TEMA 5. Modelado y Verificación Formal
Parte 2 de 2

Calidad del Software

Dr. José Luis Abellán Miguel

Grado en Ingeniería Informática

Dr. José Luis Abellán Miguel - Tlf: (+34) 968 278 657
Universidad Católica San Antonio de Murcia www.ucam.edu

Índice

- Introducción
- Estrategia de cuarto limpio
- Especificación funcional
- Diseño de cuarto limpio
- Conceptos de métodos formales
- Lenguajes de especificación formal

Bibliografía

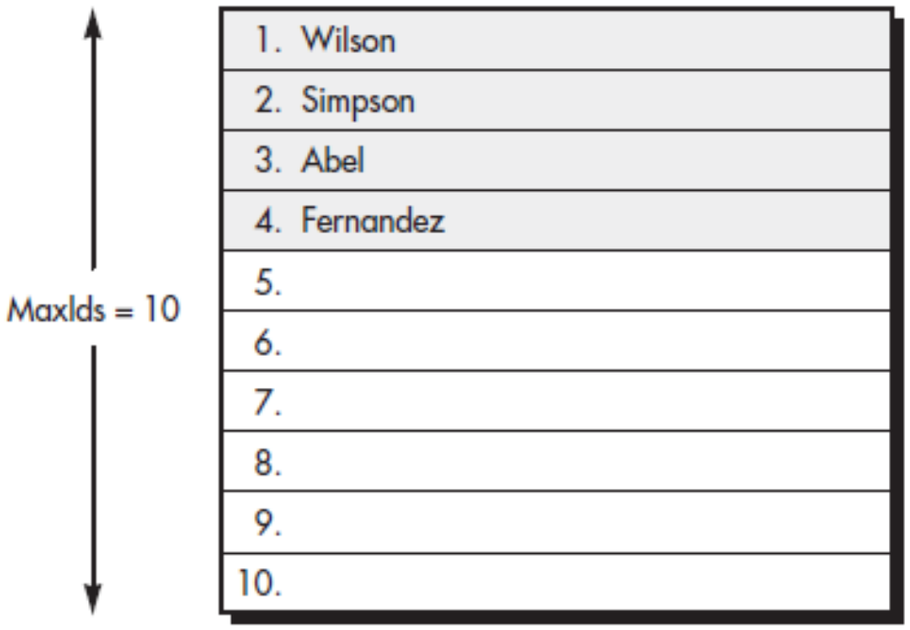
- Pressman, R. ***Ingeniería del Software: Un enfoque práctico***. 7ª edición. Madrid: McGraw Hill, 2010. ISBN: 9701054733 (disponible en la biblioteca UCAM) → **Capítulo 21**

Conceptos de Métodos Formales (1/3)

- Definición [Encyclopedia of Software Engineering'01]:
  - "*Los métodos formales son técnicas con base matemática para describir las propiedades del sistema. Tales métodos formales proporcionan marcos conceptuales dentro de los cuales las personas pueden especificar, desarrollar y verificar los sistemas en forma sistemática*"
- Los métodos de especificación permiten que los requerimientos o el diseño se interpreten solo en una forma, lo que elimina la ambigüedad de interpretación.
  - Las facilidades descriptivas de la teoría de conjuntos y la notación lógica permiten un enunciado claro de los requerimientos.

Conceptos de Métodos Formales (2/3)

