Hacking Lucene for Custom Search Results

Doug Turnbull
OpenSource Connections

Hello

Me

@softwaredoug

dturnbull@o19s.com

Us

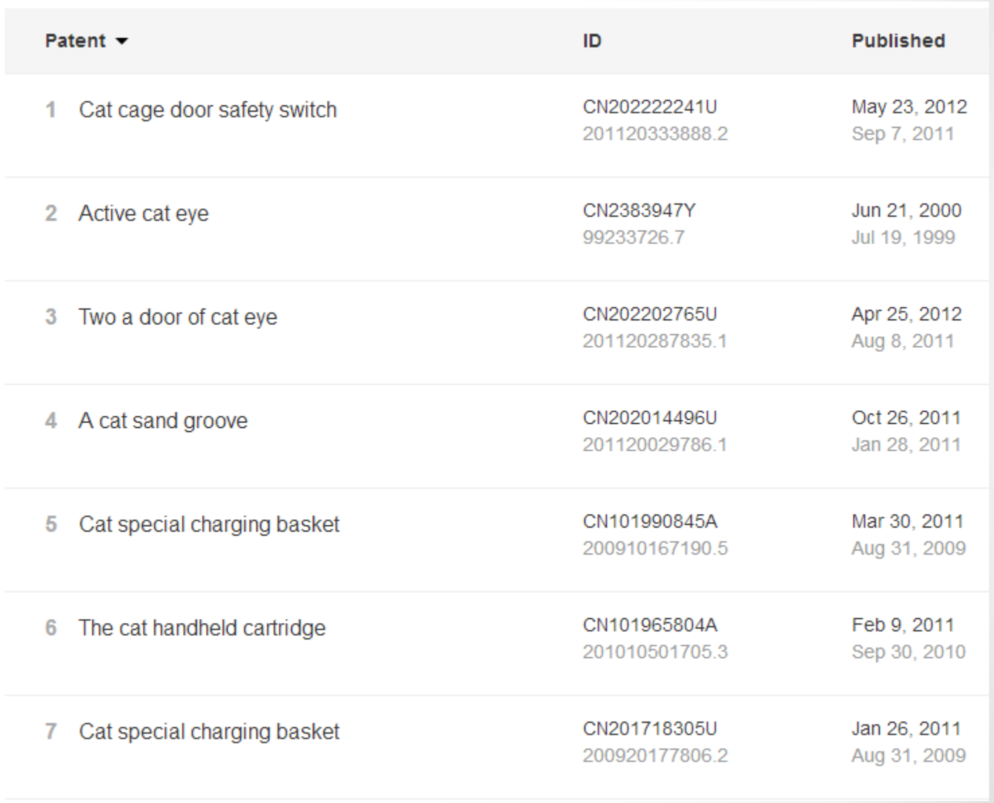
http://o19s.com

- Trusted Advisors in Search, Discovery & Analytics

Tough Search Problems

- We have demanding users!

Switch these two!



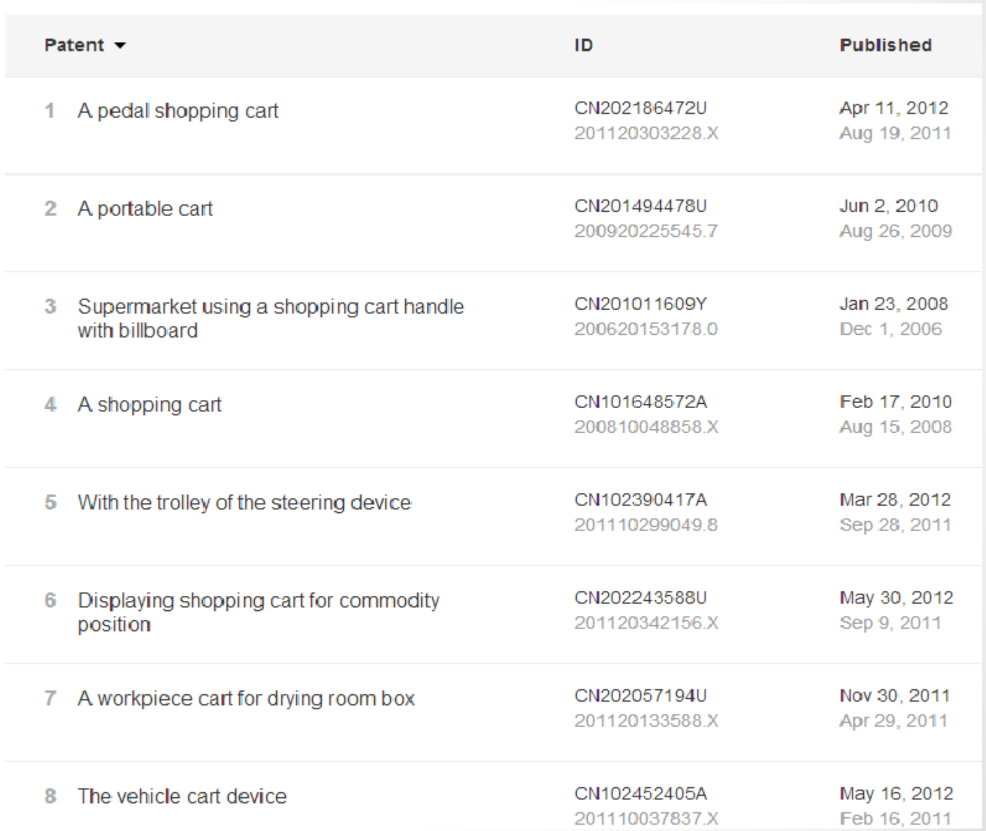
Patent	ID	Published
1 Cat cage door safety switch	CN202222241U 201120333888.2	May 23, 2012 Sep 7, 2011
2 Active cat eye	CN2383947Y 99233726.7	Jun 21, 2000 Jul 19, 1999
3 Two a door of cat eye	CN202202765U 201120287835.1	Apr 25, 2012 Aug 8, 2011
4 A cat sand groove	CN202014496U 201120029786.1	Oct 26, 2011 Jan 28, 2011
5 Cat special charging basket	CN101990845A 200910167190.5	Mar 30, 2011 Aug 31, 2009
6 The cat handheld cartridge	CN101965804A 201010501705.3	Feb 9, 2011 Sep 30, 2010
7 Cat special charging basket	CN201718305U 200920177806.2	Jan 26, 2011 Aug 31, 2009

Tough Search Problems

- Demanding users!

WRONG!

