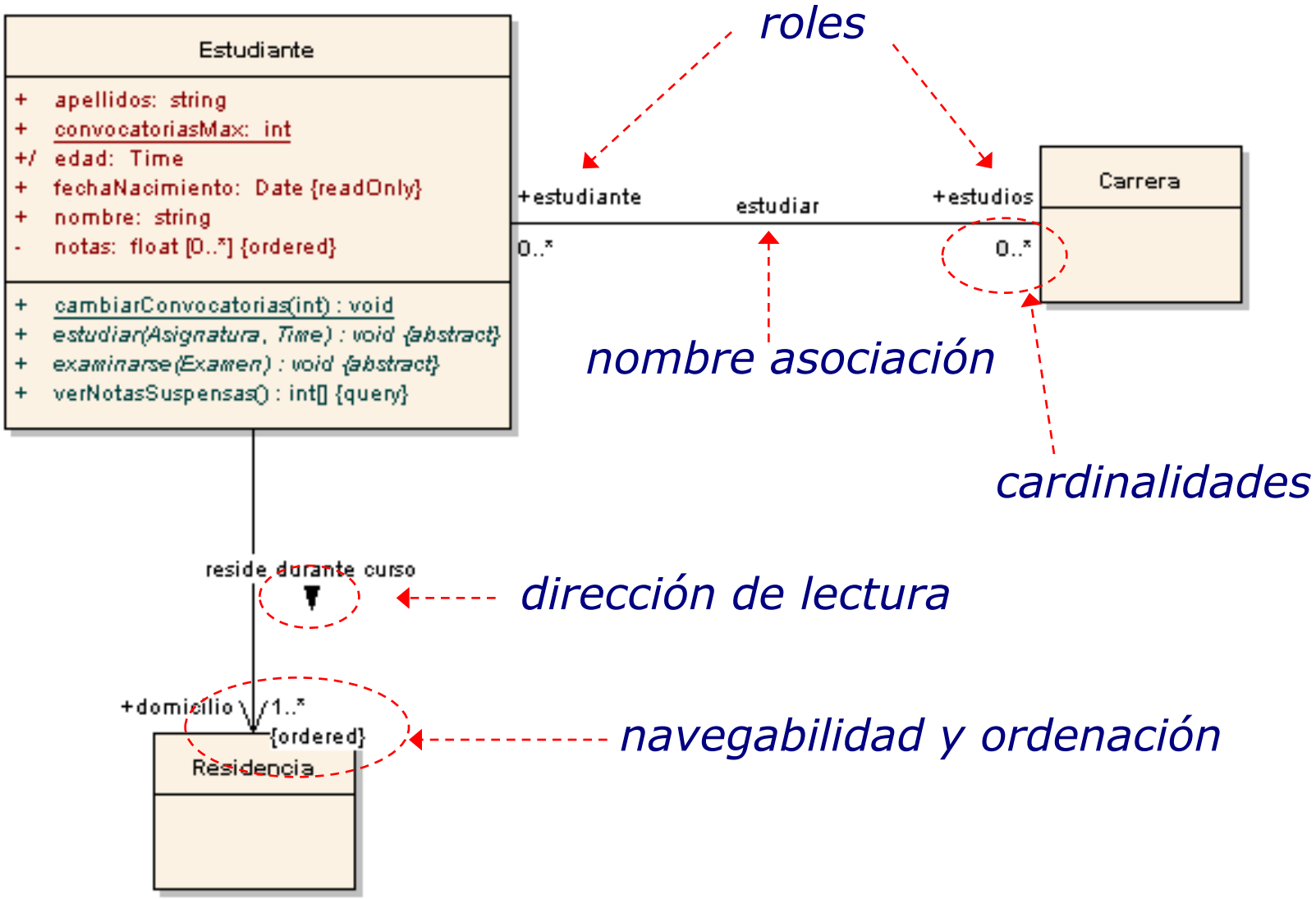
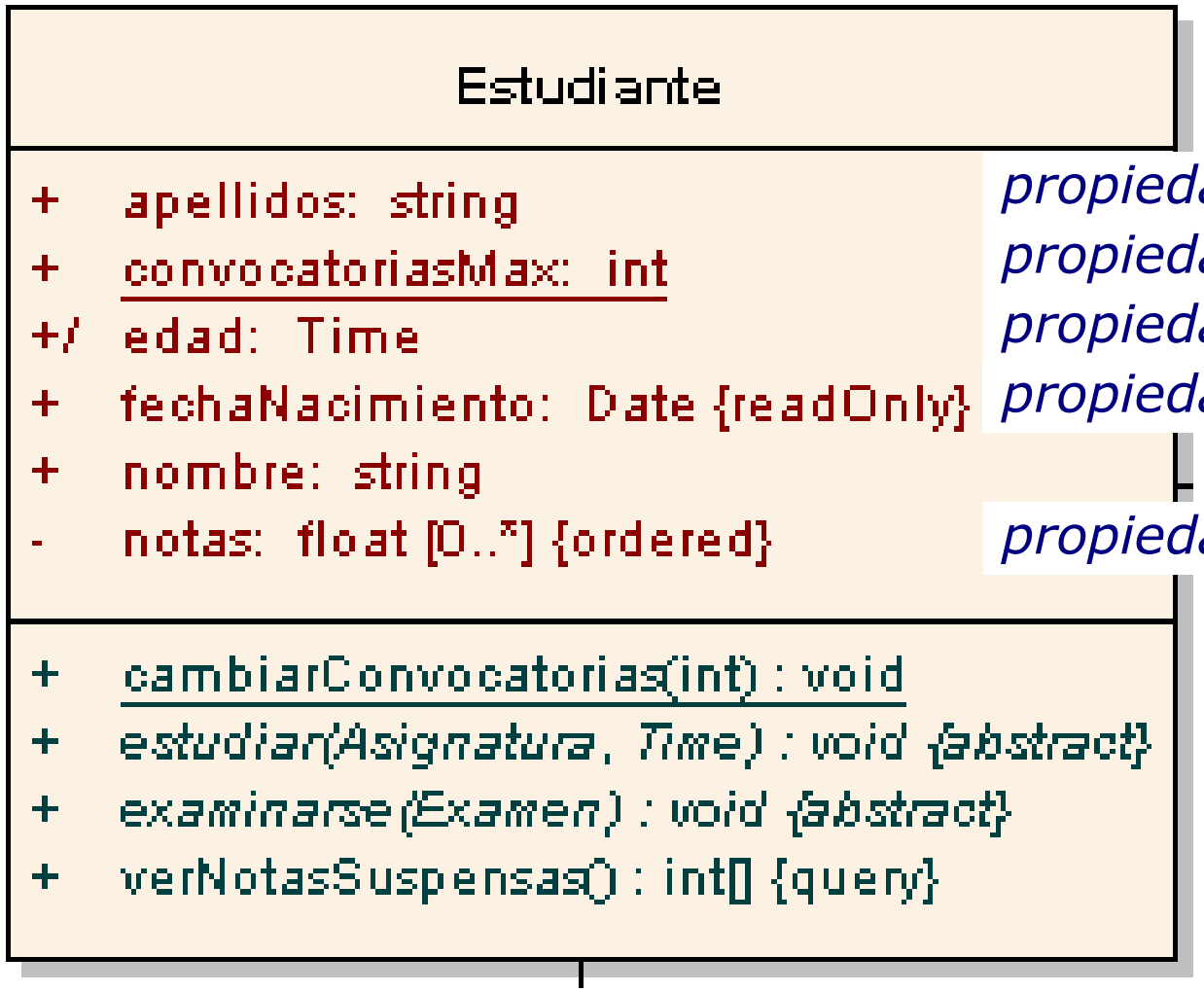
Implementación de modelos de dominio UML en Java

