COE528 (Winter 2024) - Lab3

General Lab Rules
All the necessary files of this lab should be in the lab3 directory.
All the Java files in this lab should have the following package declaration:

```
package coe528.lab3;
```

Duration: one week.

Objectives

- Use Java interface
- Use abstract class

In this lab, you will design and implement a few classes that model an odometer. An odometer is a counter with a specified number of digits. There is no *a priori* limit on an odometer's number of digits. The count can be incremented or decremented by one. If all digits are 9, incrementing will cause all digits to become 0. If all digits are 0, decrementing will cause all digits to become 9. The commands an odometer must support are *increment*, *decrement*, and *reset*. A query *count* provides the current value.

Design:

Since there is no limit on the number of digits, we cannot use a simple integer counter. In fact, we cannot return the value of the odometer as an *int*. We will return the value of the odometer as a *String*.

The odometer is designed as a sequence of digits, each with a value from 0 through 9. When the odometer is incremented, if the right-most digit is less than 9, it is incremented by 1. If that digit is 9, it is set to zero, and the process is repeated for the next digit.

For any given digit, the increment algorithm will be:

```
void increment () {
    if (value < 9)
      value = value + 1;
    else {
      value = 0;
      increment digit to the left
    }
}
```

Decrement is similar:

```
void decrement () {
    if (value > 0)
      value = value - 1;
    else {
      value = 9;
      decrement digit to the left
    }
}
```

Two questions need to be addressed:

- how is a digit represented?
- what happens with the left-most digit?

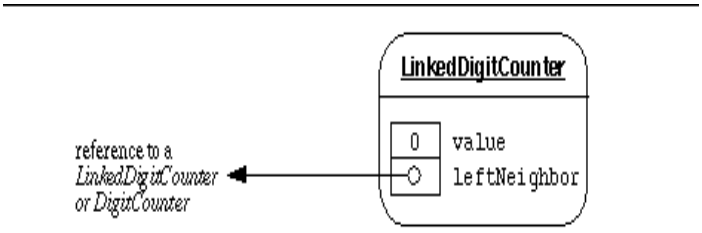
A digit can be represented as a counter that counts from zero to nine and has essentially the same operations as an odometer: commands *reset increment*, and *decrement*, and a *String*-returning query *count*. We name the class representing digits *LinkedDigitCounter*.

The left-most digit behaves like the others, except it never references the "digit to the left." We need a class to model this kind of digit: a digit with no left neighbour. We call it *DigitCounter*.

How do we put the digits together to form an odometer? The answer to this will simply be based on what a *LinkedDigitCounter* should know:

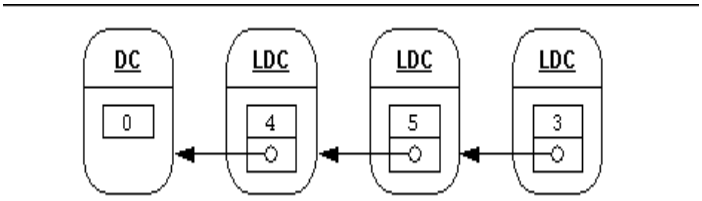
- it must know its current value;
- it must know its left neighbour.

A *LinkedDigitCounter* will look like this:



A *DigitCounter*, on the other hand, need only know its current value.

A four-digit odometer, for example, will consist of three *LinkedDigitCounter* instances, and one *DigitCounter*:



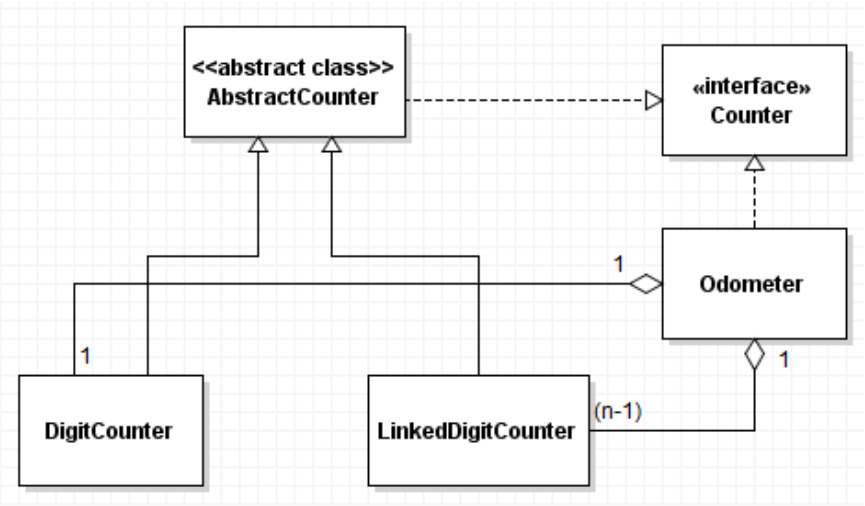
The above grouping represents the value 0453.