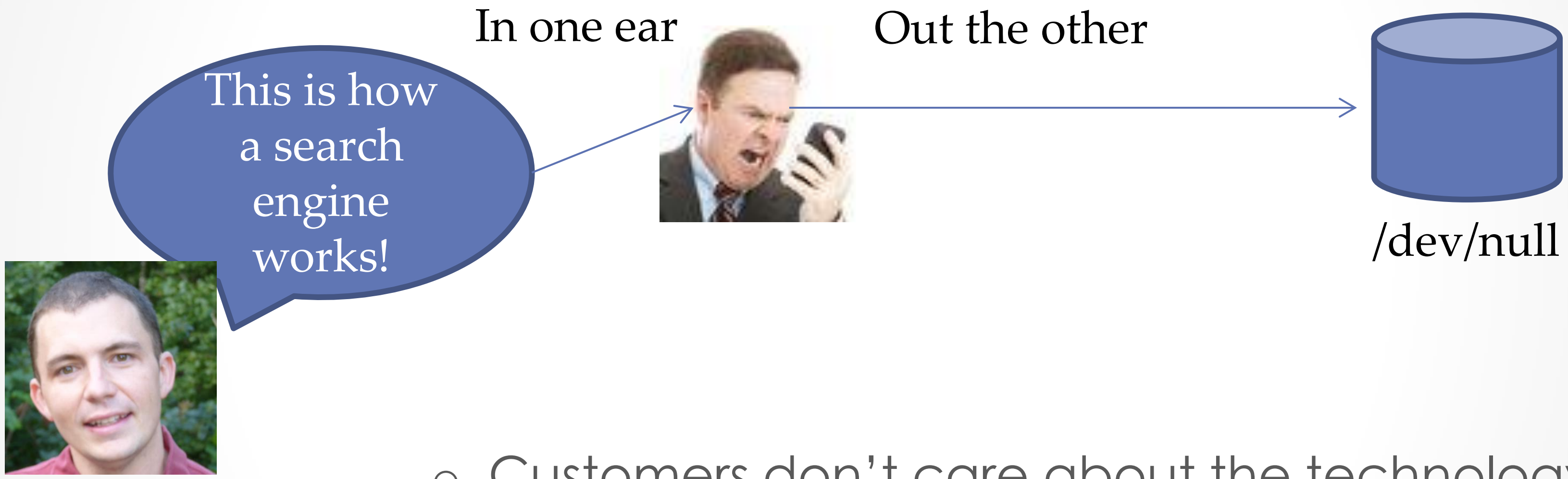
Make search do what is in my head!



Patent	ID	Published
1 A pedal shopping cart	CN202186472U 201120303228.X	Apr 11, 2012 Aug 19, 2011
2 A portable cart	CN201494478U 200920225545.7	Jun 2, 2010 Aug 26, 2009
3 Supermarket using a shopping cart handle with billboard	CN201011609Y 200620153178.0	Jan 23, 2008 Dec 1, 2006
4 A shopping cart	CN101648572A 200810048858.X	Feb 17, 2010 Aug 15, 2008
5 With the trolley of the steering device	CN102390417A 201110299049.8	Mar 28, 2012 Sep 28, 2011
6 Displaying shopping cart for commodity position	CN202243588U 201120342156.X	May 30, 2012 Sep 9, 2011
7 A workpiece cart for drying room box	CN202057194U 201120133588.X	Nov 30, 2011 Apr 29, 2011
8 The vehicle cart device	CN102452405A 201110037837.X	May 16, 2012 Feb 16, 2011

Tough Search Problems

- Our Eternal Problem:



  - Customers don't care about the technology field of Information Retrieval: **they just want results**
  - BUT we are constrained by the tech!

Satisfying User Expectations

- Easy: The Search Relevancy Game:
  - Solr/Elasticsearch **query operations** (boosts, etc)
  - **Analysis** of query/index to enhance matching

- Medium: Forget this, lets write some Java
  - Solr/Elasticsearch **query parsers**. Reuse existing Lucene Queries to get closer to user needs

That Still Didn’t Work

- Look at him, he’s angrier than ever!
- For the toughest problems, we’ve made search complex and brittle
- **WHACK-A-MOLE:**
  - Fix one problem, cause another
  - We give up,

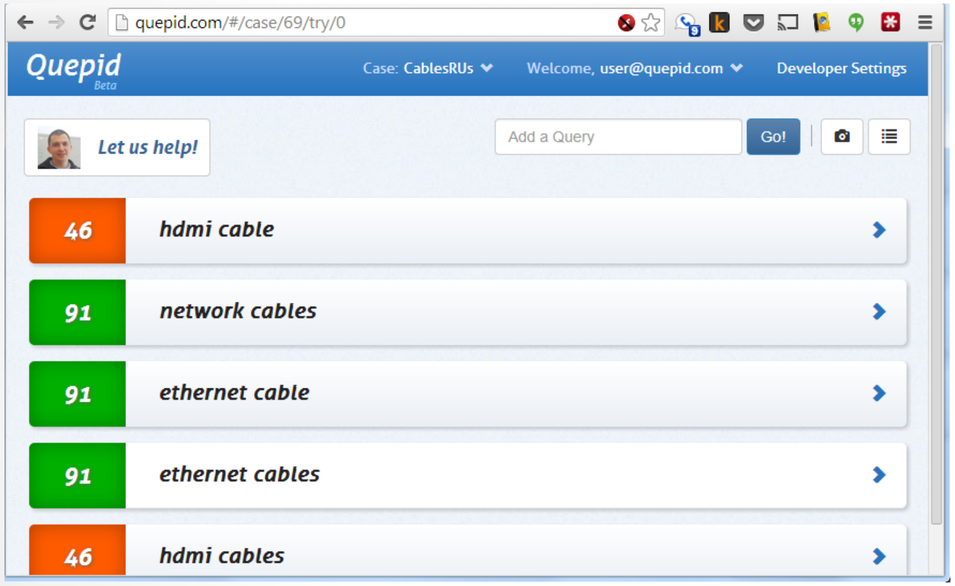
Next Level

- Hard: **Custom Lucene Scoring** – implement a query and scorer to explicitly control matching and scoring

This is the Nuclear Option!

Shameless Plug

- How do we know if we're making progress?



quepid.com/#/case/69/try/0

46	hdmi cable
91	network cables
91	ethernet cable
91	ethernet cables
46	hdmi cables

- Quepid! – our search test driven workbench

Lucene Lets Review

- At some point we wrote a Lucene index to a directory
- Boilerplate (open up the index):

```
Directory d = new RAMDirectory();
IndexReader ir = DirectoryReader.open(d);
IndexSearcher is = new IndexSearcher(ir);
```

Boilerplate setup of:

- Directory Lucene's handle to the FS
- IndexReader – Access to Lucene's data structures
- IndexSearcher – use index searcher to perform search

Lucene Lets Review

- Queries:

Make a Query and Search!

