Note: Let me know in any of the below links are not working. I will see if I have any alternate notes.

AtomicReference:

AtomicReference class was categorized under scalars. Below link includes a very interesting example on tracking # page views & # accesses that failed due to an error in a Web application. Do check it out and in general you can bookmark the link javamex.com as it has pretty good articles. The link also discusses about AtomicMarkableReference and AtomicStampedReference.

http://www.javamex.com/tutorials/synchronization_concurrency_7_atomic_reference.shtml

Field Updater Classes:

As mentioned earlier, field updater classes include AtomicLongFieldUpdater, AtomicIntegerFieldUpdater, AtomicReferenceFieldUpdater. These are reflection-based utility classes that wrap around corresponding volatile variables and can perform atomic operations on them. One use-case is that for a given shared variable, you would like it to be mostly used as a simple volatile variable, but occasionally you would want to perform some atomic operations on them. For this, using scalar atomic classes could be unnecessary. So, you can use these field updater classes only when atomicity is required and for the rest of the time the volatile field would be used as-is. Check out the below link, which explains this use-case with a very nice example involving BufferedInputStream. The link also talks about a second use-case.

http://www.javamex.com/tutorials/synchronization_concurrency_7_atomic_updaters.shtml

Volatile Boolean vs AtomicBoolean:

Checkout the following links for a discussion on this. Below is an excerpt from one of the answers.

"*I use volatile fields when said field is ONLY UPDATED by its owner thread and the value is only read by other threads, you can think of it as a publish/subscribe scenario where there are many observers but only one publisher. However if those*

observers must perform some logic based on the value of the field and then push back a new value then I go with Atomic vars or locks or synchronized blocks, whatever suits me best. In many concurrent scenarios it boils down to get the value, compare it with another one and update if necessary, hence the compareAndSet and getAndSet methods present in the Atomic* classes*."

http://stackoverflow.com/questions/3786825/volatile-boolean-vs-atomicboolean

http://stackoverflow.com/questions/4876122/when-is-it-preferable-to-use-volatile-boolean-in-java-rather-than-atomicboolean

Other Useful References:

http://stackoverflow.com/questions/6671020/java-atomic-access-to-field-within-object

http://stackoverflow.com/questions/4818699/practical-uses-for-atomicinteger