Repositorios de Información

Clases y Asociaciones





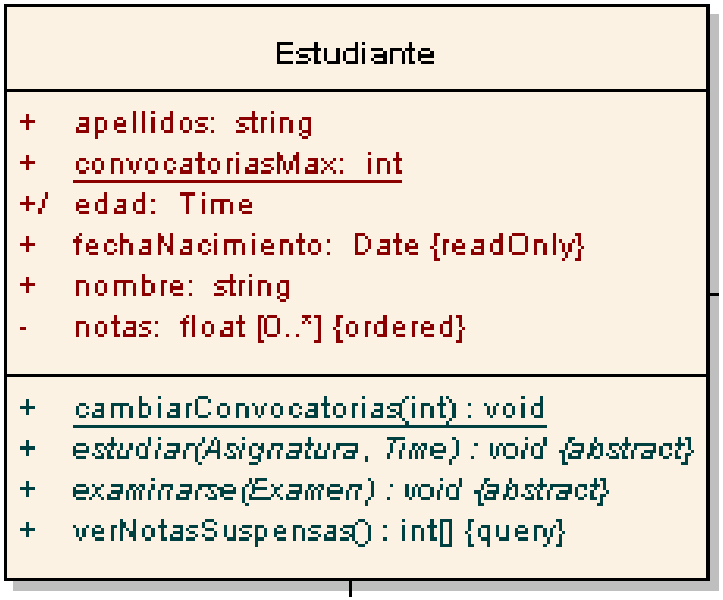
propiedad de lectura y escritura

propiedad estática

propiedad calculada

propiedad de sólo lectura

propiedad privada




```
public class Estudiante {

    private static int convocatoriasMax;
    private String apellidos;
    private String nombre;
    private Date fechaNacimiento;
    private float notas[];
    ...
```

propiedad estática

```
public class Estudiante {
    ...
    private static int convocatoriasMax;
    ...
    public static int getConvocatoriasMax() {
        return convocatoriasMax;
    }
    public static void setConvocatoriasMax(int convocatoriasMax) {
        Estudiante.convocatoriasMax = convocatoriasMax;
    }
    ...
```

propiedad de lectura y escritura

```
public class Estudiante {
    ...
    private String apellidos;
    ...
    public String getApellidos() {
        return apellidos;
    }
    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }
    ...
```

propiedad sólo lectura

```
public class Estudiante {
    ...
    private Date fechaNacimiento;
    ...
    public Date getFechaNacimiento() {
        return fechaNacimiento;
    }
    // no tiene setter
    ...
```

propiedad calculada

```
public class Estudiante {
    ...
    public Time getEdad() {
        return DateUtil.today().substract( edad ).asTime();
    }
    ...
```

propiedad privada

```
public class Estudiante {
    ...
    private float notas[];
    ...
    // sin getters ni setters, manipulada internamente
    ...
```

Cuidado en getters

Algunos getters pueden romper la encapsulación

```
public class Item {
    private String name;
    . . .
    private Date endDate;
    . . .
    private Set<Image> images = new HashSet<Image>();

    public Date getEndDate() {
        return endDate;
    }
    public String getName() {
        return name;
    }
    public Set<Image> getImages() {
        return images;
    }
}
```

Peligro!!!

java.util.Date es mutable

Seguro, String es inmutable

Peligro!!!