- TermQuery: basic term search for a field

```
Term termToFind = new Term("tag", "space");
TermQuery spaceQ = new TermQuery(termToFind);
termToFind = new Term("tag", "star-trek");
TermQuery starTrekQ = new TermQuery(termToFind);
```

- Queries That Combine Queries

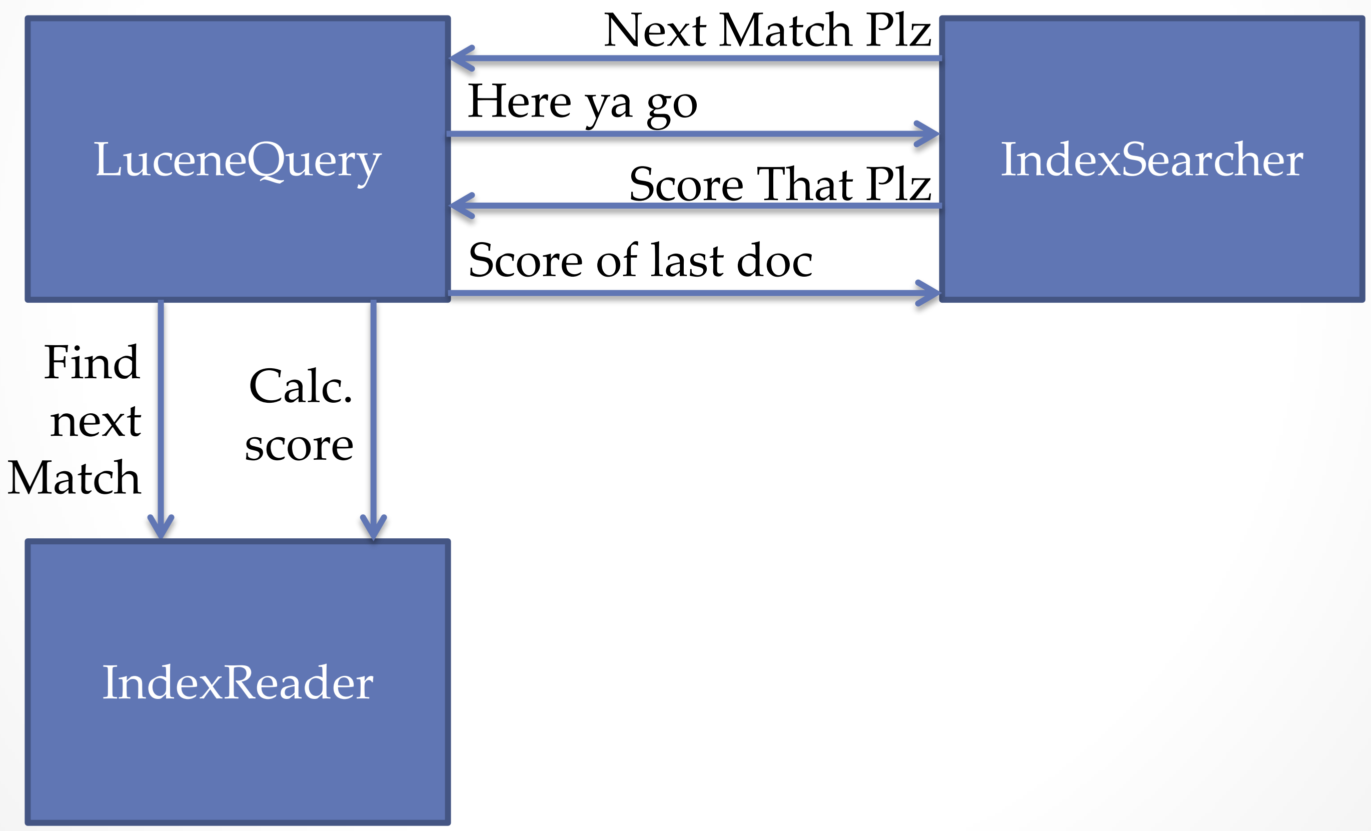
```
BooleanQuery bq = new BooleanQuery();
BooleanClause bClause = new BooleanClause(spaceQ, Occur.MUST);
BooleanClause bClause2 = new BooleanClause(starTrekQ, Occur.SHOULD);
bq.add(bClause);
bq.add(bClause2);
```

Lucene Lets Review

- Query responsible for specifying search behavior
- Both:
  - **Matching** – what documents to include in the results
  - **Scoring** – how relevant is a result to the query by assigning a score

