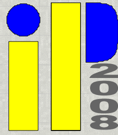


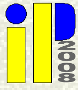
Ingeniería software
4º de Físicas

Diseño Detallado y Generación de Código

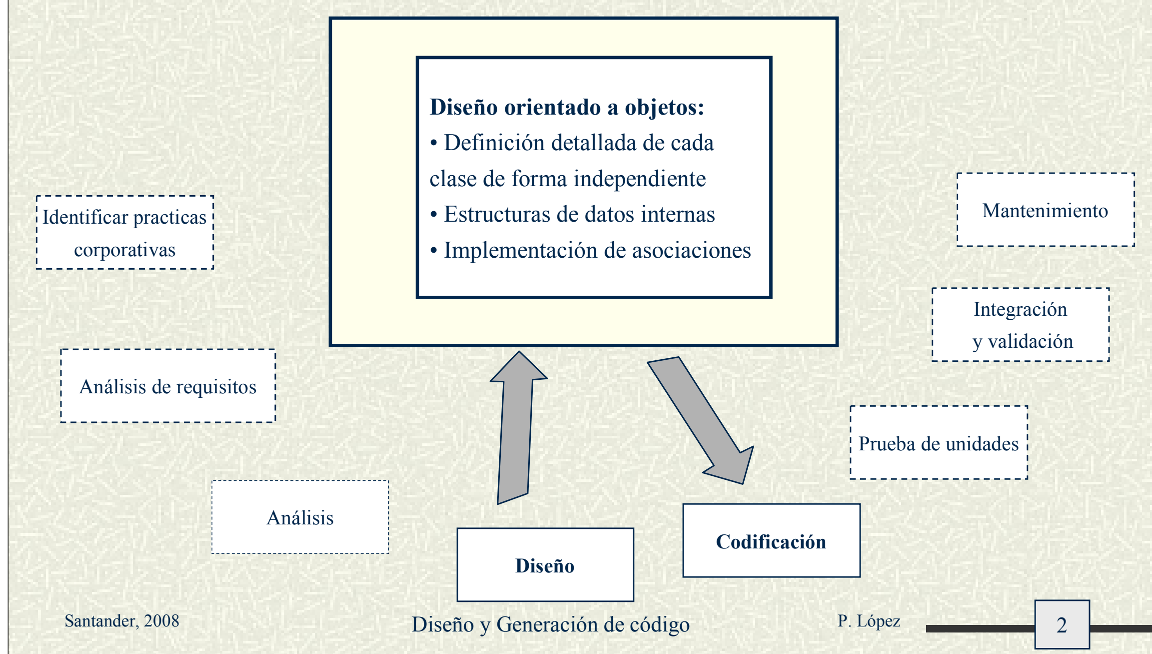
Patricia López
Computadores y Tiempo Real

Santander, 2008

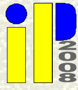
1



Objetivo de este tema



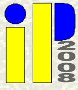
El objetivo primordial del diseño detallado es dejar el modelo preparado para su implementación. Los deben poder implementar el diseño detallado sin que surjan dudas esenciales, sólo aquellas referentes a aspectos del código.
Se optimiza el modelo de análisis eligiendo la arquitectura de diseño adecuada, seleccionando la tecnología y aplicándola al modelo de análisis. En el diseño se justifica y describe "el
como" las características establecidas en el análisis son implementadas. Corresponde a una vista mas detallada, en la que los detalles son relevantes en esta fase. Un diseño es una solución particular de las muchas posibles.
La fase de diseño detallado elabora los aspectos internos de los objetos y clases, y su ámbito se reduce a cada clase de forma independiente.
Los aspectos de la clases que son habitualmente tratados en esta fase son:
•La estructuras de datos internas de las fases.
•La implementación y optimización de los algoritmos y su descomposición cuando son complejos.
•La optimización de la máquina de estados de los objetos.
•La estrategias de elaboración de los objetos.
•Las estructuras con las que se elaboran las asociaciones, en particular cuando son múltiples y requieren mecanismos contenedores.
•La implementación de las visibilidades, en especial cuando son cruzadas y no son directamente implementables, así como los criterios de modularización y encapsulamiento de las clases.
Hay muchos criterios y guías para realizar el diseño de las clases, y suele corresponder con las estrategias de programación básica. No suelen basarse en patrones sino en bloques de sentencias proporcionadas por el lenguaje.
El resultado de esta fase son clases completamente definidas, en las que su estructuras de datos internas quedan perfectamente definidas, así como los diagramas de estados y actividades que especifican las operaciones y métodos de su interfaces.



