Tarea Independiente 24/07/2025

David Núñez Franco

July 26, 2025

Inventario de Conceptos Claves (punto 0)

- Diagrama de Venn L-C-P (L: Lenguaje, P: Paradigma, C: Compilador/Traductor)
- Paradigma en filosofía de la ciencia (Thomas Kuhn)
- Realidad Objetiva versus Subjetiva. Positivismo
- Paradigma de Programación
- OOP: Sustantivos primero. Verbo supeditado a sustantivo (conducta del sustantivo)
- FP: Verbo primero. Pueden existir sin sustantivo
- LP: Relaciones primero. Pueden existir sin sustantivo

Problema (punto 1)

"La boda": Suponga que se está planificando una gran boda y hay distintos tipos de personas involucradas: novios, padres, padrinos, invitados, fotógrafos, músicos, cocineros y potencialmente otros más. Y hay tambien muchas funciones o roles: quién baila el primer vals, quién da el discurso, quién corta el pastel, quién firma como testigo, etc. hay relaciones estáticas (se conocen en tiempo de compilación). La relación entre personas y roles no es estática (es dinámica) y no es funcional en ninguna dirección. Su solución tiene casos de prueba separados del modelo y compila y corre desde una consola. Ningún rol es completamente abstracto.

Planteamiento del problema

> "La boda": Suponga que se está planificando una gran boda y hay distintos tipos de personas involucradas: novios, padres, padrinos, invitados, fotógrafos, músicos, cocineros y potencialmente otros más. Y hay tambien muchas funciones o roles: quién baila el primer vals, quién da el discurso, quién corta el pastel, quién firma como testigo, etc. hay relaciones estáticas (se conocen en tiempo de compilación). La relación entre personas y roles no es estática (es dinámica) y no

es funcional en ninguna dirección. Su solución tiene casos de prueba separados del modelo y compila y corre desde una consola. Ningún rol es completamente abstracto.

Análisis del Problema

ENTIDADES

- **Persona**: Clase padre para los asistentes novios, padres, padrinos, invitados....
- **Rol**: Acciones a desempeñar una persona (bailar el primer vals, dar discurso...).
- **Boda**: Coordinar personas y roles.

Relaciones

- **Persona ¡-¿ Rol**: Se comportará como una relación dinámica y no funcional. Una persona puede tener múltiples roles, y un rol puede ser desempeñado por múltiples personas.
- **Subclases Persona**: Novio, Padre, Padrino, Invitado, Fotógrafo, Músico.
- **Subclases Rol**: BailarPrimerVals, DarDiscurso, CortarPastel, FirmarTestigo

subsubsection*A considerar

1. Crear personas con distintos tipos y crear roles concretos.
2. Asignar dinámicamente uno o varios roles a personas, y viceversa.
3. Saber qué persona tiene qué rol y viceversa.

Solución al planteamiento del problema en Java, aplicando paradigma OOP

```
1         public abstract class Persona {
2             protected String nombre;
3
4             public Persona(String nombre) {
5                 this.nombre = nombre;
6             }
7
8             public String getNombre() {
9                 return nombre;
10            }
11
12            @Override
13            /*toString que permite obtener el nombre concreto de la
                 clase, para mostrarlo en el mensaje*/
```

```
14          public String toString() {
15              return this.getClass().getSimpleName() + ": " +
                    nombre;
16          }
17      }
```

Listing 1: Clase Persona en Java

```
1       public class Novio extends Persona{
2
3           public Novio(String nombre) {
4               super(nombre);
5           }
6       }
```

Listing 2: Clase Novio en Java

```
1       public class Padre extends Persona{
2
3           public Padre(String nombre) {
4               super(nombre);
5           }
6       }
```

Listing 3: Clase Padre en Java

```
1       public class Padrino extends Persona{
2
3           public Padrino(String nombre) {
4               super(nombre);
5           }
6       }
```

Listing 4: Clase Padrino en Java

```
1       public class Invitado extends Persona{
2
3           public Invitado(String nombre) {
4               super(nombre);
5           }
6       }
```

