Question 3: Abstraction

1. Instruction

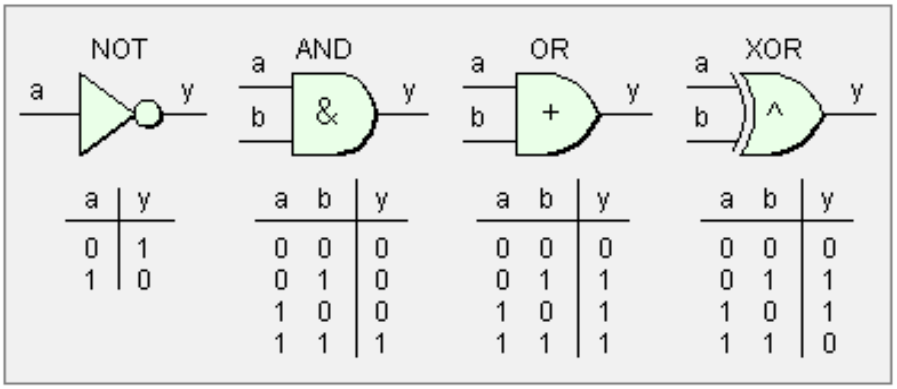
1. Create an IntelliJ project named "**Q3**" inside your folder CP_seatno_{ID}_{firstname} inside c:\temp
2. Copy all folders in folder "Q3" of the given zip file to your project **src** folder.
3. You are to implement or fill in the blanks for the following classes (details for each class are given in section 3.)
   i. BaseLogicGate (package **LogicGate**)
   ii. OrGate (package **LogicGate**)
   iii. NotGate (package **LogicGate**)
   iv. XorGate (package **LogicGate**)
   v. Adder (package **LogicGate**)
4. You also need to create a UML Diagram containing all classes in the package LogicGate. Save it as **umlQ3.png** at the **root** folder of your project.
5. JUnit for testing is in the package "student".
6. Export jar file (**q3.jar**) that **includes your source code and your .class files** and put it in **root** folder of your project. YOU MUST EXPORT JAR FILE. IF YOU DO NOT, YOUR CODE WILL NOT BE MARKED.

You should read the whole document before start coding. You may need to use some methods from the already provided classes.

2. Problem Statement: Logic gates

Logic gates are fundamental building blocks in digital electronics. AND gates output a 1 signal only when all inputs are 1s, OR gates output a 1 signal if any input is 1, NOT gates invert the input signal, and XOR gates output a high signal if the number of 1s inputs is odd. These gates are essential for processing binary data.

You might be familiar with all these gates but with only 2 inputs and operating with 1-bit data like this.



a	y
0	1
1	0

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

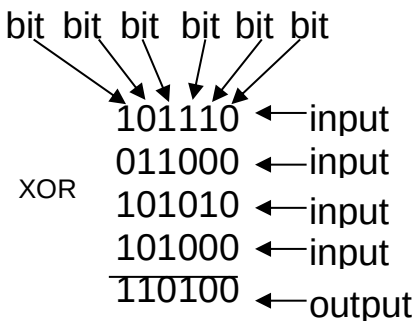
Fig: 1-bit, 2 inputs logic gates.

IMPORTANT

Here we will be implementing something beyond, we will be creating n-bit, m-inputs logic gates when n, m are of your choosing.

When dealing with multiple-bit calculation, just like addition(+), subtraction(-), or multiplication(), you do it digit-by-digit, in this case for logic gates(AND, OR, XOR, NOT), bit-by-bit.

Ex. a 6-bit, 4-input XOR gate.



AND-gate: Output bit is 1 when all the input bits are 1, otherwise 0.

OR-gate: Output bit is 1 when any input bits are 1, otherwise 0.

NOT-gate: Output bit is 1 when input bit is 0, and vice versa.

XOR-gate: Output bit is 1 when there are an odd number of 1s from the input bits, otherwise 0.

---

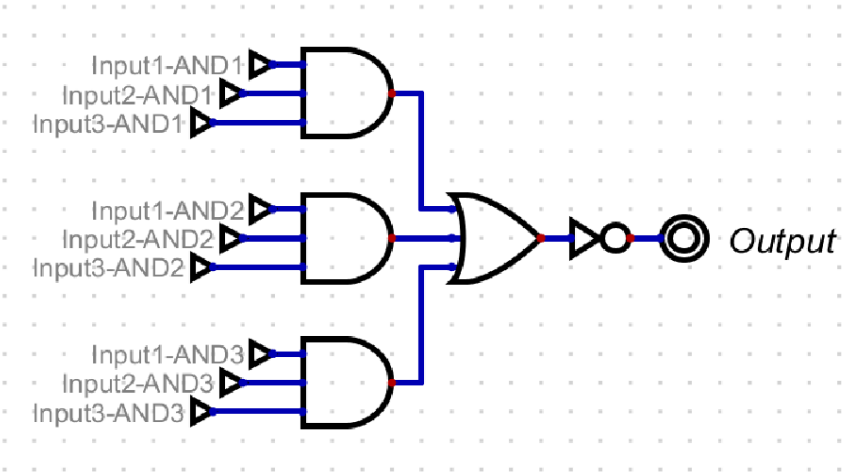
3. Implementation Details

The class packages are summarized below. Note that the important methods that you either need to write and methods that you need to complete this question are listed below with /* you must implement from scratch*/ or /*Partially Provided*/.