DigitCounter, *LinkedDigitCounter*, and *Odometer* have the same functionality. We can specify the functionality using an interface.

```
/* A basic up-down counter. */
public interface Counter {

   //The current value of this Counter as a String of digits.
   String count();

   //Increment this Counter.
   void increment();

   //Decrement this Counter.
   void decrement();

   //Reset this Counter.
   void reset();
}
```

Implementing the classes:

A common structural pattern is to define an abstract class that implements an interface, defines data common to all interface implementations, and provides some default method implementations. Concrete classes can then extend the abstract class.



The figure above shows the class diagram. As shown in the class diagram, we will define an abstract class *AbstractCounter* that implements the abstract methods of the interface *Counter* and defines an *int* component variable to contain the value of the counter. The abstract class also defines a constructor that initializes the value to 0. The classes *DigitCounter* and *LinkedDigitCounter* will both extend this abstract class. The letter *n* in the class diagram represents the number of digits in the odometer.

In the Netbeans program, click on Project > New Project and save it as "Ex1" on your lab3 directory.

In this lab, you **must** have one interface (*Counter* interface) and five classes (*AbstractCounter*, *DigitCounter*, *LinkedDigitCounter*, *Odometer*, *OdometerDriver*).

- Implement the class *AbstractCounter*. Be aware that the data will be shared by its subclasses, and those subclasses should be able to modify it.
- Implement the classes *DigitCounter* and *LinkedDigitCounter*. The class *LinkedDigitCounter* adds a new attribute, *leftNeighbor*. The value of *leftNeighbor* might reference a *LinkedDigitCounter* or a *DigitCounter*. Its type should be *Counter*, and the *LinkedDigitCounter* constructor should require a *Counter* as an argument.
- Implement the class *Odometer*. An *n*-digit odometer contains (*n*-1) *LinkedDigitCounter* instances, and one *DigitCounter* instance. We should be able to specify *n* while creating an *Odometer* instance. The value of *n* should be at least 1. The *Odometer* constructor **must** throw IllegalArgumentException whenever a value of *n* that is less than 1 is specified.
- Compile and test your implementation. A test driver, *OdometerDriver* class that contains the *main* method is provided in the Appendix-1. You **must** copy this *OdometerDriver* class and its *main* method as given. You **must not modify** this *main* method. Your console output **must** match the output given in Appendix-2 for a 0-digit, 4-digit, 5-digit, 6-digit odometer.

Submitting your lab

Deadline: The submission deadline is 11:59 PM EST, the night before the corresponding lab session during the week of February 12, 2024.

You must include the duly filled and signed standard cover page with your submission. The cover page can be found on the departmental website: Standard Assignment/Lab Cover Page

You must submit your lab electronically on D2L. Please zip up your NetBeans project containing all source files and submit it to the respective assignment folder on D2L.

Appendix-1: The test driver class OdometerDriver

```
import java.util.Scanner;
public class OdometerDriver {
    public static void main(String[] args){

        try{
            //read number of digits for odometer from console
            System.out.print("Enter number of digits for odometer: ");
            Scanner s = new Scanner( System.in);
            int numOfDigits = s.nextInt();

            Odometer odometer = new Odometer(numOfDigits);

            //increment 130 times and print the count.
            for ( int i = 0; i < 130; ++i ) {
                odometer.increment();
            }
            System.out.println(odometer.count());

            //decrement 31 times and print the count.
            for ( int i = 0; i < 31; ++i ){
               odometer.decrement();
            }
            System.out.println(odometer.count());

            //increment 1001 times and print the count.
            for ( int i = 0; i < 1001; ++i ){
                odometer.increment();
            }
            System.out.println(odometer.count());

            // decrement 1101 times and print the count.
            for ( int i = 0; i < 1101; ++i ){
                odometer.decrement();
            }
            System.out.println(odometer.count());

            //reset the odometer and print the count.
            odometer.reset();
            System.out.println(odometer.count());

            //decrement once and print the count.
            odometer.decrement();
            System.out.println(odometer.count());

            //increment once and print the count.
            odometer.increment();
            System.out.println(odometer.count());
            }
            catch(IllegalArgumentException ex){
              System.out.println("Number of digits in odometer must be at least 1");
            }
    }
}
```

Appendix-2: The Output

Console output for 0-digit odometer

```
Enter number of digits for odometer: 0
Number of digits in odometer must be at least 1
```

Console output for 4-digit odometer

```
Enter number of digits for odometer: 4
0130
0099
1100
9999
0000
9999
0000
```

Console output for 5-digit odometer

```
Enter number of digits for odometer: 5
00130
00099
01100
99999
00000
99999
00000
```

Console output for 6-digit odometer

```
Enter number of digits for odometer: 6
000130
000099
001100
999999
000000
999999
000000
```