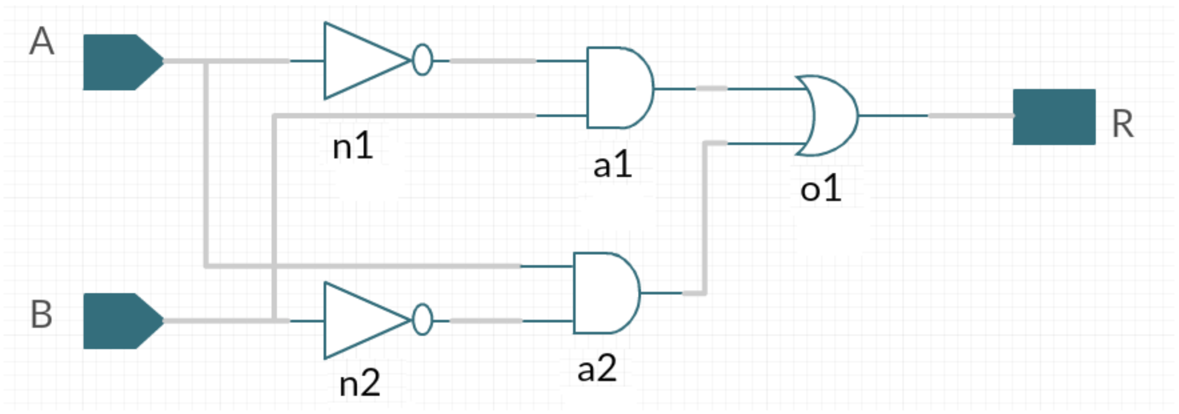
Incremental Development of a Java Program

You are to read this document in its entirety once or twice (and look at the provided code) before you do anything. You can't just code and expect to be successful. You have to understand the program ideas and constraints before you have any hope of completing this project. The code that you write should be elegant. If you sense that your code is ugly, stop, think, and ask questions before you proceed. This assignment is for you to learn the structure of a codebase before programming and before hacking. It is to give you first-hand experience at incremental development -- that is building and designing programs one unit of functionality (a.k.a. 'feature') at a time.

A **logic circuit** is shown below. It consists of 2 input ports (**A**,**B**) and 1 output port (**R**). This circuit determines whether the TRUE/FALSE value at port **A** equals the TRUE/FALSE value at port **B**. The true/false answer is at port **R**.



Ports are public inputs/outputs that are externally visible to a circuit and to which a TRUE/FALSE value can be assigned to each input port and a TRUE/FALSE value can be read from each output port. There are AND, OR, and NOT gates, which have input and output **pins.** AND and OR have precisely 2 input **pins**, named "i1" and "i2", and one output port named "o1". NOT gates have one input pin ("i1") and one output pin ("o1"). These names are NOT visible in a circuit diagram -- they represent standard naming conventions. Each **wire**, shown in gray, connects an input port or output pin of a gate to an input port of a gate or output port. Wires don't have explicit names.

> **Note**: unlike a previous homework assignment, this design allows a circuit or gate to have multiple input AND multiple output pins/ports.

Java code that implements this circuit is shown below:

```
InputPort a = new InputPort("a");
InputPort b = new InputPort("b");
OutputPort r = new OutputPort("r");

Not n1 = new Not("n1");
Not n2 = new Not("n2");

And a1 = new And("a1");
And a2 = new And("a2");

Or o1 = new Or("o1");

new Wire(a,n1,"i1");
new Wire(n1,a1,"i1");
new Wire(b,a1,"i2");

new Wire(a,a2,"i1");
new Wire(b,n2,"i1");
new Wire(n2,a2,"i2");

new Wire(a1,o1,"i1");
new Wire(a2,o1,"i2");
new Wire(o1,r);
```

As you know, every logic circuit is an object diagram, which means it has a metamodel with constraints that verifies that circuits, such as the above, are sane.Your assignment is to incrementally develop a Java program that allows you to:

- define logic circuits in terms of input ports, output ports, AND, OR, NOT gates, their pins, and wires,
- print the tuples of a database that defines a circuit that you create using Java constructor calls (such as the above),
- evaluate metamodel constraints to verify that a circuit definition is "sane", and
- evaluate the functionality of a circuit given truth values of its inputs (ex: assigning port A above to be TRUE, port B to be FALSE, and reading FALSE from port R).

Gates Netbeans Project

Here is the "shell" NetBeans project (which you can translate into Eclipse if you like). I have removed the bodies of almost all methods and have removed some fields (created for convenience in my implementation). You likely don't need to add more methods, but you might want to add some fields. You'll recognize the empty methods as there are TODO comments. Anyways, what you add to this program is up to you. The minimal program is given. It compiles, but obviously does not run. You will have to replace empty or useless method bodies with something meaningful.