El que recibe esta colección le puede añadir o quitar elementos descontroladamente

Algunos getters pueden romper la encapsulación

```
Item i = ...
print( i.getEndDate() ); // shows 12/12/2012

Date date = i.getEndDate();
date.setDay(25);

print( i.getEndDate() ); // shows 25/12/2012
```

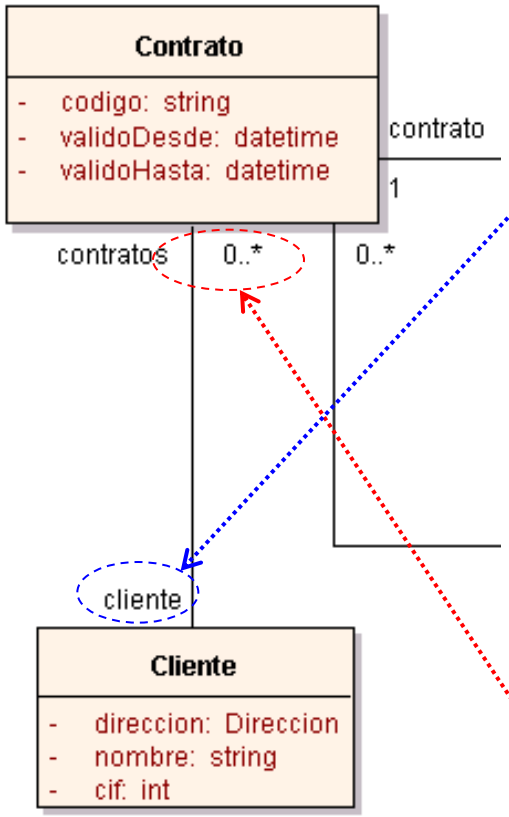
Posibilidades

- *Hacer tipos inmutables* — *Sin setters*
- *Devolver copias* — *Valores por constructor*

```
public Date getEndDate() {
    return endDate.clone();
}
```

```
public Set<Image> getImages() {
    return Collections.unmodifiableSet(images);
}
```

Implementación de asociaciones



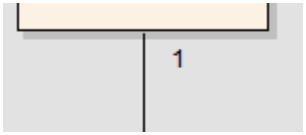

```java
public class Contrato{
    private Long id;
    private String codigo;
    private Date validoDesde;
    private Date validoHasta;

    private Cliente cliente;
    private Set<Equipo> equipos = new HashSet<Equipo>();
    private Set<Contacto> contactos = new HashSet<Contacto>();
```

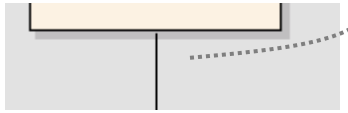
```java
public class Cliente implements Serializable{
    private Long id;
    private String cif;
    private String nombre;
    private Direccion direccion;

    private Set<Contrato> contratos = new HashSet<Contrato>();
```

Cardinalidades

- *Uno a uno*
- *Uno a muchos*
- *Muchos a muchos*

Extremos UNO

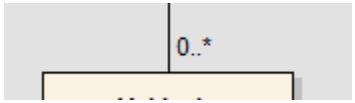


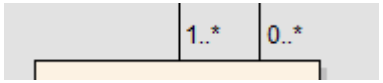


Si el diagrama no especifica cardinalidad se suele interpretar como UNO

```
public class Averia   {
      ...
      private Factura factura;
      ...
```

Extremos MUCHO





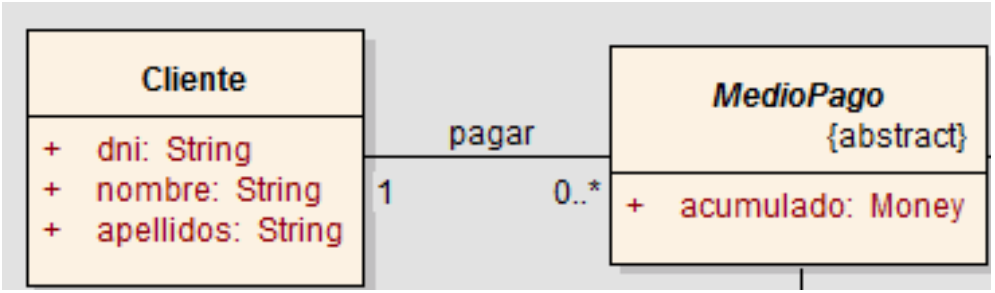
- *Set*
- *List*
- *Collection*

```
public class Factura {
      ...
      private Set<Averia> averias = new HashSet<Averia>();
      ...
```

Navegabilidad

- *Bidireccional*
- *Unidireccional*

Bidireccional




```
public class Cliente {
    ...
    private Set<MedioPago> mediodDePago = new HashSet<MedioPago>();
    ...
}
```

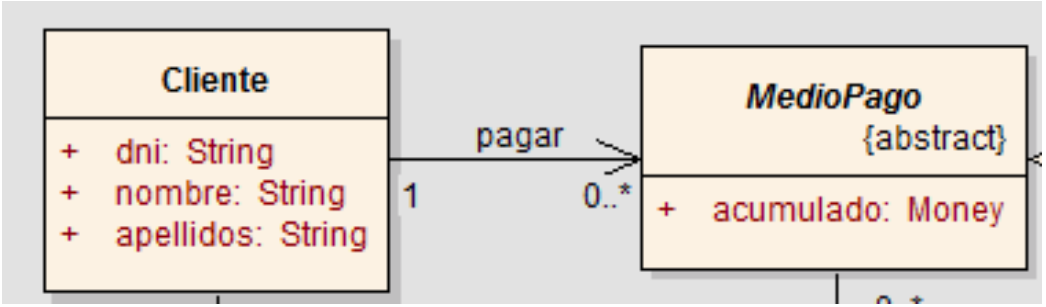
```
public abstract class MedioPago {
    ...
    private Cliente cliente;
    ...
}
```

Referencia cruzadas

Navegabilidad

- *Bidireccional*
- *Unidireccional*

Unidireccional




```
public class Cliente {
    ...
    private Set<MedioPago> mediodDePago = new HashSet<MedioPago>();
    ...
}
```

```
public abstract class MedioPago {
    ...
    // no tiene atributo cliente
    ...
}
```

