Reuso de Código

Herencia vs. Composición

Herencia de Clases

- Herencia total: debo conocer todo el código que se hereda -> Reutilización de Caja Blanca
- Usualmente debemos redefinir o anular métodos heredados
- Los cambios en la superclase se propagan automáticamente a las subclases
- Herencia de Estructura vs. Herencia de comportamiento
- Es útil para extender la funcionalidad del dominio de aplicación

Composición de Objetos

- Los objetos se componen en forma Dinámica -> Reutilización de Caja Negra
- Los objetos pueden reutilizarse a través de su interfaz (sin conocer el código)
- A través de las relaciones de composición se pueden delegar responsabilidades entre los objetos

Herencia vs. Composición de Objetos

- Las clases y los objetos creados mediante herencia están *estrechamente acoplados* ya que cambiar algo en la superclase afecta directamente a la/las subclases.

- Las clases y los objetos creados a través de la composición están *débilmente acoplados*, lo que significa que se pueden cambiar más fácilmente los componentes sin afectar el objeto contenedor.

Cómo hacer un mal uso de la herencia

```
import java.util.ArrayList;

 public class Stack<T> extends ArrayList<T> {

     public void push(T object) {
         this.add(object);
     }
     public T pop() {
         return this.remove(this.size() - 1);
     }
     public boolean empty() {
         return this.size() == 0;
     }
     public T peek() {
         return this.get(this.size() - 1);
     }
}
```

Cómo hacer un mal uso de la herencia

- Esta clase ***Stack*** funcionará como una pila, pero su interfaz es *voluminosa*; está formada por mensajes que hay que anular o redefinir!!

- La interfaz pública de esta clase no es solo push y pop, (esperable para una Pila), también incluye
- add en cualquier posición por índice,
- remove de una posición a otra,
- clear y muchos otros mensajes heredados de ArrayList que son inapropiados para una Pila.

Cómo hacer un mal uso de la herencia

El ejemplo tiene errores de diseño

- uno semántico:

"una pila es una ArrayList" no es cierto; Stack no es un subtipo adecuado de ArrayList (No cumple con Is-a). Se supone que una pila aplica el último en entrar, primero en salir, una restricción que se satisface con la interfaz push/pop, pero que no se cumple con la interfaz de ArrayList que es mucho más extensa.

- Otro mecánico:

heredar de ArrayList viola el encapsulaminto; usar ArrayList para contener la colección de objetos de la pila es una opción de implementación que puede y debe ocultarse.

Composición, no Herencia

```
import java.util.ArrayList;

 public class Stack<T> {
     private ArrayList<T> elementos


     public void push(T object) {
         elementos.add(object);
    }
     public T pop() {
        return elementos.remove(elementos.size() - 1);
    }
     public boolean empty() {
         return elementos.size() == 0;
    }
     public T peek() {
         return elementos.get(elementos.size() - 1);
    }
}
```

Cómo solucionar un mal uso de la herencia

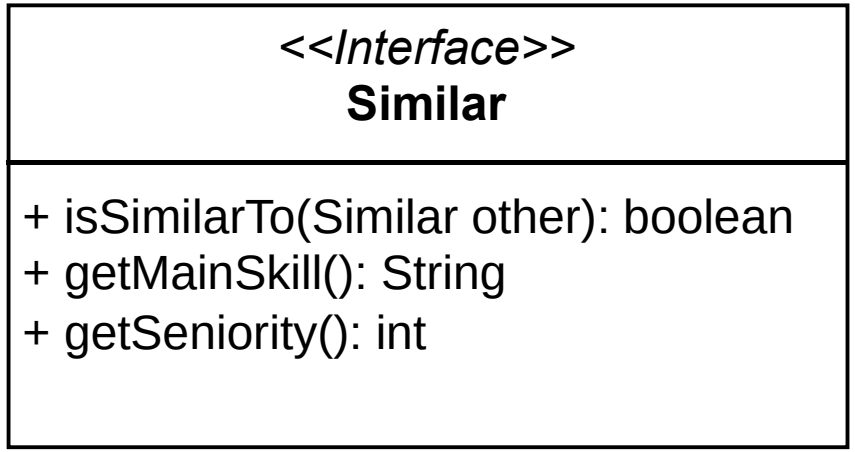
Composición, no herencia…

- Mecánicamente, heredar de ArrayList no cumple con el encapsulamiento; en cambio,
- **componer con ArrayList para contener la colección de objetos de la pila es una opción de implementación que permite ocultarla públicamente.**
- En este caso, en vez de heredar, el uso o composición permite reuso y mantiene el encapsulamiento!!

