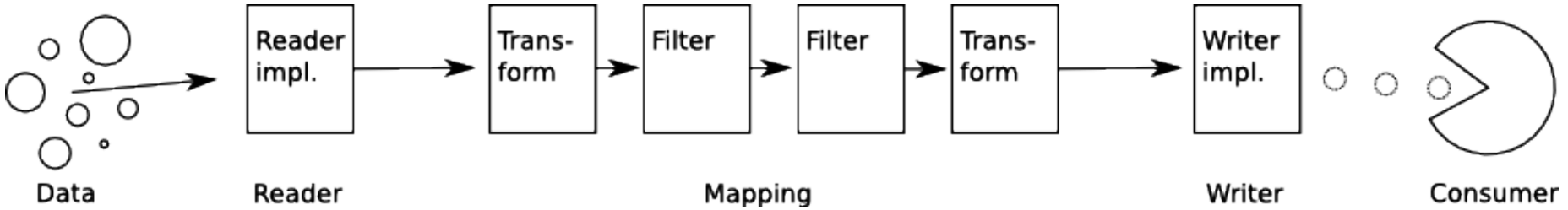
Feature Extraction Tutorial and/or Review

The Basics




```
class Reader<K,V> {
  K getCurrentKey();
  V getCurrentValue();
  boolean nextKeyValue();
  void close();
}
```

```
class Output<K,V> {
  void write(K key, V value)
  void close();
}
```

```
class Mapping<Ki,Vi,Ko,Vo> {
  void map(Ki key, Vi val, Output<Ko, Vo> out);
}
```

General workflow:

```
Reader<Ki, Vi> reader;
Mapping<Ki, Vi, Ko, Vo> mapping;
Output<Ko, Vo> out;

while (reader.nextKeyValue()) {

   Ki key = reader.getCurrentKey();
   Vi value = reader.getCurrentValue();
   mapping.map(key, value, out);
}

out.close();
```

.. but you don't do that,
the execution environment does
(and does it better too)

Basic (out of box) execution environment

```
BasicExecutionEnvironment env;

Reader<Ki, Vi> reader;
Mapping<Ki, Vi, Ko, Vo> mapping;
Output<Ko, Vo> out;

env.execute(reader, mapping, out);
```

... but remember everything an execution environment is supposed to do: Error handling, parallelism, define “workflows”, etc.

Error handling

- Mapping impls are defined to be stateless
  - Order of execution should not matter, execution environment is free to re-order as it sees fit
- Output impls are defined to be idempotent
  - The same value can be re-submitted an arbitrary number of times without consequence
- Reader impls are NOT stateless, and are designed to be iterated through only once per execution job

Error handling

- Coping strategy: for a given key+value pair, if a mapping or write to Output fails, try again

```
BasicExecutionEnvironment env;
ExceptionListener el;

Reader<Ki, Vi> reader;
Mapping<Ki, Vi, Ko, Vo> mapping;
Output<Ko, Vo> out;

env.execute(reader, mapping, out, el);
```

Error handling

One way to do it (anonymous inner class):

This example always re-submits

```
env.execute(reader, mapping, out, new ExceptionListener() {

  public void exceptionThrown(ExecutionTaskException e) {
                e.getTask().run();
            }
  });
```

Problem: What if the exception is always thrown, e.g. a bug in code
Vs a transient network effect?

Solution: could keep a count of failures per key/value pair and stop at
A fixed number.

Question: Would it be useful to have a utility to do so, or have a wrapper
for BasicExecutionEnvironment that does that automatically?

Error handling

- Behind the scenes, BasicExecutionEnvironment is keeping track of each key+value operation.
  - Adds job to “the list” if it fails and is not addressed yet
  - If calling code does retry(), these get priority, and emptied from the queue when (if) they succeed
  - If calling code does getTask().run(), the Runnable will do this accounting.. and of course notify the listener if the task fails yet again.
- So BasicExecutionEnvironment guarantees every key+value pair is run at least once. Unless...

Error Handling

```
BasicExecutionEnvironment env;
env.setDropFailedTasks(true);
ExceptionListener el;

Reader<Ki, Vi> reader;
Mapping<Ki, Vi, Ko, Vo> mapping;
Output<Ko, Vo> out;

env.execute(reader, mapping, out, el);
```

- If set to drop failed tasks, will allow exceptionListener to simply ignore exception (and allow a key+value pair to be dropped)
- If no exceptionListener given, execute() will throw an exception if one is encountered, so an exceptionListener is really needed in order to selectively drop failed tasks.

Parallelism

```
ExecutorService exe;

BasicExecutionEnvironment env =
    new BasicExecutionEnvironment(exe);
```

- Every key+value pair mapping operation is wrapped in a Runnable (sound familiar?) and submitted to a java ExecutorService.
- ExecutorService responsible for the performance characteristics of an extraction/transformation job

Parallelism

- Executors.newCachedThreadPool()
  - Java-provided, re-uses an existing thread if available, or starts a new one. Can result in lots of threads, though!
- Executors.newFixedThreadPool(N)
  - Java provided, limits the number of threads. BUT, submitted tasks are put into an infinitely large queue to wait until a thread becomes available.!