Navegación de la asociación

- *Extremos UNO*
- *Extremos MUCHO*

Extremos UNO

```
public class Averia  {
    ...
    private Factura factura;
    ...
    public void setFactura(Factura factura) {
        this.factura = factura;
    }
    public Factura getFactura() {
        return factura;
    }
    ...
```

Extremos MUCHO

```
public class Factura {
    ...
    private Set<Averia> averias = new HashSet<Averia>();
    ...
    public Set<Averia> getAverias() {
        return averias;
    }
    // no necesita setter
    ...
```

Mantenimiento de la asociación

- *Unidireccional: sencillo*
- *Bidireccional: Fundamental mantener las referencias cruzadas*

```
Factura f = ...
Averia a = ...

// asociar
f.getAverias().add( a );
a.setFactura( f );

// desasociar
f.getAverias().remove( a );
a.setFactura( null );
```

Mantenimiento de la asociación

Fundamental mantener las referencias cruzadas

```
Factura f = ...
Averia a = ...

// asociar
f.getAverias().add( a );
a.setFactura( f );

// desasociar
f.getAverias().remove( a );
a.setFactura( null );
```

Código repetitivo

Rompe encapsulación

Práctica NO recomendada

Mantenimiento de la asociación

Alternativa mejor: ***añadir métodos de mantenimiento en uno de los dos extremos***

```
Factura f = ...
Averia a = ...

// asociar
f.addAveria( a );

// desasociar
f.removeAveria( a );
```

Mantenimiento de la asociación

Alternativa mejor: ***añadir métodos de mantenimiento en uno de los dos extremos***

```
Factura f = ...
Averia a = ...

// asociar
f.addAveria( a );

// desasociar
f.removeAveria( a );
```

```
public class Factura {
    ...
    protected Set<Averia> averias = new HashSet<Averia>();
    ...
    public void addAveria(Averia a) {
        a.setFactura( this );
        this.averias.add( a );
    }

    public void removeAveria(Averia a) {
        this.averias.remove( a );
        a.setFactura( null );
    }

    public Set<Averia> getAverias() {
        // devolver copia solo lectura del Set
        return Collections.unmodifiableSet( averias );
    }
```

¡El orden importa!

Fundamental mantener las referencias cruzadas

Mantenimiento de la asociación

Alternativa mejor: ***añadir métodos de mantenimiento en uno de los dos extremos***

```
Factura f = ...
Averia a = ...

// asociar
f.addAveria( a );

// desasociar
f.removeAveria( a );
```

```
public class Averia {
    ...
    private Factura factura;
    ...

    /*package*/ void _setFactura(Factura factura) {
        this.factura = factura;
    }

    public Factura getFactura() {
        return factura;
    }

    ...
}
```

Fundamental mantener las referencias cruzadas

Mantenimiento de la asociación

Alternativa recomendada: ***añadir métodos de mantenimiento estáticos en clase dedicada***

```
Factura f = ...
Averia a = ...

// asociar
Association.Facturar.link(a, f);

// desasociar
Association.Facturar.unlink(a, f);
```

Mantenimiento de la asociación

Alternativa recomendada: ***añadir métodos de mantenimiento estáticos en clase dedicada***

```
Factura f = ...
Averia a = ...

// asociar
Association.Facturar.link(a, f);

// desasociar
Association.Facturar.unlink(a, f);
```

```
public class Association {
    public static class Facturar {
        public static void link(Factura f, Averia a) {
            a._setFactura( f );
            f._getAverias().add( a );
        }

        public static void unlink(Factura f, Averia a) {
            f._getAverias().remove( a );
            a._setFactura( null );
        }
    }
}
```

¡El orden importa!

Fundamental mantener las referencias cruzadas

Mantenimiento de la asociación

Alternativa recomendada: ***añadir métodos de mantenimiento estáticos en clase dedicada***

```
Factura f = ...
Averia a = ...

// asociar
Association.Facturar.link(a, f);

// desasociar
Association.Facturar.unlink(a, f);
```

```
public class Factura {
    ...
    private Set<Averia> averias = new HashSet<>();
    ...

    /* package */ Set<Averia> _getAverias() {
        return averias;
    }

    public Set<Averia> getAverias() {
        return new HashSet<>( averias );
    }
}
```

