Evaluation subjects

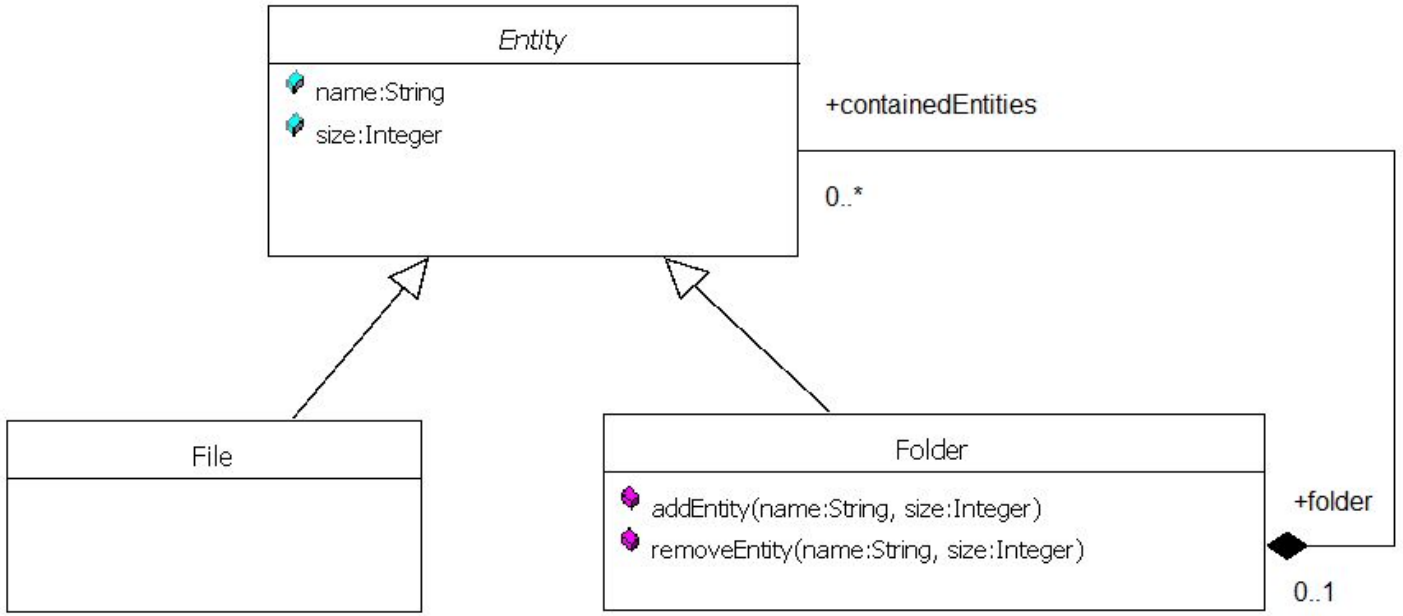
Dan Chiorean

The management of files system

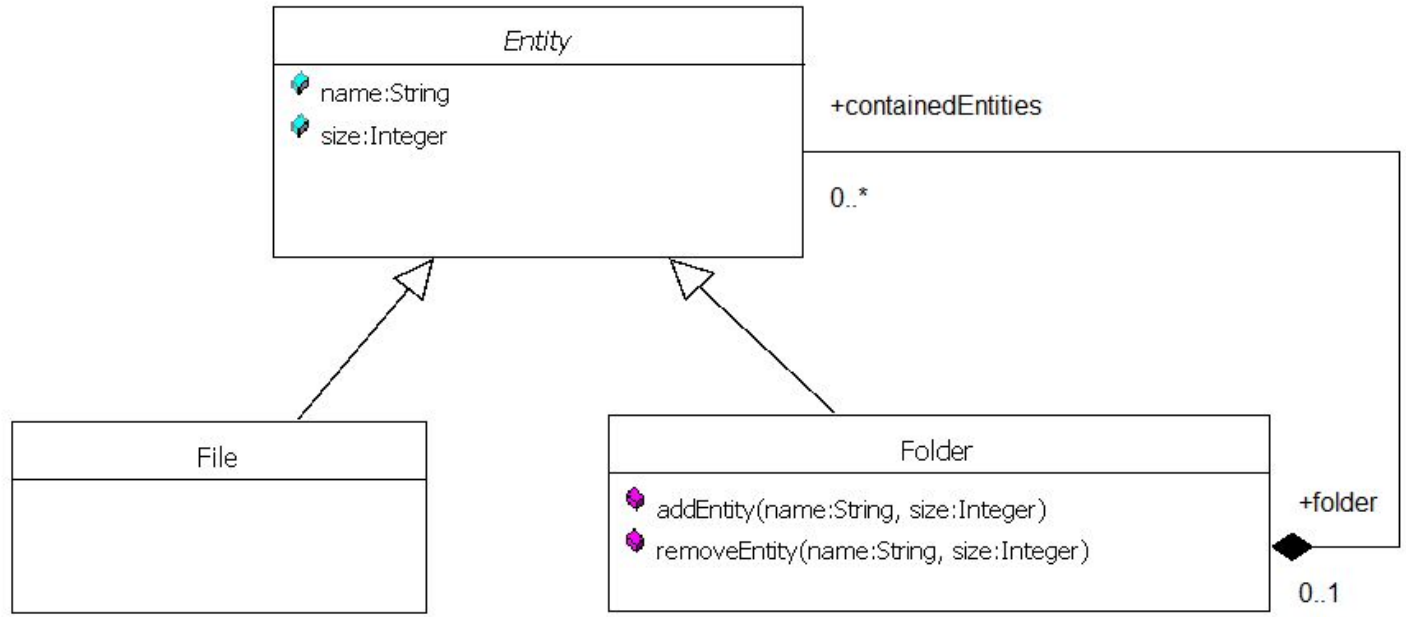
- In computers, the persistency of information is managed by using files grouped in folders (directories). Each element of the file system is characterized by name and size. In a namespace the uniqueness of element'name is mandatory. File system elements are hierarchicaly organized. Using UML, please specify the structure of a file system complying with the above informal specification. Using OCL please specify an invariant used to check the above-mentioned constraint. . In order to support the quote "It is better to prevent than cure.", please specify the same constraint as a precondition of the operation addFile(fileName: String, size:Integer) that is implemented in the Folder namespace.

The management of files system _2

- Each file has a size measured in bytes. Please specify another precondition for the operation addFile(fileName:String, size:Integer) that checks that adding the size of the new file to the sum of size for all the files stored in the drive, the drive capacity is not exceeded.