3.1 Package application /*Already Provided*/

3.1.1. Class Main /*Already Provided*/

This class implements an AND-OR-invert (AOI) logic component that is constructed from AND-gates, OR-gates and a NOT-gate of which you, the student taking this test will be the one providing. After you've finished creating the gates, you can try using them here. Disclaimer: running this class is not mandatory and does not count towards the score, don't waste time here if you haven't finished implementing all the gates.



Methods

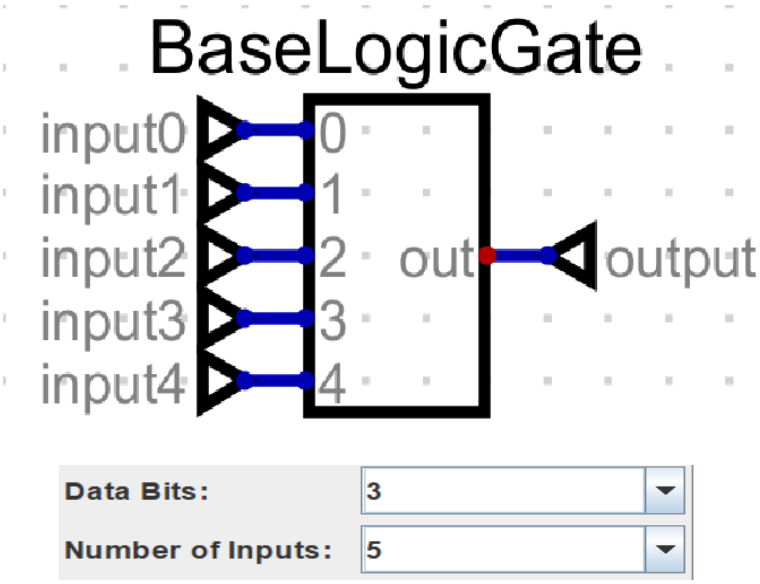
Method	Description
+ main(String[] args)	Run AOI program. 1. Receive “Amount of bits of each input” 2. Receive “Amount of AND-gates” 3. Receive “Amount of inputs per AND-gate” 4. Receive “Strings of inputs” Ex. “0010010” for 7 bit input.

3.2 Package LogicGate

This package contains the base class of all logic gates and all its subsequent variations (AND, OR, NOT, XOR)

3.2.1 **Class BaseLogicGate** /* you must implement from the scratch*/

This class represents BaseLogicGate which is the template of all other functional gates, It cannot be initialized. All logic gates have 2 fundamental values which need to be stated, first is the number of inputs that the gate will be calculating with, and second is the number of bits this gate will be working on.



Fields

Field	Description
- int numOfInput	Amount of inputs the gate takes in to calculate.
- int numOfBit	Amount of bits the gate takes in to calculate.
- ArrayList<String> datas	Stores the inputs. It should have the size “numOfInput”. Ex. data at index 3 = input3
- String output	Stores the output String after calculation.

Methods

Method	Description
+ BaseLogicGate(int NumOfInput, int NumOfBit)	BaseLogicGate constructor. Set numOfInput to NumOfInput. Set numOfBit to NumOfBit.

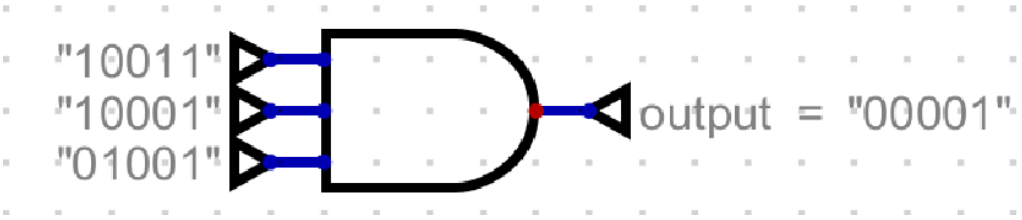
	Set output to 0 value string corresponding to the number of bits used. Ex. numOfBit = 5, then 0 value string = “00000”. Set datas to an arrayList<String> that is filled with 0 value Strings and its size corresponds to numOfInput. Ex. numOfInput = 3 numOfBit = 2 datas = [“00”, “00”, “00”] (This is not the code, just the representation of what it should look like.)
+ String toString()	This method is to print out the output of the gate, Hint: Use method provided in “Package Utils”.
+ String evaluate()	This method calculates the output from the given inputs according to the gates’ function (AND, OR, NOT, XOR), it does not have a body here and Its implementation is in each subclass.
+ boolean isDataValid(String data)	1. Check if the length of the data, if it is not the same as numOfBit, returns false. 2. Check if all the char in the data string is either a “0” or a “1” (Basically, check if the string is a binary number). If not return false. *Hint : Use “charAt()”*