Ejemplo1: Tabla simbólica

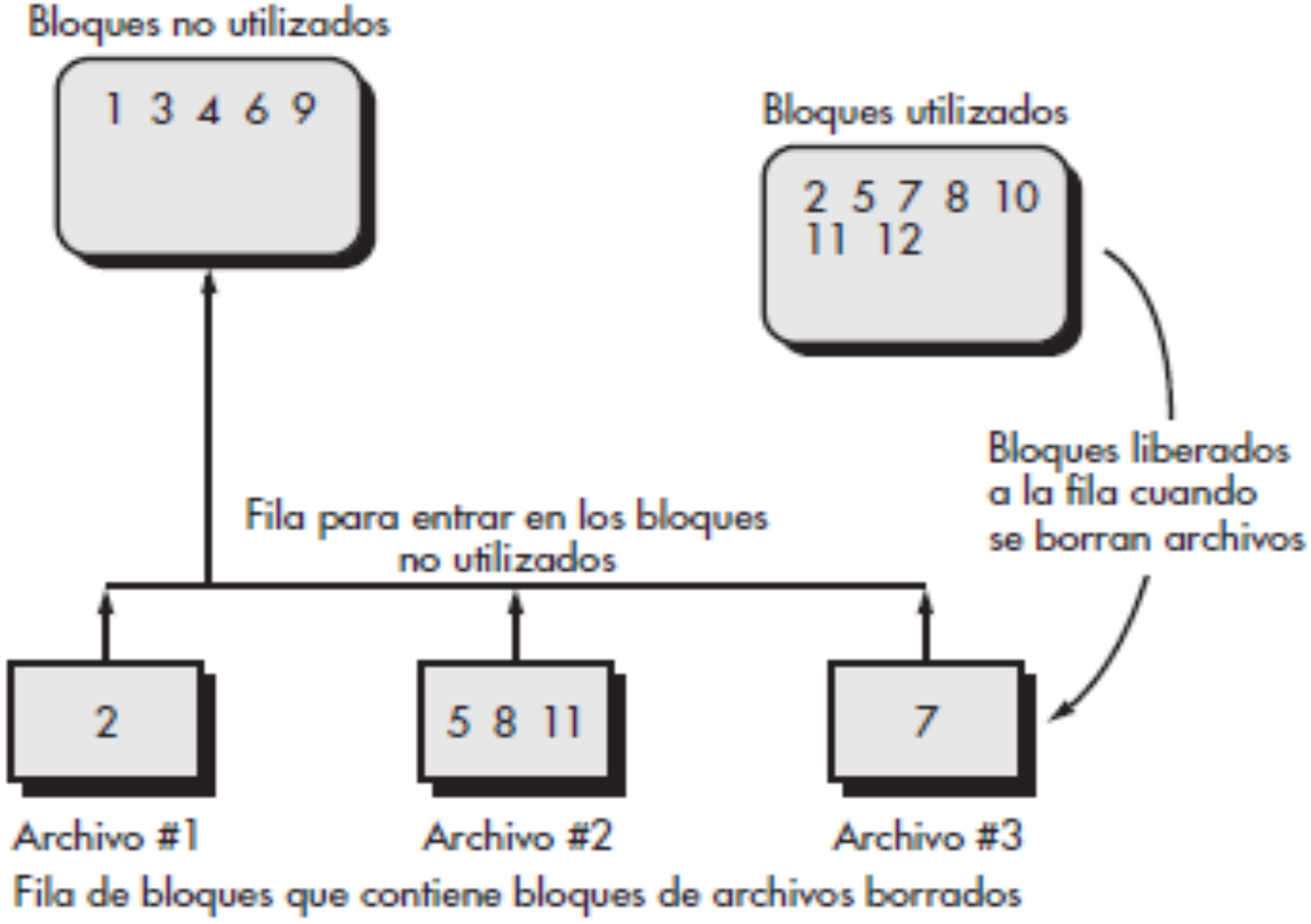


- Un programa se usa para mantener una tabla simbólica
- Restricciones:
  - Restr1: Colección de ítems sin duplicación
  - Restr2: No se pueden almacenar más de MaxIds nombres

- **Estado**: datos almacenados en el sistema (tabla simbólica)
- **Invariante de datos**: lo que no cambia (Restr2).
- **Operaciones**: leer o escribir datos (add() y remove())
  - **Precondición**: circunstancias en las que es válida una operación (cuando se cumplen Restr1 & Restr2)
  - **Postcondición**: lo que se garantiza que es verdadero hasta completar una operación (para add() la tabla aumentará con el nuevo identificador)

Conceptos de Métodos Formales (2/3)

Ejemplo2: Manipulador de bloques (1/3)



Conceptos de Métodos Formales (2/3)

Ejemplo2: Manipulador de bloques (2/3)

- **Estado**: colección de bloques libres, la colección de bloques usados y la fila de bloques reciclados
- **Invariante de datos**:
  - Ningún bloque se marcará como no utilizado y usado al mismo tiempo
  - Todos los conjuntos de bloques que se conservan en la fila serán subconjuntos de la colección de los bloques actualmente utilizados
  - Ningún elemento de la fila contendrá el mismo numero de bloque
  - La colección de bloques utilizados y bloques que no se usan será la colección total de bloques que constituyen los archivos

Conceptos de Métodos Formales (2/3)

Ejemplo2: Manipulador de bloques (3/3)

- **Operaciones**: *add(bloques), remove(bloques) y check()(Comprueba si la fila de bloques esta vacía)*
  - **Precondición**: *add* (los bloques que se van a agregar deben estar en la colección de bloques usados); *remove* (la fila debe tener al menos un item); *check* (NULL).
  - **Postcondición**: *add* (la coleccion de bloques ahora se encuentra al final de la fila); *remove* (los bloques deben agregarse a la colección de bloques no utilizados); *check* (entrega el valor *true* si la fila esta vacía y false en otro caso).

Conceptos de Métodos Formales (3/3)

♦ Ejercicio: en grupos de dos/tres alumnos especificar un marco conceptual basado en metodología formal para una función que describa una operación sobre una estructura de datos que el grupo considere.