Listing 5: Clase Invitado en Java

```
1       public abstract class Rol {
2           protected String nombre;
3
4           public Rol(String nombre) {
5               this.nombre = nombre;
6           }
7
```

```
8           public String getNombre() {
9               return nombre;
10          }
11
12          @Override
13          public String toString() {
14              return this.getClass().getSimpleName() + ": " +
                    nombre;
15          }
16      }
```

Listing 6: Clase Rol en Java

```
1       public class BailarPrimerVals extends Rol{
2           public BailarPrimerVals() {
3               super("Bailar el primer vals");
4           }
5       }
```

Listing 7: Clase BailarPrimerVals en Java

```
1       public class DarDiscurso extends Rol{
2
3           public DarDiscurso() {
4               super("Dar el discurso");
5           }
6       }
```

Listing 8: Clase DarDiscurso en Java

```
1       public class CostarPastel extends Rol{
2           public CostarPastel() {
3               super("Cortar el pastel");
4           }
5       }
```

Listing 9: Clase CortarPastel en Java

```
1       public class FirmarComoTestigo extends Rol{
2           public FirmarComoTestigo() {
3               super("Firmar como testigo");
4           }
5       }
```

Listing 10: Clase FirmarComoTestigo en Java

```
1       import java.util.*;
2
3       public class Boda {
4           private List<Persona> personas;
5
```

```
6           /*Este atributo va a representar la relacion dinamica
               entre roles y personas
7           * La clave es Rol, y el valor (SetPersona) es quien
               desempena.
8           * Se una un set para evitar duplicados, ya que una misma
                persona no aparece en dos roles*/
9           private Map<Rol, Set<Persona>> rolesAsignados;
10
11          public Boda() {
12              this.personas = new ArrayList<>();
13              this.rolesAsignados = new HashMap<>();
14          }
15
16          public void agregarPersona(Persona persona) {
17              personas.add(persona);
18          }
19
20          public void asignarRol(Persona persona, Rol rol) {
21              rolesAsignados.computeIfAbsent(rol, k -> new HashSet
                   <>()).add(persona);
22          }
23
24          /*Se va a recorrer la lista de personas y, para cada una
               , busca en el mapa de roles si se encuentra
25          * asignada a algun rol*/
26          public void mostrarRolesPorPersona() {
27              for (Persona persona : personas) {
28                  System.out.println(persona);
29
30                  for(Map.Entry<Rol, Set<Persona>> entry :
                       rolesAsignados.entrySet()) {
31                      if (entry.getValue().contains(persona)) {
32                          System.out.println(" -> " + entry.getKey
                               ().getNombre());
33                      }
34                  }
35              }
36          }
37
38          /*Muestra todas las personas agrupadas por el rol a
               desempenar*/
39          public void mostrarPersonasPorRol() {
40              for(Map.Entry<Rol, Set<Persona>> entry :
                   rolesAsignados.entrySet()) {
41                  System.out.println(entry.getKey().getNombre() +
                       ":");
42
```

```
                for(Persona persona : entry.getValue()) {
                    System.out.println(" - " + persona.getNombre
                        ());
                }
            }
        }
    }
```

Listing 11: Clase Boda en Java

```
public class Main {
    public static void main(String[] args) {
        Boda boda = new Boda();

        Persona novio = new Novio("David");
        Persona novia = new Novio("Lucia");
        Persona padre = new Padre("Ernesto");
        Persona padrino = new Padrino("Chavarria");
        Persona invitado =  new Invitado("Gabriel");

        boda.agregarPersona(novio);
        boda.agregarPersona(novia);
        boda.agregarPersona(padre);
        boda.agregarPersona(padrino);
        boda.agregarPersona(invitado);

        Rol vals = new BailarPrimerVals();
        Rol discurso = new DarDiscurso();
        Rol pastel = new CostarPastel();
        Rol testigo = new FirmarComoTestigo();

        boda.asignarRol(novio, vals);
        boda.asignarRol(novia, vals);
        boda.asignarRol(padre, discurso);
        boda.asignarRol(padrino, testigo);
        boda.asignarRol(invitado, pastel);

        System.out.println("==ROLES POR PERSONA==");
        boda.mostrarRolesPorPersona();

        System.out.println("\nPERSONAS POR ROL");
        boda.mostrarPersonasPorRol();
    }
}
```