Fundamental mantener las referencias cruzadas

Tipos de clases en modelos de dominio

- *Entidades*
- *Tipos valor: ValueTypes*

Inmutables

No se rompe la encapsulación al devolverlos en los getters

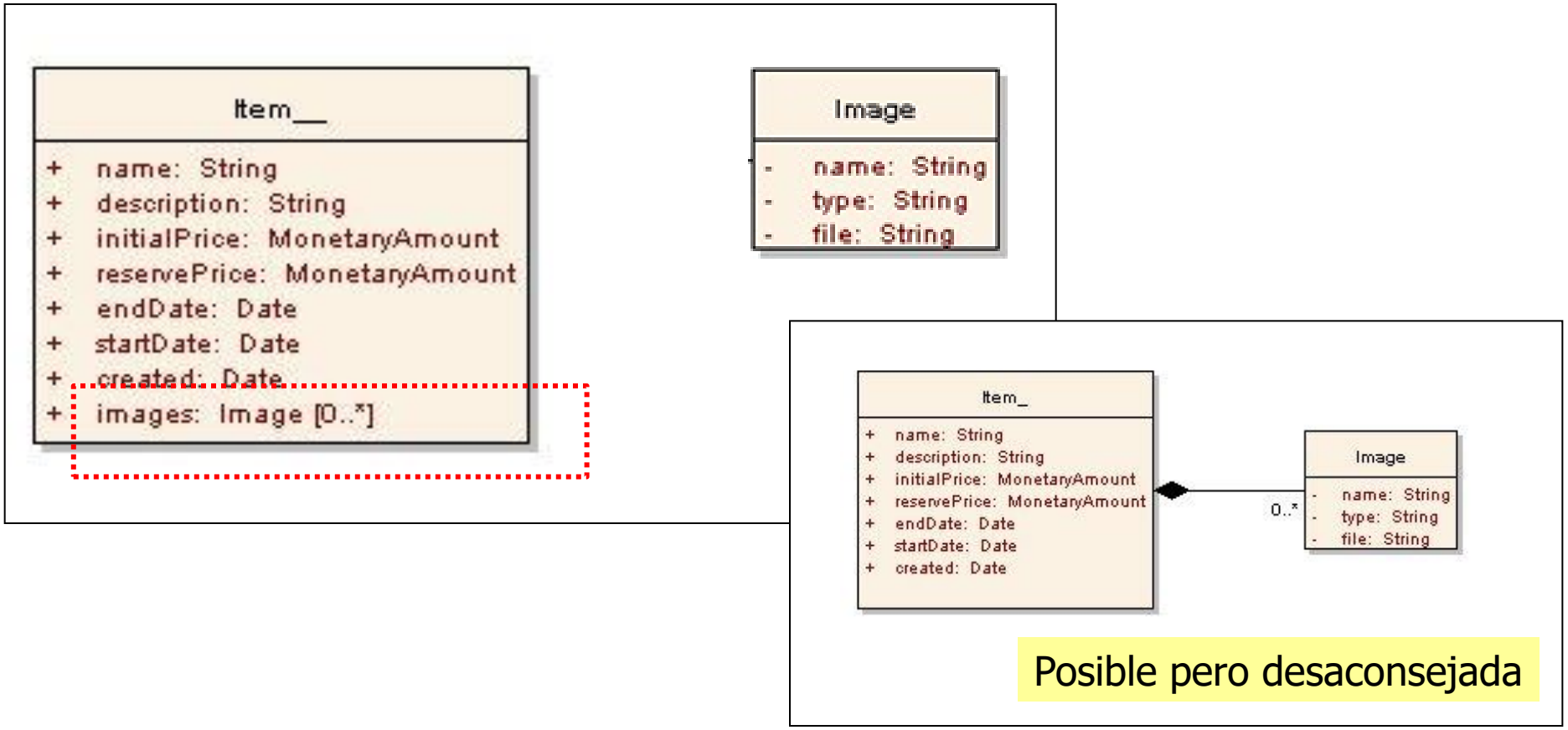
Entidades

- Un entidad representa la existencia de "algo" en el dominio (en la realidad) que tiene identidad propia
  - Sus propiedades pueden cambiar a lo largo del tiempo pero sigue siendo "ella"
  - Avería, Factura, Vehículo, Cliente…
- Puede estar asociada con otras entidades
- Su ciclo de vida es independiente de otras entidades
- Debe tener una **identidad** (lo que en BDD llamaríamos clave primaria)
  - Debe haber algún atributo (o combinación) que sirva de identificación

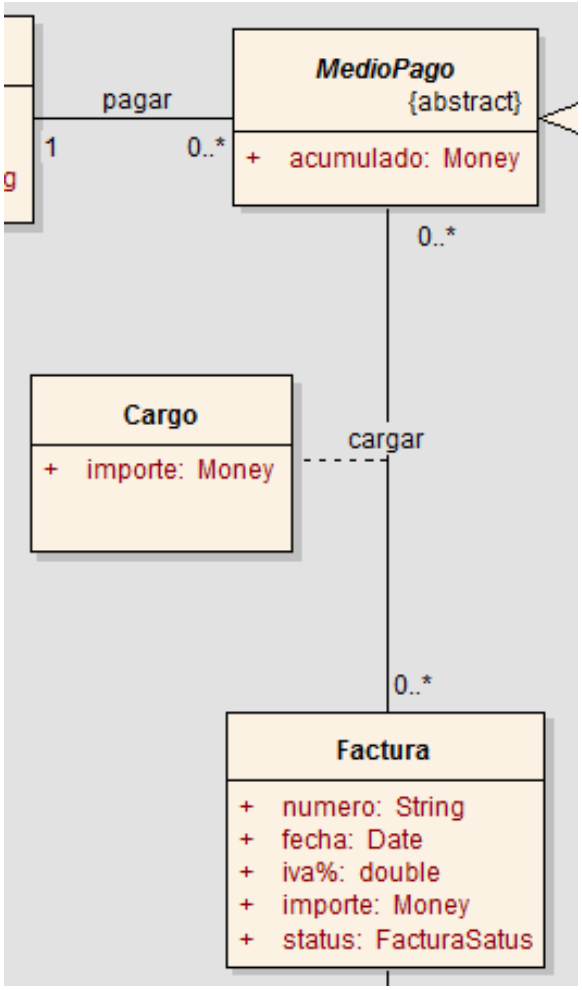
Tipos Valor

- Representan un valor, no tienen identidad
  - Una moneda de 2€ no importa si es ésta o aquella, sólo importa que es de 2€.
  - Su valor es inalterable
- Son atributos de una entidad
- Su ciclo de vida depende enteramente de la entidad que las posee
- Son inmutables → no tienen setters
  - Así son los tipos básicos Java: Integer, Double, String, etc., menos Date (?)