Diseño orientado a objetos

- Objetivo de la fase de diseño: Preparar el modelo completo para su implementación
- El resultado son clases completamente definidas, en las que su estructuras de datos internas quedan perfectamente definidas, así como los diagramas de estados y actividades que especifican las operaciones y métodos de sus interfaces.
- El diseño orientado a objetos es un método de diseño del software de una aplicación basada en construirla con una estructura modular en la que los módulos software se corresponden con abstracciones de los objetos del problema
- Siguiendo un desarrollo orientado a objetos el mapeado es directo, los módulos software son directamente los identificados en la fase de análisis.

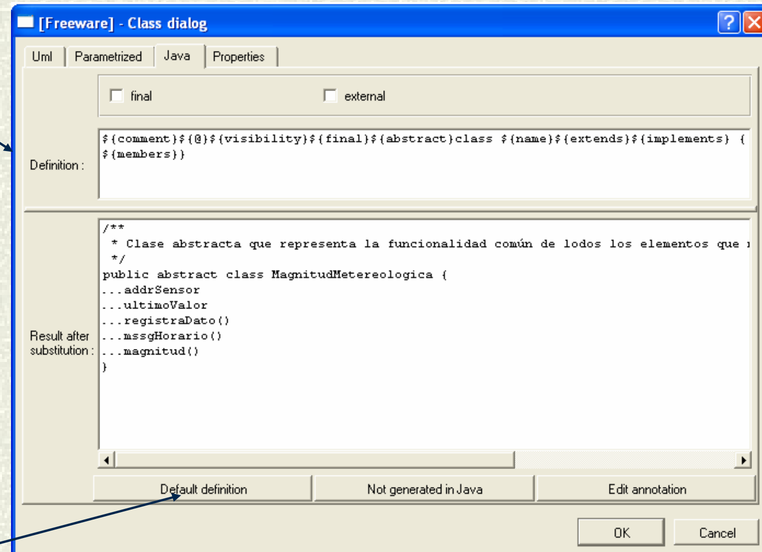
Diseño orientado a objetos: Es un método de diseño del software de una aplicación basada en construirla con una estructura modular en la que los módulos software se corresponden con abstracciones de los objetos del problema.
La fase de diseño detallado elabora los aspectos internos de los objetos y clases, y su ámbito se reduce a cada clase de forma independiente.
Los aspectos de la clases que son habitualmente tratados en esta fase son:
•La estructuras de datos internas de las fases.
•La implementación y optimización de los algoritmos y su descomposición cuando son complejos.
•La optimización de la máquina de estados de los objetos.
•La estrategias de elaboración de los objetos.
•Las estructuras con las que se elaboran las asociaciones, en particular cuando son múltiples y requieren mecanismos contenedores.
•La implementación de las visibilidades, en especial cuando son cruzadas y no son directamente implementables, así como los criterios de modularización y encapsulamiento de las clases.
•Estudio de los mecanismos de garantizar en tiempo de ejecución que las precondiciones son satisfechas y como responder cuando no se satisfacen.
Hay muchos criterios y guías para realizar el diseño de las clases, y suele corresponder con las estrategias de programación básica. No suelen basarse en patrones sino en bloques de sentencias proporcionadas por el lenguaje.
El resultado de esta fase son clases completamente definidas, en las que su estructuras de datos internas quedan perfectamente definidas, así como los diagramas de estados y actividades que especifican las operaciones y métodos de su interfaces.



