# INF5570v19 - Distributed Objects
# Mandatory Assignment 2 (Oblig 2)

Eric Jul
Department of Informatics
University of Oslo

February 21st, 2019

## Introduction

For questions regarding the task, feel free to post on the Piazza forum. If you have not done so yet, you should sign up to Piazza here:
http://piazza.com/uio.no/spring2019/inf5570/
We prefer that you use Piazza for all questions so that everyone will get the same advice.

## Exercise 1 - Break-even for call-by-visit

Call-by-visit may provide a performance improvement. However, if the parameter to an invocation is never called, call-by-visit is a waste of time and negatively affects performance. On the other hand, if the parameter is accessed billions of times during the call then call-by-visit has a positive influence on performance even when the parameter object is large. So for a given parameter object, there is a point between zero calls and one billion calls where call-by-visit performance changes from a negative impact to a positive impact. Such a point is called a break-even point. For small objects, the break-even point is usually around one.

| Unfortunately, call-by-visit and call-by-move do NOT work properly in our current implementation. You are to emulate call-by-visit using explicit moves of the parameter object. Such emulation will be described in the class - and sample code will be put on the web site. Obviously, such emulation has a negative effect on the break-even point – however, it does not change the overall experiment. |
|---|

Write and run a program to find the break-even point for the emulated call-by-visit with objects of a given size. Find break-even for a parameter object that has an attached instance variable referring to an array of integers with the following number of elements:

- 50
- 100
- 500
- 1,000
- 10,000

Experiment and find the break-even points. Explain why the array must be attached.
(Beware: the subject of *break-even point* will likely be one of the exam questions at the oral exam at the end of the course.)

## Exercise 2 - Time Collector

Write and run a program like `kilroy` that visits each active node and collects the local time in an array. Explain how you deal with times collected in different time zones.
Test the program locally on at least four machines. IF we get the problems of running on Planetlab fixed then run on at least four different machines in at least four different countries on Planetlab. Upon return, the program should print the collected times along with the node information (e.g., node-number, IP number, name, whatever...)

## Exercise 3 - Time Synchronization

Write a program that puts a time agent on a number of nodes and then at regular intervals (for testing, e.g., once a minute) makes a call to all its time agents to collect the time on each machine. Using a time synchronization method of your choice, compute a time that represents a reasonable approximation of the actual time. For each computed actual time, print the time and the difference for each of the times on the other nodes. Your program should work despite some of the nodes running the agents are inaccessible or down, i.e., you should use `unavailable` handlers.
Test the program locally on at least four machines. Test your program on Planetlab with at least four machines in at least four countries. The more nodes, the merrier... (but four is sufficient)
Explain the algorithm that you use for time synchronization. You are free to pick whatever algorithm that you fancy as long as you provide good arguments for your choice. Simple is good. Advanced is for the ambitious.

## Delivery in Devilry

The assignment must be submitted in devilry.

In addition to the source code, you should deliver a `.txt` file with the output from a run of the programs and a short report (2-5 pages) commenting upon your decisions and results. When relevant, feel free to include graphs showing your results. When relevant, explain your choice of how many iterations of the test that you ran and what kind of result, you are reporting, e.g., min/max, average values, median values, etc.

## DEADLINE

The deadline is set to **Tuesday March 12th, 2019 at 23:59:00 CET**.

**Have fun!**