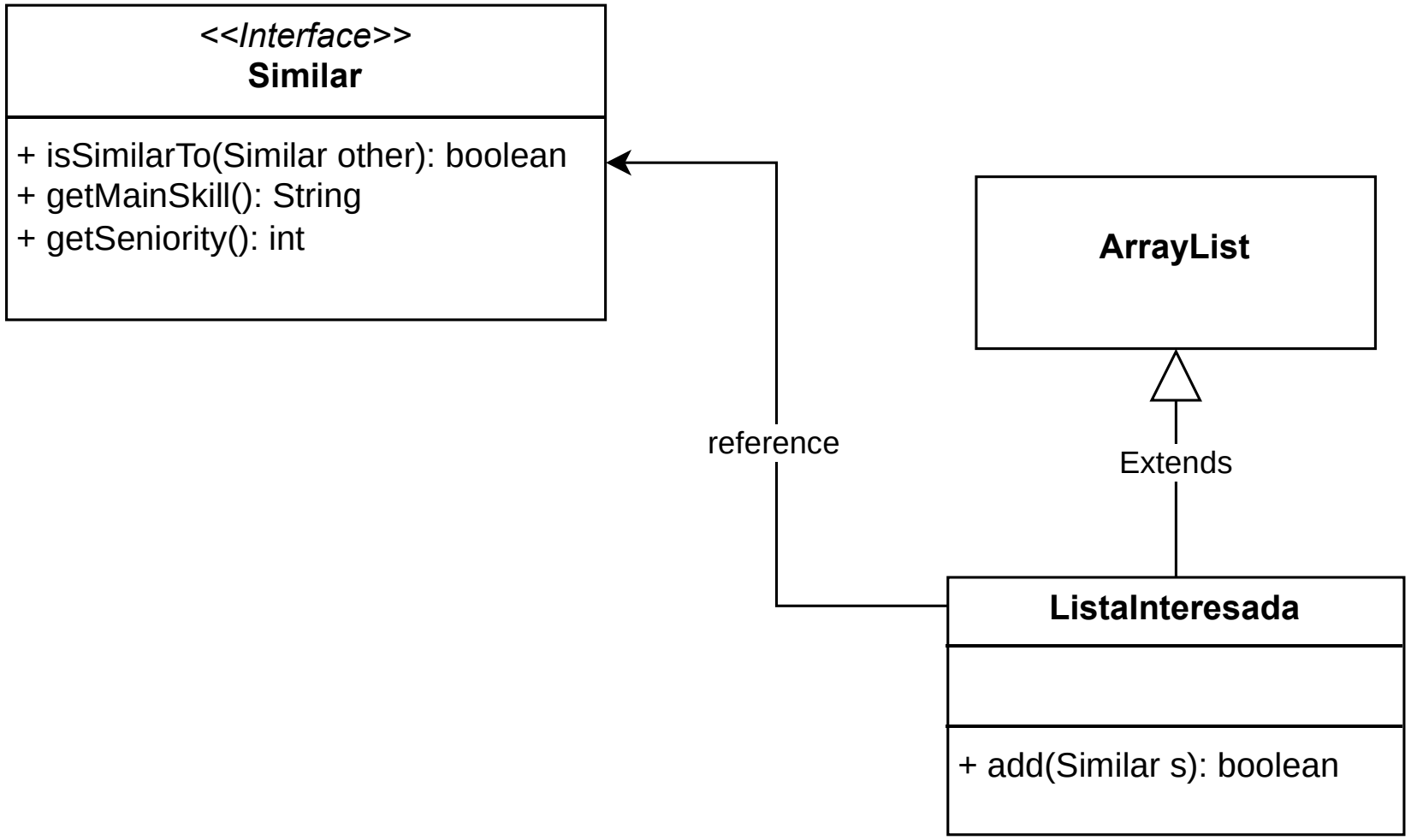
Rehuso herencia – un caso correcto e interesante

