

Observer Pattern

Requerimiento inicial:

Crear un servicio que nos permita dar de alta una nueva matricula

Primera solución:

```
public class ServicioMatriculacion {  
    public void alta (Matricula nueva) {  
        System.out.printf("Creando nueva matricula de %s para %s.\n",  
                           nueva.getCurso(), nueva.getAlumno());  
    }  
}
```


Nuevo requerimiento:

Enviar un mensaje al alumno con la información de la nueva matrícula

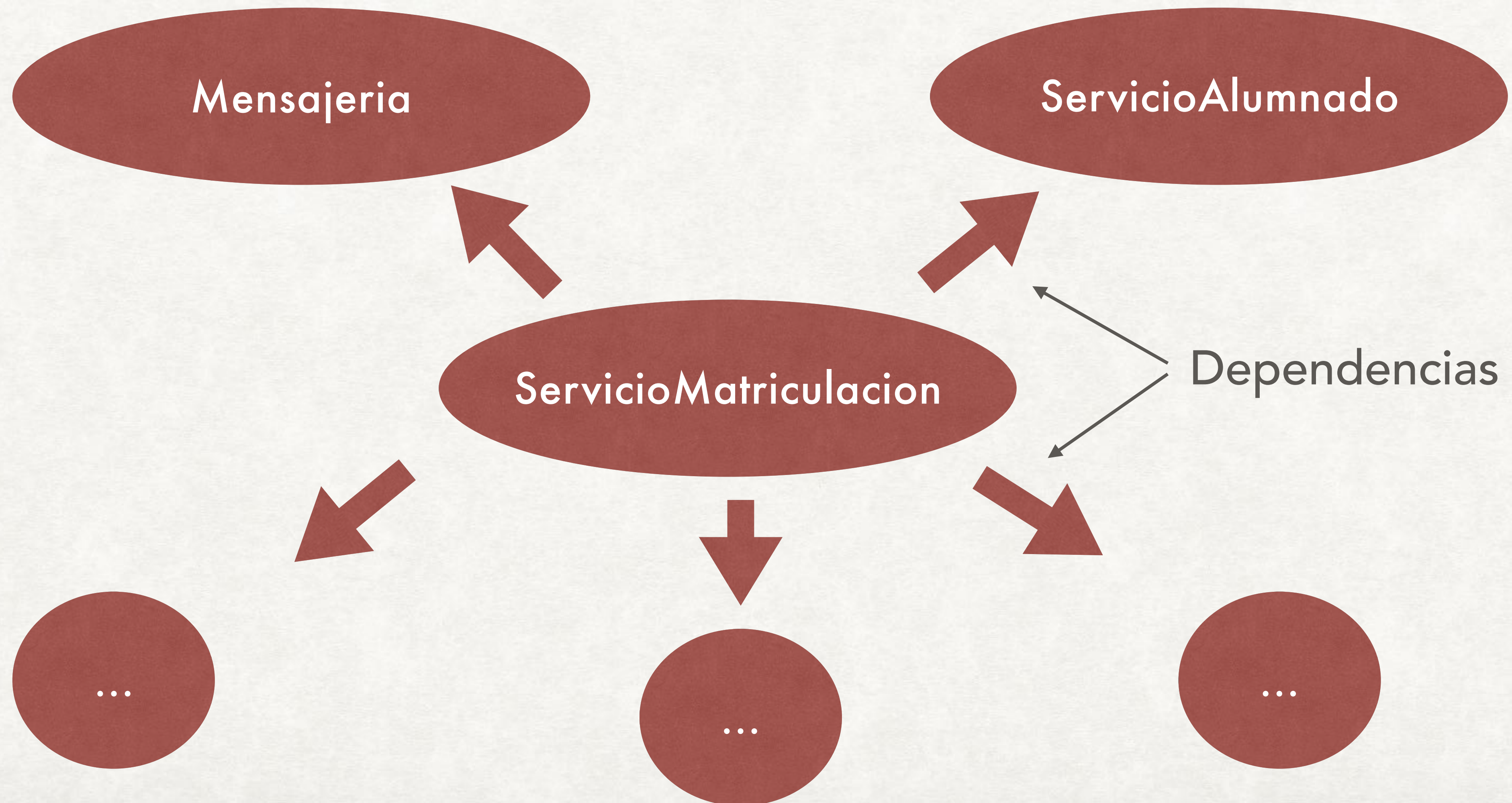
```
public class ServicioMatriculacion {  
  
    public void alta (Matricula nueva) {  
  
        System.out.printf("Creando nueva matricula de %s para %s.\n",  
                           nueva.getCurso(), nueva.getAlumno());  
  
        new Mensajeria().enviaMensajeAlta(nueva.getAlumno(), nueva.getCurso());  
  
    }  
  
}
```