- **Definir el estado, invariante de datos**, al menos una **operación** con su **precondición y postcondición**

Lenguajes de Especificación Formal (1/3)

- Se componen de tres componentes principales:
  - Sintaxis que define la notación especifica con la que se representa la especificación
    - Notación estándar de la teoría de conjuntos y cálculo de predicados
  - Semántica para ayudar a definir un "universo de objetos" que se usaran para describir el sistema
    - Cómo representa el lenguaje los requerimientos del sistema
  - Un conjunto de relaciones que definen las reglas que indican qué objetos satisfacen adecuadamente la especificación

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (1/13)

- Notación formal desarrollada de modo que los usuarios de UML puedan agregar mas precisión a sus especificaciones.
- Diagramas UML: clase, estado o actividad
  - Se agregan expresiones OCL y hechos de estado acerca de elementos de los diagramas:
    - **Invariantes**: Ej., El propietario de un coche ha de ser mayor de edad
    - **Pre/Postcondiciones**: Ej., La operación se ejecuta para un conjunto no vacío. El resultado de la operación añade un elemento al conjunto
  - Dichas expresiones se llaman **restricciones**; cualquier implementación derivada del modelo debe asegurar que cada una de las restricciones siempre sigue siendo verdadera.

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (2/13)

- Notación OCL:

x.y	Obtiene la propiedad y del objeto x. Una propiedad puede ser un atributo, el conjunto de objetos al final de una asociación, el resultado de evaluar una operación y otras cosas, dependiendo del tipo de diagrama UML. Si x es un Conjunto, entonces y se aplica a cada elemento de x; los resultados se recopilan en un nuevo Conjunto.
c->f()	Aplica la operación f interna de OCL a la Colección c en sí (en oposición a cada uno de los objetos en c). A continuación se mencionan ejemplos de operaciones internas.
and, or, =, <>	And lógica, or lógica, igual, no es igual.
p implica q	Verdadero si q es verdadero o p es falso.

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (3/13)

- Notación OCL:

Muestra de operaciones sobre colecciones (incluidos conjuntos y secuencias)

C−>size()	El número de elementos en la Colección c.
C−>isEmpty()	Verdadero si c no tiene elementos, falso de otro modo.
c1−>includesAll(c2)	Verdadero si cada elemento de c2 se encuentra en c1.
c1−>excludesAll(c2)	Verdadero si ningún elemento de c2 se encuentra en c1.
C−>forAll(elem | boolexpr)	Verdadero si boolexpr es verdadera cuando se aplica a cada elemento de c. Conforme se evalúa un elemento, se enlaza a la variable elem, que puede usarse en boolexpr. Esto implementa cuantificación universal, que se estudió anteriormente.
C−>forAll(elem1, elem2 | boolexpr)	Igual que el anterior, excepto que boolexpr se evalúa para cada posible *par* de elementos tomados de c, incluidos casos donde el par tiene el mismo elemento.
C−>isUnique(elem | expr)	Verdadero si expr evalúa un valor diferente cuando se aplica a cada elemento de c.

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (4/13)

- Notación OCL:

Muestra de operaciones específicas para conjuntos

s1–>intersection(s2)	El conjunto de aquellos elementos que se encuentran en s1 y también en s2.
s1–>union(s2)	El conjunto de aquellos elementos que se encuentran en s1 o en s2.
s1–>excluding(x)	El conjunto s1 con la omisión del objeto x.

Muestra de operación específica a secuencias

Seq–>first()	El objeto que es el primer elemento en la secuencia seq.

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (5/13)

- Una expresión OCL involucra operadores que operan sobre objetos
- Los objetos pueden ser instancias de la clase **Collection** OCL:
  - **Set** (conjunto) y **Sequence** (secuencia) son dos subclases
- El resultado de una expresión completa siempre debe ser booleana (V o F)
- El objeto **self** es el elemento del diagrama UML en cuyo contexto se evaluara la expresión OCL