Representación en UML de Tipos Valor

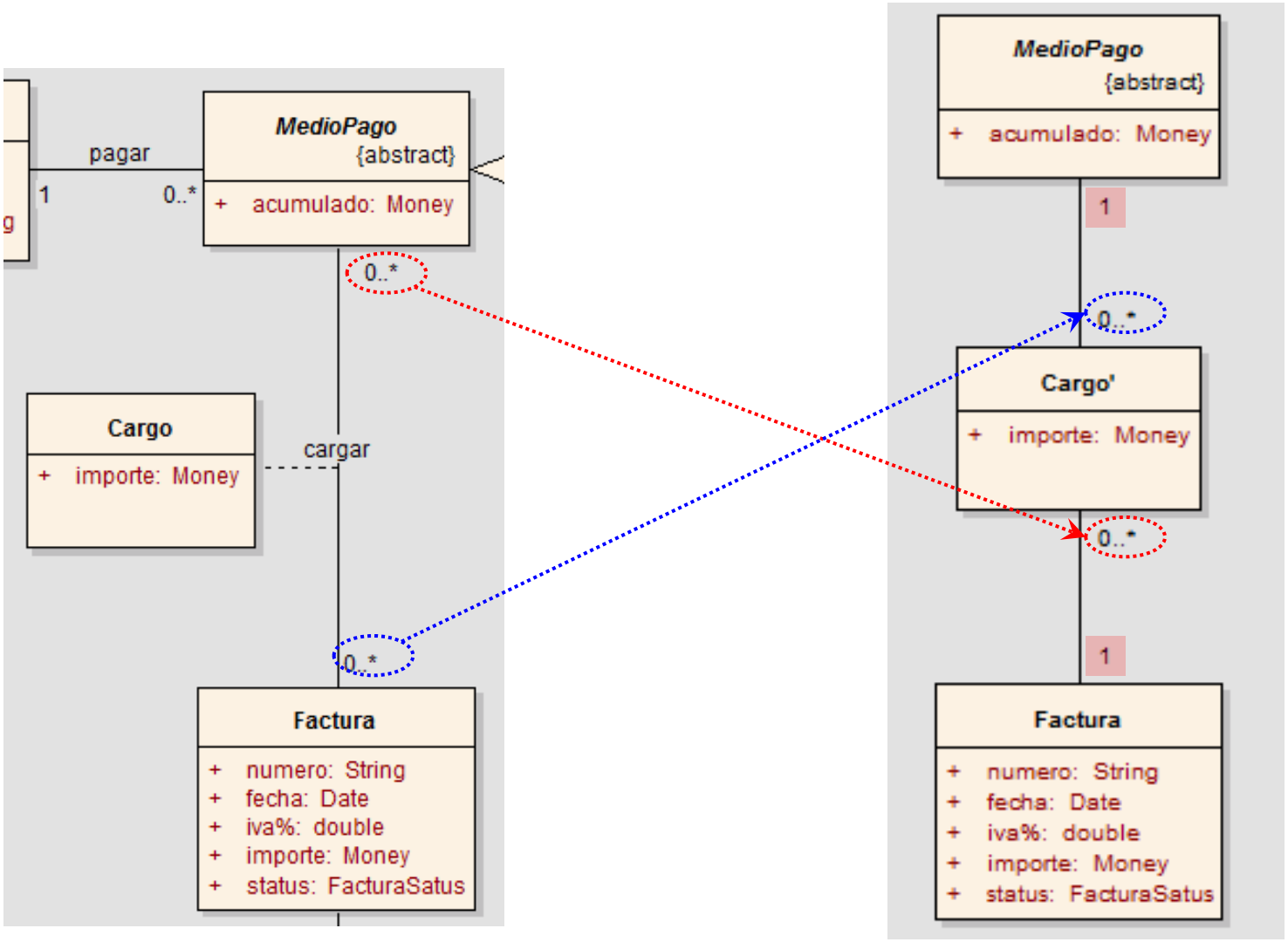


Implementación de clases asociativas

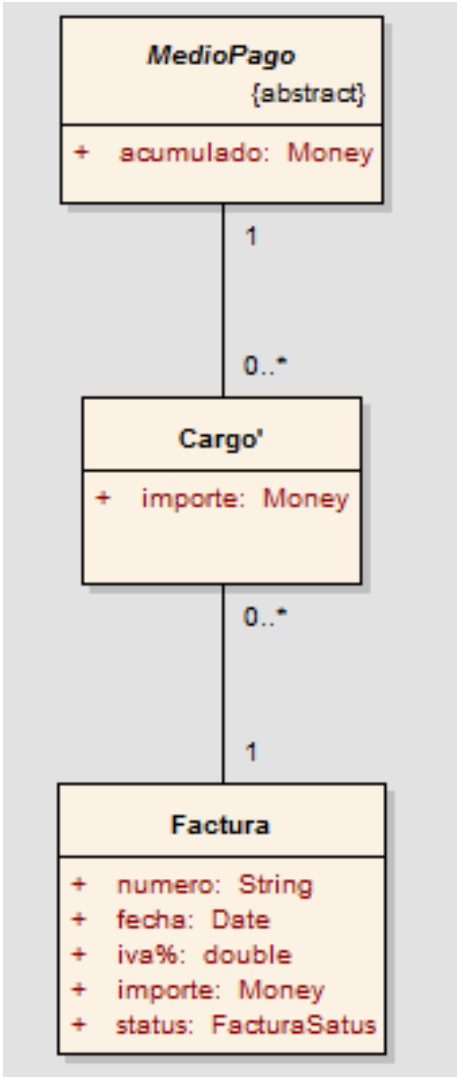


- Representan a la vez clase y asociación
- Permiten añadir atributos y funcionalidad a una asociación
- En java se implementan con una clase
- Cada instancia representa un enlace
- Mismas consideraciones de cardinalidad y navegabilidad
- Identidad compuesta por los dos extremos →dos objetos sólo pueden estar asociados una vez

