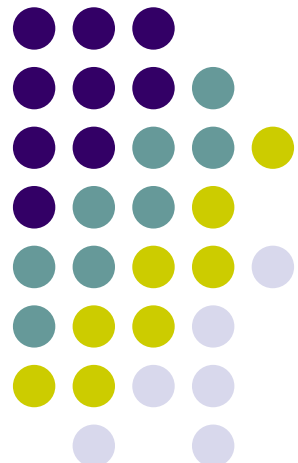
# Patrones de diseño

## Patrón básico *Handler*



# Patrones de diseño
## Introducción


- Objetivos:
  - Diseño específico para el problema, pero general para poder adecuarse a futuros requisitos
  - Evitar el rediseño en la medida de lo posible
  - Evitar resolver cada problema partiendo de cero
  - Reutilizar soluciones que han sido útiles en el pasado

- Idea:
  - Patrones recurrentes de clases y comunicación entre objetos en muchas soluciones de diseño
  - Reutilizar diseños abstractos que no incluyan detalles de la implementación

# Patrones de diseño
## Introducción


- Qué son:
  - Descripción del problema y la esencia de su solución, que se puede reutilizar en casos distintos
  - Solución adecuada a un problema común
  - Documentar la experiencia en el diseño

- Tipos:
  - De creación: implica el proceso de instanciar objetos
  - Estructurales: composición de objetos
  - De comportamiento: cómo se comunican los objetos, cooperan y distribuyen las responsabilidades para lograr sus objetivos

# Patrones de diseño
## Estructura de un patrón


- Nombre del patrón
  - Describe el problema de diseño, soluciones y consecuencias
  - Vocabulario de diseño
- Problema
  - Describe cuándo aplicar el patrón (aplicabilidad)
  - Explica el problema y su contexto (motivación)
- Solución
  - Elementos que forman el diseño, relaciones, responsabilidades
  - No un diseño concreto, sino una plantilla que puede aplicarse en muchas situaciones distintas
- Consecuencias
  - Resultados, ventajas e inconvenientes de aplicar el patrón
  - Por ejemplo: relación entre eficiencia en espacio y tiempo, cuestiones de implementación, etc.

# Patrón básico *Handler*

## Propósito


- Sirve para manejar *identificadores* de objetos de manera independiente a su implementación

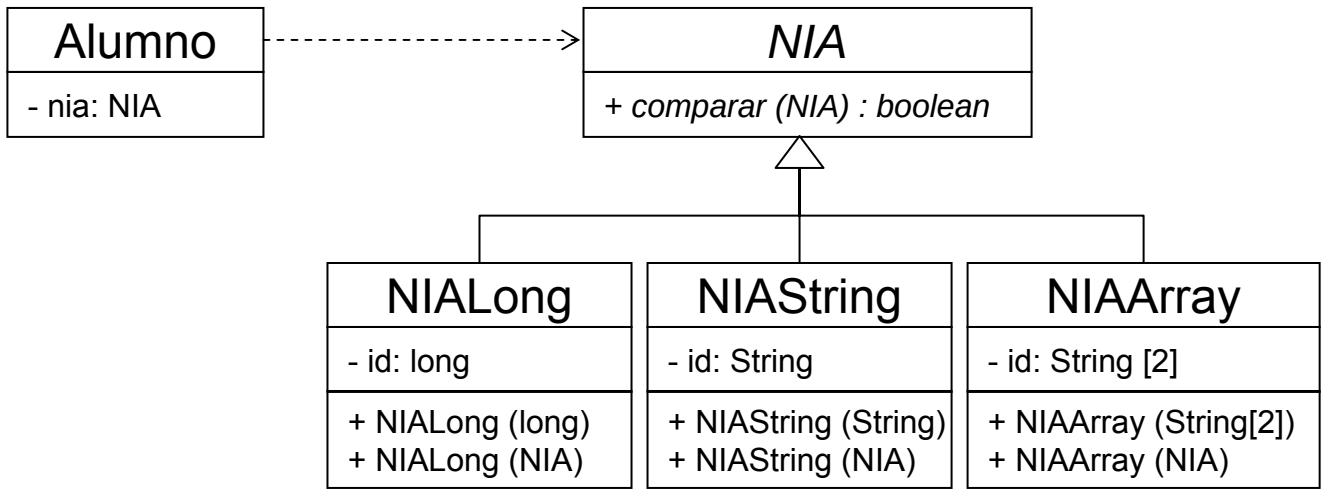
- Permite cambiar fácilmente la implementación de un identificador (`int`, `String`, ...) a cualquier tipo básico o clase primitiva, sea sencilla o compuesta

# Patrón básico *Handler*
## Motivación


- A veces una clase tiene un identificador de cierto tipo, y puede que el tipo cambie en futuras versiones de la clase
- Subclases de una misma clase común usan identificadores de distinto tipo
- Ej: Una aplicación de gestión de alumnos
  - Los alumnos se identifican por un NIA de 10 dígitos
- Solución:
  - Atributo del tipo especificado: no es flexible, si el tipo cambia la clase puede requerir muchos cambios
  - Definir una clase “de envoltura” para el identificador