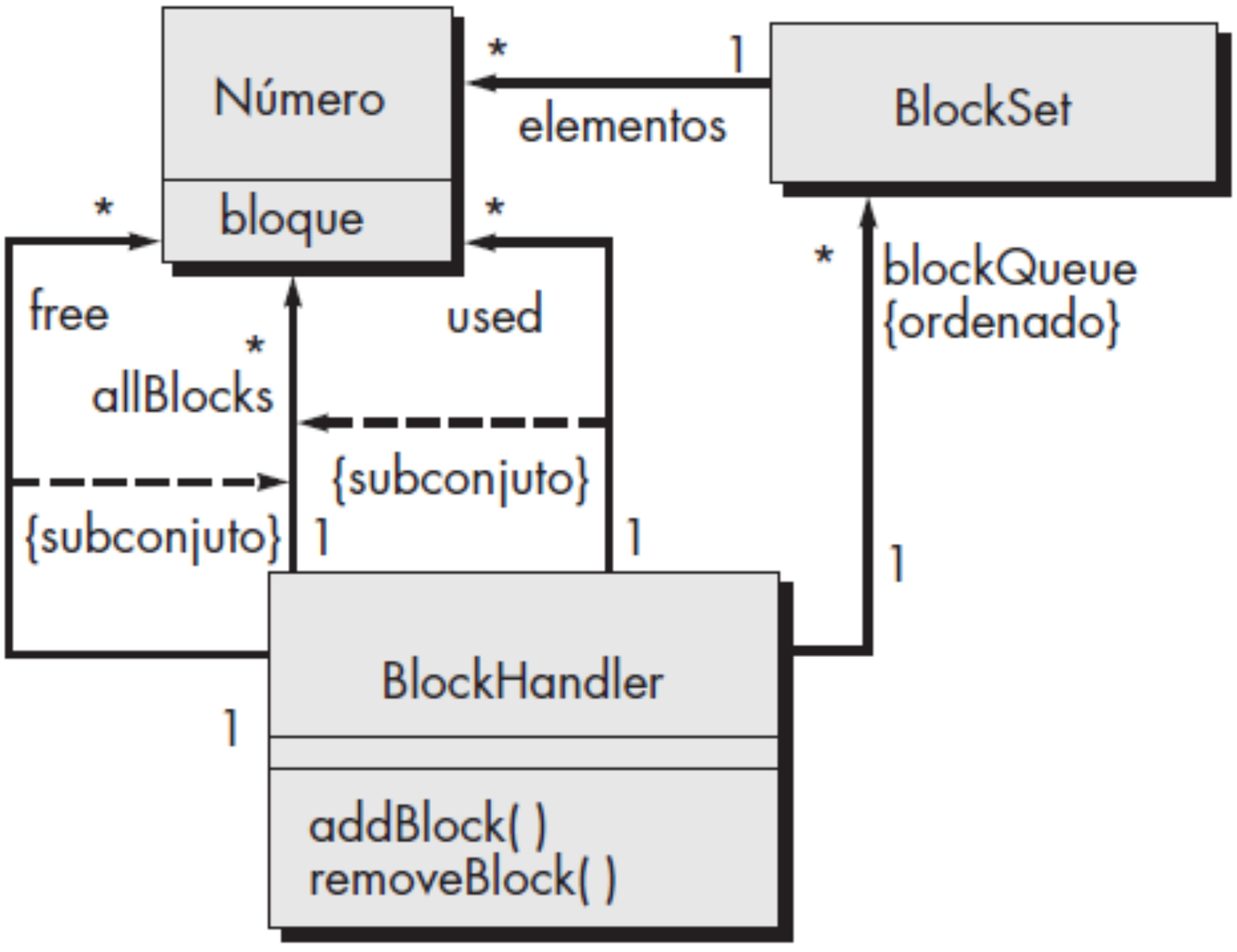
Lenguajes de Especificación Formal (2/3)
Lenguaje de restricción de objeto (OCL) (6/13)

- El símbolo **.** del objeto **self** permite obtener otros objetos:
  - Si **self** es la clase **C**, con atributo **a**, entonces **self.a** evalúa al objeto almacenado en **a**
  - Si **C** tiene una asociacion uno a muchos llamada *assoc* con otra clase **D**, entonces **self.assoc** evalúa un **Set** cuyos elementos son del tipo **D**
  - Si **D** tiene el atributo **b**, entonces la expresión **self.assoc.b** evalúa al conjunto de todos los **b** que pertenecen a todos los **D**

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (7/13)

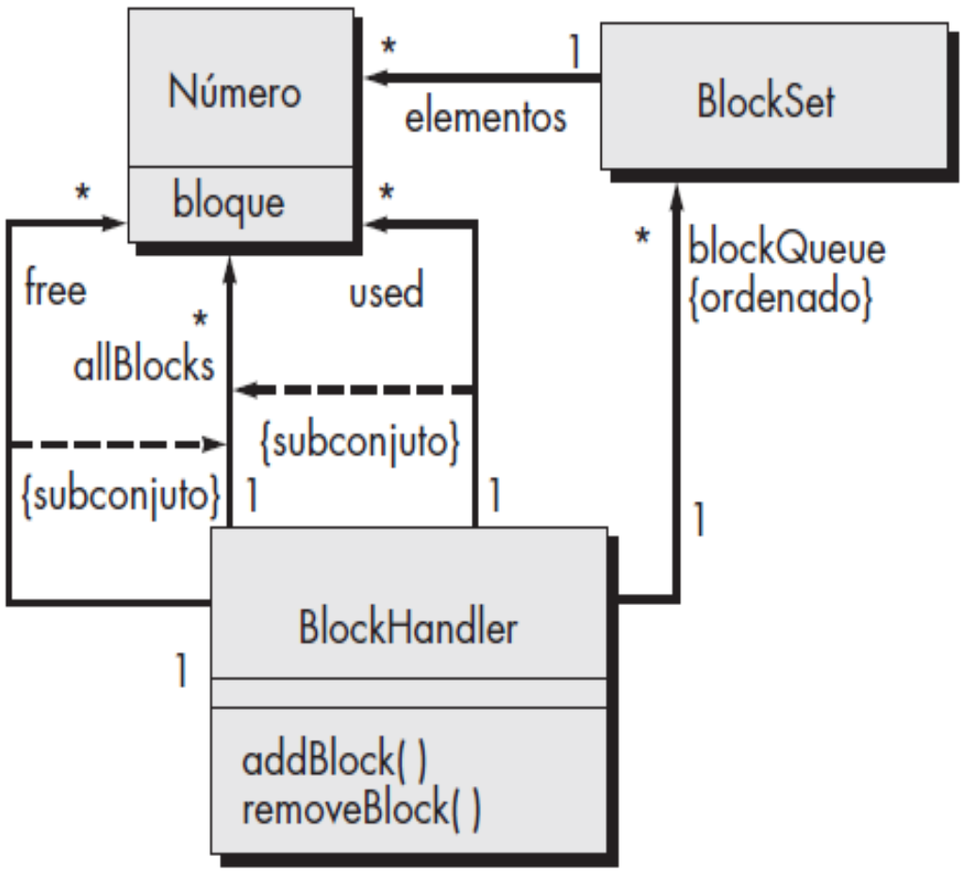
- Ej: Diagrama de Clase para Manipulador de Bloques



Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (8/13)

- Ej: Expresiones OCL de las restricciones



Ningún bloque se marcará como no utilizado y usado al mismo tiempo

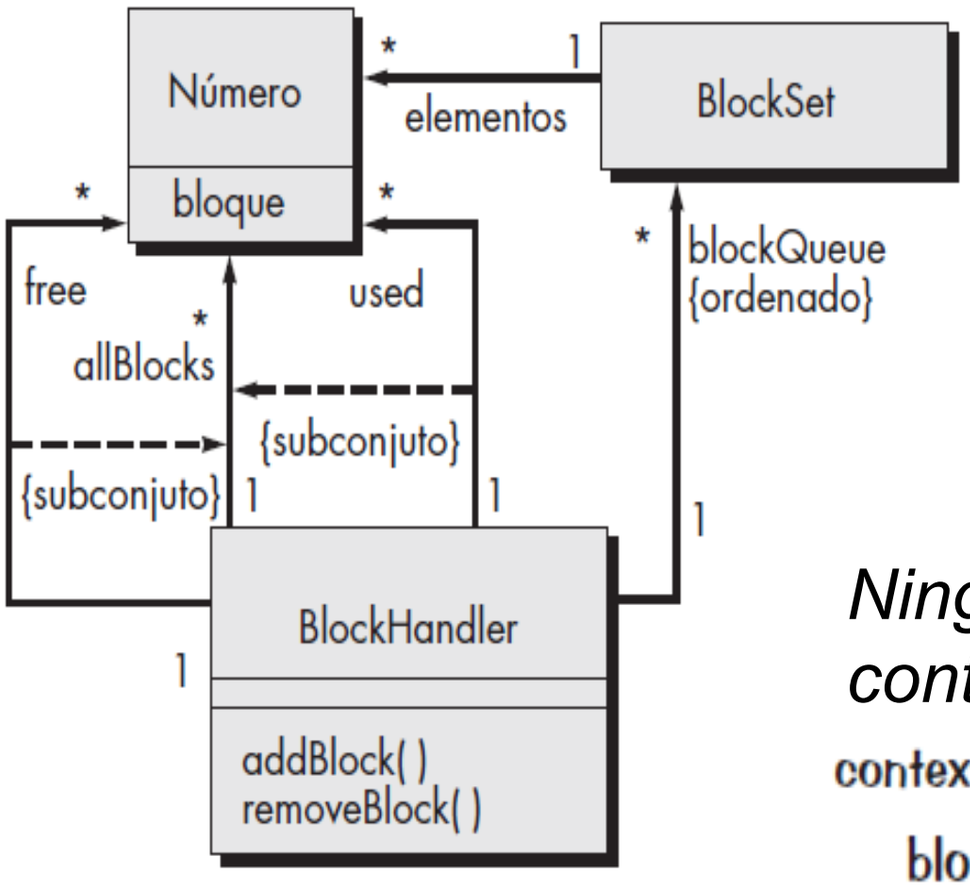
```
context BlockHandler inv:
    (self.used->intersection(self.free)) ->isEmpty()
```

- Palabra clave **context**
  - Elemento del diagrama UML que restringe la expresión
- Palabra clave **self**
  - Refiere a la instancia de BlockHandler
  - Se puede omitir

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (9/13)

- Ej: Expresiones OCL de las restricciones



Todos los conjuntos de bloques que se conservan en la cola (blockQueue) serán subconjuntos de la colección de los bloques actualmente utilizados

```
context BlockHandler inv:
    blockQueue->forAll(aBlockSet | used->includesAll(aBlockSet ))
```