Generación código Java en Bouml

Edit sobre una clase =>
Pestaña *Java*

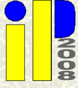
Patrón de generación



MagnitudMetereologica
addrSensor : int
ultimoValor : float
+ *registraDato() : void*
+ *mssgHorario() : String*
+ magnitud() : float

Resultado

Cuando se pulsa
“Default definition”,
se aplica el patrón de generación



Modificación del modo de generación de código

Proyecto => *Edit* => *Edit Generation Settings*

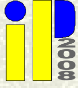
[Freeware] - GenerationSettings dialog

Types | Stereotypes | Java[1] | Java[2] | Java[3] | Java[4] | Description | Directory

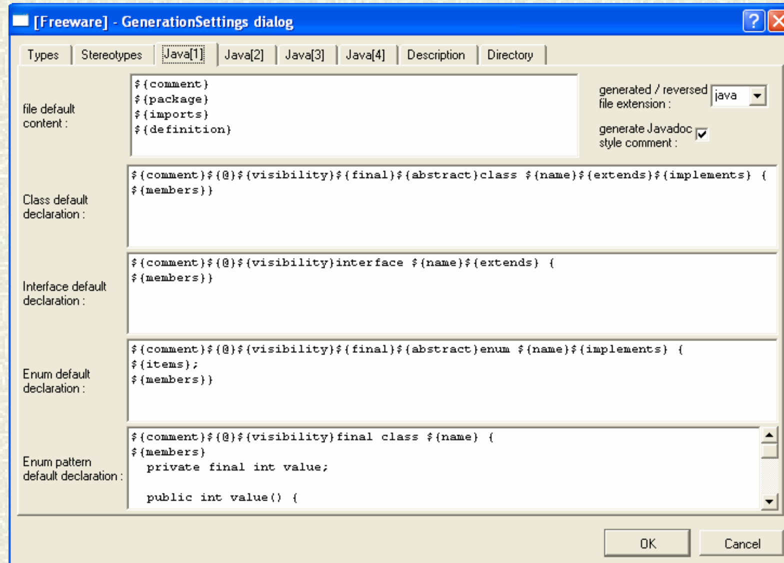
Types correspondence, and C++ operation argument default passing for them :

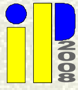
	Uml	C++	Java	Idl	C++ in	C++ out	C++ in ou	C++ retur	do
1	void	void	void	void	${type}	${type} &	${type}	${type}	
2	any	void *	Object	any	const ${ty	${type}	${type} &	${type}	
3	bool	bool	boolean	boolean	${type}	${type} &	${type} &	${type}	
4	char	char	char	char	${type}	${type} &	${type} &	${type}	
5	uchar	unsigned char	char	octet	${type}	${type} &	${type} &	${type}	
6	byte	unsigned char	byte	octet	${type}	${type} &	${type} &	${type}	
7	short	short	short	short	${type}	${type} &	${type} &	${type}	
8	ushort	unsigned short	short	unsigned short	${type}	${type} &	${type} &	${type}	
9	int	int	int	long	${type}	${type} &	${type} &	${type}	
10	uint	unsigned int	int	unsigned long	${type}	${type} &	${type} &	${type}	
11	long	long	long	long	${type}	${type} &	${type} &	${type}	
12	ulong	unsigned long	long	unsigned long	${type}	${type} &	${type} &	${type}	
13	float	float	float	float	${type}	${type} &	${type} &	${type}	
14	double	double	double	double	${type}	${type} &	${type} &	${type}	
15	string	string	String	string	${type}	${type} &	${type} &	${type}	
16					${type}	${type} &	${type} &	${type}	

OK | Cancel



Patrones de generación





Generación código Java de atributos

MagnitudMetereologica
- # addrSensor : int
- # ultimoValor : float
- + *registraDato() : void*
- + *mssgHorario() : String*
- + magnitud() : float

[Freeware] - Attribute dialog

Uml | Java | Properties

class : MagnitudMetereologica [Analisis::Subsistema EMA]

name : ultimoValor

stereotype :

type : float

multiplicity :

initial value : Editor

public / protected (selected) / private / package

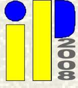
static attribute / volatile / read-only

description : Editor Default — Almacena el último valor medido de la magnitud meteorológica.

constraint : Editor

OK Cancel

Santander, 2008 Diseño y Generación de código P. López 7



Generación código Java de atributos

MagnitudMetereologica
- # addrSensor : int
- # ultimoValor : float
- + *registraDato() : void*
- + *mssgHorario() : String*
- + magnitud() : float