Listing 12: Clase Main en Java

Solución al planteamiento del problema en JavaScript, aplicando paradigma FP

```
/*Crear a la Persona, con el nombre y el tipo*/

function crearPersona(nombre, tipo) {
    return {
        nombre,
        tipo,
        toString: function () {
            return `${tipo}: ${nombre}`;
        },
    };
}

/*Crear el Rol, con el nombre*/

function crearRol(nombre) {
    return { nombre };
}

/*Funcion que estructura la boda*/

function crearBoda() {
    const personas = [];
    const rolesAsignados = new Map(); /*Es decir, Map<Rol,
       Set<Persona>>*/

    function agregarPersona(persona) {
        personas.push(persona);
    }

    function asignarRol(persona, rol) {
        if (!rolesAsignados.has(rol)) {
            rolesAsignados.set(rol, new Set());
        }
        rolesAsignados.get(rol).add(persona);
    }

    function mostrarRolesPorPersona() {
        /*Recorremos todas las personas agregadas a la boda
           */
        for (const persona of personas) {
            console.log(persona.toString());

            /*Recorremos todas las asignaciones de roles*/
            for (const [rol, personasAsignadas] of
```

```
                    rolesAsignados.entries()) {
43                      /*Si la persona tiene ese rol asignado, lo
                            muestra*/
44                      if (personasAsignadas.has(persona)) {
45                          console.log(` -> ${rol.nombre}`);
46                      }
47                  }
48              }
49          }
50
51          function mostrarPersonasPorRol() {
52              /*Recorremos las asignaciones de roles*/
53              for (const [rol, personasAsignadas] of
                    rolesAsignados.entries()) {
54                  console.log(`${rol.nombre}:`);
55
56                  /*Muestra el nombre de cada persona que tiene
                        asignado ese rol*/
57                  for (const persona of personasAsignadas) {
58                      console.log(` - ${persona.nombre}`);
59                  }
60              }
61          }
62
63          return {
64              agregarPersona,
65              asignarRol,
66              mostrarRolesPorPersona,
67              mostrarPersonasPorRol,
68          };
69      }
70
71      /*Crear la boda*/
72      const boda = crearBoda();
73
74      /*Personas*/
75      const david = crearPersona("David", "Novio");
76      const lucia = crearPersona("Lucia", "Novia");
77      const ernesto = crearPersona("Ernesto", "Padre");
78      const chavarria = crearPersona("Chavarria", "Padrino");
79      const sofia = crearPersona("Sofia", "Invitada");
80
81      /*Agregar personas*/
82      boda.agregarPersona(david);
83      boda.agregarPersona(lucia);
84      boda.agregarPersona(ernesto);
85      boda.agregarPersona(chavarria);
```

```
86          boda.agregarPersona(sofia);
87
88          /*Roles*/
89          const vals = crearRol("Bailar el primer vals");
90          const discurso = crearRol("Dar el discurso");
91          const pastel = crearRol("Cortar el pastel");
92          const testigo = crearRol("Firmar como testigo");
93
94          /*Asignar roles*/
95          boda.asignarRol(david, vals);
96          boda.asignarRol(lucia, vals);
97          boda.asignarRol(ernesto, discurso);
98          boda.asignarRol(chavarria, testigo);
99          boda.asignarRol(sofia, pastel);
100
101         console.log("=== ROLES POR PERSONA ===");
102         boda.mostrarRolesPorPersona();
103
104         console.log("\n=== PERSONAS POR ROL ===");
105         boda.mostrarPersonasPorRol();
```

Listing 13: Implementación 'La Boda' en JavaScript

Problema (punto 2)

En modelamiento algebraico de datos (abstract data types, ADTs) hay datos puros que no tiene métodos de lógica especial (a lo más getters y formas de serialización como hileras, alis toString). Son inmutables (no se puede cambiar su estado, solo inicializarlo durante creación). En Java a ese tipo de dato se le dice un POJO (plain old data object). Su solución tiene casos de prueba separados del modelo y compila y corre desde una consola. Restricción: no usar 'instanceOf'

- Modele los números naturales como ADTs.
- Modele la suma de naturales sin usar la suma primitiva durante la operación. Añada un toString que permita ver al nat como número en decimal

POJO (Plain Old Java Object) en Java

Características fundamentales:

- Dato puro que no tiene métodos de lógica especial (a lo sumo getters, hileras, toString) y funciones puras (add, succ).
- Inmutables.