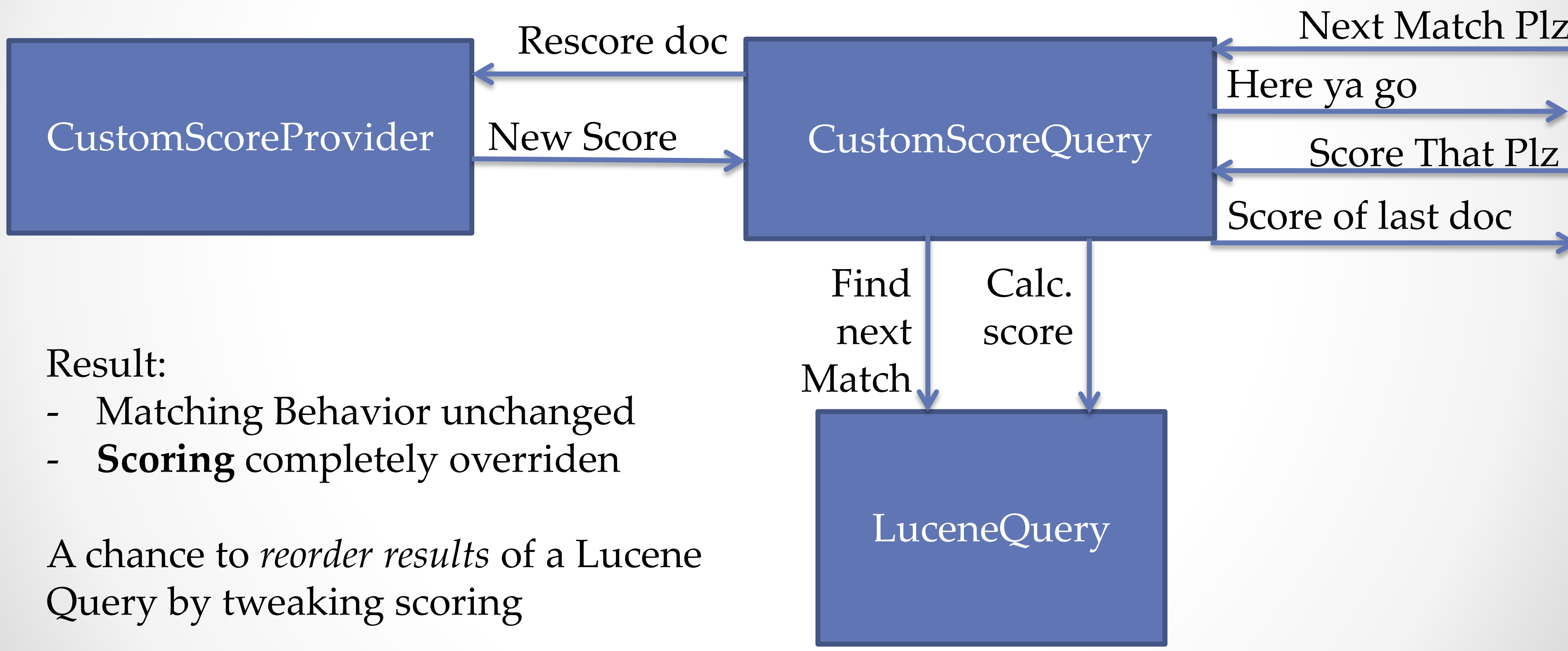
Lucene Queries, 30,000 ft view

Aka, “not really accurate, but what to tell your boss to not confuse them”



First Stop CustomScoreQuery

- Wrap a query but override its score



Result:
- Matching Behavior unchanged
- **Scoring** completely overriden

A chance to *reorder results* of a Lucene Query by tweaking scoring

How to use?

- Use a normal Lucene query for **matching**

```
Term t = new Term("tag", "star-trek");
TermQuery tq = new TermQuery(t);
```

- Create & Use a CustomQueryScorer for **scoring** that wraps the Lucene query

```
CountingQuery ct = new CountingQuery(tq);
```

Implementation

- Extend CustomScoreQuery, provide a CustomScoreProvider

```
protected CustomScoreProvider getCustomScoreProvider(
          AtomicReaderContext context) throws IOException {
    return new CountingQueryScoreProvider("tag", context);
}
```

(boilerplate omitted)

Implementation

- CustomScoreProvider rescores each doc with IndexReader & docId

```
// Give all docs a score of 1.0
public float customScore(int doc, float subQueryScore, float
valSrcScores[]) throws IOException {
    return (float)(1.0f); // New Score
}
```

Implementation

- Example: Sort by number of terms in a field

```
// Rescores by counting the number of terms in the field
public float customScore(int doc, float subQueryScore, float
valSrcScores[]) throws IOException {
    IndexReader r = context.reader();
    Terms tv = r.getTermVector(doc, _field);
    TermsEnum termsEnum = null;
    termsEnum = tv.iterator(termsEnum);
    int numTerms = 0;
    while((termsEnum.next()) != null) {
        numTerms++;
    }
    return (float)(numTerms); // New Score

}
```

CustomScoreQuery, Takeaway

- **SIMPLE!**
  - Relatively few gotchas or bells & whistles (we will see lots of gotchas)
- Limited
  - No tight control on what matches
- If this satisfies your requirements: You should get off the train here

Lucene Circle Back

- I care about overriding **scoring**
  - CustomScoreQuery

- I need to control custom scoring and **matching**
  - Custom Lucene Queries!

Example – Backwards Query

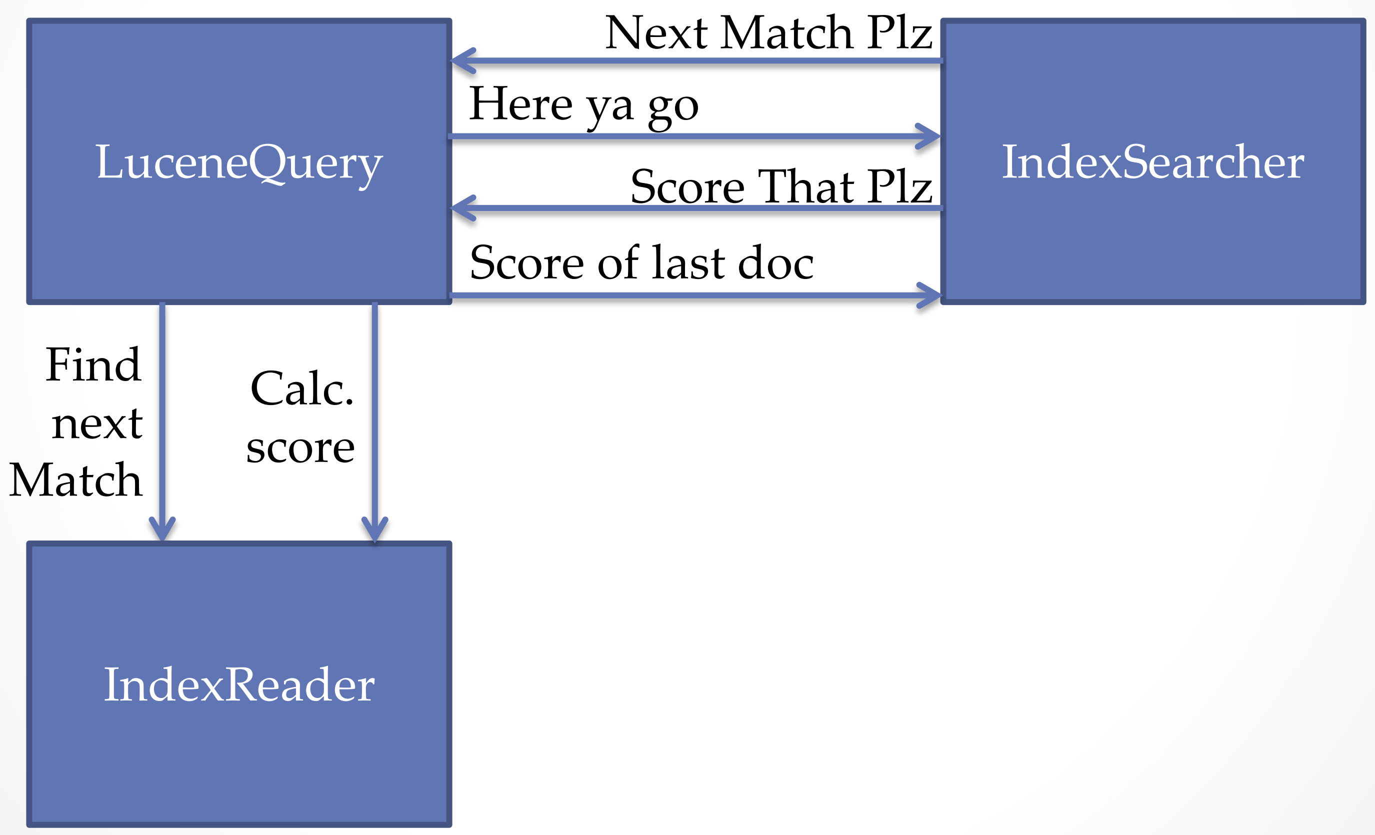
- Search for terms backwards!
  - Instead of banana, lets create a query that finds **ananab** matches and scores the document (5.0)
  - But lets also match forward terms **(banana)**, but with a lower score (1.0)

