# Qualité Logicielle – Project 2

Quentin DEBESSON – Valentine ISAUTIER

## Stack

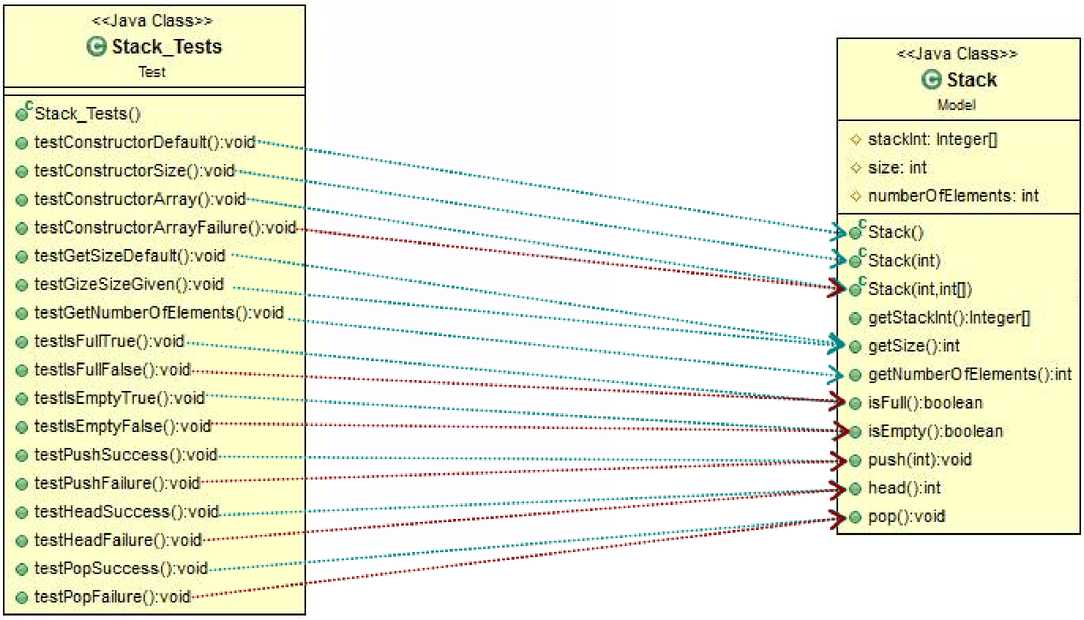
### Question 3.

If we had to implement differently the `Stack` class without changing all the methods, we would have used generic types instead of hard-defined types (i.e. `int`). It would avoid a lot of changes, and the object would be more flexible in the end.

The only difference would be in the types and not in the global architecture.

### Question 4.

We wouldn't need to rewrite all the tests. The only one using a strongly defined type (`head()`, `push()` or some constructors) would be rewritten.



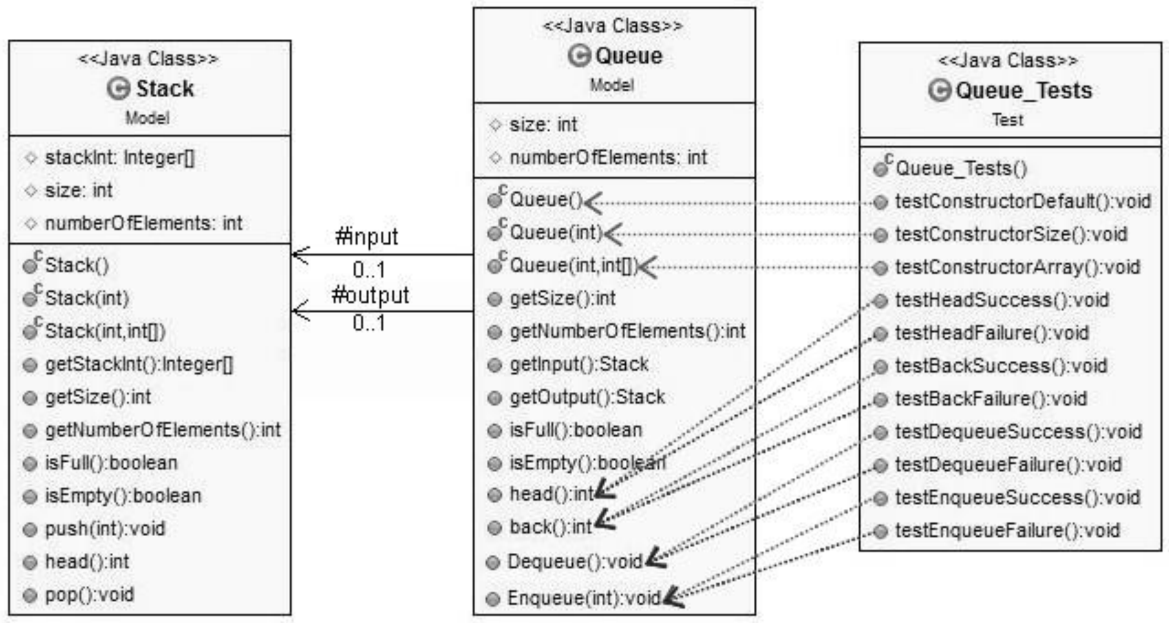
# Queue

## Question 9.

We won’t need to rewrite all the tests. The only tests that are needed are for `dequeue()`, `enqueue()`, `head()`, `back()` (in order to check the algorithm for these 4 methods), and the constructors : `Queue()`, `Queue(int)`, `Queue(int, int[])`.

Some methods are exactly the same as the ones in the `Stack` class but used in the `Queue` context, calling `Stack` objects: therefore, the tests that imply Stacks are useless.