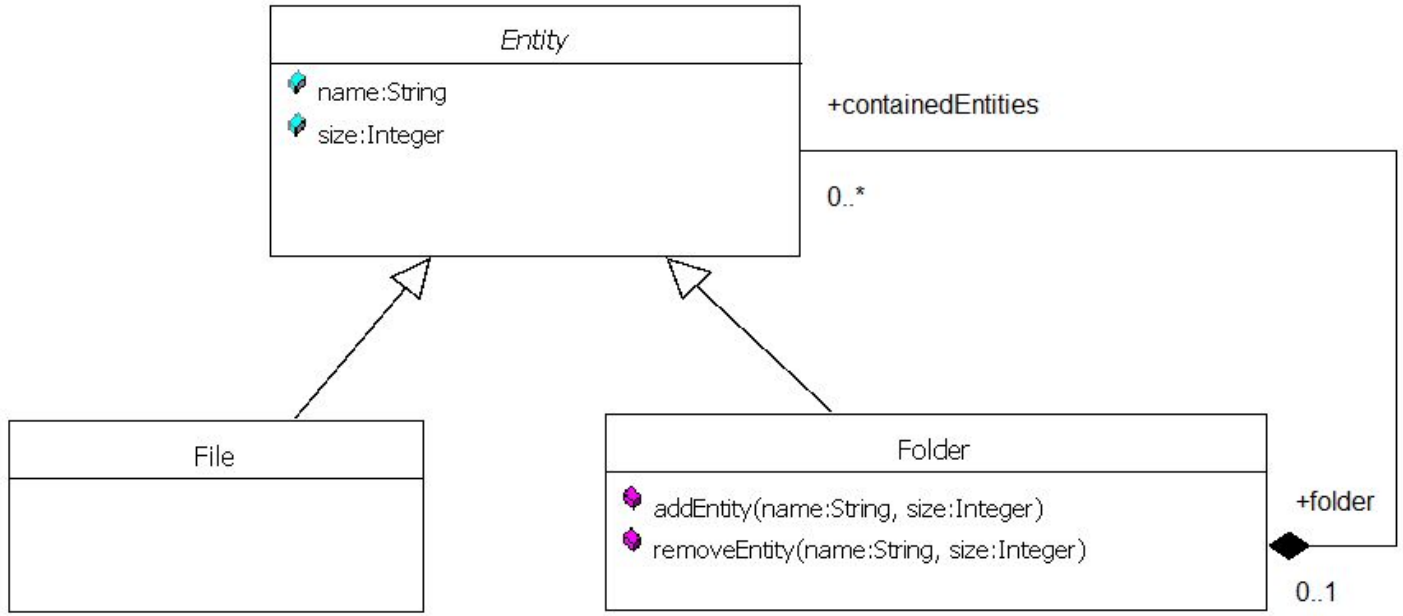


The management of files system _3




```
context Folder
        inv onlyOneRoot:
           Folder.allInstances->one(f:Folder | f.oclAsType(Entity).folder->isUndefined)


context Folder::addEntity(n:String, s:Integer)

    pre:
        let root:Folder = Folder.allInstances->select(f | f.oclAsType(Entity).folder->isUndefined)->any(f |true) in
        self.containedEntities->excludes(self) and root.size - root.containedEntities.size->sum > s

    post:
        self.containedEntities->includes(self) and self.size = self@pre.size + s
```

The management of information in drivers_ ^

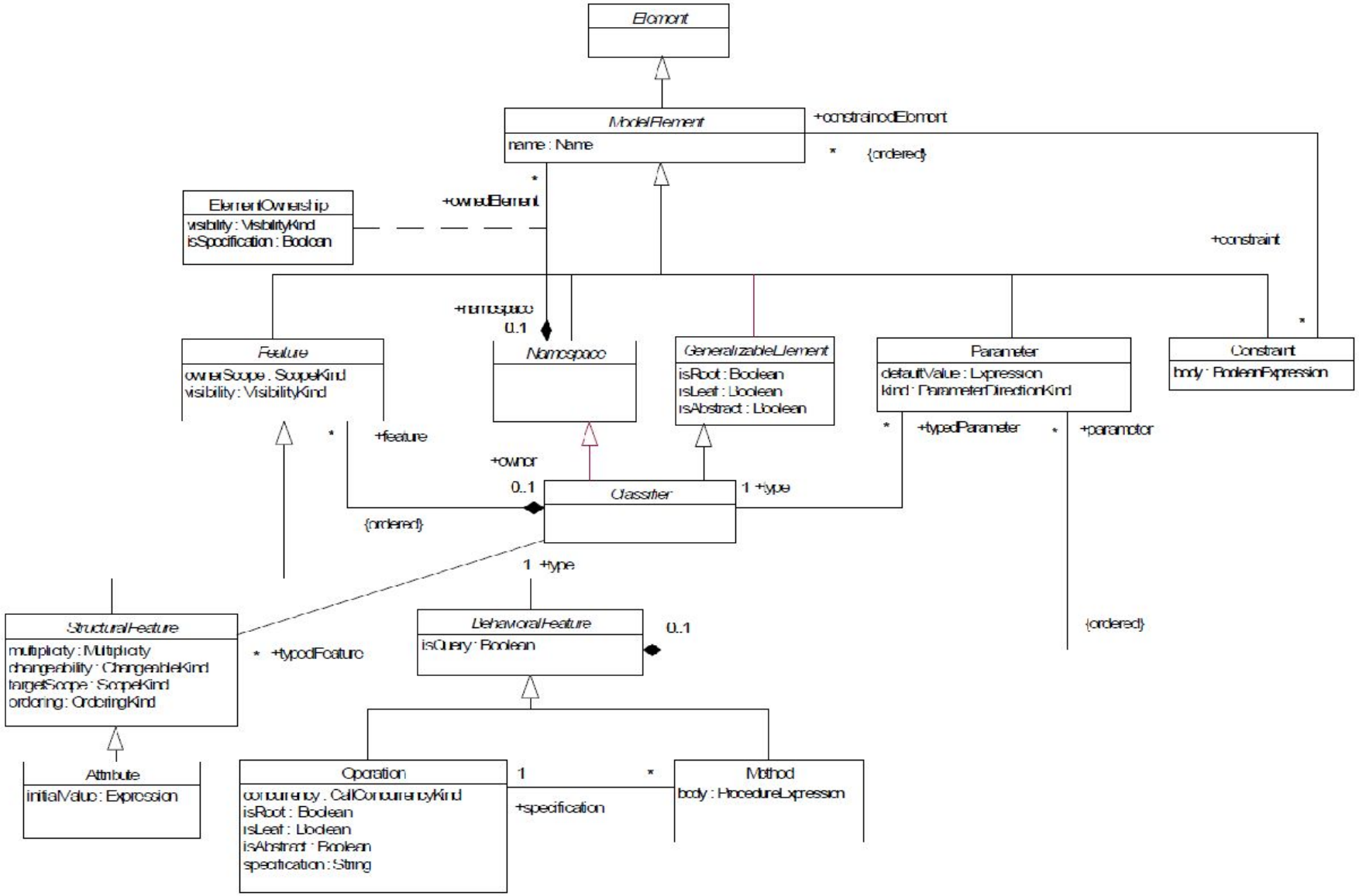



```
context Folder

        inv nameUnicity:
            let cElementsName:Bag(String) = self.containedEntities.name in
            self.containedEntities->reject(cE:Entity | cElementsName->count(cE.name)=1)->isEmpty
endmodel

? Bag{1, 2, 3, 1, 4, 5}->one(i:Integer | i = 1)

? Bag{1, 2, 3, 1, 4, 5}->one(i:Integer | i = 3)

? Bag{1, 2, 3, 1, 4, 5}->count(1)
```

Identify Metaclasses of the precedent UML class diagram

The Figure 3 describe the structure of main modeling concepts in UML 1.5. For each element of the model proposed in previous slides, please mention the metaclass whose instance is, if the metaclass is represented in Figure 3.