Y otro requerimiento:

Si es la primera matrícula, darlo de alta en el servicio de alumnado

```
public class ServicioMatriculacion {  
  
    public void alta (Matricula nueva) {  
  
        System.out.printf("Creando nueva matricula de %s para %s.\n",  
                           nueva.getCurso(), nueva.getAlumno());  
  
        new Mensajeria().enviaMensajeAlta(nueva.getAlumno(), nueva.getCurso());  
  
        if ( nueva.isPrimeraMatricula() ) {  
            new ServicioAlumnado().nuevoAlumno(nueva.getAlumno());  
        }  
  
    }  
  
}
```


Y otro, y otro, y otro ...



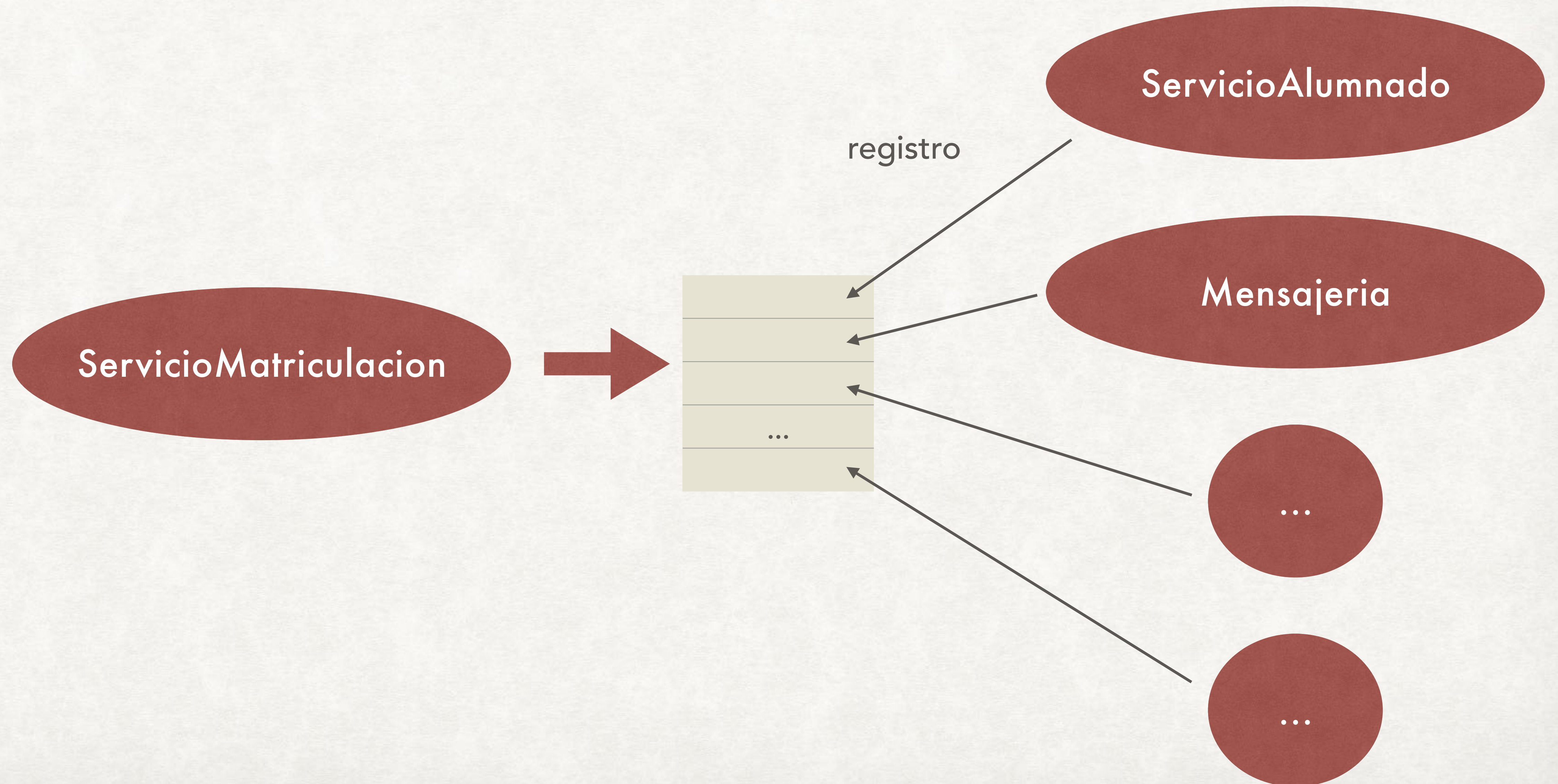

```
public class ServicioMatriculacion {  
  
    public void alta (Matricula nueva) {  
  
        System.out.printf("Creando nueva matricula de %s para %s.\n",  
                           nueva.getCurso(), nueva.getAlumno());  
  
        new Mensajeria().enviaMensajeAlta(nueva.getAlumno(), nueva.getCurso());  
  
        if ( nueva.isPrimeraMatricula() ) {  
            new ServicioAlumnado().nuevoAlumno(nueva.getAlumno());  
        }  
  
        // ...  
        // nuevos servicios  
        // ...  
  
    }  
  
}
```

