**Exercise 1**

1. We can consider using asynchronous invocations:

* When a host object distribute services amongst multiple objects.
* When an object does not immediately need the result of an invocation.
* When invocations that are logically asynchronous.
* When classes and methods are split in order to increase concurrency and reduce liveness problems.

1. In direct invocations, asynchronization is achieved by avoiding synchronization. Without a synchronization block, caller will not need to wait. However, to apply this pattern, we will need to have immutable helpers. If we have mutable helpers, then we can use partially synchronize helper call.
2. Early reply is a pattern where a caller invokes a host and waits host to return result. In this pattern, synchronization is retained in host. Host calls helper and then waits for helper to fill-in a result to one slot buffer. However, host doesn’t block call on the helper. After host gets a result returns this to the caller. Also, host need to perform some cleanup activity on buffer after computing and returning the result to the caller.
3. Futures doesn’t block client. Host creates a future when a client calls for a request on host. Then, host return the reference of this future to client. Client waits for result by using this reference to future. In this pattern, we don’t have a buffer, host doesn’t deal with clean up operation.
4. Futures allow clients to continue their own process while the host computes the requested value. So, if you have so stuff to do after invoking the request on host, then futures are better than early replies.

**Exercise 2**

1. Calling Thread.currentThread().join() doesn’t make sense in a thread’s concurrent run method because it will cause a deadlock. Join methods makes current thread to wait until the specified thread terminates, so if you use this inside a thread’s run method, because it’s the current executing thread at that time, then it will begin to wait itself to terminate which doesn’t make sense.
2. Calling notifyAll() in a code block without synchronization will create an IllegalMonitorStateException because calling thread needs to own the object’s monitor. A thread can own an objects monitor with synchronization.
3. After a thread finishes execution in tun method, it’s terminated, also it’s state is changed to terminated.
4. Main application and GUI components can be rendered in different threads, so if thread termination is not properly implemented for all threads in the application then main application may not terminate.
5. You can use a while loop to implement a busy-waiting algorithm, this while loop controls concurrency until condition happens or you can use a while loop to implement a monitor algorithm, again also here while loop will be used to ensure safety.