- Disclaimer: its probably possible to do this with easier means!

https://github.com/o19s/lucene-query-example/

Lucene Queries, 30,000 ft view

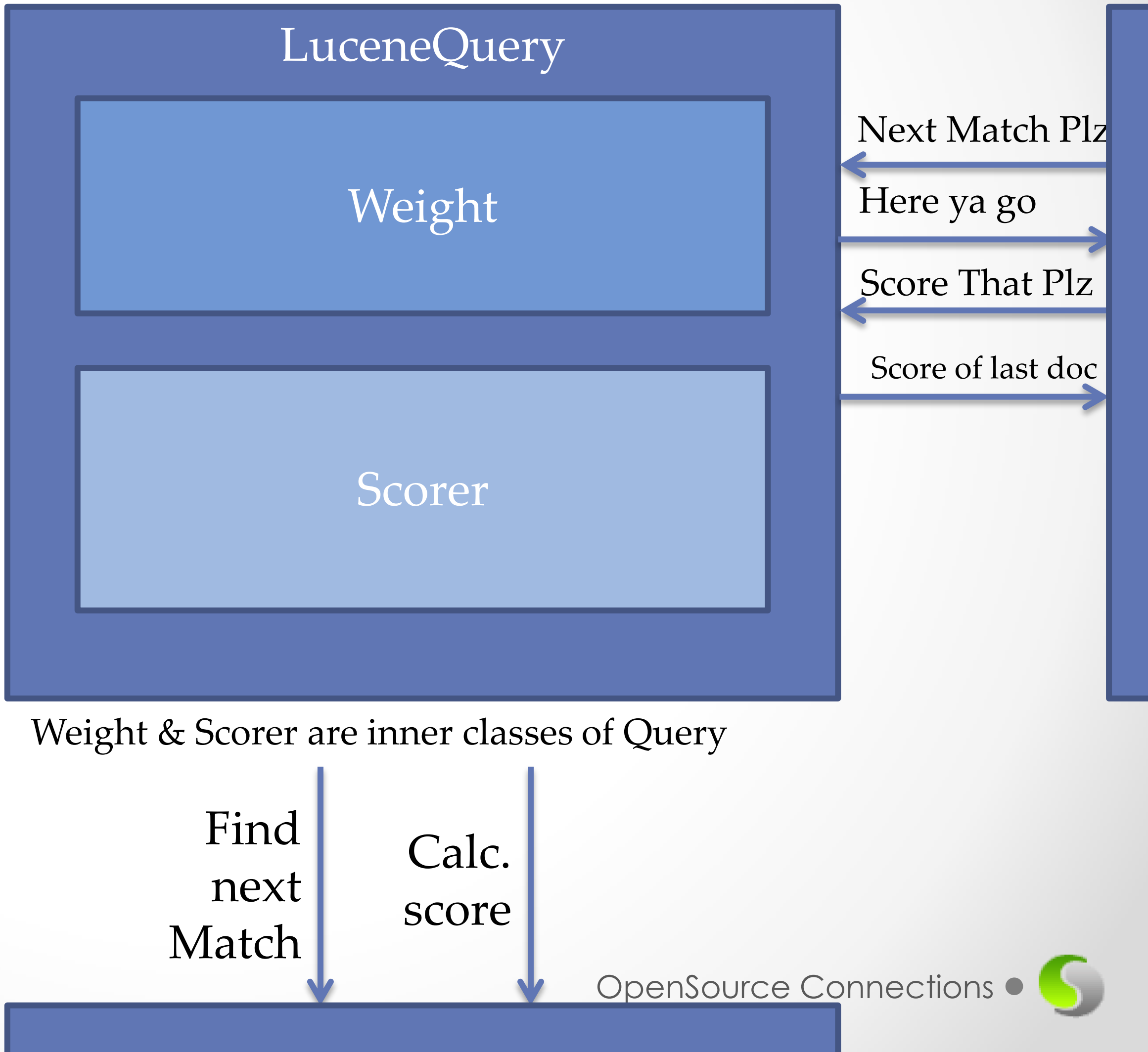
Aka, "not really accurate, but what to tell your boss to not confuse them"



Anatomy of Lucene Query

A Tale Of Three Classes:

- *Queries* Create *Weights:*
  - Query-level stats for this search
  - Think "IDF" when you hear weights
- *Weights* Create *Scorers:*
  - Heavy Lifting, reports **matches** and returns a **score**



Backwards Query Outline

```
class BacwkardsQuery {


    class BackwardsScorer {
        // matching & scoring functionality
    }


    class BackwardsWeight {
        // query normalization and other “global” stats

        public Scorer scorer(AtomicReaderContext context, …)
    }


    public Weight createWeight(IndexSearcher)

}
```

How are these used?

When you do:

```
Query q = new BackwardsQuery();
idxSearcher.search(q);
```

This Setup Happens:

```
Weight w = q.createWeight(idxSearcher);
normalize(w);
foreach IndexReader idxReader:
    Scorer s = w.scorer(idxReader);
```

Important to know how Lucene is calling your code

Weight

What should we do with our weight?

```
Weight w = q.createWeight(idxSearcher);
normalize(w);
```

<u>**IndexSearcher Level Stats**</u>

- Notice we pass the IndexSearcher when we create the weight
    - Weight tracks IndexSearcher level statistics used for scoring

<u>**Query Normalization**</u>

- Weight also participates in query normalization

Remember – its your Weight! **Weight can be a no-op and just create searchers**

Weight & Query Normalization

Query Normalization – an optional little ritual to take your Weight instance through:

```
float v = weight.getValueForNormalization();
float norm = getSimilarity().queryNorm(v);
weight.normalize(norm, 1.0f);
```

What I think my weight is

Normalize that weight against global statistics

Pass back the normalized stats

Weight & Query Normalization

```
float v = weight.getValueForNormalization();
float norm = getSimilarity().queryNorm(v);
weight.normalize(norm, 1.0f);
```

