sOFTWARE 2 PRACTICAL

## Classes & Exceptions

Week 5 – Practical 5

# Problem 1

Today’s problem is to implement a counter tally. A tally counter is a mechanical (as shown here), electronic, or software device used to incrementally count something, typically fleeting. One of the most common things tally counters are used for is counting people, animals, or things that are quickly coming and going from some location.

The aim of this exercise is to create our first type of object (class) that provides the same functionality (services) as a tally counter. Therefore, we need to think first of the design of the class before starting any implementation. What does a tally counter do?

1. We should be able to increment the counter by one (using one button)
2. We should be able to reset the counter to zero (using a second button)
3. Let assume that our counter has three digits (i.e. can count from 0 to 999). Once we have reached 999, the next click to increment the counter should throw a checked exception of type **InvalidOperationException** that you must implement yourself.
4. We should be able to read the value of the counter.

Points 1-4 represent the behaviour of our type, what we need to define as well are the attributes (to store the states of the object). There is only one state needed: the current value of the counter.

### Step 1

* Create a new project called “Tally counter”.
* Create a new class TallyCounter within a package named tools.
* Create an attribute counter of type int. Should it be public or private? (discuss with one of your peers)
* Create a constructor that takes no parameter and initialise the counter to 0.
* Implement the method toString(). The String returned should be of the form 001, 011 or 999. If you cannot find a way to do that (even after making a search on the internet), just return a String of the form 1, 11 or 999.

### Step 2

We want to test our code so far. Rather than writing a main(String[]) method in our class, we are going to create another class which will be our main application, e.g. the one we call to run the program. This class will use the type we defined in TallyCounter.

* In the project “Tally counter”, create a new class MainApp, and make sure it creates the default main(String[]) method.
* In the main method, create an instance tallyCount of the class TallyCounter
* Still in the main method, print the object tallyCount.
* Run the MainApp program (the output should be 000).

### Step 3

Now we need to implement the remaining services:

* Implement the checked exception **InvalidOperationException** class within the package tools.
* Implement the method increment() that increments the counter by one. Note that if the counter is already at 999, it throws a **InvalidOperationException** that you have implemented.
* Implement the method read(), which returns the current value of the counter
* Implement the method reset(), which reset the counter to zero

Notice that I did not tell you if the method should be private or public, and I did not say if the method should return a value or not. You must decide and justify your choice. Talk to one of your peers to see if you agree or not.

In addition, after each method implementation, you should check if your code is working using the MainApp class. What code should you write for each method?

# Problem 2 – Refactoring

After writing our class, our client realises that he wants more services/functionalities from our class TallyCounter. We need to modify our code to satisfy our client new requirements, we call this refactoring our code.

* The client wants to be able to create tally counter object with 3 or more digits. For example, a counter using 5 digits can go up to 99999. No counter should have less than 3 digits. In addition, the client would like to have counter with 3 digits as a default.
  + Do I need additional attributes?
  + Should I overload the constructor?
  + Which method(s) needs refactoring?
  + Once the refactoring is done, is your MainApp still working?
* The client would like to add a decrement() functionality, which decrement the counter by one.   
  Note that the counter cannot become negative, so if the counter is already at 0, decrementing the counter should throw an InvalideOperationException.
  + Make sure you write the necessary code to test your new method.