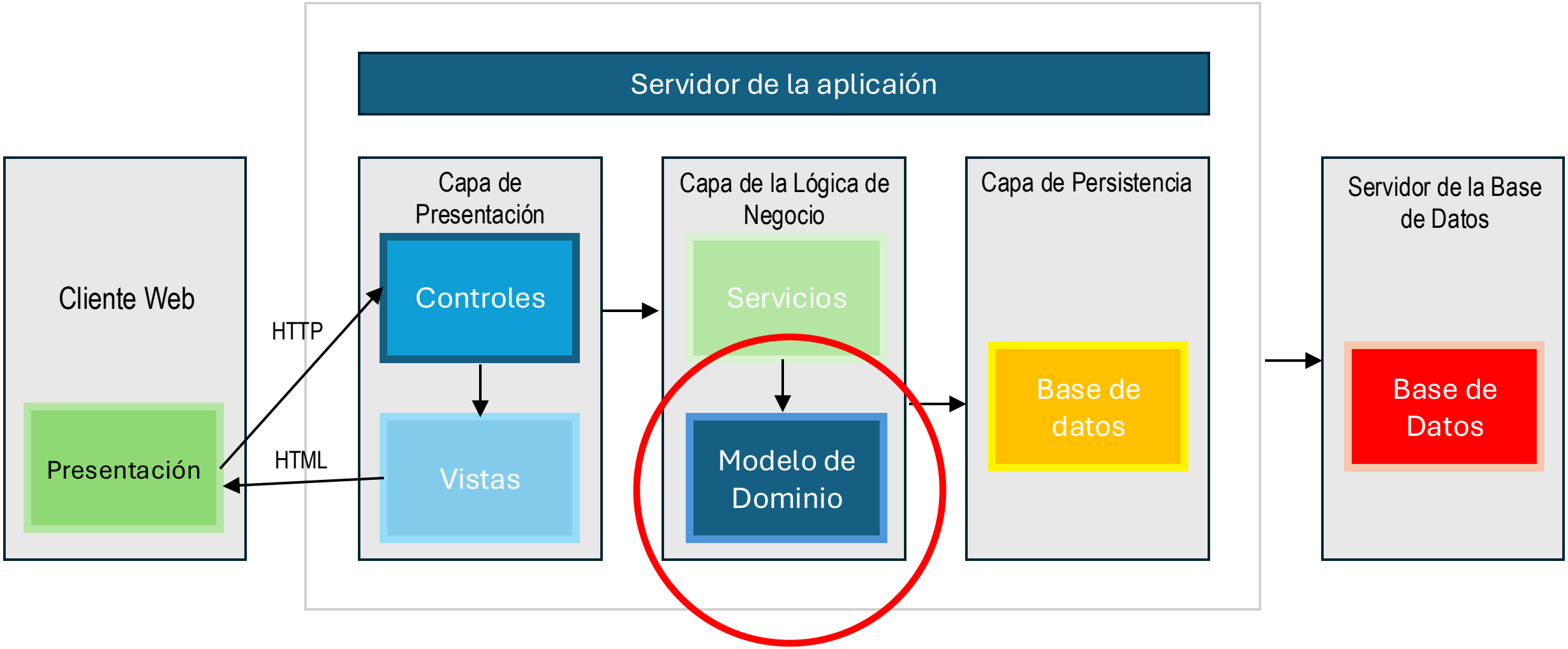
Recruiter

- Queremos una lista, que en su constructor tome un Similar.
- Cuando se agrega un elemento se verifica que sea similar a lo que se definió como parámetro, sino levanta una excepción.



* If a collection refuses to add a particular element for any reason
* other than that it already contains the element, it must throw
* an exception (rather than returning {@code false}). This preserves
* the invariant that a collection always contains the specified element
* after this call returns.


<<Interface>>
Similar
+ isSimilarTo(Similar other): boolean
+ getMainSkill(): String
+ getSeniority(): int
ArrayList
reference
Extends
ListaInteresada
+ add(Similar s): boolean


Servidor de la aplicaión
Cliente Web
Presentación
HTTP
HTML
Capa de Presentación
Controles
Vistas
Capa de la Lógica de Negocio
Servicios
Modelo de Dominio
Capa de Persistencia
Base de datos
Servidor de la Base de Datos
Base de Datos