- For TermQuery:
  - The result of all this ceremony is the *IDF* (inverse document frequency of the term).
- This code is fairly abstract
  - All three steps are pluggable, and can be totally ignored

BackwardsWeight

- Custom Weight that completely ignores query normalization:

```
@Override
public float getValueForNormalization() throws IOException {
    return 0.0f;
}

@Override
public void normalize(float norm, float topLevelBoost) {
    // no-op
}
```

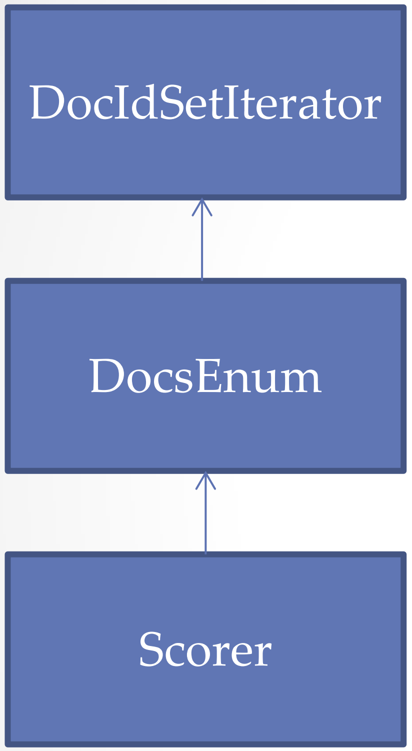
Weights make Scorers!

```
@Override
public Scorer scorer(AtomicReaderContext context, boolean
scoreDocsInOrder, boolean topScorer, Bits acceptDocs) throws
IOException {

    return new BackwardsScorer(...);
}
```

- Scorers Have Two Jobs:
  - **Match!** – iterator interface over matching results
  - **Score!** – score the current match

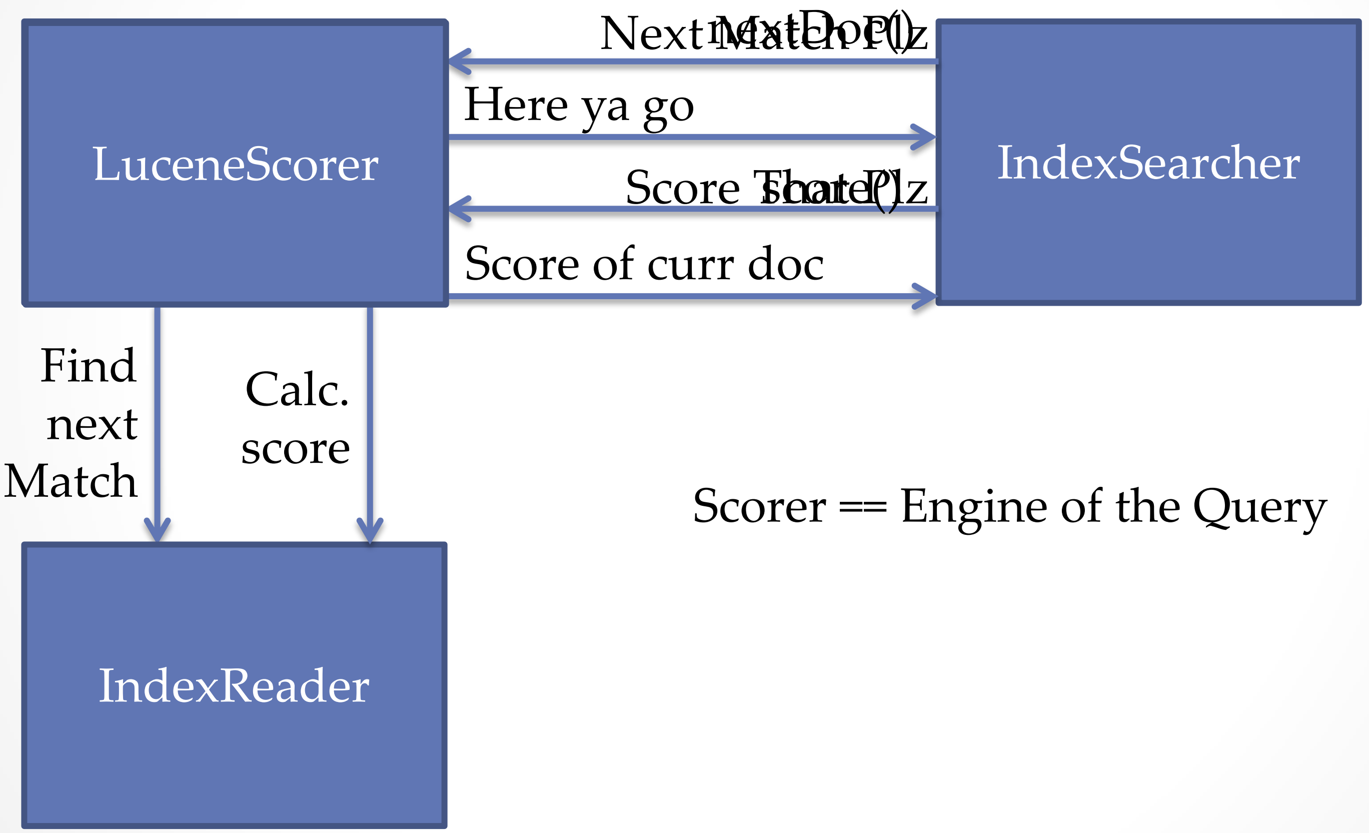
Scorer as an iterator



- Inherits the following from DocsEnum:
- **nextDoc()**
  - Next match
- **advance(int docId)** –
  - Seek to the specified docId
- **docID()**
  - Id of the current document we're on

In other words…

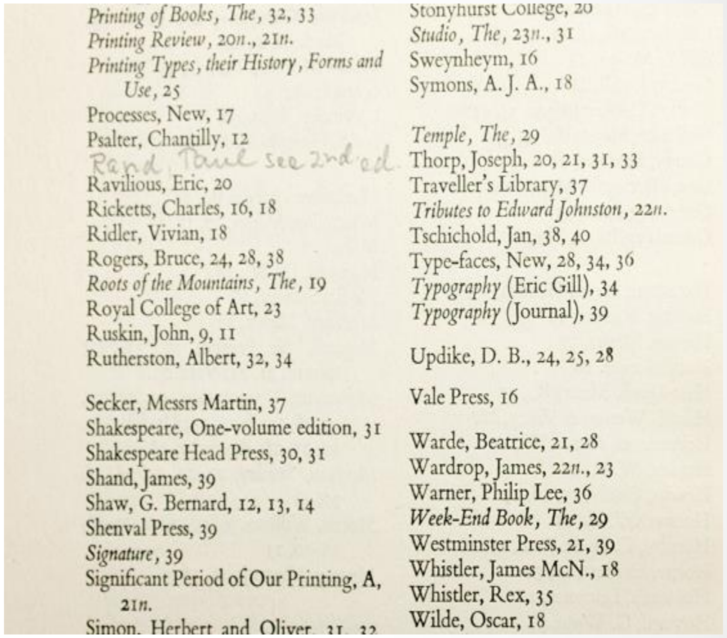
- Remember THIS? …Actually…



Scorer == Engine of the Query

What would nextDoc look like

- Remember search is an **inverted index**
  - Much like a book index
  - Fields -> Terms -> Documents!



