Tutorial 4: Iteration

Part I: Tutorial Exercises

These exercises are designed to help you practice using and designing `Iteration` abstractions.

1. Write a program named `First10Primes` that creates a list containing the first 10 prime numbers. Print the elements of this list out on the standard output.

2. Write a program named `OddAlphabetList` that creates a list containing all the letters of the English alphabet, whose character codes are odd numbers. Print this list out on the standard output using its iterator.

3. Write a program named `RandomNums` that creates a list containing 10 randomly generated numbers in the range `[1,100]`. Print the elements of this list out on the standard output.

4. Write a program named `OddAlphabet` that creates a list `l1` containing all the character codes of the letters of the English alphabet and uses an Iterator of `l1` to create another list `l2` containing a sub-set of `l1` that includes only the odd character codes. Print the elements of both `l1` and `l2` out on the standard output.

`l1` is a stored list.

5. Specify and implement a sub-type of `LinkedList<Integer>` called `IntegerLinkedList` and iteration abstraction `IntegerLinkedList.evenIterator` that returns an `Iterator` for only the even elements of the list.

6. Specify and implement a sub-type of `LinkedList<Integer>` called `PrimeLinkedList` and iteration abstraction `PrimeLinkedList.primeIterator` that return an `Iterator` for only the prime elements of the list.

Part II: Modified textbook exercises

6.1. Specify a procedure, `isPrime`, that determines whether an integer is prime, and then implement it using `PrimeList`. To do this, you need to design and implement the `PrimeList` class. Note that, unlike `LinkedList`, `PrimeList` is an auto-populated collection.

6.6. Implement the following iterator:

```
/**
 * @requires g contains only Integers
 * @effects
 *   if g or x is null
 *     throws NullPointerException
 *   else
 *     returns a generator that produces in order, each exactly one,
 *     all elements e produced by g, for which x.checker(e) = true
 */
static Iterator filter(Iterator g, Check x)
```

Here `Check` is a type whose objects have a method:

```
public boolean checker(Integer)
```

that determines whether its argument satisfies some conditions. For example, `checker` might determine if its input is an odd number.

(*) Hints:

- Create a class and give it some name, like `TEx6` (meaning "*textbook exercise 6*")
- In the same package, implement the class named `Check` that has the method named `checker` as given in the requirement. Alternatively, you could also implement `Check` as a private inner class of `TEx6`.
- Implement a generator `TEx6.FilteredGen` (i.e. an inner class of `TEx6`), which uses a `Check` object to filter the elements provided by the input `Iterator` to return the next element.
- Implement method `TEx6.filter` as specified in the requirement. It must create and return an instance of `FilteredGen` using the input arguments.
- Implement the main method `TEx6.main` which does the followings:
    1) Defines an `Integer` collection object using any of the collection-typed classes that you have learnt (e.g. `LinkedList`, `PrimeList`, `Vector`, `List`, etc).
    2) Create a `Check` instance.
    3) Call method `filter` with an `Iterator` from the collection and the `Check` object and display the result to the standard output.