Transformación previa



Paso a Java



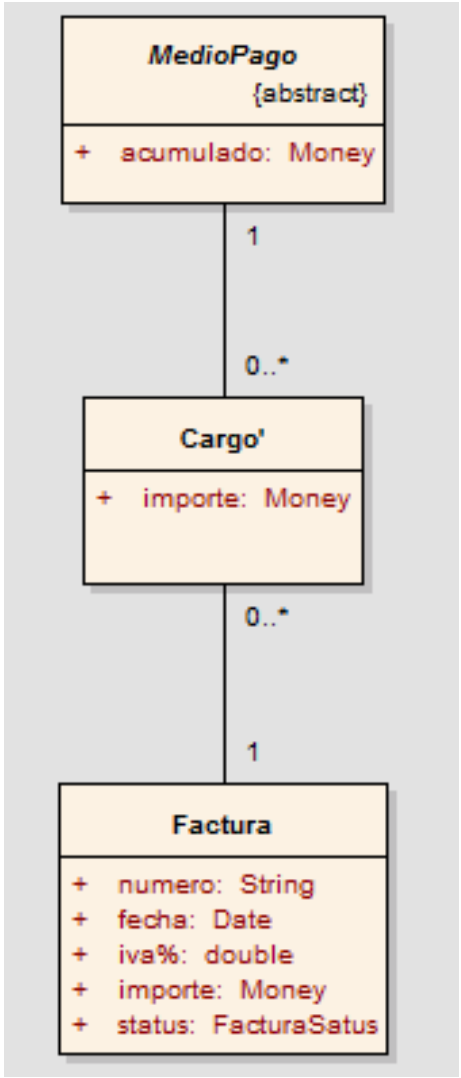

```
public class cargo {
    ...
    private Factura factura;
    private MedioPago medioPago;

    /* package */ void _setFactura(Factura f) {
        this.factura = f;
    }
    /* package */ void _setMedioPago(MedioPago mp) {
        this.medioPago = mp;
    }

    public Cargo(Factura f, MedioPago mp) {
        Association.Cargar.link(f, this, mp);
    }

}
```

Los dos ramales de la asociación deben estar sincronizados

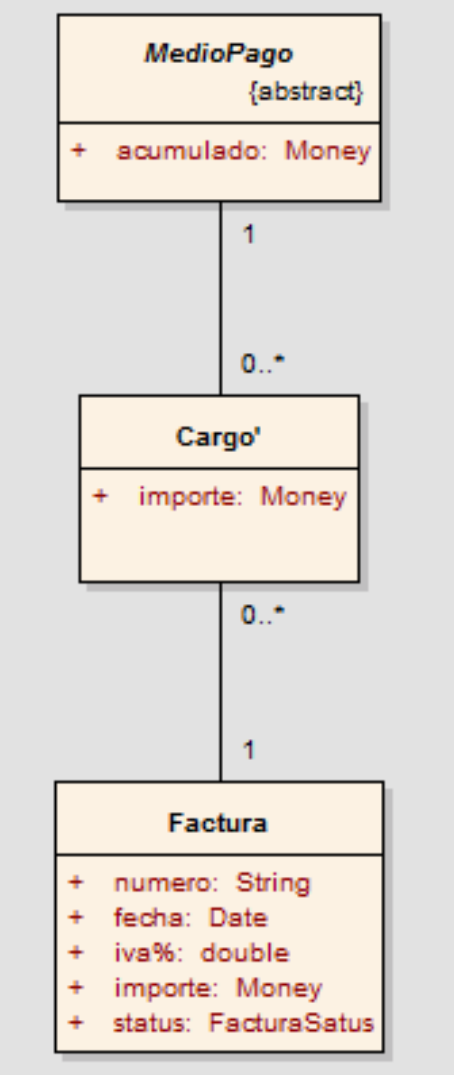
Paso a Java

```
public class Association {
    public static class Cargar {
        public static void link(Factura f, Cargo c, MedioPago mp) {
            c._setFactura( f );
            c._setMedioPago( mp );

            f._getCargos().add( c );
            mp._getCargos().add( c );
        }

        public static void unlink(Cargo c) {
            c.getFactura()._getCargos().remove( c );
            c.getMedioPago()._getCargos().remove( c );

            c._setFactura( null );
            c._setMedioPago( null );
        }
    }
}
```

Los dos ramales de la asociación deben estar sincronizados



Paso a Java




```
public abstract class MedioPago {
    ...
    private Set<Cargo> cargos = new HashSet<Cargo>();
    /* package */ Set<Cargo> _getCargos() {
        return cargos;
    }
    public Set<Cargo> getCargos() {
        return Collections.unmodifiableSet(cargos);
    }
    ...
}
```

_getCargos() Acceso paquete

Sólo accesible a la clase Cargo

getCargos() Acceso public

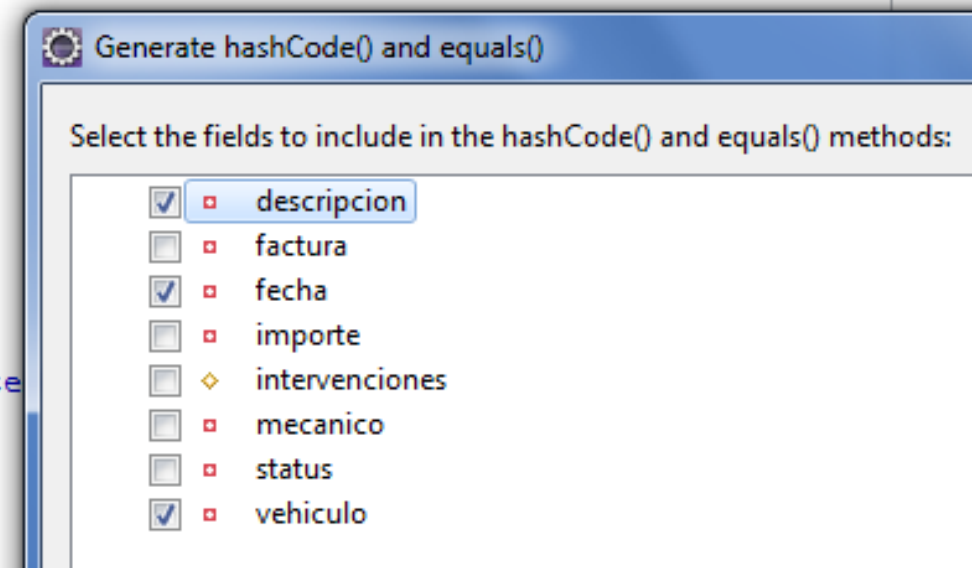
Accesible al resto de la aplicación

```
public class Factura  {
    ...
    private Set<Cargo> cargos = new HashSet<Cargo>();
    /* package */ Set<Cargo> _getCargos() {
        return cargos;
    }
    public Set<Cargo> getCargos() {
        return Collections.unmodifiableSet(cargos);
    }
    ...
}
```

equals() y hashCode()