Printing of Books, The, 32, 33
Printing Review, 20n., 21n.
Printing Types, their History, Forms and Use, 25
Processes, New, 17
Psalter, Chantilly, 12
Rand, Paul see 2nd ed.
Ravilious, Eric, 20
Ricketts, Charles, 16, 18
Ridler, Vivian, 18
Rogers, Bruce, 24, 28, 38
Roots of the Mountains, The, 19
Royal College of Art, 23
Ruskin, John, 9, 11
Rutherston, Albert, 32, 34

Secker, Messrs Martin, 37
Shakespeare, One-volume edition, 31
Shakespeare Head Press, 30, 31
Shand, James, 39
Shaw, G. Bernard, 12, 13, 14
Shenval Press, 39
Signature, 39
Significant Period of Our Printing, A, 21n.
Simon, Herbert and Oliver, 31, 32

Stonyhurst College, 20
Studio, The, 23n., 31
Sweynheym, 16
Symons, A. J. A., 18

Temple, The, 29
Thorp, Joseph, 20, 21, 31, 33
Traveller's Library, 37
Tributes to Edward Johnston, 22n.
Tschichold, Jan, 38, 40
Type-faces, New, 28, 34, 36
Typography (Eric Gill), 34
Typography (Journal), 39

Updike, D. B., 24, 25, 28

Vale Press, 16

Warde, Beatrice, 21, 28
Wardrop, James, 22n., 23
Warner, Philip Lee, 36
Week-End Book, The, 29
Westminster Press, 21, 39
Whistler, James McN., 18
Whistler, Rex, 35
Wilde, Oscar, 18

IndexReader == our handle to inverted index:

- Much like an index. Given **term**, return **list of doc ids**
- **TermsEnum:**
  - Enumeration of terms (actual logical index of terms)
- **DocsEnum**
  - Enum. of corresponding docIDs (like list of pages next to term)

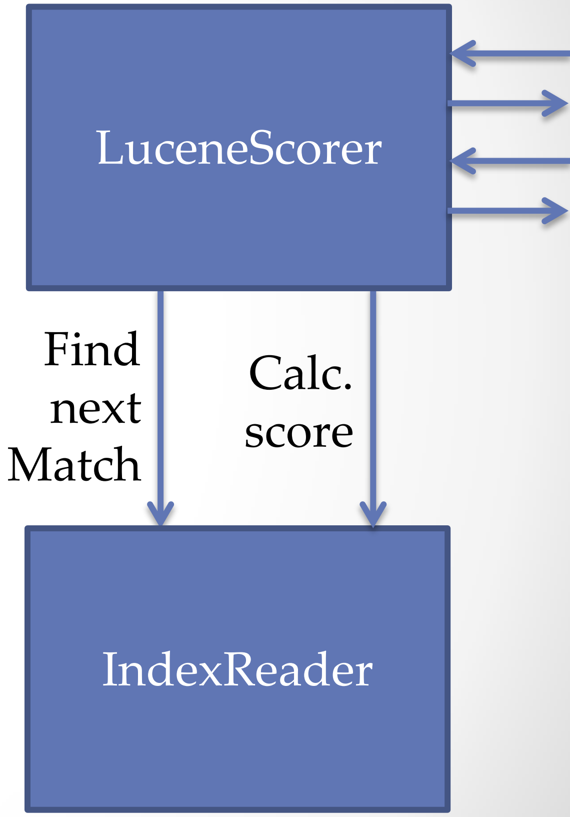
What would nextDoc look like?

- TermsEnum to lookup info for a Term:

```
final TermsEnum termsEnum =
     reader.terms(term.field()).iterator(null);
termsEnum.seekExact(term.bytes(), state);
```

- Each term has a DocsEnum that lists the docs that contain this term:

```
DocsEnum docs = termsEnum.docs(acceptDocs, null);
```

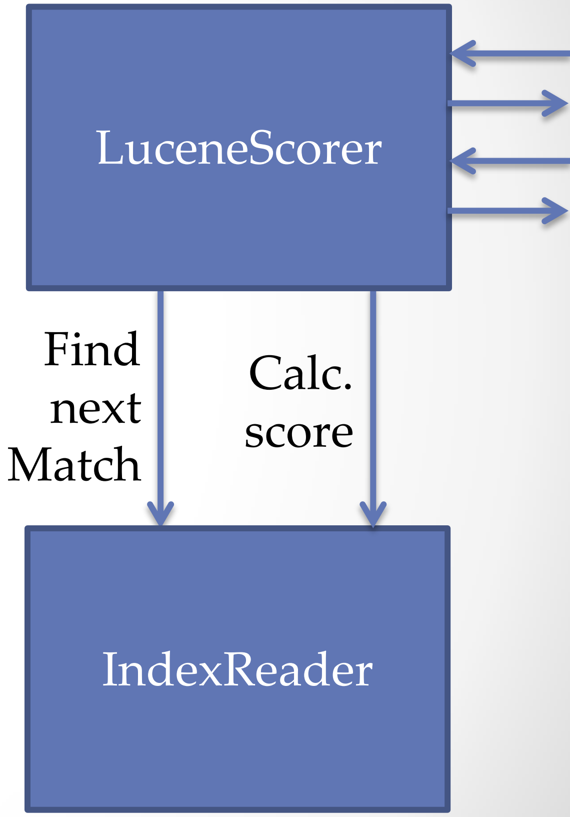


What would nextDoc look like?

- Wrapping this enum, now I can return matches for this term!

```
@Override
 public int nextDoc() throws IOException {
   return docs.nextDoc();
 }
```

- You’ve just implemented TermQuery!



