Prof. Dr. Oscar Nierstrasz
Pascal Gadient, Mohammadreza Hazhirpasand

Solution
Assignment 07 — 30.10.2019 – v1.0
Liveness and Asynchrony

Exercise 1 (2.5 Points)

Answer the following questions about liveness and asynchrony:

a) When should you consider using asynchronous invocations? **Answer:**

- *When an object can distribute services among multiple clients.*
- *When an object does not immediately need the result of an invocation to continue doing useful work.*
- *When invocations are logically asynchronous, regardless of whether they are coded using threads.*
- *During refactoring, when classes and methods are split in order to increase concurrency and reduce liveness problems.*

b) In what sense can a direct invocation be asynchronous? **Answer:**

If you invoke helpers directly, but asynchronously without any synchronization, you are able to create asynchronous invocations with direct invocations. In this case the host is free to accept other requests while the host's caller may has to wait for the reply.

c) What is an "early reply"? **Answer:**

*"Early reply" is a feature that allows a host to perform useful activities after returning a result to the client. It means that the host can continue to work after its response to do some operations that are not of interest for the client (*e.g., *some cleanup work).*

d) What are "futures"? **Answer:**

"Futures" is a feature that allows a client to continue in parallel with a host until the future value is needed. It means that the client can start a process even if he doesn't need that result yet, and can go ahead as long as he doesn't really need the data that is to be computed by the host environment.

e) When are futures better than early replies? **Answer:**

In general, the idea is to choose futures instead of early replies if it is required that the client has to complete some extra tasks that are not related to the communication partner. For example, you can use futures if you know that a functionality will take a lot of time to execute. In this case a client initiates a call to the host requesting the functionality as soon as possible and proceeds with its work while the host remains computing. You can use early replies when you need to modify a data structure that you always want to keep optimized. In these cases a client asks to add an element to the structure and when the operation is done, the host can reply to the request and then proceed with further optimizations of the structure.

Exercise 2 (2.5 Points)

Answer the following questions about Java VM and implementation details:

a) Why does the call *Thread.currentThread().join()* not make much sense in a thread's concurrent run method? **Answer:**

Because the thread would wait for it's own death and that forces the thread to remain in the wait state. As these threads do not die on their own, it must be ensured manually that they will be terminated when the main application is closed.

b) Why can you encounter an IllegalMonitorStateException when calling notifyAll() in a code block that is not synchronized? **Answer:**

The method notifyAll() can only be called successfully when the caller has the monitor, because the caller will then give the monitor to the waiting callee that reacts to the notifyAll() statement.

c) What happens with the thread when the code execution gets to the end of the thread's run method (suppose no loop is involved in the run method)? **Answer:**

The thread transitions into the "terminated" state, i.e., *is not running any longer, similar to a closed application.*

d) Why do some Java apps not terminate, even though their GUI has been closed, and how can you mitigate that problem? **Answer:**

Per default, the window manager does not force threads to terminate. Consquently, the threads were not notified about the closing of the window and the threads remain active. As a result, the application remains running on the system although the user interface thread terminated. In other words, running or waiting background threads hinder the main app from exiting gracefully. A solution would be to call `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`.

e) Name one reason for using a while loop in a thread's run method. Justify your selected reason. **Answer:**

Graphical (user interface) draw calls that need to be performed repeatedly. There exist different approaches for this problem, like spawning new threads for every UI update, but then the configuration (= state) has to be injected into each newly instanciated thread. That's not efficient and often more complex than just letting a single thread loop several times, or even infinitely during the host application's runtime.

Exercise 3 (5 Points)

The winter is coming and so is the snow. We created a thread-based snowflake environment to simulate winter conditions. In this setup each snowflake represents one thread. Consequently, each snowflake draws itself on the Java Graphics2D object from the main application. Your task is to synchronize threads and beautify the snowflakes and their movement. We provide several tasks with importance of their order:

- Make the snowflake a runnable thread.
- Instantiate some snowflakes and draw them!
- Movement of the snowflakes has to be synchronized. If it wouldn't be synchronized, each snowflake would just move so fast, you wouldn't even see any movement at all. In other words, they should fall to the ground each with a constant velocity.
- Java2D draw calls on the Graphics2D object have to be synchronized, because this measure greatly reduces flickering. It can be handled in several different ways, but the easiest one is probably to combine the draw call synchronisation with the movement synchronisation.
- Add individual horizontal and vertical movement parameters to the snowflakes.

Some things that don't matter and thus are not mandatory for this exercise (however, you can still work on these):

- Snowflakes themselves don't need to rotate
- Overlapping is not an issue
- Different sizes of the snowflakes are not required

You will find additional comments and hints in the code that may help and guide you. The SCG Snowflake Simulation Environment is available on GitHub. You can find the project on `https://github.com/pgadient/concurrency_e07t03.git`. Please submit your project within a zip file. You can easily achieve that by exporting the project with the Eclipse export assistant. **Answer:**

You can find a sample implementation on GitHub at `https://github.com/pgadient/concurrency_e07t03_solution.git`. Please clone immediately as the repository will be turned back into a private one during the afternoon.