	3. If the data is at the correct length of bits and is indeed a binary number. Return true.
+ getter/setter	For setters, For the number of inputs 1. If it's a NOT-gate then the number of inputs is always 1. 2. If it's another kind of gate then the number of inputs must be at least 2, if not make it so. The number of bits must be at least 1, if not make it so. If the output is not valid data, do nothing. *Do not make a getter/setter for datas ArrayList, we will be creating custom methods for them.*
+ String getData(int n)	Getter for datas ArrayList. Return the string at index n from datas ArrayList.
+ void setDatas(int n, String Data)	Setter for datas ArrayList. Make sure String Data is valid (use method isDataValid that you implement for this class), if not then do nothing. Store String Data into datas ArrayList (at position n).

3.2.2 **Class AndGate** /* Already Provided*/

This class represents AND-gate. It's a type of BaseLogicGate. All logic gates calculate their output bit-by-bit. For AND-gate, The output bit is 1 only when all the input bits are 1.

Ex.



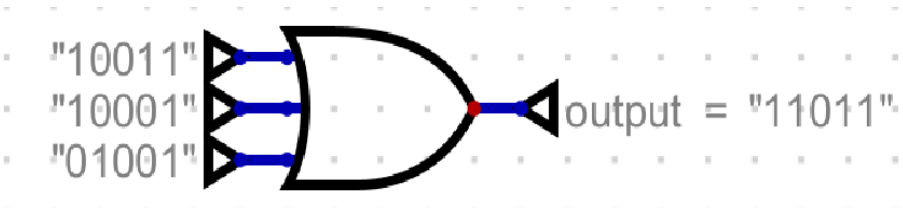
Methods

Method	**Description**
+ AndGate(int NumOfInput, int NumOfBit)	Constructor for AndGate. Use super constructor.
+ String evaluate()	(Use this as an example for other classes) This method calculates the output from the given inputs for AND-gate. The output bit is 1 only when all the input bits are 1s. Set output to the calculated string and also return the calculated string. Ex. "101111" AND "001101" AND "101110" = "001100"

3.2.3 **Class OrGate** /* you must implement from the scratch*/

This class represents Or-gate. It's a type of BaseLogicGate. All logic gates calculate their output bit-by-bit. For Or-gate, The output bit is 1 when any of the input bits are 1.

Ex.



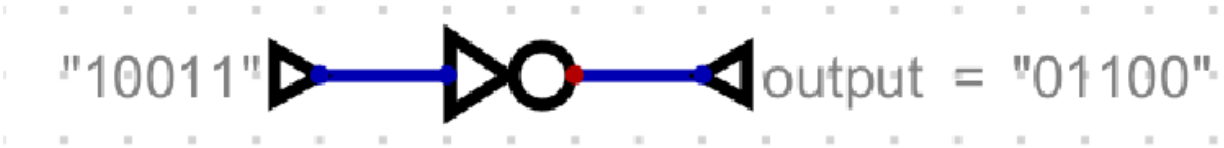
Methods

Method	**Description**
+ OrGate(int NumOfInput, int NumOfBit)	Constructor for OrGate. Use super constructor.
+ String evaluate()	This method calculates the output from the given inputs for Or-gate. The output bit is 1 when any of the input bits are 1. Set output to the calculated string and also return the calculated string. Ex. "101111" OR "001101" OR "101110" = "101111"

3.2.3 **Class NotGate** /* you must implement from the scratch*/

This class represents NOT-gate. It's a type of BaseLogicGate. All logic gates calculate their output bit-by-bit. For NOT-gate, The output bit is the inverse of the input.

Ex.