[Freeware] - Attribute dialog

Uml | Java | Properties

transient

Declaration :

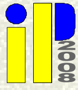
```
${comment}${@}${visibility}${static}${final}${transient}${volatile}${type} ${name}${value};
```

Result after substitution :

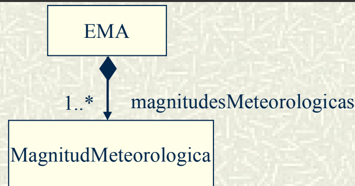
```
/**
 * Almacena el último valor medido de la magnitud meteorológica.
 */
protected float ultimoValor;
```

Default declaration | Not generated in Java | Edit annotation

OK | Cancel



Generación código Java de asociaciones múltiples



[Freeware] - Relation dialog

Uml | Java | Properties

name : <directional composition>

type : ↔ directional composition stereotype :

association :

in EMA [Analisis::Subsistema EMA]

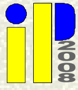
name : magnitudesMetereológicas

multiplicity : 1..* initial value : Editor

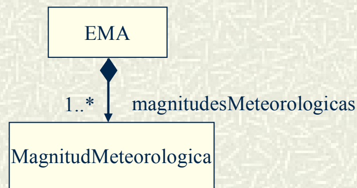
static relation volatile read-only public protected private package

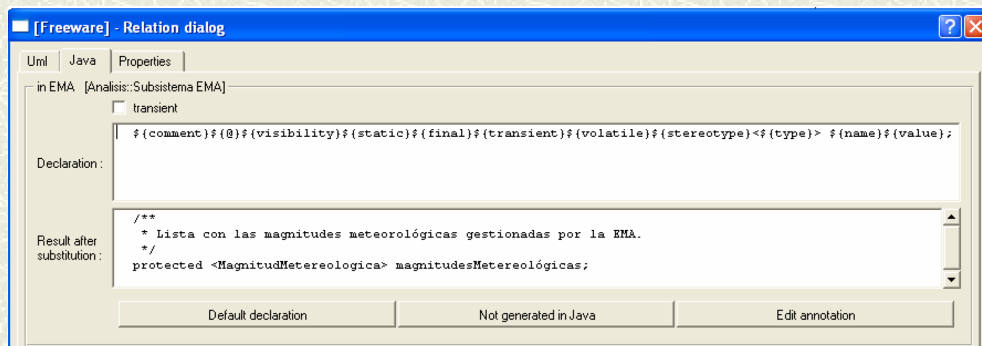
description : Lista con las magnitudes meteorológicas gestionadas por la EMA. Editor Default

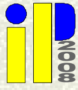
constraint : Editor



Generación código Java de asociaciones múltiples (incorrecto)

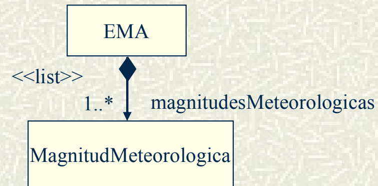


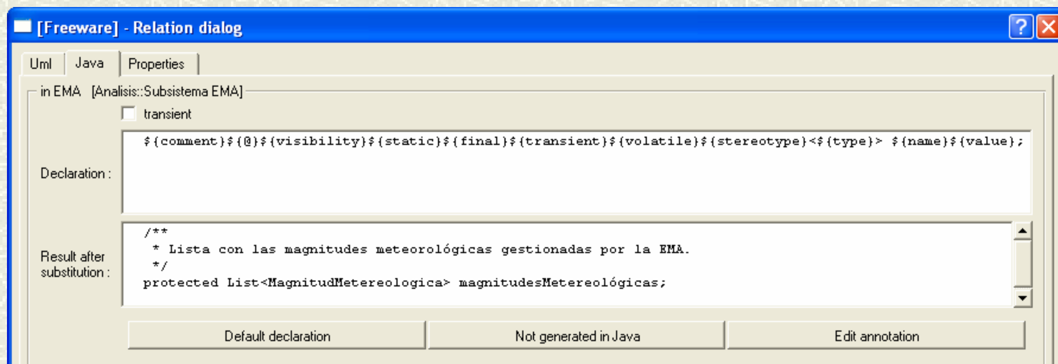




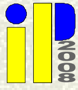
Generación código Java de asociaciones múltiples

