Marcel ZAUDER 16-124-836
Pascal GERIG 16-104-721

7.1 Several Questions

a) *When should you consider using asynchronous invocations?*
The intent of using asynchronous invocations is to avoid waiting for a request. This is achieved by decoupling the sending activity from the receiving activity. This can be useful if a client requests a value of a resource but do not need it immediately. When using asynchronous invocations the client can reutrn to its activities, while not needing to wait for the host to return the requested value, therefore this can lead to a possible improvement ins speed of the algorithm.

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

c) *What is an "early reply"?*
An "early reply" means that a host already returns a requested resource but afterwards still performs useful activities.

d) *What are "futures"?*
Futures are objects that used when a client requests a resource but do not need the value immediately. Therefore a Future object can be returned by the host before it is actually calculating its value, such that the client can perform activities which do not need the value of the future until it reaches the point where it needs the value and then waits for the value to be given to the future by the host.

e) *When are futures better than early replies?*
Futures should be used if the calculation of a requested resource can take a long time. In such a case the client can request a future from a host such that the host can start the calculation of the value of the requested resource and the client can perform other activities which do not need the Future's value. Therefore a huge waiting time can be prevented.

7.2 Several Questions 2

a) *Why does the call Thread.currentThread().join() not make much sense in a thread's concurrent run method?*
Thread.currentThread().join() blocks the current thread forever which can lead to a whole application preventing from exiting and therefore to be stuck. This is because the when calling these methods it will lead to the currentThread to wait for itself to terminate which will never happen because it will wait for itself.

b) *Why can you encounter an IllegalMonitorStateException when calling notifyAll() in a code block that is not synchronized?*
The caller of notifyAll() (as well as wait() and notify()) is required to own the monitor for which it's invoking these methods. If the call is made from within an unsynchronized block, the caller does not own the monitor/lock which is necessary and therefore a IllegalMonitorStateException is thrown.

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)?*
After a thread reached the end of its run method it will terminate and will not be in the state of a Runnable anymore.

d) *Why do some Java apps not terminate, even though their GUI has been closed, and how can you mitigate that problem?*
If a thread that has a loop in its run method it will never reach the end of it, so it will never terminate. Another cause can be that a thread is still waiting on something and was never notified. Because the thread is still in the Runnable state the Java app will not terminate. In order to mitigate this we can implement a stop method, which is called when the GUI is called which will terminate all threads that are still running, like interrupting them or "forcefully kill" them (the first one is the better option).

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

Marcel ZAUDER 16-124-836
Pascal GERIG 16-104-721

7.3 Snow Flakes