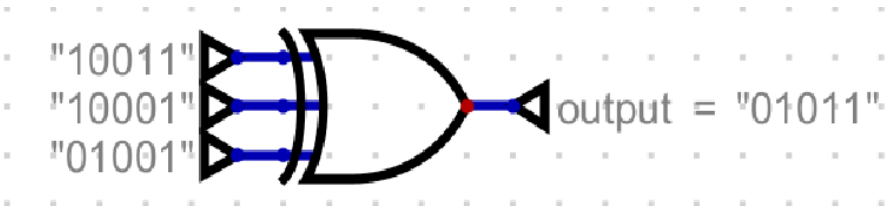
Methods

Method	Description
+ NotGate(int NumOfBit)	Constructor for NotGate. Use super constructor. NOT-gate always has only 1 input.
+ String evaluate()	This method calculates the output from the given inputs for NOT-gate. The output bit is the inverse of the input bit. Set output to the calculated string and also return the calculated string. Ex. "101111" NOT = "010000"

3.2.4 **Class XorGate** /* you must implement from the scratch*/

This class represents XOR-gate. It's a type of BaseLogicGate. All logic gates calculate their output bit-by-bit. For XOR-gate, The output bit is 1 when the number of 1s from the inputs is odd.

Ex.



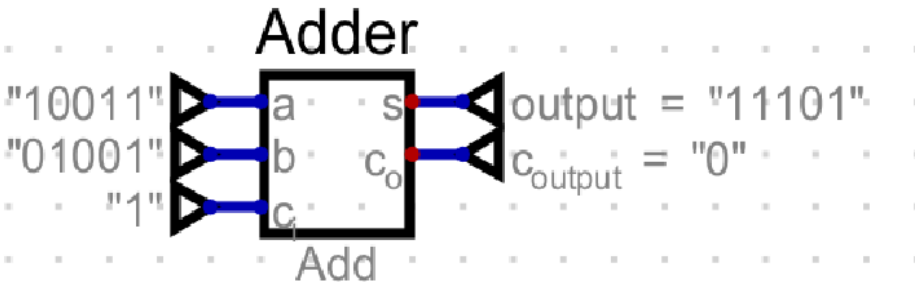
Methods

Method	**Description**
+ XorGate(int NumOfInput, int NumOfBit)	Constructor for XORGate. Use super constructor.
+ String evaluate()	This method calculates the output from the given inputs for XOR-gate. The output bit is 1 when the number of 1s from the inputs is odd. Set output to the calculated string and also return the calculated string. Ex. "101101" XOR "001101" XOR "101010" = "001010"

3.2.5 **Class Adder** /*Partially Provided*/

This class represents a full adder (which is a type of basic logic gate), a component which receives three numbers, two of which being numbers that are to be added together (A+B) and the other number is the carry-over bit that will be added as well (A+B+C). This adder comprises of AND-gates, OR-gates, and XOR-gates of which, if you've created them correctly, the adder should be working properly. Your work here is to create the constructor for the class.

Ex.



Fields

Field	**Description**
- String c	The carry-over bit
- String outputC	The carry-over output bit

Methods

Method	**Description**
+ Adder(int numOfBit, String c)	/* you must implement this method*/ Constructor for Adder. Use super constructor. Only has 2 inputs. Set c to c. Set outputC to "".

+ String evaluate()	This method calculates the output from the given inputs and c.
+ setter/getter	

3.3. Package Utils /*Already Provided*/

3.3.1. Class FormatString /*Already Provided*/

This class has a method that will format toString BaseLogicGate class' output.

Methods

Method	Description
+ String toString(BaseLogicGate gate)	Return output in a designated manner.

4. Scoring Criteria

The scoring criteria is listed below. There are JUnit files to test as well as the UML Diagram file. The total score of this section is 70, which will be factored into a net score of 10.

1. OrGateTest **15 points**
    a. constructorTest 2 points
    b. constructorTest2 2 points
    c. isDataValidTest 1 point
    d. setDataTest 1 point
    e. getter/setterData 2 points
    f. toStringTest 1 point
    g. evaluateTest1 3 points
    h. evaluateTest2 3 points
2. NotGateTest **15 points**
    a. constructorTest 2 points
    b. constructorTest2 2 points
    c. isDataValidTest 1 point
    d. setDataTest 1 point
    e. getter/setterData 2 points

f. toStringTest 1 point

g. evaluateTest1 3 points

h. evaluateTest2 3 points

3. XorGateTest **15 points**

a. constructorTest 2 points

b. constructorTest2 2 points

c. isDataValidTest 1 point

d. setDataTest 1 point

e. getter/setterData 2 points

f. toStringTest 1 point

g. evaluateTest1 3 points

h. evaluateTest2 3 points

4. AdderTest **10 points**

a. constructorTest 2 points

b. getterSetterTest 2 points

c. evaluateTest1 2 points

d. evaluateTest2 2 points

e. evaluateTest3 2 points

5. PolymorphismTest **5 points**

a. testPoly 5 points

6. UML diagram **10 points**

a. All classes are present with correct variables and methods 5 point

b. Inheritance and abstract structures are correct 5 point

Export jar (**q3.jar**) file that includes source code and .class files and **put it in root folder** of your project. YOU MUST EXPORT JAR FILE. IF YOU DO NOT, YOUR CODE WILL NOT BE MARKED.