Object-Oriented Programming

Worksheet 4

Tom Lin Jamie Willis

February 1, 2021

1 Domain modelling: Parity

1. Integers can be divided into two categories, *even* and *odd*, this property is also known as the *parity*[1] of a number.

   Basic additions and multiplications with even and odd numbers are a special case of modulo 2 arithmetic with the following rules:

   - $even \pm even = even$
   - $even \pm odd = odd$
   - $odd \pm odd = even$
   - $even \times even = even$
   - $even \times odd = even$
   - $odd \times odd = odd$

   (a) Implement two classes, `Even` and `Odd`, they should be constructed by a single integer of the matching parity. Identify and assert the invariant in your constructor, throwing exceptions where sensible.

   Also implement the corresponding `toString`, `equals`, and `hashCode` methods.

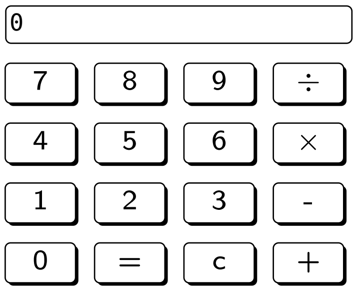
   † (b) For both `Even` and `Odd`, implement the method `add` and `multiply` such that the addition and multiplications rules are satisfied. The return type of these two methods must be of either parity and not a plain `int`.

   (c) Test your parity classes to verify whether the returned types match the above rules.

[1] https://en.wikipedia.org/wiki/Parity_(mathematics)

2 Domain modelling: Calculator

†† 1. Consider a very basic 8-digit integer only four-function (add, subtract, multiply, divide) calculator. The calculator has the following appearance:



Like an object in the real world, you should be able to have more than one calculator.

(a) A calculator can be modelled very simply using a class with two public methods:

- `int value()` – returns the current result of the calculator so far.
- `void press(char key)` – executes a valid key press or fails otherwise.

It should only support integer operations. In order to make a clean and maintainable design, it is good to use plenty of private methods to break up responsibility.

- Implement a private method `private void pressOp(char key)` which handles button press for `+`, `-`, `*`, `/`.
- Implement a private method `private void pressDigit(char key)` which appropriately handles the pressing of a digit key (`0-9`).
- Implement a private method `private void eval()` which is used to handle `=`.
- Implement the public method `public void press(char)` using the `pressOp(char)`, `pressNum(char)`, and `eval()`, you should also implement the clear operation (`C`) here.

(b) Test your calculator to verify whether the behavior matches a real world four function calculator. Make sure having more that one calculator is not an issue, i.e. calculators do not interfere with one another.

(c) Suppose we later found out that real world calculators have finite battery life, so we need to implement a power button to turn the calculator off when not in use.

Implement the power button where any input made while the calculator is off will not be registered. The clear button `C` should power on the calculator. For simplicity, once switched on, the calculator remains on.

† † † (d) Make use of the calculator model by implementing an user interface of your choice, e.g. command line or a GUI, to produce a fully functional calculator program.