Identify Metaclasses of the precedent UML class diagram

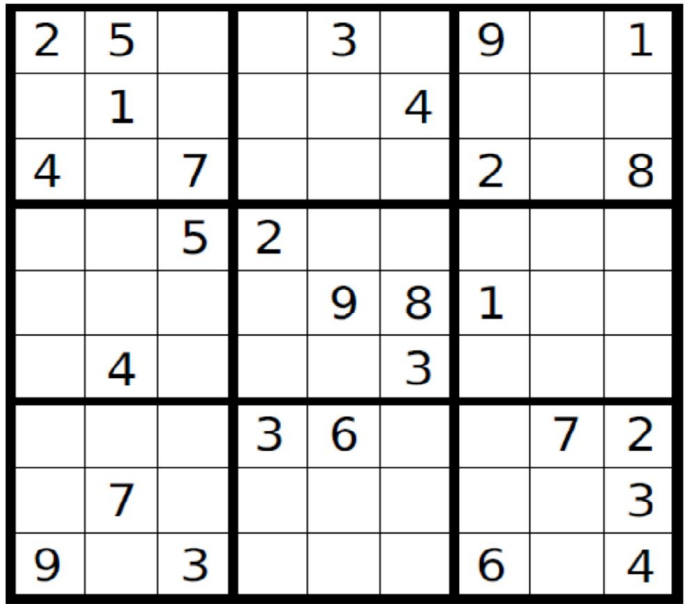


Sudoku

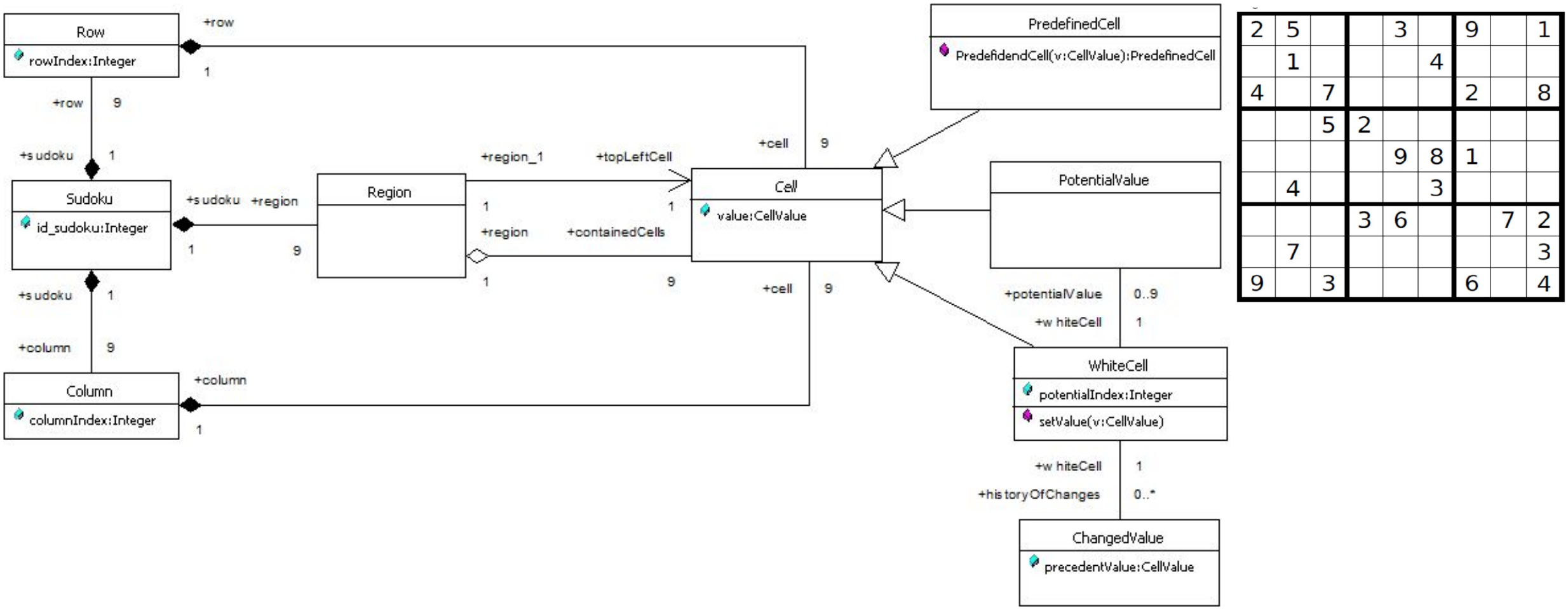
- A Sudoku puzzle is shown in the figure below. To complete this puzzle requires the puzzler to fill every empty cell with an integer between 1 and 9 in such a way that every number from 1 up to 9 appears once in every row, every column and everyone region of the small 3 by 3 boxes highlighted with thick borders. Each cell corresponds to a rowIndex, which represents the row of the cell, a columnIndex, which represents the column of the cell and has a value, which represents the content of the cell. A cell belongs to a region; each region is defined by the topLeft cell and by the contained cells. Initially, the puzzle contains predefined cells whose values remain unchanged and by empty cells, also named whiteCells. During the game, the puzzler starts by setting the value of empty cells trying to comply with the constraints mentioned in the beginning. . Once the value of an empty cell was set, the type of the cell is a new kind, named PotentialValue.

Sudoku_2

- If later in the game the puzzler notices that the value of the PotentialValue cell is incorrect, then this must be changed. To support new potential changes of the value, the player keeps changes history in a sequence of so named ChangedValue cells.
  - Using the UML, please construct a model containing the concepts needed to support the player
  - Using OCL please specify in the context of a cell an invariant constraining the value of the cell, which is not empty, to be unique both on his row, column and region.