The project consists of 5 packages, one of which is RegTest -- a version of which was posted. Use the version of RegTest that I give you for this project.

Watch the animated GIF below. Recall an earlier lecture where I showed you how to explain a class diagram in simple, incremental way. Well, I'm doing the same thing here but in a different way. There are 4 functionalities in this program:

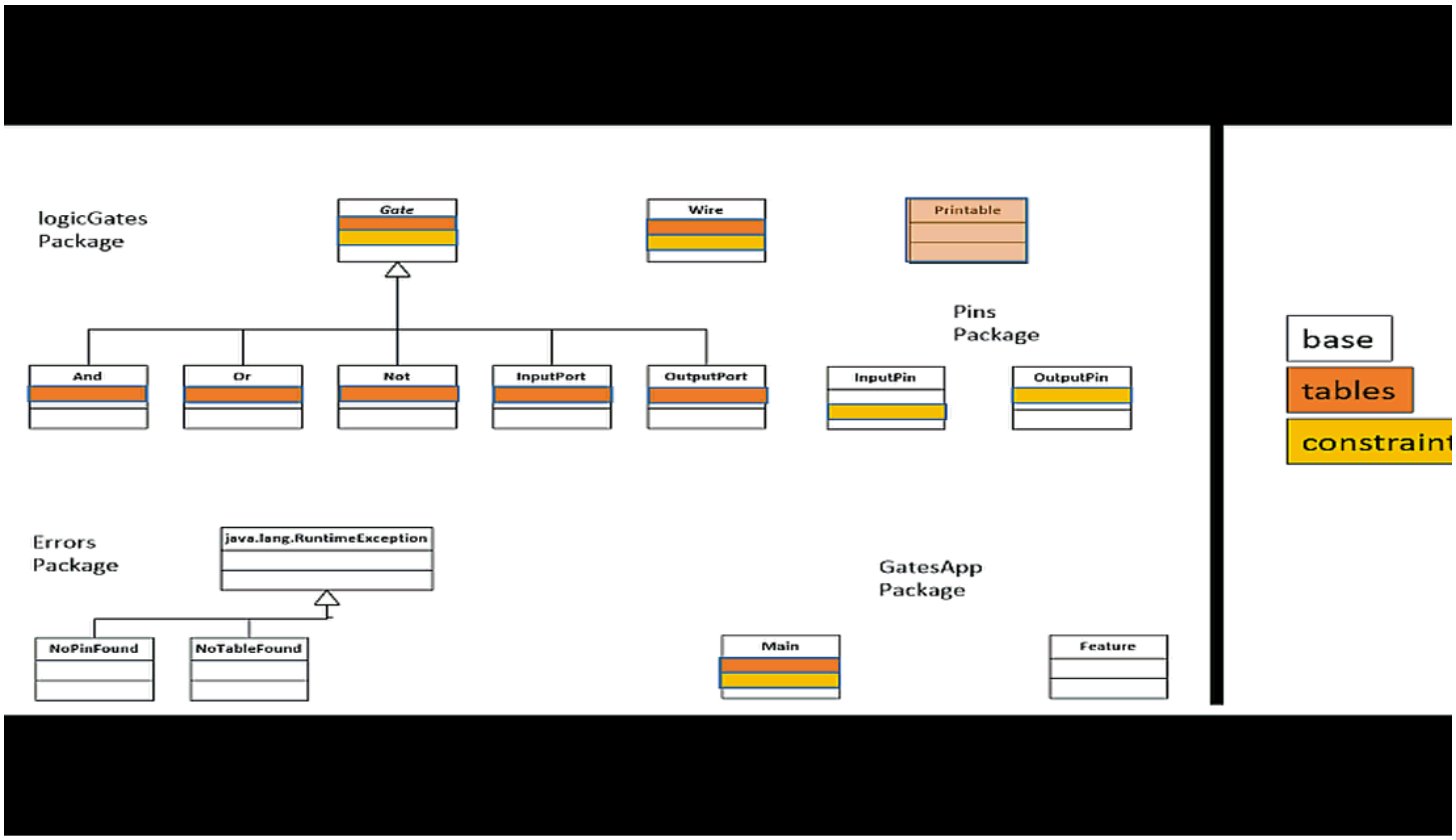
- **Base** -- which allows you to encode circuits (as above)
- **Table** -- which allows you to print tables
- **Constraints** -- that make circuits sensible (these are metamodel constraints that you are to evaluate)
- **Eval** -- the ability to place a boolean value on each input pinand extract a boolean value at each output pin.