- Entidades:
  - HashCode y equals redefinido SÓLO sobre los atributos que determinan identidad

```
public class Averia {
    private String descripcion;
    private Date fecha;
    private Double importe;
    private AveriaStatus status;
    private Factura factura;
    private Mecanico mecanico;
    private Vehiculo vehiculo;

    protected Set<Intervencion> inte
```



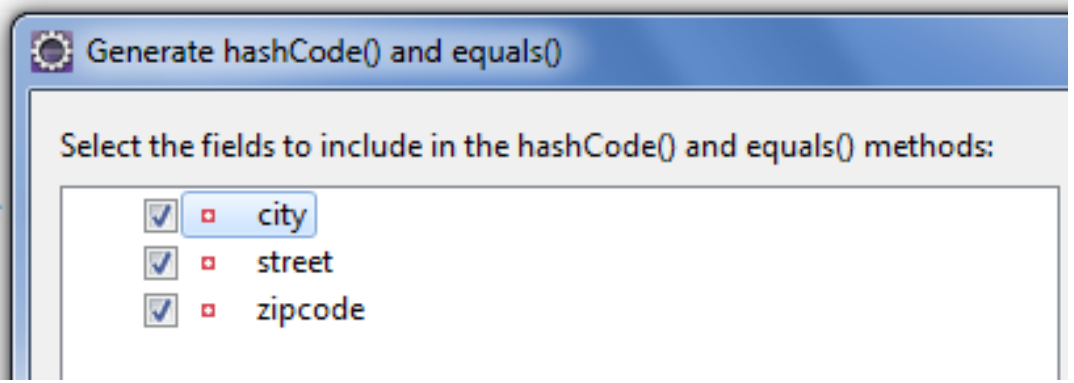
equals() y hashCode()

- Entidades:
  - Atributos que determinan identidad
  - ¿Cuáles?
    - Atributo (o combinación) que permite distinguir una entidad de las demás (un objeto de otro)
      - Cuantos menos mejor
    - Son atributos de sólo lectura
      - No cambiarán nunca
      - No llevan setters, sus valores se pasan en el constructor
    - Ya aparecen en el modelo del dominio
      - Si no los encuentras tu modelo de dominio está mal diseñado

equals() y hashCode()

- Tipos Valor:
  - HashCode y equals redefinido sobre TODOS los atributos

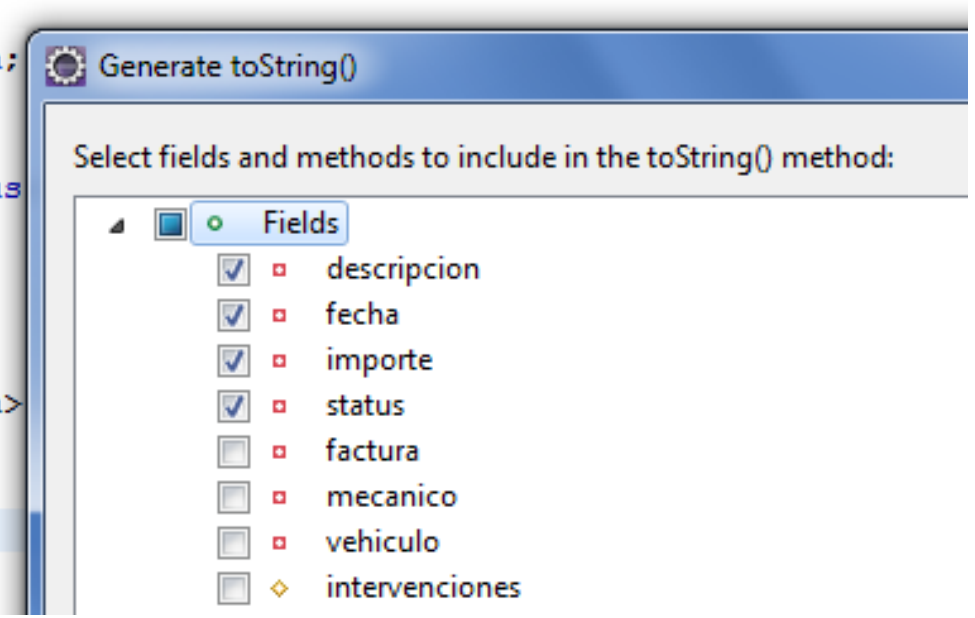
```
public class Address implements Serializable {
    private String street;
    private String zipcode;
    private String city;

    /**
     * No-arg constructor for
     */
    public Address() {
    }
```



toString()

- Redefinir toString()
  - Sin interés funcional pero útil para depuración
  - Generar automático con el IDE
  - Por defecto no incluir referencias a Entidades

```
public class Averia {
    private String descripcion;
    private Date fecha;
    private Double importe;
    private AveriaStatus status
    private Factura factura;
    private Mecanico mecanico;
    private Vehiculo vehiculo;

    protected Set<Intervencion>
```



Serializable

- Serializable
  - Habilita la clase para que pase por un Stream
    - A fichero, a otra máquina, etc
  - Necesario si hay capas remotas (app web)
  - Incluir número de serie para evitar problema de versionado