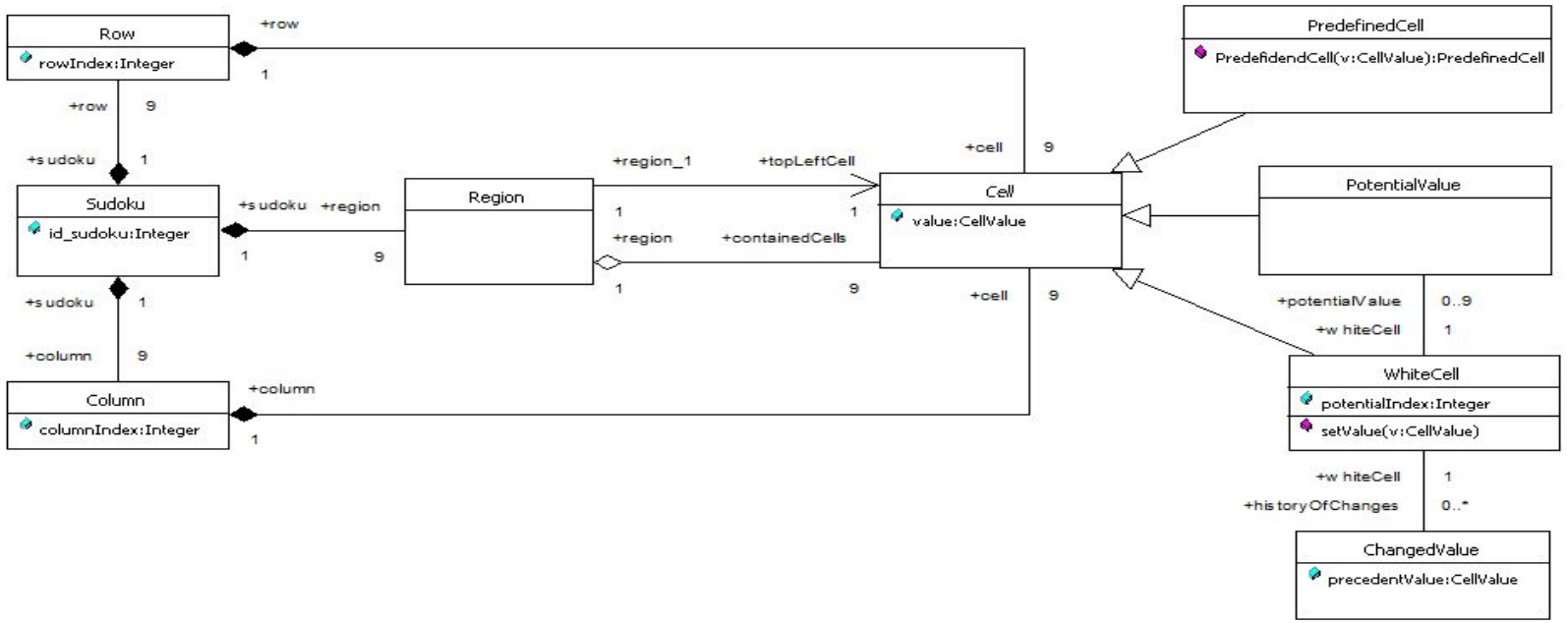


Sudoku_3



2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

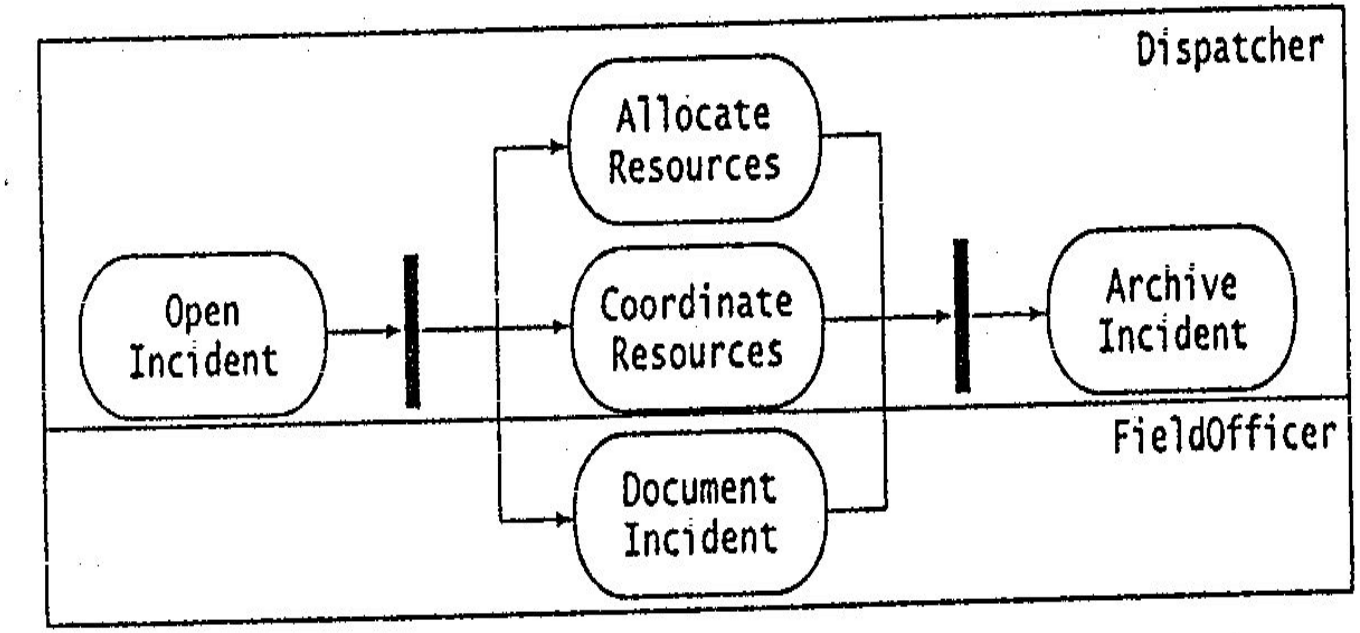
Sudoku 4




```
context Cell
    inv uniqueValue:
        if self.value.isUndefined
            then true
            else
                  (self.row.cell->reject(c:Cell|c.value.isUndefined)->
                   select(c:Cell | c.value = self.value)->size = 1 and
                self.column.cell->reject(c:Cell|c.value.isUndefined)->
                     select(c:Cell|c.value = self.value)->size = 1 and
                self.region.containedCells->reject(c:Cell
c.value.isUndefined)->select(c:Cell | c.value = self.value)->size = 1)
        endif
```