- NO debe extender de otra clase (no puede ser clase hija).
- Usar implements de una interfaz simple.
- NO debe usar anotaciones sofisticadas ni frameworks.

Modelo de números naturales como ADT's, aplicando concepto de POJO en Java

```java
/*Interfaz que representa los naturales*/
public interface Nat {
    int toInt(); // Serializar los numeros a enteros
    String toString(); // Representar como texto
}
```

Listing 14: Clase Nat en Java

```java
/*Representa el numero 0*/

public final class Zero implements Nat {

    public Zero() {

    }

    @Override
    public int toInt() {
        return 0;
    }

    @Override
    public String toString() {
        return "cero";
    }
}
```

Listing 15: Clase Zero en Java

```java
/*Representa el sucesor de un numero natural*/
public final class Succ implements Nat {
    private final Nat prev;

    public Succ(Nat prev) {
        this.prev = prev;
    }

    public Nat getPrev() {
        return prev;
```

```
        }

        @Override
        public int toInt() {
            return 1 + prev.toInt();
        }

        @Override
        public String toString() {
            return "Sucesor(" + prev.toString() + ")";
        }
    }
```

Listing 16: Clase Succ en Java

```
    public class Main {
        public static void main(String[] args) {
            Nat zero = new Zero();
            Nat one = new Succ(zero);
            Nat two = new Succ(one);
            Nat three = new Succ(two);

            System.out.println("cero: " + zero.toString() + " =
               " + zero.toInt());
            System.out.println("uno: " + one.toString() + " = "
               + one.toInt());
            System.out.println("dos: " + two.toString() + " = "
               + two.toInt());
            System.out.println("tres: " + three.toString() + " =
                " + three.toInt());
        }
    }
```

Listing 17: Main en Java

Modelo de suma de números naturales sin usar suma primitiva, aplicando concepto de POJO en Java

```
    /*Interfaz que simula los numeros naturales*/
    public interface Nat {
        Nat add(Nat other); // Esta es nuestra suma recursiva
        int toInt(); // Conversion a Int
        String toString(); // Texto decimal
    }
```

Listing 18: Clase Nat en Java

```
    /*Representa el numero 0*/
```

```java
public final class Zero implements Nat{
    public Zero() {

    }

    @Override
    public Nat add(Nat other) {
        /*0 + n = n*/
        return other;
    }

    @Override
    public int toInt() {
        return 0;
    }

    @Override
    public String toString() {
        /*Convierte un entero a su representacion en cadena
           */
        return String.valueOf(toInt());
    }
}
```

Listing 19: Clase Zero en Java

```java
/*Representa el numero natural sucesor*/
public final class Succ implements Nat{
    private final Nat prev;

    public Succ(Nat prev) {
        this.prev = prev;
    }

    public Nat getPrev() {
        return prev;
    }

    @Override
    public Nat add(Nat other) {
        // Suma(n) + m = Suma(n + m)
        return new Succ(prev.add(other));
    }

    @Override
    public int toInt() {
        return 1 + prev.toInt();
    }
```

```
23
24              @Override
25              public String toString() {
26                  return String.valueOf(toInt());
27              }
28          }
```

Listing 20: Clase Succ en Java

```
1           public class Main {
2               public static void main(String[] args) {
3                   Nat zero = new Zero();
4                   Nat one = new Succ(zero);
5                   Nat two = new Succ(one);
6                   Nat three = new Succ(two);
7
8                   Nat suma1 = two.add(three); // 2 + 3
9                   Nat suma2 = three.add(three); // 3 + 3
10                  Nat suma3 = three.add(zero); // 3 + 0
11
12                  System.out.println(suma1.toString());
13                  System.out.println(suma2.toString());
14                  System.out.println(suma3.toString());
15              }
16          }
```

Listing 21: Main en Java