BackwardsScorer nextDoc

- Recall our Query has a Backwards Term (ananab):

```
public BackwardsQuery(String field, String term) {
    backwardsTerm = new Term(field, new StringBuilder(term).reverse().toString());
     ...
}
```

- Later, when creating a Scorer. Get a handle to **DocsEnum** for our backwards term:

```
     public Scorer scorer(AtomicReaderContext context, boolean scoreDocsInOrder,
                          boolean topScorer, Bits acceptDocs) throws IOException {

          Term bwdsTerm = BackwardsQuery.this.backwardsTerm;
          TermsEnum bwdsTerms = context.reader().terms(bwdsTerm.field()).iterator(null);
          bwdsTerms.seekExact(bwdsTerm.bytes());
          DocsEnum bwdsDocs = bwdsTerms.docs(acceptDocs, null);
```

Terrifying and verbose Lucene speak for:
1. Seek to term in field via TermsEnum
2. Give me a DocsEnum of matching docs

BackwardsScorer nextDoc

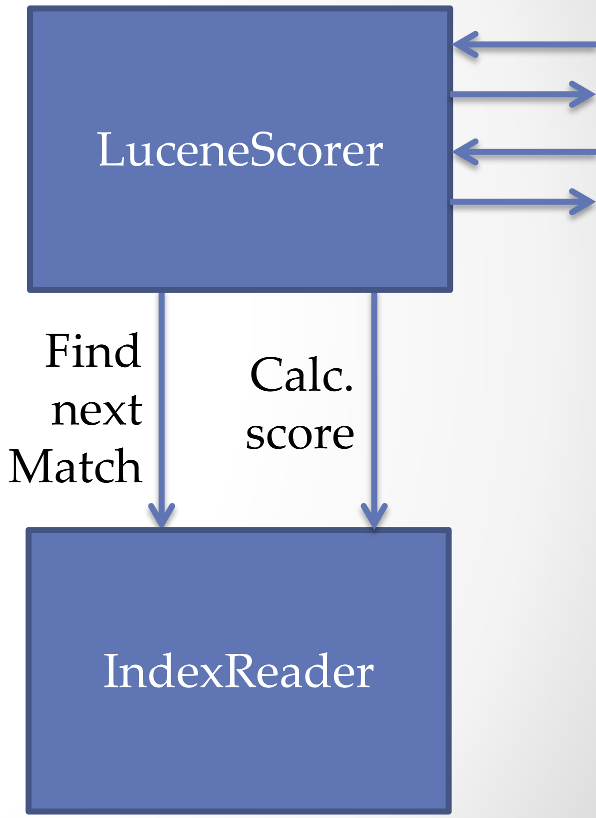
- Our scorer has **bwdDocs** and **fwdDocs**, our nextDoc just walks both:

```
@Override
public int nextDoc() throws IOException {
    int currDocId = docID();
    // increment one or both
    if (currDocId == backwardsScorer.docID()) {
        backwardsScorer.nextDoc();
    }
    if (currDocId == forwardsScorer.docID()) {
        forwardsScorer.nextDoc();
    }
    return docID();
}
```

Scorer for scores!

- Score is easy! Implement **score**, do whatever you want!

```
@Override
public float score() throws
IOException {
    return 1.0f;
}
```



BackwardsScorer Score

- Recall, match a backwards term (ananab)score = 5.0, fwd term (banana) score = 1.0
- We hook into docID, update score based on current posn

```
@Override
public int docID() {
     int backwordsDocId = backwardsScorer.docID();
     int forwardsDocId = forwardsScorer.docID();
     if (backwordsDocId <= forwardsDocId && backwordsDocId != NO_MORE_DOCS) {
          currScore = BACKWARDS_SCORE;
          return backwordsDocId;
     } else if (forwardsDocId != NO_MORE_DOCS) {
          currScore = FORWARDS_SCORE;
          return forwardsDocId;
     }
     return NO_MORE_DOCS;
}
```

We call docID() in nextDoc()

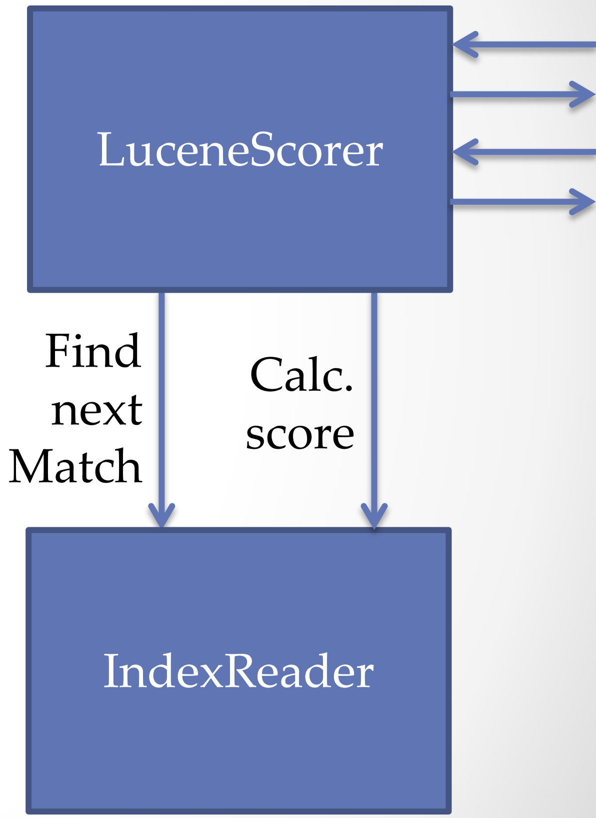
Currently positioned on a bwds doc, set currScore to 5.0

Currently positioned on a fwd doc, set currScore to 1.0

BackwardsScorer Score

- For completeness sake, here’s our **score**:

```
@Override
public float score() throws
IOException {
    return currScore;
}
```



So many gotchas!

- Ultimate POWER! But You will have weird bugs:
  - Do all of your searches return the results of your first query?
    - In **Query** Implement **hashCode** and **equals**
  - Weird/Random Test Failures
    - Test using LuceneTestCase to ferret out common Lucene bugs
      - Randomized testing w/ different codecs etc
      - IndexReader methods have a certain ritual and very specific rules, (enums must be primed, etc)

Extras

- Query **rewrite** method
  - Optional, recognize you are a complex query, turn yourself into a simpler one
    - BooleanQuery with 1 clause -> return just one clause
- Weight has optional **explain**
  - Useful for debugging in Solr
  - Pretty straight-forward API

Conclusions!

- These are nuclear options!
    - You can achieve SO MUCH before you get here (at much less complexity)
    - There's certainly a way to do what you've seen without this level of control

- Fun way to learn about Lucene!

QUESTIONS?