Please explain the diagram below

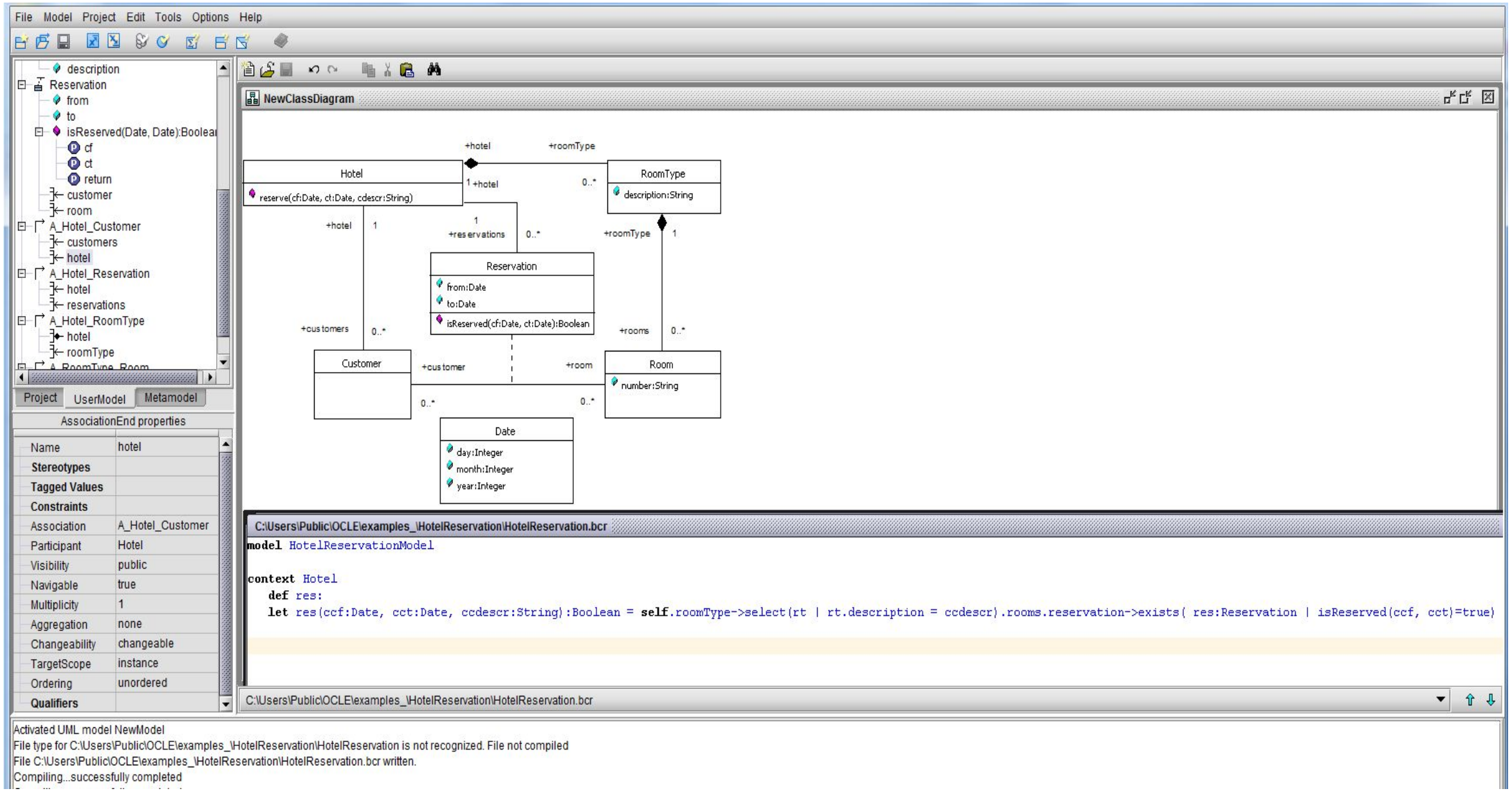
- The figure below represents a UML diagram describing how an incident is managed after a dispatcher receives an incident report from a field officer. Please name the kind of this diagram, describe in English the semantics of this diagram and name each modeling concepts represented.



Hotel booking

- A Hotel has different types of rooms. When a potential client intents to reserve a room, then it specifies the interval [from:Date ... to:Date] and the type of room he wants to reserve. On the hotel site, each type of room has a textual description. A software system receives the client request and answer if the reservation is possible. In other words, if in the interval mentioned there is at least a room of the type mentioned not occupied.
  - By means of a UML class diagram, please specify a UML model supporting the above mentioned functionality.
  - In the Hotel class, please specify, using OCL, an observer returning a Boolean value; (true if the reservation is possible and false if it's not). In the context of the Reservation class there is implemented an observer isReserved(cf:Date, ct:Date):Boolean that returns true if the room is not reserved any day in the interval[cf:Date ... ctDate].

