What you will do

Turn in: **Gate.java**

Requirements and method names for you *Gate* class are given in the UML diagram below. Name your fields and methods exactly as specified so that the included test program will work for your class.

Gate
+ IN : int = 1 + OUT : int = –1 + CLOSED : int = 0 – mSwing : int
+ Gate() //constructor + setSwing(direction:int) : boolean + open(direction:int) : boolean + close() : void + getSwingDirection() : int + thru(count:int) : int + toString() : String

Create a *Gate* class that can be used to represent a gate for a livestock pen containing animals (in our case those animals will be snails). There will be a single field for each instance of *Gate,* named *mSwing* which is an *int.* The *mSwing* field will contain the current swing direction of the gate. To make our programs more readable, create the following three *public static final int* values to indicate the swing direction.

- *OUT* = −1 to swing outward allowing the snails to leave the pen or enclosed area
- *IN* = 1 to swing inward allowing the snails to enter the pen or enclosed area
- CLOSED = 0 to swing closed preventing any snails from movement between the pen and enclosed area

Note that each gate will only allow snails to move in a single direction, either *IN* or *OUT* of the pen. When an instance of *Gate* is first constructed, it should be closed to prevent any movement (that is the job of the constructor).

As shown in the UML diagram above, create a getter (*getSwingDirection()*) and a setter (*setSwing()*) for the the *mSwing* field **.** When setting the *mSwing* direction through the *setSwing()* method, return a boolean value to indicate if the swing direction parameter was valid (*true,* which means successfully set) or if an invalid swing direction parameter was given (*false,* which means not successfully set). In the case of an invalid swing direction parameter do not change the current value of *mSwing*.

When attempting to open the gate, the caller is required provide a swing direction. The *open()* method performs this task by receiving a swing direction parameter and then calling the *setSwing()* helper method to set the swing direction to the passed value. However, since it only makes sense to open a gate to either let snails in or out of the pen, the open() method should first ensure that the direction parameter value is either IN or OUT before calling the setSwing() helper method. If the input direction parameter is neither IN nor OUT, then open() should return false (and not change the gate). Otherwise, it should open the gate and return true to indicate that the gate was successfully opened in one of the two valid directions. The *close()* method simply closes the gate (it is a void method).

The primary purpose of the gate is to allow snails to pass through the gate in either the *IN* or *OUT* direction. Suppose *n* snails attempt to go through an instance of *Gate.* If this gate is set to the swing *OUT* position, the snails will leave the pen and the total number of snails in the pen will be decreased. If the gate is set to the swing *IN* position, snails will be entering the pen and the number of penned snails will be increased. If the gate is closed, there should be no change to the number of snails in the pen.

Finally, you are required to override the *toString()* method to exactly match the output shown in the following samples (tip: use copy/paste to avoid spelling/typing mistakes).

// for a gate that is closed
This gate is closed

// for a gate that has opened to swing IN
This gate is open and swings to enter the pen only

// for a gate that been opened swing OUT
This gate is open and swings to exit the pen only

// for a gate that has a swing value other than IN, OUT, or CLOSED
This gate has an invalid swing direction

After you have created your *Gate* class in the *Gate.java* file you should test it using the JUnit tests in the supplied Android Studio. Running your JUnit test is simple: just right-click on the file *GateUnitTests.java* and choose Run 'GateUnitTests". These tests will call the *Logic.process()* method in the *Logic.java* file, which will instantiate a *Gate* object and invoke some of its methods.

Source code aesthetics *(commenting, indentation, spacing, identifier names)*:
If you choose to have your solution reviewed via the optional peer grading mechanism you'll be evaluated against a number of criteria. For example, you are required to properly indent your code and will lose points if you make significant indentation mistakes. No line of your code should be over 100 characters long (even better is limiting lines to 80 characters). You should use a consistent programming style, including the following:

- Consistent indenting
- Use of "whitespace" and blank lines to make the code more readable
- Use of comments to explain pieces of complex code

```java
package mooc.vandy.java4android.gate.logic;

/**
 * This file defines the Gate class.
 */
public class Gate {
    // TODO -- Fill in your code here
    public static final int IN = 1;
    public static final int OUT = -1;
    public static final int CLOSED = 0;

    private int mSwing;

    public Gate(){
        mSwing = CLOSED;
    }

    public boolean setSwing(int direction) {
        if(direction == IN || direction == OUT || direction == CLOSED) {
            mSwing = direction;
            return true;
        }
        return false;
    }

    public boolean open(int direction) {
        if (direction == IN || direction == OUT) {
            this.setSwing(direction);
            return true;
        }
```

```
31          return false;
32      }
33
34      public void close() {
35          mSwing = 0;
36      }
37
38      public int getSwingDirection() {
39          return mSwing;
40      }
41
42      public int thru(int count) {
43          if(mSwing == IN)
44              return count;
45          else if(mSwing == OUT)
46              return - count;
47          else
48              return 0;
49      }
50
51      @Override
52      public String toString() {
53          if(mSwing ==0)
54              return "This gate is closed";
55          else if(mSwing == IN)
56              return "This gate is open and swings to enter the pen only";
57          else if (mSwing == OUT)
58              return "This gate is open and swings to exit the pen only";
59          else
60              return "This gate has an invalid swing direction";
61      }
62
63
64  }
```