- Hay que definir el tipo de asociación mediante un estereotipo. BOUML ofrece como posibilidades:
  - List : Mapeado a la interfaz List de Java
  - Set : Mapeado a la interfaz Set de Java
  - Map : Mapeado a la interfaz Map de Java



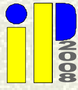


Santander, 2008 Diseño y Generación de código P. López 11



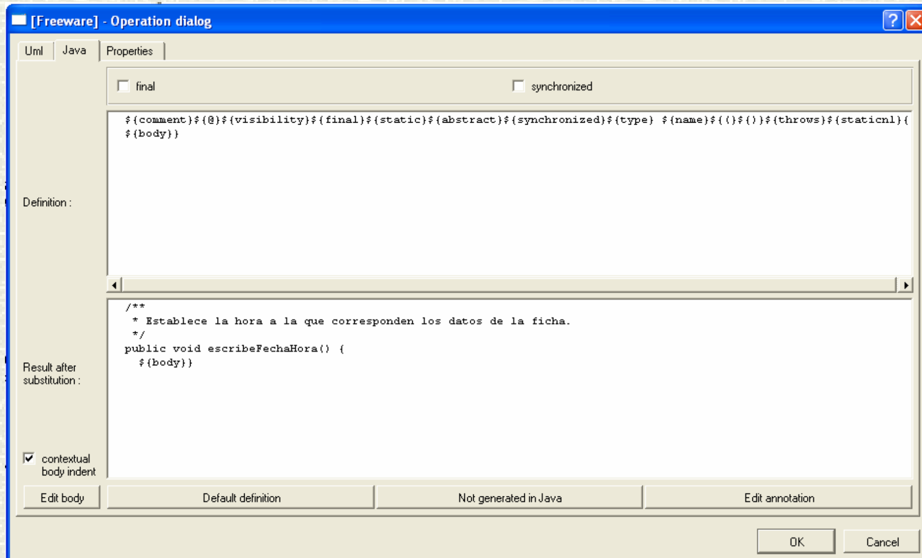
Generación código Java de operaciones

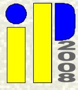
Definir siempre el valor de retorno, incluso cuando sea void (excepto en constructores)



Generación código Java de operaciones (incorrecto)

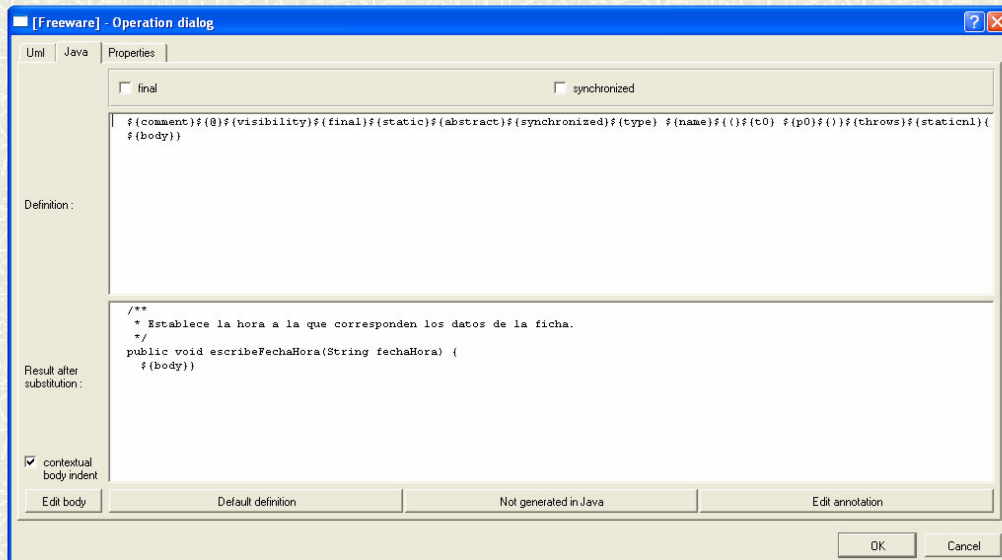
La implementación obtenida inicialmente no muestra los argumentos de entrada