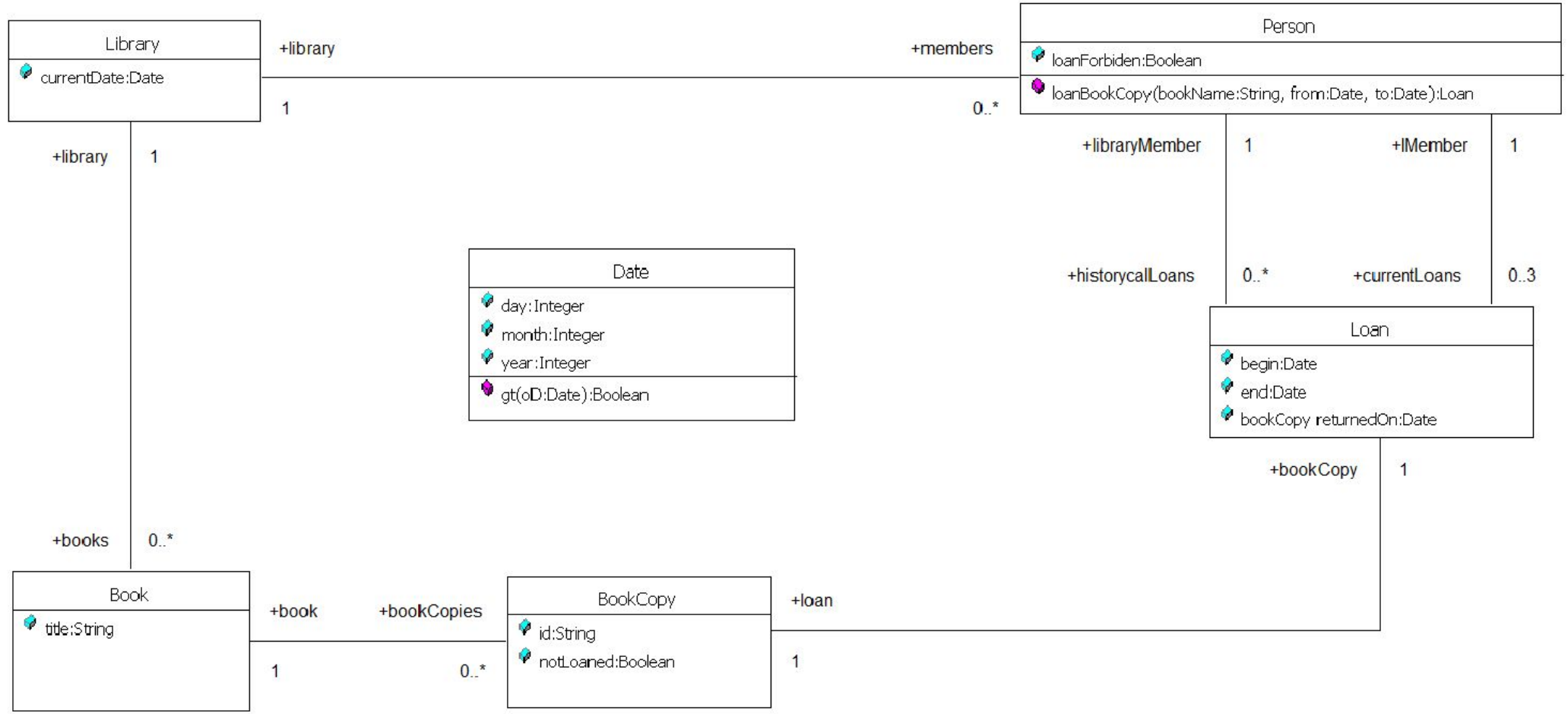
Hotel booking



Library management

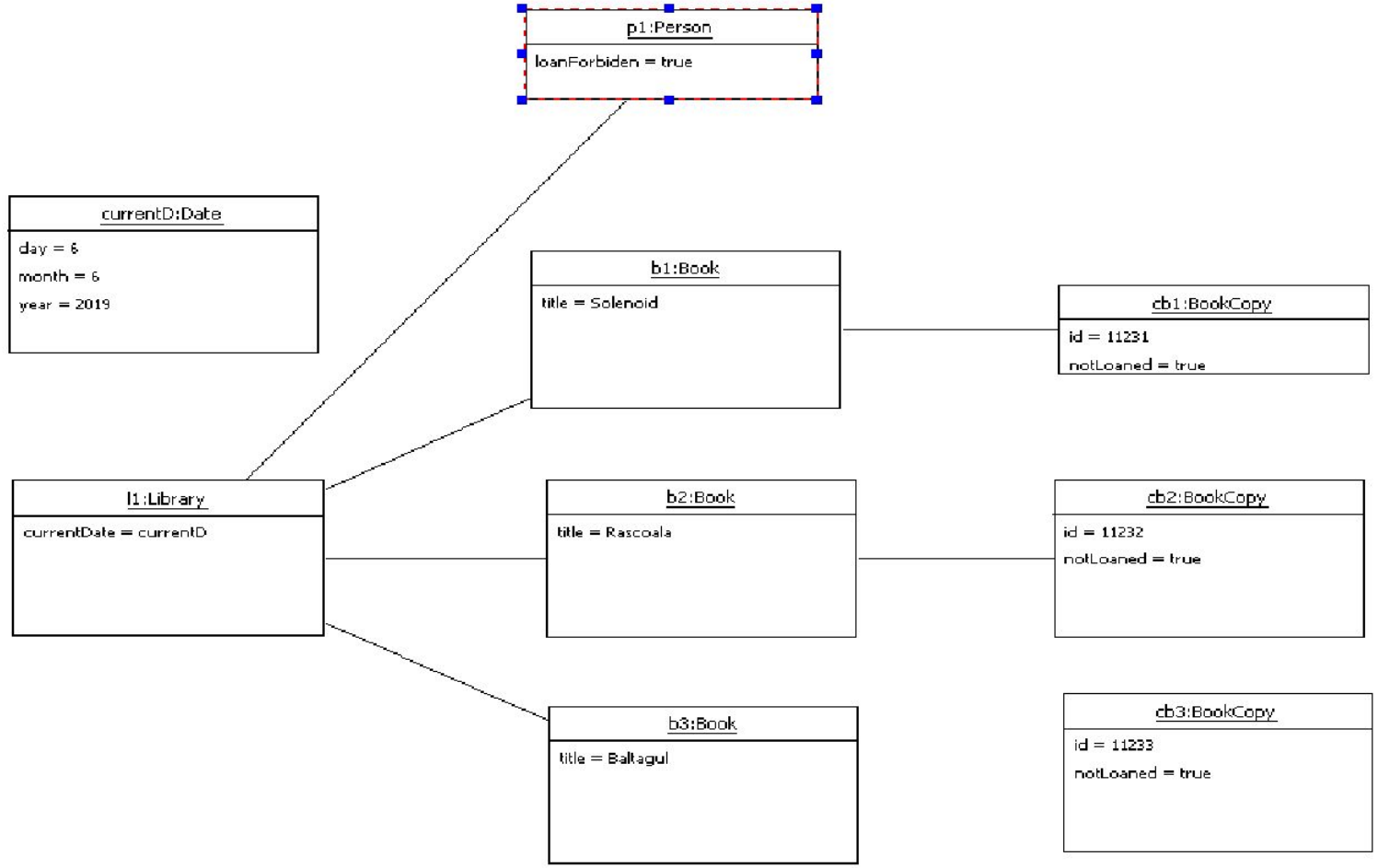
- A library owns a set of books. Each book has a title which is unique. Usually, for each book there can be many book copies, characterized by an ID unique and an information notLoaned, having the value **true** if the book copy is not loaned at the current date. The library has a set of persons which are library members. Each member may loan book copies if hi/she is not penalized (loan forbidden) by the library. Each loan concerns only a book copy. The maximum number of current loans is 3. The library keeps also the list of historical loans, in which apart of loan dates, the data of returning the book copy is stored. The current date is stored at the level of library. In the Person class there is implemented an operation **loanBookCopy(bookTitle:String, from:Date, to:Date):Loan**.
- 1. Please represent the architecture of the above-mentioned problem by means of a UML class diagram. 2 pt
- 2. Please specify a precondition for the operation **loanBookCopy(bookTitle:String, from:Date, to:Date):Loan** checking that: in the library books there is a book having the same title with the book mentioned on the loan requirement, and, in the book copies there is at least one which is not loan; that the member has less than 3 current loans, and between these there are not books with the same title or loans for which the current date is greater than the end date. (In the class Date there is an observer gt(od:Date):Boolean, returning true if the od is greater than the current day. 1 pt
- 3. Use a snapshot to represent 2 cases in which the precondition required for the **loanBookCopy** operation are violated.

