añaden opciones para el atributo Jutsus, los cuales permiten tanto añadir listas o eliminarlas, además se verifica que cada ninja tenga como mínimo el atributo *'name'* para ser creado.

```
public NinjaBuilder addJutsus(Collection<Jutsu> jutsus){  no usages
    if(jutsus!=null) for (Jutsu j : jutsus) addJutsu(j);
    return this;
}
public NinjaBuilder clearJutsus(){  no usages
    this.jutsus.clear();
    return this;
}
public Ninja build(){  5 usages
    if(name == null || name.isBlank())
        throw new IllegalStateException("Es obligatorio asignar el nombre");
    return new Ninja(name, rank, village, stats, jutsus);
}
```

Este patrón permite una construcción clara, es flexible ya que pueden añadirse objetos sin necesariamente contar con todos los atributos, poseen una validación centralizada todas las reglas requeridas están en *'build',* a largo plazo es extensible en caso de que se quieran añadir nuevos atributos.

Ejecución:

1. Se añaden nuevos ninjas, para cada uno se asignan nombres y rangos, los únicos atributos que no están definidos por cada fabrica (Aldea). Aun así, si solo se añade el nombre no habría inconveniente, ya que es la información básica para crear un ninja.

```
// NINJAS
Ninja naruto    = konoha.createNinja( name: "Naruto",   Rank.GENIN);
Ninja gaara     = suna.createNinja( name: "Gaara",      Rank.CHUNIN);
Ninja zabuza    = kiri.createNinja( name: "Zabuza",     Rank.JONIN);
Ninja deidara   = iwa.createNinja( name: "Deidara",     Rank.JONIN);
Ninja killerBee = kumo.createNinja( name: "Killer Bee", Rank.JONIN);
```

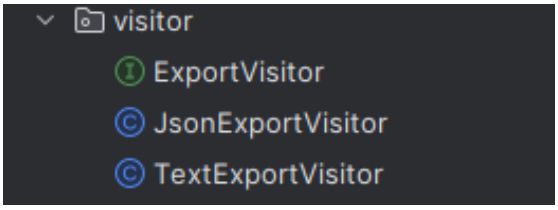
```
Ninja: Naruto [GENIN - KONOHA]  ATK=8 DEF=8 CHAKRA=14
  Jutsus:
   - Rasengan (coste 8)
   - Kage Bushin (coste 5)
Ninja: Gaara [CHUNIN - SUNA]  ATK=7 DEF=8 CHAKRA=13
  Jutsus:
   - Sabaky Kyuu (coste 7)
   - Futoon Sabaku Kyuu (coste 6)
```

Visitor

Se patrón se utilizó para crear informes de los ninjas y sus misiones. La clase ***'ExportVisitor'*** es la interfaz que define los métodos de visita, uno por cada tipo de objeto, en este caso *Ninja* y *Mission*.

```java
public interface ExportVisitor {
    void visit(Ninja ninja);
    void visit(Mission mission);
    String getReport();
}
```

Cada metodo de visita posee una implementación concreta de exportación dependiendo de cómo se quiera reportar. En este caso se desea exportar la información en formato Json y Text.



Es útil porque no implica cambiar significativamente la clase de dominio de los objetos que se visten: Ninja y Misiones, en su lugar, se añade un método *accept* que da paso al visitante. Esto delega toda la lógica de exportación a *Visitor*, lo cual facilita la adición de nuevas operaciones y mantiene un código limpio y claro.

Dominio Ninja

```java
public class Ninja {
    private final String name;
    private final Rank rank;
    private final Village village;
    private final Stats stats;
    private final List<Jutsu> jutsus;

    public Ninja(String name, Rank rank, Village village, Stats stats, List<Jutsu> jutsus) {
        this.name = name;
        this.rank = rank;
        this.village = village;
        this.stats = stats;
        this.jutsus = new ArrayList<>(jutsus);
    }

    public String getName() { return name; }
    public Rank getRank() { return rank; }
    public Village getVillage() { return village; }
    public Stats getStats() { return stats; }
    public List<Jutsu> getJutsus() { return Collections.unmodifiableList(jutsus); }
```

```
    // 🔑 Método para Visitor
    public void accept(ExportVisitor visitor) { visitor.visit( ninja: this); }
}
```

Dominio Mission

```
public class Mission {
    private final String name;
    private final MissionRank rank;
    private final int reward;
    private final Rank requiredRank;

    public Mission(String name, MissionRank rank, int reward, Rank requiredRank) {
        this.name = name;
        this.rank = rank;
        this.reward = reward;
        this.requiredRank = requiredRank;
    }

    public String getName() { return name; }
    public MissionRank getRank() { return rank; }
    public int getReward() { return reward; }
    public Rank getRequiredRank() { return requiredRank; }

    // 🔑 Método para Visitor
    public void accept(ExportVisitor visitor) { visitor.visit( mission: this); }
}
```

Ejecución:

1. Se utilizan los métodos Visitor en la función Main.

```
        // VISITOR TEXTO
        ExportVisitor textVisitor = new TextExportVisitor();
        naruto.accept(textVisitor);
        gaara.accept(textVisitor);
        mission.accept(textVisitor);
        System.out.println("\n== TEXT EXPORT ==\n" + textVisitor.getReport());

        // VISITOR JSON
        ExportVisitor jsonVisitor = new JsonExportVisitor();
        naruto.accept(jsonVisitor);
        gaara.accept(jsonVisitor);
        mission.accept(jsonVisitor);
        System.out.println("\n== JSON EXPORT ==\n" + jsonVisitor.getReport());
    }
```

2. **Terminal- Salida:** Reporte de ninjas con sus misiones, en formato Json y Text.

```
== TEXT EXPORT ==
Ninja: Naruto [GENIN - KONOHA]  ATK=8 DEF=8 CHAKRA=14
  Jutsus:
   - Rasengan (coste 8)
   - Kage Bushin (coste 5)
Ninja: Gaara [CHUNIN - SUNA]  ATK=7 DEF=8 CHAKRA=13
  Jutsus:
   - Sabaky Kyuu (coste 7)
   - Futoon Sabaku Kyuu (coste 6)
Mission: Escoltar al maestro puente [C]  Reward=300 RequiredRank=GENIN


== JSON EXPORT ==
{"items":[{"type":"ninja","name":"Naruto","rank":"GENIN","village":"KONOHA","stats":
```

Combate

```
⚔️ Combate entre Naruto y Gaara
 - Poder de Naruto: 23
 - Poder de Gaara: 21
🏆 Ganador: Naruto
```