ICSI 404 – Assignment 1, the humble bit

For this assignment, you need to create a class, called bit, to represent one bit. You must internally (private) use an integer to represent the one bit; the valid values are 0 (off) and 1 (on).

You must fully implement this interface (source file is provided):

```
public interface IBit {
void set(int value); // sets the value of the bit
void toggle(); // changes the value from 0 to 1 or 1 to 0
void set(); // sets the bit to 1
void clear(); // sets the bit to 0
int getValue(); // returns the current value
bit and(bit other); // performs and on two bits and returns a new bit set to the result
bit or(bit other); // performs or on two bits and returns a new bit set to the result
bit xor(bit other); // performs xor on two bits and returns a new bit set to the result
bit not(); // performs not on the existing bit, returning the result as a new bit
@Override
 String toString(); // returns “0” or “1”
}
```

You must implement the logic for these operations – you cannot use the logic (&, &&, |, ||) operators for these operations. You may use “if” or “switch”.

You must provide a file (bit_test.java) that has a method (void runTests()). Each method of “bit” must be tested in bit_test.java. These tests could throw an exception on failure, print expected and actual values or print “pass” or “fail”, for example. The tests must be adequate to prove that your bit class really works. For example – not should test both cases (start with 0, not yields 1 and start with 1 and yield a 0). Your test cases should be independent of one another. You do not want one failed test to cause another failed test to occur. Your main method should call runTests on bit_test. There should be output that allows a grader to determine that the tests pass or fail. The graders will also have different tests that they will run.

You must submit buildable .java files for credit.

Rubric	Poor	OK	Good	Great
Comments	None/Excessive (0)	“What” not “Why”, few (5)	Some “what” comments or missing some (7)	Anything not obvious has reasoning (10)
Variable/Function naming	Single letters everywhere (0)	Lots of abbreviations (5)	Full words most of the time (8)	Full words, descriptive (10)
Unit Tests	None (0)	Partial Coverage (7)	All methods covered, needs more cases (13)	All methods/cases covered (20)
Accessors/Mutators /toString	None (0)	Some implemented (7)	All implemented, some fail (13)	All implemented, all tests pass (20)
And	None(0)		Implemented, wrong (5)	Implemented, correct (10)
Or	None(0)		Implemented, wrong (5)	Implemented, correct (10)
Not	None(0)		Implemented, wrong (5)	Implemented, correct (10)
Xor	None(0)		Implemented, wrong (5)	Implemented, correct (10)