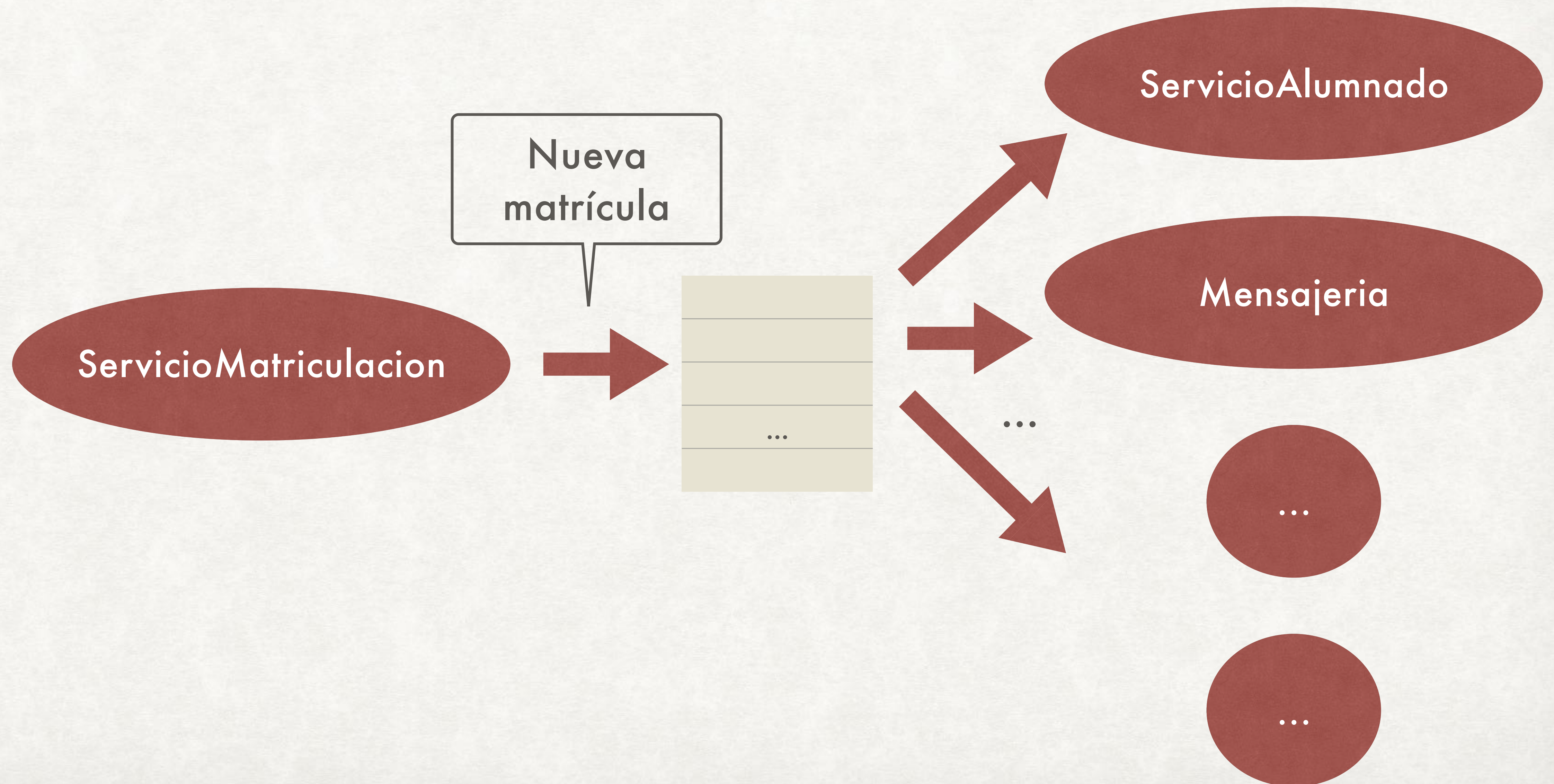

“

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and update automatically.

— *GoF*

”





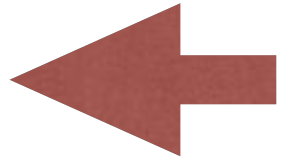

```
public interface NuevaMatriculaObserver {  
  
    public void nueva (Matricula nueva);  
  
}
```



```
public class Mensajeria implements NuevaMatriculaObserver {  
  
    @Override  
    public void nueva(Matricula nueva) {  
        System.out.printf("Nuevo email: ¡Enhorabuena %s!, te has matriculado de %s.\n",  
            nueva.getAlumno(), nueva.getCurso() );  
    }  
}
```


Registro

```
public class ServicioMatriculacion {  
    private Set<NuevaMatriculaObserver> observers = new LinkedHashSet<>();  
  
    public void registra (NuevaMatriculaObserver observer) {  
        observers.add(observer);  
    }  
  
    public void alta (Matricula nueva) {  
        ...  
    }  
}
```



Configuración

```
ServicioMatriculacion matriculador = new ServicioMatriculacion();  
matriculador.registra(new Mensajeria());  
matriculador.registra(new ServicioAlumnado());
```


Observed

```
public class ServicioMatriculacion {  
  
    private Set<NuevaMatriculaObserver> observers = new LinkedHashSet<>();  
  
    public void registra (NuevaMatriculaObserver observer) {  
        observers.add(observer);  
    }  
  
    public void alta (Matricula nueva) {  
  
        System.out.printf("Creando nueva matricula de %s para %s.\n",  
                           nueva.getCurso(), nueva.getAlumno());  
  
        for (NuevaMatriculaObserver observer: observers) {  
            observer.nueva(nueva);  
        }  
    }  
}
```


Observer Pattern