Item 68
git_comments:

1. Use "" instead of str so don't consume chars (fillBuffer) from the input on throwing IAE below:

git_commits:

1. **summary:** LUCENE-3969: fix PatternTokenizer to not consume chars from the input Reader if it throws IAE
   message: LUCENE-3969: fix PatternTokenizer to not consume chars from the input Reader if it throws IAE git-svn-id: https://svn.apache.org/repos/asf/lucene/dev/branches/lucene3969@1311318 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Use all (non-deprecated) analysis ctors in TestRandomChains
   description: We made TestRandomChains in LUCENE-3919, which reflects all analysis components from the classpath and builds analyzers from them, then checks consistency. but currently it only supports some tokenizers/tokenfilters/charfilters, because it hardcodes at certain ctors e.g. Tokenizer(Reader) and Tokenizer(Version+Reader). Instead we should use all ctors, just filling them in with random data of whatever argument type they take.

jira_issues_comments:

1. First cut at a patch: supports all of our analysis components, which means basically every time you run this test now, it finds a bug. The test ignores any invocations of UOE or IAE, and i fixed most/all trivial issues related to that, but there are serious problems...
2. updated patch... just fixing some more bugs.
3. **body:** Hi Robert, no generics complaints, it cannot be done better (means warning-free, that's a backwards-API-bug in Java - as always. Reflection-APIs and other APIs in general should return collections not raw arrays; the horrible thing here is that reflection have to create a new array on every invocation (clone), as it could be modified later. With Lists it could return unmodifiableList... Oracle should fix this and break backwards, as we should with CR.getSequentialSubReaders as there is not overhead at all). I am a little bit irritated about the catch block: {noformat} + if (e instanceof UnsupportedOperationException) { + // ignore + System.err.println("WARNING: " + e + " for " + clazz); + } else if (e instanceof InvocationTargetException) { + if (e.getCause() instanceof IllegalArgumentException || + e.getCause() instanceof UnsupportedOperationException) { + // thats ok + if (VERBOSE) { + System.err.println("Ignoring IAE/UOE from ctor:"); + e.printStackTrace(); + } + } else { + // not ok + System.err.println(clazz); + throw new RuntimeException(e); + } + } else { + throw new RuntimeException(e); + } {noformat} The first check for Unsupported cannot come from the called ctor only from reflection code itsself, but reflection should not throw UOE. When you call a ctor with reflection, every Throwable the CTOR throws is wrapped by InvocationTargetException. So the code in my opinion should *only* catch InvocationTargetException and then check for the corresponding "valid" causes. All other causes (not the ITE) should be rethrown using o.a.l.util.Rethrow (and not wrapped with Runtime). And any other reflection-based Ex should not be catched at all. I will play around and remove the big Exception bug.
   label: code-design
4. **body:** Patch with the above mentioned cleanups. I also changed some code to be more consistent (isAnnotationPresent also for classes, Class.getName instead toString). This thing fails almost always, in most cases with "too many tokens".

label: code-design

5. **body:** New patch: - I now have found out where the UOE comes from, it's the random parameter generator. But this is nasty and should be solved better. I readded the catch block with a comment. In general, the code should be refactored to not call getConstrcutors all the time. Instead the global list of List<Class<T>> should be replaced by List<Constructor<T>>, then we only have one list where to choose the ctor from (the class is implicit). Will work on a patch.
   label: code-design
6. New patch with the Constructors moved up the chain to be top-level citizens, analysis classes are no longer explicitely used. It would be goot to fix the randomParameter generator to never fail but instead the reflection code in beforClass() to check the constructor args against a Set<Class<?>> validArgs of valid parameters and throw away all invalid ctors from the beginning: validArgs.containsAll(Arrays.asList(ctor.getParameterTypes()))
7. **body:** we could also fail the test here if the random generator is out of date... is that wrong? its definitely simple :) warnings will likely be missed when the thing is running from hudson...
   label: documentation
8. **body:** Here a patch with a more flexible argument generator. Its still a little bit ugly how the three special cases are handled (and the Sets need to be kept in sync!), but at least the standard types are created very simple.
   label: code-design
9. thats nice! {quote} Its still a little bit ugly how the three special cases are handled (and the Sets need to be kept in sync!), but at least the standard types are created very simple. {quote} Yeah but thats only the exceptional cases, so its no big deal. I tested the patch, by adding a fake ctor with an unsupported type to BulgarianStemFilter: {noformat} public BulgarianStemFilter(final TokenStream input, BitSet ignored) { super(input); } {noformat} Test failed nicely at startup: {noformat} [junit] Caused by: java.lang.AssertionError: public org.apache.lucene.analysis.bg.BulgarianStemFilter(org.apache.lucene.analysis.TokenStream,java.util.BitSet) has unsupported parameter types {noformat} Thanks Uwe!
10. https://svn.apache.org/repos/asf/lucene/dev/branches/lucene3969 is open for heavy committing, in case anyone else wants to pull our their hair debugging this thing :)
11. In the branch this test is failing about 2% of the time now... getting closer to sanity. still some bugs to fix and some things to figure out...
12. This now fails only very rarely (< 1%), finding real bugs. Mergeing the test and bugfixes to trunk now.
13. Bulk close for 3.6.1