## Question 10.



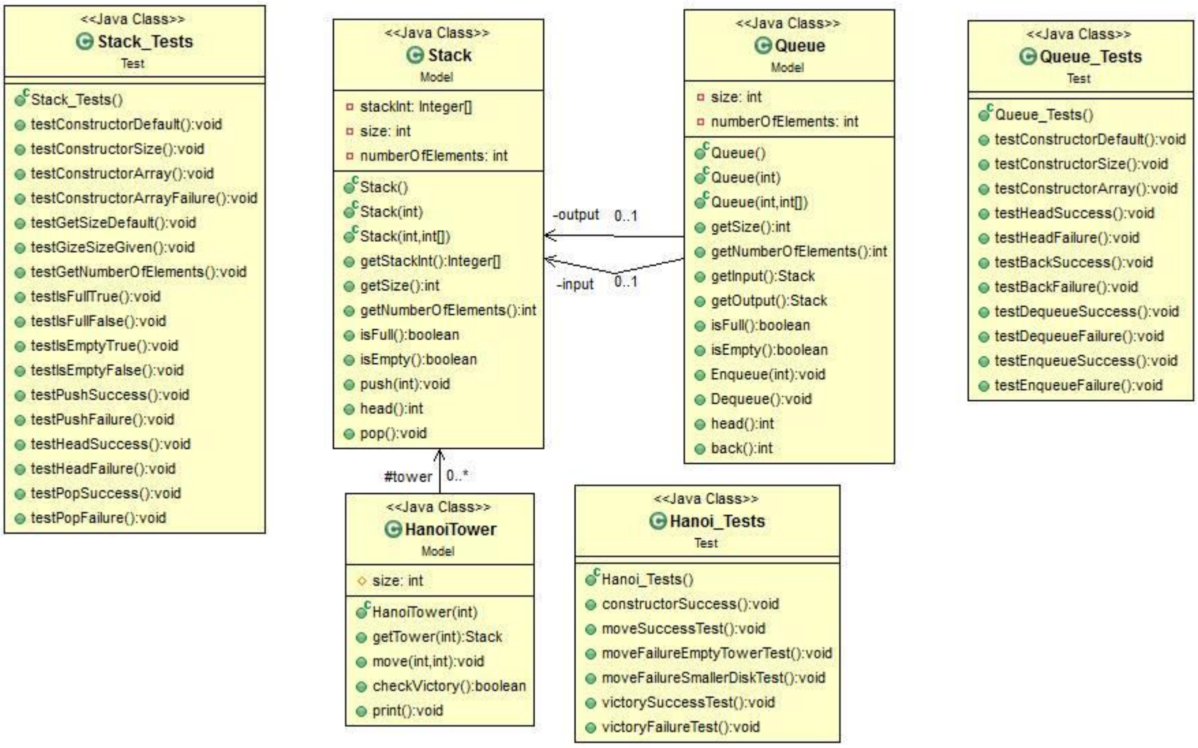
## Question 11.

We have reused code from the `Stack` class to develop the `Queue` class, and from the `Stack_tests` class to develop the `Queue_tests` class.

Another solution would have been to make the `Queue` class extend from `Stack`: this way, the code wouldn’t need to be reused, the only thing that we could refactor would be the 2nd array inside the table to manage the `input` and the `output`. The new functions would have to be coded (`head`, `back`) and the old functions to be refactored (`push` and `pop` become `Enqueue` and `Dequeue`).

# Hanoi Towers

Here is the final class diagram for the Hanoi Towers project:



<<Java Class>>
Stack_Tests
Test
Stack_Tests()
testConstructorDefault():void
testConstructorSize():void
testConstructorArray():void
testConstructorArrayFailure():void
testGetSizeDefault():void
testGizeSizeGiven():void
testGetNumberOfElements():void
testIsFullTrue():void
testIsFullFalse():void
testIsEmptyTrue():void
testIsEmptyFalse():void
testPushSuccess():void
testPushFailure():void
testHeadSuccess():void
testHeadFailure():void
testPopSuccess():void
testPopFailure():void
<<Java Class>>
Stack
Model
stackInt: Integer[]
size: int
numberOfElements: int
Stack()
Stack(int)
Stack(int,int[])
getStackInt():Integer[]
getSize():int
getNumberOfElements():int
isFull():boolean
isEmpty():boolean
push(int):void
head():int
pop():void
-output
0..1
-input
0..1
<<Java Class>>
Queue
Model
size: int
numberOfElements: int
Queue()
Queue(int)
Queue(int,int[])
getSize():int
getNumberOfElements():int
getInput():Stack
getOutput():Stack
isFull():boolean
isEmpty():boolean
Enqueue(int):void
Dequeue():void
head():int
back():int
<<Java Class>>
Queue_Tests
Test
Queue_Tests()
testConstructorDefault():void
testConstructorSize():void
testConstructorArray():void
testHeadSuccess():void
testHeadFailure():void
testBackSuccess():void
testBackFailure():void
testDequeueSuccess():void
testDequeueFailure():void
testEnqueueSuccess():void
testEnqueueFailure():void
#tower
0..*
<<Java Class>>
HanoiTower
Model
size: int
HanoiTower(int)
getTower(int):Stack
move(int,int):void
checkVictory():boolean
print():void
<<Java Class>>
Hanoi_Tests
Test
Hanoi_Tests()
constructorSuccess():void
moveSuccessTest():void
moveFailureEmptyTowerTest():void
moveFailureSmallerDiskTest():void
victorySuccessTest():void
victoryFailureTest():void