Nordstrom Credit Technology Engineering Interview Exercise

1. Your first task is to design and implement an in-memory cache that has the following characteristics: The initial ask is for entries to use an Object as the key and an Object as the stored value. Duplicate keys are not allowed. What other options might you use? If you would rather use a custom type for either the key or the value, please explain why.

   I would rather use a custom type of Integer for the key. Having an integer as the key would make more sense for this program because …

   (Integer key, Object Value)

   The cache supports a fixed number of total entries and does not grow. Rather, the cache will support various eviction strategies to removing old entries (e.g. least recently used).

   Supported operations of this cache include the following:

   - Create(size) -- creates the cache with a maximum number of entries, specified by size
   - Add(key, value) -- adds an entry to the cache, if an object already exists with the same key, replace the value with the new value, if the key does not already exist in the cache, then add the entry specified by key/value into the cache, evicting an existing entry, if necessary
   - Get(key) -- retrieves the value of the entry specified by key or null if no entry exists
   - Exists(key) -- returns true if an entry with key exists in the cache, false otherwise

   For this first part, please provide two implementations of the cache. The two variants of the cache will differ in their approach to evicting existing entries when the cache is full. First, provide an implementation that evicts the least recently used (LRU) entry from the cache when the cache is full. For LRU consideration, any add, get, or exists operation will update the LRU time stamp. Second, provide an alternate implementation that randomly evicts an entry from the cache when the cache is full. For the purposes of this first approach, you've been asked to implement your two variants using either sub-classing or the strategy pattern. Please choose one and explain the choice.

   I used sub-classing to implement the two variants.

2. Now we have run into a situation in which the server hosting your cache continuously crashes. Being an in-memory system, this causes us to lose the data in the cache resulting in our application performance to take a significant hit. In this exercise, you should re-design the architecture of your cache implementation for robustness and high-availability. For example, how would you deal with situations in which the server hosting your cache has crashed?

I would ensure that my application didn't have a single point of failure, and use a load balancer to leverage multiple servers to balance a high volume of requests.