# Patrón básico *Handler*

## Motivación

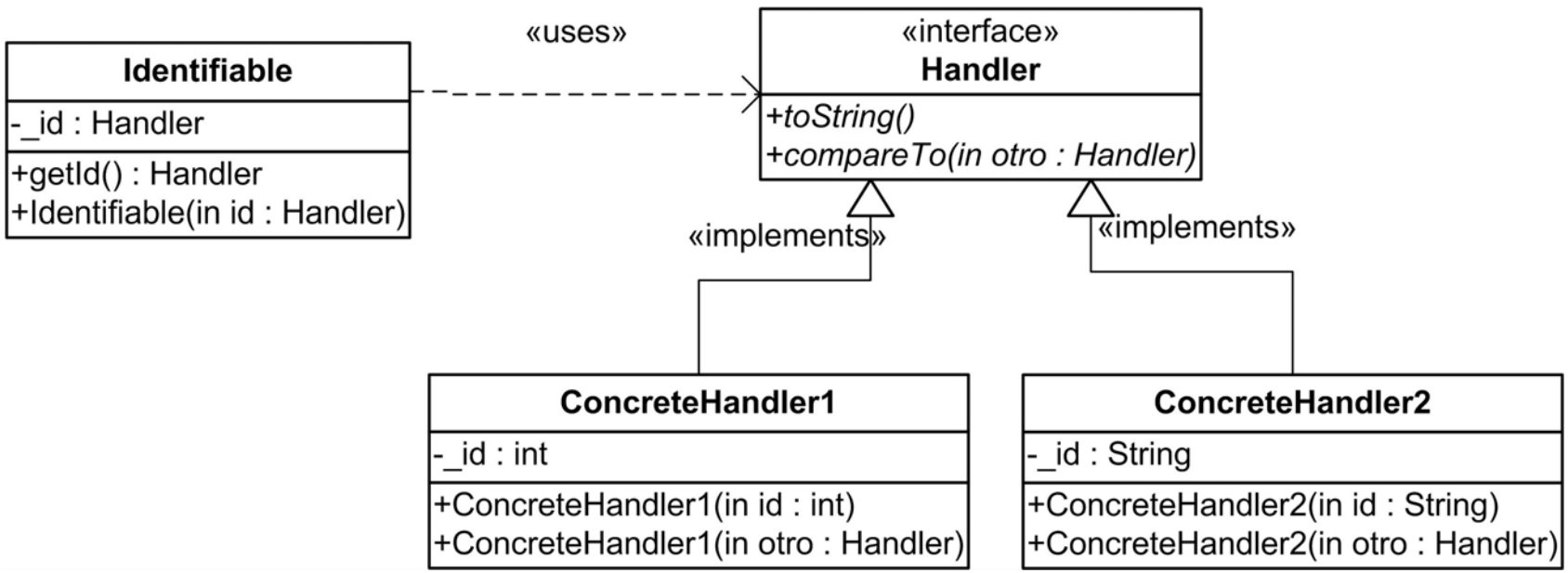



## Aplicabilidad

- Usa el patrón *Handler* cuando:
  - Tienes una clase que usa algún tipo de identificador

# Patrón básico *Handler*

## Estructura



# Patrón básico *Handler*
## Participantes


- **Identifiable (*Alumno*)**: clase cliente que necesita identificar a sus objetos a través de un atributo identificador

- **Handler (*NIA*)**: interfaz para declarar los identificadores de los objetos de la clase `Identifiable`

- **ConcreteHandler (*NIALong*, *NIAString*, *NIAArray*)**: implementación concreta de la interfaz `Handler`

# Patrón básico *Handler*

## Ejemplo de Implementación


```
interface Handler {
   String toString();
   int compareTo(Handler otro);
}

class Identificador implements Handler {
   private int _id;
   Identificador (String id) throws NumberFormatException {
      _id = new Integer(id).intValue();
   }
   Identificador (Handler otro) throws NumberFormatException {
      _id = new Integer(otro.toString()).intValue();
   }
   public String toString() { return new Integer(_id).toString(); }
   public int compareTo (Handler otro) {
      return toString().compareTo(otro.toString());
   }
}

class Identifiable {
   Handler _id;
   public Identifiable(String id) {_id = new Identificador(id); }
}
```

# Patrón básico *Handler*
## En java…


- `java.lang.Comparable`
- Implementado por clases de envoltura (`Integer`, `Long`, etc.), `String`, `File`, `Date`, ...
- `public int compareTo (Object o) throws ClassCastException`
  - Invariantes:
    `sgn(x.compareTo(y)) = -sgn(y.compareTo(x))`
    `(x.compareTo(y)>0 and y.compareTo(z)) → x.compareTo(z)>0`
    `x.compareTo(y)=0 → sgn(x.compareTo(z))=sgn(y.compareTo(z)) para todo z`
  - Consistente con equals:
    `(x.compareTo(y)=0) = (x.equals(y))`