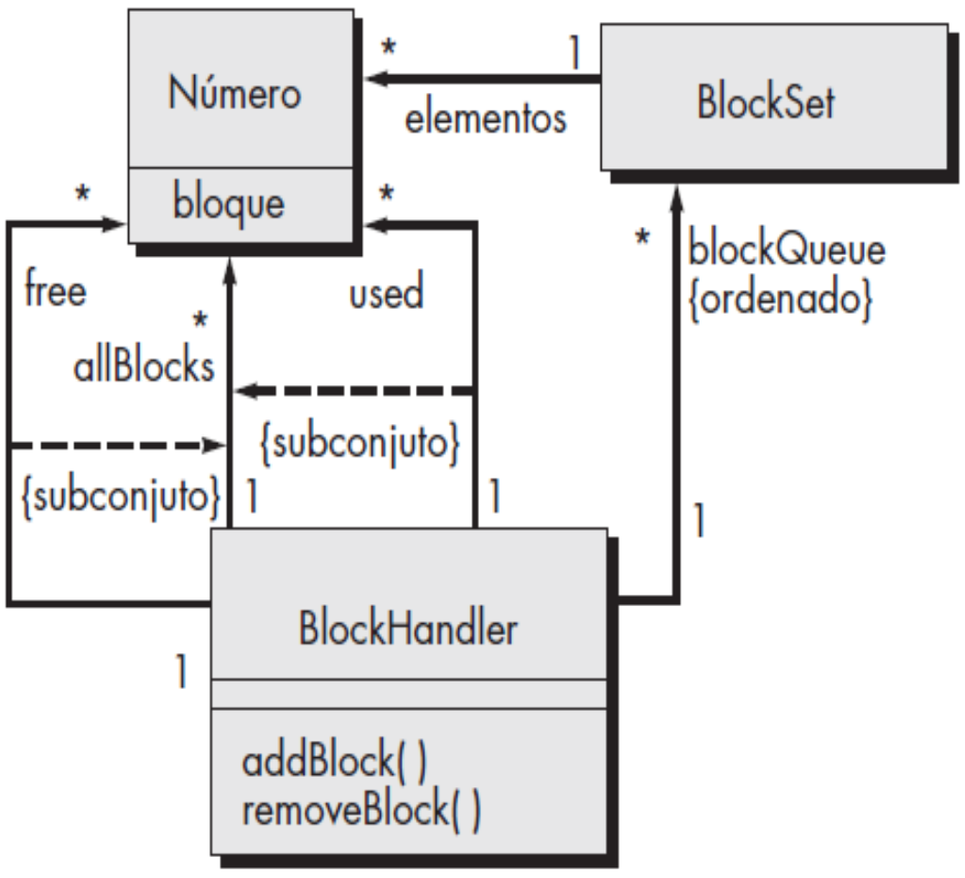
Ningún elemento de la cola (blockQueue) contendrá el mismo número de bloque

```
context BlockHandler inv:
    blockQueue->forAll(blockSet1, blockSet2 |
        blockSet1 <> blockSet2 implies
        blockSet1.elements.number->excludesAll(blockSet2elements.number))
```

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (10/13)

- Ej: Expresiones OCL de las restricciones



La colección de bloques utilizados y bloques que no se utilizan será la colección total de bloques que constituyen los archivos

```
context BlockHandler inv:
    allBlocks = used->union(free)
```

La colección de bloques no utilizados no tendrá números de bloque duplicados

```
context BlockHandler inv:
    free->isUnique(aBlock | aBlock.number)
```

La colección de bloques utilizados no tendrá números de bloque duplicados

```
context BlockHandler inv:
    used->isUnique(aBlock | aBlock.number)
```

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (11/13)

- OCL para precondiciones y postcondiciones de las operaciones

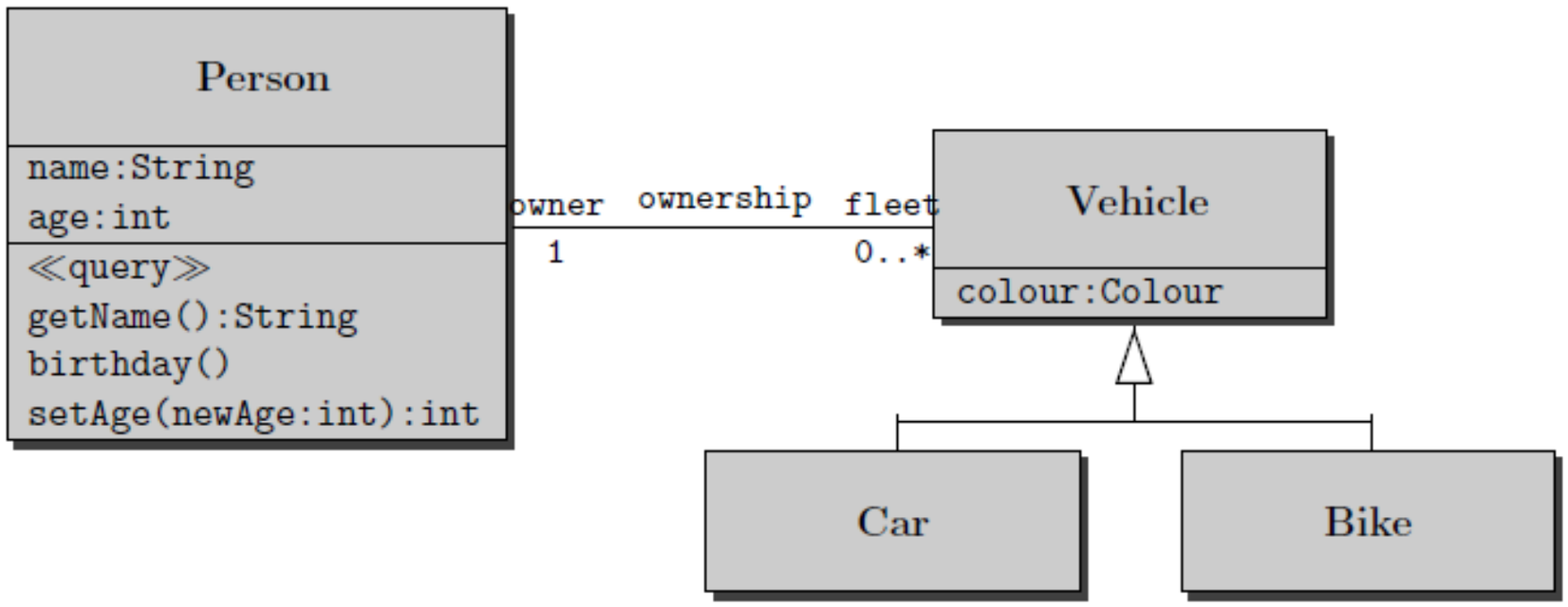
```
context BlockHandler::removeBlocks()
    pre: blockQueue->size() >0
    post: used = used@pre-blockQueue@pre->first() and
        free = free@pre->union(blockQueue@pre->first()) and
        blockQueue = blockQueue@pre->excluding(blockQueue@pre->first)
```

