Generic Load Driver TODO

* **Main sampler implementation.** Assume that SamplerImpl.run() is called at irregular intervals, make sure I generate regular samples. Test happy path and all edge cases.
  1. Very small intervals
  2. Very large intervals – longer than the sampling interval, repeatedly
* **Using a Class as type selector.** Return to CounterValues. getFailureTypes(), Throwable type, find a better solution.
* **SamplerImplStressTest** 
  + Return to severalSamplingCycles()
  + Improve it with testing other sampling-related results that are going to be implemented.
* **Failure Counters**
  + Implement support for failure counters. Return to CounterValues.
  + Implementation and test for different failures
  + Return to NonBlockingCounterTest
* **System-wide Metrics**
  + Loadaverage (average per sample)
  + Cpuloads (average per sample)
  + Memories (average per sample)
  + Future parameters
* **Sampler**
  + Identify the current sampler limits
  + Abstract out Sample (which can have a configurable number of fields, depending on the load/service)
  + Introduce the Sampler interface and the implementation.
  + Implement prototype based on an internal thread
  + Keep in mind that I want to zip and ship results remotely
  + Test edge cases for when the internal thread takes a long time to process stuff – the result should be eventually generated correctly
  + Unit tests – happy case and edge cases
  + Bolt it in
  + Test locally
  + Test in amazon
* **From activeMQ load testing:** Count messages based on their ID.
* **Switch to cliff.** 05/03/15.
* **The Hierarchical Repository Key Reading is Slow.** Large variation in key loading time, locally. Try to load it in parallel and also look into how I am reading the keys from the file – the way I am doing it might be inefficient.
* **Compression.** A generic compression service? Apply it to the cache only?
* **Multiple independent agents.** This way I can increase the amount of load.
* **log4j**. Externalize log4j.xml so I can modify it directly, don't have to unjar for it. Ideally, the debugging should be turned on by --debug.
* Make it easy to dump the runtime configuration – implement it as a “dry-run” where I only have to specify “dry-run” somehow and not modify anything else.
* Introduce a CacheLoadStrategy and make all cache load strategies subclasses of that, the same Send and Receive are subclasses of JmsLoadStrategy.
* Implement support for --provider Currently we assume Infinispan for “keyvalue” and ActiveMQ for “message”.
* Move the Content’s StorageStrategy to Configuration.
* Make sure cld load embedded hierarchical fills up the embedded hash with the content of the hierarchical repo (on one and multiple threads).
* Isolate HierarchicalStorageStrategy. toHex(…) into its own class (or Util), test the heck out of it and update https://home.feodorov.com:9443/wiki/Wiki.jsp?page=SHA1
  + 10 different values – calculate hex sha1 with git and write 10 unit tests for them. Use / \* and other characters that are not FS-friendly
* Unify KeyStore and StorageStrategy.
  + Rename KeyStore to LocalStore – it stores keys and values.
  + Verify which of the top-level ConfigurationImpl attributes still make sense.
  + Keep all strategies in the Configuration and not in commands
* Exact distributions of max-operations between runner threads.
* If CLI starts in background, it cannot be controlled with System.in.read – find an alternative solution.
* All “strategies” should end in Strategy.
* Understand the commented out ReadTest tests, and either uncomment or write equivalent ones.
* The ReadThenWriteOnMiss.synthethicValue is created with the default value size of 1024. Make it more user friendly, hint somehow that the values in the cache are larger (or of different size).
* Move Util.getInstance() and associated test to Nova Ordis Utilities. \* Dynamic Logging in interactive mode until I find a way to get the information I need from infinispan and not rely on runtime.
* Reconcile the new style of load manager that preconfigures the operation inside, with the old style SingleThreadedRunner op.initialize(ThreadLocalRandom.current(), lastWrittenKey, keyStore); Old logic in "OldLoad" and "OldSingleThreadedRunner"

saving keys into the file bug: because the collector is doing it asynchronously, some of the keys will stay in memory until "press the key" so we may end up with less keys on disk.

* Change the name of Connect to "interactive" or something similar.