Watch the animated GIF to see the class contents of the Base application, the additions made by Table functionality, then the Constraint functionality, and finally the Eval functionality. **You are to build, debug, and test one functionality at a time, not doing it in a big-bang mess.**

In the Gates Project, there are 3 packages:

- The **logicGates** package defines an abstract class Gate and concrete subclasses for each kind of gate: And, Or, Not, InputPort and OutputPort. There is also a Wire class whose instances are wires.
- The **Pins** package contains a pair of classes: one for input pins and another for output pins. You will discover errors in designs -- such as references to undefined pins, undefined tables, and values (truth values that are input and output from circuits). The **Errors** package contains 4 such classes, all subclasses of **java.lang.RuntimeException**, but not all are needed at the start of your programming.
- The **GatesApp** package, with a Main, Value, and Feature class.

You will also find MainTest.java, which has 3 fully-defined regression tests complete with correct outputs.



A Tour of Gates

Look at the Main class, a sketch of which is given here. There's not much to it, except for the **aCircuit( )** method which builds the logic circuit of the figure at the top of this assignment, invokes the code to print out its model database (i.e., the Gate tuples/objects and wires that define it), invokes a Java method to evaluate all metamodel constraints, and evaluates the circuit by defining 'boolean' values at its A and B inputs and reading its 'boolean' output R. Study this code, because you will need to run it and make sure that your implementations of other classes support its required functionality.

Now, I quoted 'boolean' because circuits really use 3-value logic. For example, an InputPort could be set to the TRUE value, or it could be set to the FALSE value, or it might not be set (in which case it has an UNKNOWN value). I use a Java enum to define Value below:

```
public enum Value { TRUE, UNKNOWN, FALSE }
```

The Feature file is a set of constants: base, tables, constraints, and eval. **You should set all but base to be false**. You will alter the values in this file as you proceed to implement its functionality.

Look at the logicGates.Gate file. You will see comments like:

```
// TABLE METHODS
// CONSTRAINTS METHODS
// EVAL METHODS
```

You will find these comment markers to partition each file, as barriers of how to develop your program incrementally:

- Your first task is to implement every Java file up to the `@Feature(tables)` barrier. This implements base functionality.
- Your next task is to update the java file and set tables = true. And then implement the methods up to the `@Feature(constraints)` barrier. Doing so implements the table and table printing functionality.
- Your next task is to update the java file and set constraints = true and then implement the methods up to the `@Feature(eval)` barrier. The constraints that you are to check are:
  1. every gate of type G (e.g. And, Or, Not...) has a unique name.
  2. every gate of type G has all of its inputs used (see 5 below)
  3. every gate of type G has all of its outputs used (see 6 below)
  4. any constraint that you might additionally want to add. I don't expect you to add any, but if you do, go ahead and document it.
  5. every InputPin of every Gate must be connected via a wire to an OutputPin. Remember: every InputPort is a Gate with precisely 1 OutputPin.
  6. every OutputPin is connected to an InputPin of a Gate. Remember: every OutputPort is a Gate with precisely one InputPin.
- Your last task is to update the java file and set eval = true and implement all methods after the `@Feature(eval)` barrier. At this point, when you run the regression tests, you will be able to test your program's output and compare it to the output of correct solutions in the CORRECT directory.

  A hint for the eval methods. The only state that is maintained for any circuit is that of InputPorts, each of which has a Value field that can be set. So in the above circuit, there are 2 InputPorts (named A and B) that you will have to set to a TRUE or FALSE value before you evaluate the circuit. To get the output of a circuit, you get the value of an OutputPort (in the circuit above, it is named R). And by following wires and primitive gates backwards, you can obtain the inputs to primitive gates, compute their output, and pass along their output to the next gate and so on. As a hint, look at the getValue() method of logicGate.And. I commented out my implementation, rather than removing its code.

As you implement each task, you should consult the files in the CORRECT directory (which is the correct output for MainTest.java). You'll see that you'll have to match the printed output.

Hint: As you turn on features:

base first, then (base,table), then (base,table,constraints), and finally all (base,table,constraints,eval)

A test is performed for each configuration. The test really is nothing for base (as base outputs nothing), but things kick in as you add more features. You should add your own tests to make sure of the sanity of your implementations.

What to Submit

All of the below in a zip file **(including your Netbeans or Eclipse Project)**. The zip file must unzip into **<yourName>/<YourFilesAndDirectories>**.

1. Your program needs to run correctly on Linux machines, even though you may have developed them on Macs and Windoze. The TA will grade your program running on Linux.

2. A short description that the Grader needs to know to run your program, other than the above.

3. Run your tool on all "test/GatesApp/MainTest" given in the GatesShell.zip file.

4. A short writeup explaining any additional tests that you have added to your program to verify that it works.

5. A PDF file (in the required format) that the Grader should read to provide any information that is not obvious. The contents of the PDF file can be minimal.

A critical part of any design is clarity and understandability. Hence, you will be graded on the clarity of your project and its ability to work correctly. Sloppy code, documentation, or anything that makes grading or understanding your program difficult will cost you points. Beware, some of these "beauty" points are subjective.

Remember: No late assignments/submissions will be accepted.