x@pre: el objeto x antes de la operación

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (12/13)

♦ **Ejercicio**: dado el siguiente diagrama de clases definir mediante OCL las siguientes restricciones:

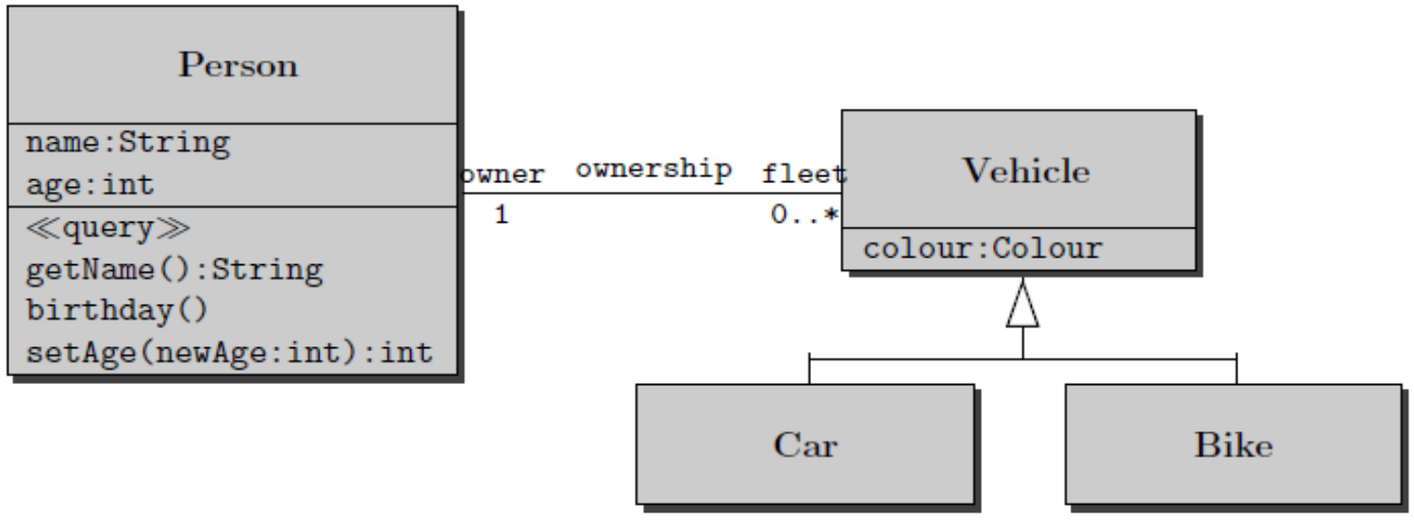


- El propietario de un vehículo tiene que tener al menos 18 años
- Ninguna persona tiene más de tres vehículos
- Todos los coches de las personas son negros
- Para operación setAge → pre: edad mayor que cero
- Para operación birthday → post: edad se incrementa en uno

Lenguajes de Especificación Formal (2/3)

Lenguaje de restricción de objeto (OCL) (13/13)

♦ **Ejercicio**:



- El propietario de un vehículo tiene que tener al menos 18 años

context Vehicle inv: self.owner.age >= 18

- Ninguna persona tiene más de tres vehículos

context Person inv: self.fleet→size() <= 3

- Todos los coches de las personas son negros

context Person inv: self.fleet→forAll(v | v.colour = black)