Library management_2



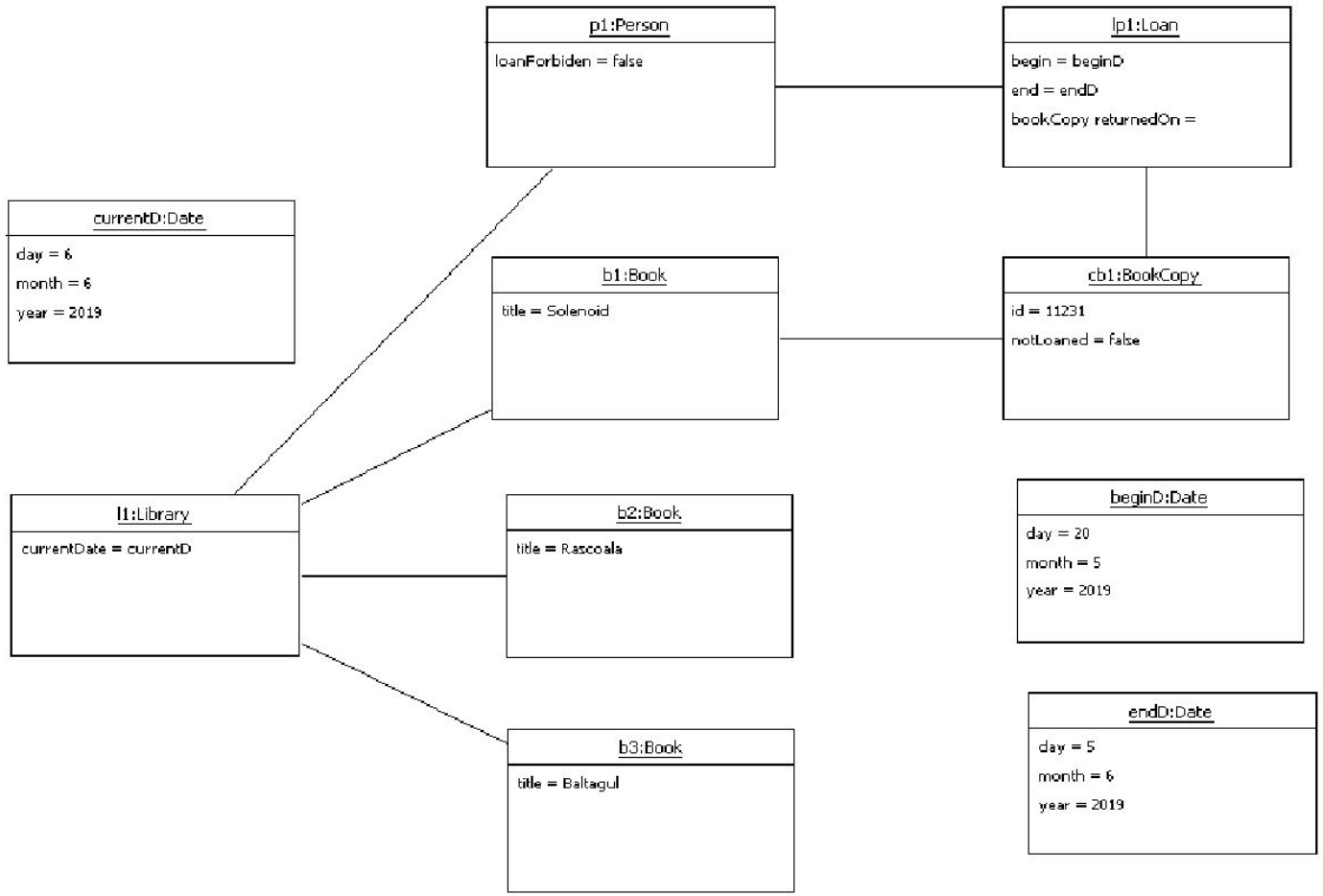

```
context Person::loanBookCopy(bookTitle:String, from:Date, to:Date):Loan
    pre:
        not self.loanForbiden and
        self.library.books->exists(b:Book | b.title = bookTitle and b.bookCopies->exists(bc | bc.notLoaned)) and
        self.currentLoans->size < 3 and
        not self.currentLoans.bookCopy.book.title->exists(bn | bn = bookTitle) and
        not self.currentLoans->exists(l:Loan | self.library.currentDate.gt(l.end))
```

Library management_3




```
context Person::loanBookCopy(bookTitle:String, from:Date, to:Date):Loan
    pre:
        not self.loanForbiden and
        self.library.books->exists(b:Book | b.title = bookTitle and b.bookCopies->exists(bc | bc.notLoaned)) and
        self.currentLoans->size < 3 and
        not self.currentLoans.bookCopy.book.title->exists(bn | bn = bookTitle) and
        not self.currentLoans->exists(l:Loan | self.library.currentDate.gt(l.end))
```

