CSC301

Asynchronous patterns & Object pools

Multi-Threading

- Multiple threads of execution running “simultaneously”
  - Well … not exactly simultaneously, but sharing the CPU makes it seem like it
  - Java also supports creating new processes, but threads are much more common.
- Multi-threaded programming is traditionally considered difficult
  - Easy to get things wrong
    For example: Deadlock, race condition, starvation, etc.
  - Hard to debug (mainly because the order of execution is nondeterministic)

Java & Multi-Threading

- The API's related to threads (and concurrency) evolved over the years
- Initially, there was just Thread
  - Either extend it (i.e. create a subclass) or construct it with a Runnable
  - Once you create a Thread instance, you can start it
  - Fairly low-level methods allow synchronization and inspection of threads
- Let's see a short code example

Java Threads - More Control

- Threads are resources (just like memory or CPU time)
- Sometimes we want more control over how resources are managed. For example:
  - Limit the number of active threads
  - Reuse threads, instead of creating new ones
  - Certain tasks should not compete with one another for CPU

Resource Pool

- Object Pool is a common design pattern (aka Resource Pool)
  - As the name suggests, an object that is responsible for managing a pool of resources
    - The resource pool manages the *lifecycle* of resources (e.g. construction, destruction, etc.)
  - Two of the most common examples for resources that can benefit from pooling:
    - Threads
    - Database connections
  - ScheduledThreadPoolExecutor is an example of a thread pool that comes built-in with Java
    - Here is really nice tutorial you might find useful

Asynchronous Callback

- Instead of waiting for a function to return a result, tell it in advance what you want to do with its result.
- That is:
  - When calling the function, pass a callback
  - The function runs in a separate thread
  - Once the function is done, the callback is called (with function’s result)

Asynchronous Callback

- Common example: AJAX
- Very natural in some languages (e.g. JavaScript)
- Java 8 makes things more convenient with lambda expressions and CompletableFuture
    - A nice tutorial on using CompletableFuture

Asynchronous Callback

- Asynchronous programming fits well with many of the tasks modern programmers deal with (because of the nature of the Internet)
- That being said, it doesn't always fit well with the syntax that modern programmers are used to
  - Chaining multiple methods can result in code that is very hard to read
    - If you are not careful, you might get yourself into a callback hell
  - Exceptions need to be handled more carefully