- Para operación setAge → pre: edad mayor que cero

context Person::setAge(newAge:int) pre: newAge > 0

- Para operación birthday → post: edad se incrementa en uno

context Person::birthday() post: self.age = self.age@pre + 1

Lenguajes de Especificación Formal (3/3)

Lenguaje Z (1/5)

- Se aplica a conjuntos escritos, relaciones y funciones dentro del contexto de la lógica de predicados de primer orden
- Las especificaciones del lenguaje Z se organizan como un conjunto de **esquemas:**
  - Estructura de lenguaje que introduce variables y que especifica la relación entre dichas variables
  - Describe los datos almacenados a los que accede y que altera un sistema (el **estado**)
  - Describe las operaciones que se aplican para cambiar el estado y las relaciones que ocurren dentro del sistema

Lenguajes de Especificación Formal (3/3)

Lenguaje Z (2/5)

- Ejemplo sobre el manipulador de bloques
  - *used* y *free* serán conjuntos de bloques.
  - *BlockQueue* será una secuencia
  - No habrá bloques comunes en la colección utilizada y en las colecciones libres de bloques.
  - La colección de bloques utilizados y de bloques libres siempre será igual a la colección completa de bloques en el sistema
  - El *i*-esimo elemento en la fila de bloques siempre será un subconjunto de los bloques utilizados.
  - Para cualesquiera dos elementos de la fila de bloques que no son el mismo, no habrá bloques comunes en dichos elementos

Lenguajes de Especificación Formal (3/3)

Lenguaje Z (3/5)

BlockHandler

$used, free : \mathbb{P}\ BLOCKS$

$BlockQueue : \text{seq}\ \mathbb{P}\ BLOCKS$

$used \cap free = \varnothing \land$

$used \cup free = AllBlocks \land$

$\forall i: \textbf{dom}\ BlockQueue \bullet BlockQueue\ i \subseteq used \land$

$\forall i, j: \text{dom}\ BlockQueue \bullet i \neq j \Rightarrow BlockQueue\ i \cap BlockQueue\ j = \varnothing$

- *used* y *free* serán conjuntos de bloques.
- *BlockQueue* será una secuencia
- No habrá bloques comunes en la colección utilizada y en las colecciones libres de bloques.
- La colección de bloques utilizados y de bloques libres siempre será igual a la colección completa de bloques en el sistema
- El *i*-esimo elemento en la fila de bloques siempre será un subconjunto de los bloques utilizados.
- Para cualesquiera dos elementos de la cola de bloques que no son el mismo, no habrá bloques comunes en dichos elementos

Lenguajes de Especificación Formal (3/3)

Lenguaje Z (4/5)

❑ Notación básica (1/2)

Conjuntos:

$S : \mathbb{P}\, X$	S se declara como un conjunto de Xs.
$X \in S$	x es miembro de S.
$x \notin S$	x no es miembro de S.
$S \subseteq T$	S es un subconjunto de T: todo miembro de S también está en T.
$S \cup T$	La unión de S y T: contiene a todo miembro de S o T o ambos.
$S \cap T$	La intersección de S y T: contiene a todo miembro tanto de S como de T.
$S \setminus T$	La diferencia de S y T: contiene todo miembro de S excepto aquellos que también están en T.
$\varnothing$	Conjunto vacío: no contiene miembros.
{x}	Conjunto de un solo elemento: sólo contiene a x.
$\mathbb{N}$	Conjunto de los números naturales 0, 1, 2, ...
$S : \mathbb{F}\, X$	S se declara como un conjunto finito de Xs.
max (S)	El máximo del conjunto no vacío de números S.

Lenguajes de Especificación Formal (3/3)

Lenguaje Z (4/5)

❑ Notación básica (2/2)

Funciones:

$f : X \rightarrowtail Y$	f se declara como una inyección parcial de X a Y.
dom f	El dominio de f: el conjunto de valores x para los cuales se define $f(x)$.
ran f	El rango de f: el conjunto de valores tomados por $f(x)$ conforme x varía sobre el dominio de f.
$f \oplus \{x \mapsto y\}$	Una función que concuerda con f excepto que x se mapea a y.
$\{x\} \lhd f$	Una función como f, excepto que x se remueve de su dominio.

Lógica:

$P \wedge Q$	P y Q: es verdadero si tanto P como Q son verdaderos.
$P \Rightarrow Q$	P implica a Q: es verdadero si Q es verdadero o P es falso.
$\theta S' = \theta S$	Ningún componente del esquema S cambia en una operación.

Lenguajes de Especificación Formal (3/3)

Lenguaje Z (5/5)

- También sirve para definir precondiciones y postcondiciones de las operaciones (removeBlocks)

RemoveBlocks

Δ *BlockHandler*

$\#BlockQueue > 0$,

$used' = used \setminus head\ BlockQueue \land$

$free' = free \cup head\ BlockQueue \land$

$BlockQueue' = tail\ BlockQueue$

- Δ *BlockHandler* da como resultado todas las variables que constituyen el estado disponible para el esquema *RemoveBlocks* y garantiza que la invariante de datos se cumplirá antes y después de ejecutar la operación.