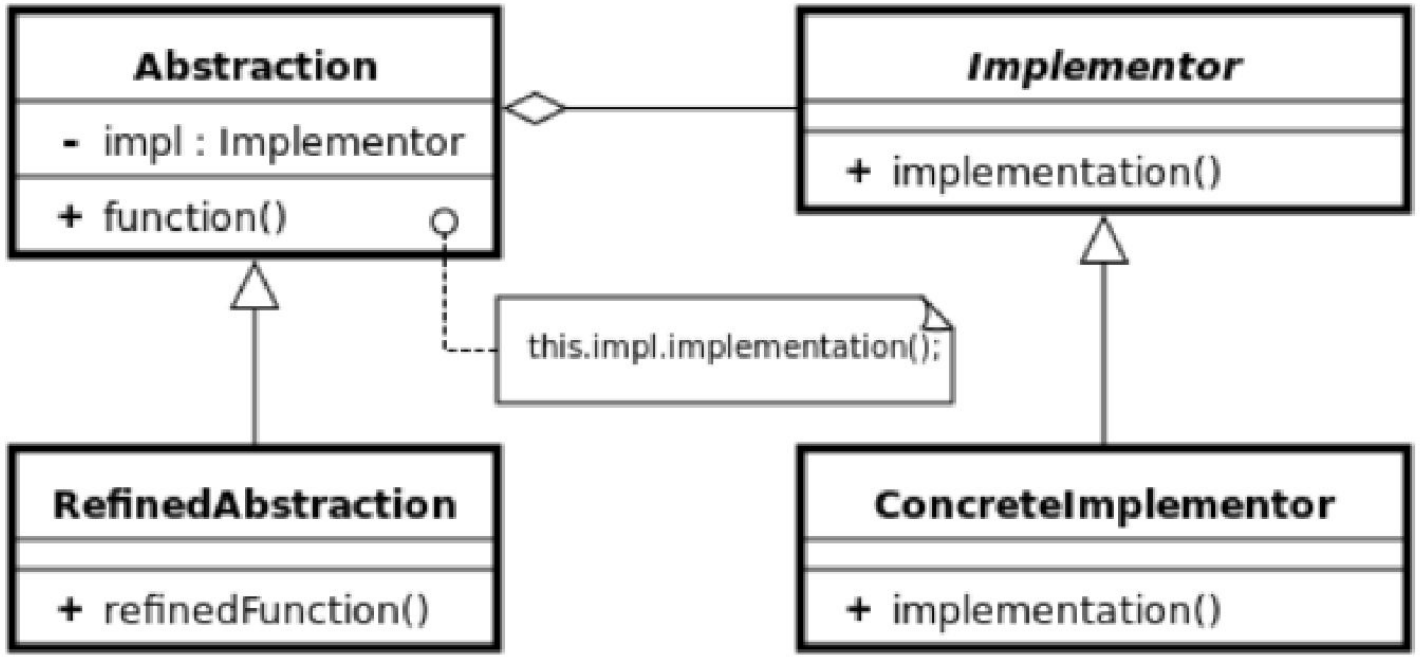
Library management_4



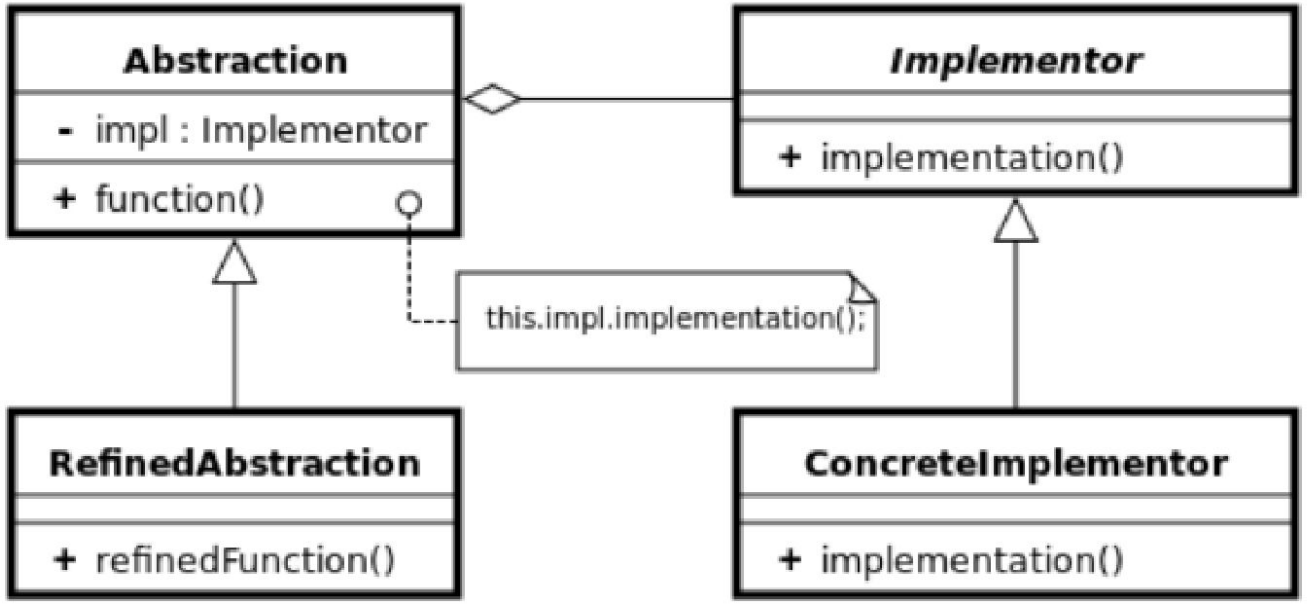

```
context Person::loanBookCopy(bookTitle:String, from:Date, to:Date):Loan
    pre:
        not self.loanForbiden and
        self.library.books->exists(b:Book | b.title = bookTitle and b.bookCopies->exists(bc | bc.notLoaned)) and
        self.currentLoans->size < 3 and
        not self.currentLoans.bookCopy.book.title->exists(bn | bn = bookTitle) and
        not self.currentLoans->exists(l:Loan | self.library.currentDate.gt(l.end))
```

Please analyze the diagram below

Please mention if in the class diagram represented below there are some informational redundancies from the point of view of the UML specification. If the diagram represents a design pattern, please name it and describe how the pattern works.

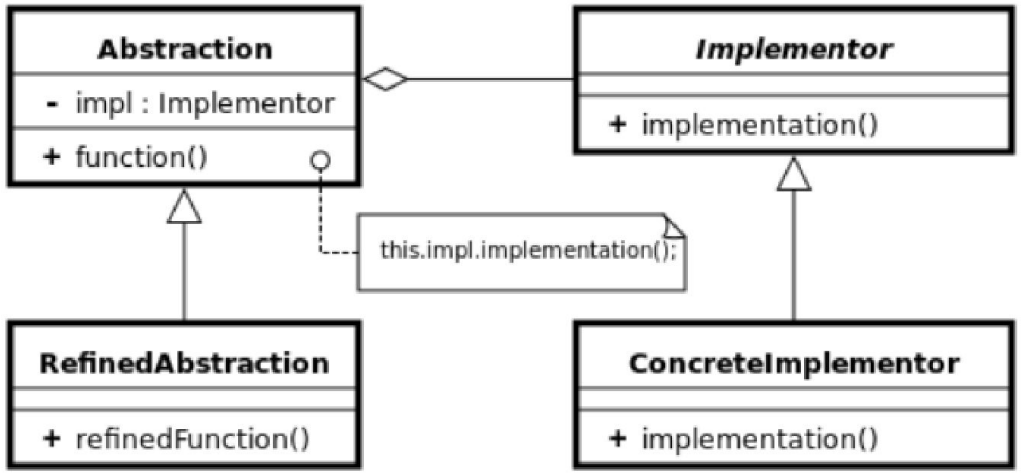


Please analyze the diagram below_2



- As we may notice, in the Abstraction interface the impl attribute (containing a reference towards the *Implementor* interface) is redundant because this is represented graphically by means of an aggregation (composition in pattern description). The diagram represents the Bridge design pattern.
- The Bridge pattern decouples abstraction from implementation so that both can vary independently. It has been achieved with delegation (composition) rather than by using inheritance.

Please analyze the diagram below_3



- The pattern contains four components.
- Abstraction: defines an interface
- RefinedAbstraction: implements abstraction:
- Implementor: defines an abstract class (interface) for implementation
- ConcreteImplementor: implements Implementor interface.
- Two orthogonal class hierarchies using delegation (and no inheritance). The Abstraction and Implementation hierarchies can vary independently. Implementation never refers Abstraction. Abstraction contains Implementation interface as a member (through composition). This composition reduces one more level of inheritance hierarchy.