Introduction

- **Ultimately, the project score should match the appropriate score on the rubric listed on the course policies page of the course. See http://cse.osu.edu/software/web/policies.html**
- For this project, the implementation is worth 5 points and the `JUnit` test fixture is worth 5 points.
- To test the student's implementation, run the given `JUnit` test fixture (`NaturalNumber3Grade`).
- Note: Some suggested point deductions for specific issues are in parenthesis. Also, if the project contains any bugs, OR is missing any test cases the project should NOT receive a 10.

Implementation

- If the student's implementation contains any bugs (i.e. fails any test cases), there should be a penalty applied.
- Any solutions that are complex (i.e., deviate significantly from the expected solution), contain loops, use recursion, or declare additional methods should be penalized. (-1/2 ea. instance)
- If the student failed to uphold the correspondence, a 1 point penalty should be applied. Even if the solution passes all test cases. (Note: if the student's solution also contains bugs, further penalties should be applied)

Constructors

The following is a list of the constructors and what a good (or expected) solution is.
If the student's solution violated the kernel purity rule in each constructor take off -1 point, otherwise -1/2 point each (as indicated below).

- `createNewRep()` : Set `this.rep = ""`.
- `NaturalNumber3()` : Set the abstract value of `this` to 0. (i.e. call `createNewRep()`).
- `NaturalNumber3(int)` : Set the value of `this` to the value of the `int`. Note: the implementation should handle the special case when the `int = 0`. The solution should not call the method `setFromInt(int)`, this is a violation of the kernel purity rule. (-1/2)
- `NaturalNumber3(String)` : Similar to `int` constructor, special case when the incoming `String = "0"`. The solution should not call the method `setFromString(String)`, this is a violation of the kernel purity rule. (-1/2)
- `NaturalNumber3(NaturalNumber)` : Similar to `String` constructor. Same special case as `String` constructor. The solution should not make a call to `copyFrom(NaturalNumber)`, this is a violation of the kernel purity rule (-1/2). It is okay for the solution to call `toString()` and `isZero()` on the incoming `NaturalNumber`, this is not a violation of the kernel purity rule.

Kernel Methods

The following is a list of the kernel methods and what a good (or expected) solution is:

- `multiplyBy10(int)` : Special case: when `this = 0` and `k = 0` (the value of this should not change). Other cases: the value of `k` can be concatenated to `this.rep`. If the solution does not handle the special case when `this = 0` and `k = 0` (-1/2 point penalty).

- `divideBy10()` : Special case when `this = 0`. Other cases, take substring of `this.rep` and grab the last digit in `this.rep` and return this value.
- `isZero()` : Check `this.rep` for the empty string (simplest solution is a one liner). Any solution which uses if-statements to set or return `Booleans` should receive comments (not necessarily a penalty) that the solution can be simplified.

JUnit

See the 'JUnitGrading' document (in the 'General' folder) for details on grading JUnit test fixtures.

- The JUnit test fixture MUST be organized based on the design pattern discussed in class (as well as in several examples.).
- The JUnit should not test each methods using every constructor. For example, when testing isZero the test fixture should not use more than one type of constructor for the isZero method.

Required Test Cases

The following test cases are required. It is suggested that 1/3 to 1/4 a point be deducted for each missing test case (the [total] amount deducted should depend on the total number of cases missed, and other factors that are being penalized for in the test fixture.)

- Default Constructor
- `Integer` Constructor (2 cases total)
  1. Construct using 0 - base case
  2. Construct using non-zero value - normal case
- `String` Constructor (2 cases total)
  1. Construct using "0" - base case
  2. Construct using non-zero - normal case
- `NaturalNumber` Constructor (2 cases total)
  1. Construct using a `NaturalNumber` with value 0 - base case
  2. Construct using a `NaturalNumber` with non-zero value - normal case
- MultiplyBy10 (4 cases total)
  - `NaturalNumber` is the constructed actual value and K is the argument passed into `multiplyBy10()`.
  1. `NaturalNumber` = 0 : K = 0
  2. `NaturalNumber` = 0 : K > 0
  3. `NaturalNumber` > 0 : K = 0
  4. `NaturalNumber` > 0 : K > 0
- DivideBy10 (3 cases total)
  - `NaturalNumber` is the constructed actual value, and Ret is the value that should be returned by the `divideBy10()` method.
  1. `NaturalNumber` = 0 : Ret = 0
  2. `NaturalNumber` > 0 : Ret = 0
  3. `NaturalNumber` > 0 : Ret > 0

- IsZero (2 cases total)
  - `NaturalNumber` is the constructed actual value, and Ret is the value that should be returned by `isZero()`.
  1. `NaturalNumber` = 0 : Ret = true
  2. `NaturalNumber` > 0 : Ret = false