Parallelism

- Common Services to the rescue!
- SynchronousExecutor()
  - Used by default in BasicExecutionEnvironment
  - Simply executes every task immediately in current thread – no parallelism at all.
- BoundedBlockingPoolExecutor(N)
  - Uses a fixed thread pool, but no queue
  - Job submission blocks until a thread is available

Ingest use case

- Potentially many feature extraction tasks to perform, each one could potentially:
  - Export data to another service (index)
  - Modify the SIP based upon extracted feature
  - Generate event(s) based upon a particular extraction
  - Allow a non-essential feature extraction job to fail and not kill the ingest
  - Force ingest to terminate if an essential feature extraction job fails

Ingest use case

```
class BasicFeatureExtraction extends IngestServiceBase {

  @Required
  void setJobs(List<Job<?,?>> jobs);

  void execute(String sipRef); // from IngestService
}

class Job<Ko, Vo> {

  @Required
  void setMapping(Mapping<String, Dcp, Ko, Vo> map);

  @Required /* but will have to change in Y3 */
  void setOutputFactory(OutputFactory<Ko, Vo> outfac);

  void setOutcomeListener(OutputListener l);

  Void setExceptionListener(ExceptionListener l);

  void setLabel(String label)

  void setFailureAllowed(boolean allowFail); // for the lazy
  void setCreateEvent(boolean createEvent); // for the lazy
}
```

Ingest use case

```
/* part of ingest service */
interface OutcomeListener {

  void onSuccess(String sipRef, IngestFramework fwk);

  void onFailure(String sipRef, IngestFramework fwk,
                 Exception e);
}

/* from feature extraction “execution” interfaces
interface OutputFactory<K,V> {

  Output<K,V> newOutput();

  /* Sneak preview of Y3 needs */
  public void close(boolean... success);
}
```

Ingest use case

```
<bean id="featureExtraction"
class="org.dataconservancy.dcs.ingest.services.BasicFeatureExtraction">

  <property name="ingestFramework"
     ref="org.dataconservancy.dcs.ingest.IngestFramework" />

  <property name="executionEnvironment"
  ref="ingestFeatureExtractionExecutionEnvironment" />

  <property name="jobs">
      <list>
        <ref bean=”dcpSolrJob” />
        <ref bean=”generalModelJob” />
        <ref bean=”registryUpdateJob” />
      </list>
  </property>
</bean>
```

Question related to DCS instance configuration. Is one list of jobs sufficient, or can/should/must jobs be added in a modular fashion. For example, a method invoking bean + addJob(Job<?,?>) method on BasicFeatureExtraction.

Ingest use case

Example general model indexing job, run every ingest.

```
<bean id="generalModelJob"
class="org.dataconservancy.dcs.ingest.services.BasicFeatureExtraction$Job">

  <property name="mapping" ref="generalModelMappingChain" />

  <property name="outputFactory" ref="generalModelIndexOutputFactory" />

  <property name=”outcomeListener” ref=”gqmIndexJobListener” />

</bean>


<bean name=”generalModelIndexOutputFactory”
class=”org.dataconservancy.dcs.index.transform.IndexOutputFactory”>

  <property name=”indexService” ref=”generalModelIndexService”/>

</bean>


<bean name=”gqmIndexJobListener”
  class=”org.dataconservancy.dcs.index.transform.IndexEventLogger”>

  <property name=”indexName” value=”GQM” />
</bean>
```

Chaining

```
<bean id="generalModelMappingChain"
class="org.dataconservancy.dcs.transform.execution.MappingChain">

  <property name="chain">
    <list>

      <bean class=”org.dataconservancy.transform.dcp.LeafNodeDuFilter” />

      <bean class=”org.dataconservancy.transform.profile.DcpProfileEmitter”>
        <property name=”detectorMap” ref=”profileDetectorMap />
      </bean>

      <bean
class=”org.dataconservancy.transform.profile.RegistryBasedDcpProfileMapper”>
         <property name=”registry” ref=”gqmTransformRegistry” />
      </bean>

    <list>
  </property>

</bean>

<bean id=”gqmTransformRegistry”
     class=”org.dataconservancy.transform.registry.SimpleRegistryImpl”>
  <property name=”map”>
  </property
</bean>
```

Registry

```
<bean id="profileDetectorMap" class="java.util.HashMap">
  <constructor-arg>
    <map>
      <entry key="dataconservancy:profiles:DryValleysSip"
             value-ref="DryValleyDetector" />

      <entry key="dataconservancy:profiles:SDSSSip"
             value-ref="SDSSDetector" />
    <map>
  </constructor-arg>
<bean>

<bean id="gqmTransformRegistry"
     class="org.dataconservancy.transform.registry.SimpleRegistryImpl">
  <property name="map">
    <map>

      <entry key="dataconservancy:profiles:DryValleysSip"
             value-ref="DryValleyGQMTransform" />

      <entry key="dataconservancy:profiles:SDSSSip"
             value-ref="SDSSGQMTransform" />

    </map>
  </property
</bean>
```