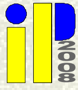




Generación código Java de operaciones

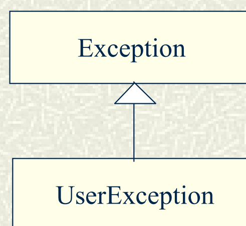
Pulsando Default Definition genera los parámetros correspondientes

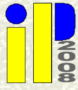




Generación de código de excepciones

- En Java una excepción extiende siempre a la clase Exception
- Para que BOUML genere el código de forma adecuada, hacemos que todas las excepciones de usuario extiendan a una clase Exception (para la cual no generaremos código).

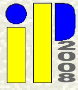




Generación cuerpo de las operaciones

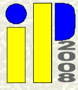
- Podemos incluso generar el código que queremos que aparezca en el cuerpo de las operaciones:
  - Para ello tenemos que asignar a la variable de entorno BOUML_EDITOR un editor cualquiera (bloc de notas, p.e.)
  - Pinchando Edit Body en la pestaña anterior, nos abre una ventana del editor elegido y editamos el cuerpo del procedimiento (corresponderá al identificador $body del patrón de generación)

- Podremos editar directamente el texto :
  - En métodos muy sencillos
  - Para que Java no de errores de compilación porque no se devuelven los tipos de datos que se requieren.



Aspectos a tener en cuenta en diseño detallado en Bouml

- Definir los estereotipos para las agregaciones múltiples
- Definir todos los parámetros de una operación, incluidos los que devuelven “void”
- Para cada operación, pulsar “Default Definition” en la pestaña Java (para que tenga en cuenta los parámetros)
- Excepciones de usuario extendiendo a Exception



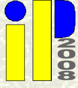
Generación de código

- Para generar el código, hay que generar un artifact (.java) para cada clase.
  - Generamos una vista de desarrollo: *New Deployment View*
  - La asociamos a la vista de clases: *Botón derecho sobre la vista de clases => Edit => deployment view*
  - Creamos un artefacto (.java) para cada clase : *Botón derecho en la clase => Create source artifact*

  Si no hubiesemos asociado la vista de despliegue a la de clases no nos ofrecería esa opción. Al crear los artifacts, aparecen en la vista de despliegue

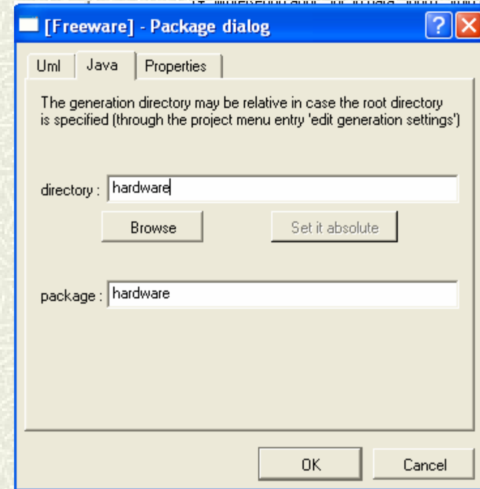
- Elegimos el directorio raíz en que se van a generar las clases: *Botón derecho en el proyecto => Edit generation settings => Directory*

Santander, 2008 Diseño y Generación de código P. López 18



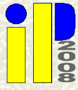
Generación de código (continuación)

- Si queremos generar el código por paquetes
  - Edit en el paquete donde esté el deployment view => Java
  - Directory: Directorio donde se generarán las clases definidas en ese paquete. Si se ha especificado un directorio raíz, es relativo a él.
  - Package: Nombre del paquete Java (el que aparece en la cabecera de la clase).



- Generamos el código : *Botón derecho en el proyecto => Generate => Java*

  Aparece una ventana que informa de los errores y warnings encontrados



Dependencias externas

- Para incluir dependencias de paquetes externos (java.util, java.io, etc), añadimos el correspondiente import directamente en el artifact.
  - Botón derecho sobre el artifact => Edit => Java Source y lo editamos directamente debajo del campo ${imports}