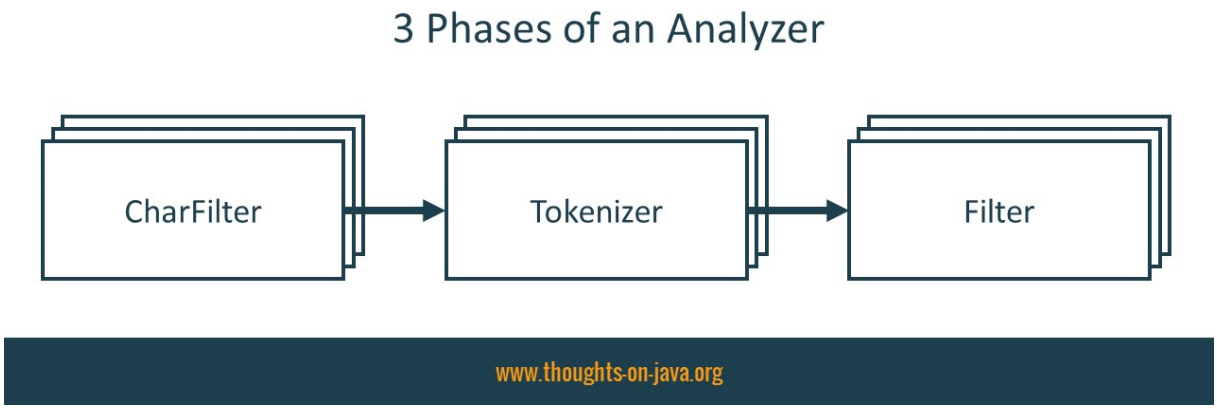
The 3 phases of an analyzer

An Analyzer consists of 3 phases, and each of them can perform multiple steps:

The *CharFilter* adds, removes or replaces certain characters. That is often used to normalize special characters like ñ or ß.

The *Tokenizer* splits the text into multiple words.

The *Filter* adds, removes or replaces specific tokens.



Define a custom Analyzer

As you can see in the following code snippet, you can define a custom analyzer with an *@AnalyzerDef* annotation.

The analyzer definition is global and you can reference it by its name. So, better make sure to use an expressive name that you can easily remember. I choose the name *textanalyzer* in this example because I define a generic analyzer for text messages. It's a good fit for most simple text attributes.

This example doesn't require any character normalization or any other form of character filtering. The analyzer, therefore, doesn't need any *CharFilter*.

But it needs a *Tokenizer*. This one is required by all custom analyzers. It splits the text into words. In this example, I want to index my twitter messages. These are simple text messages which can be split

at whitespaces and punctuations. A *Tokenizer* created by Lucene's *StandardTokenizerFactory* can split these messages easily into words.

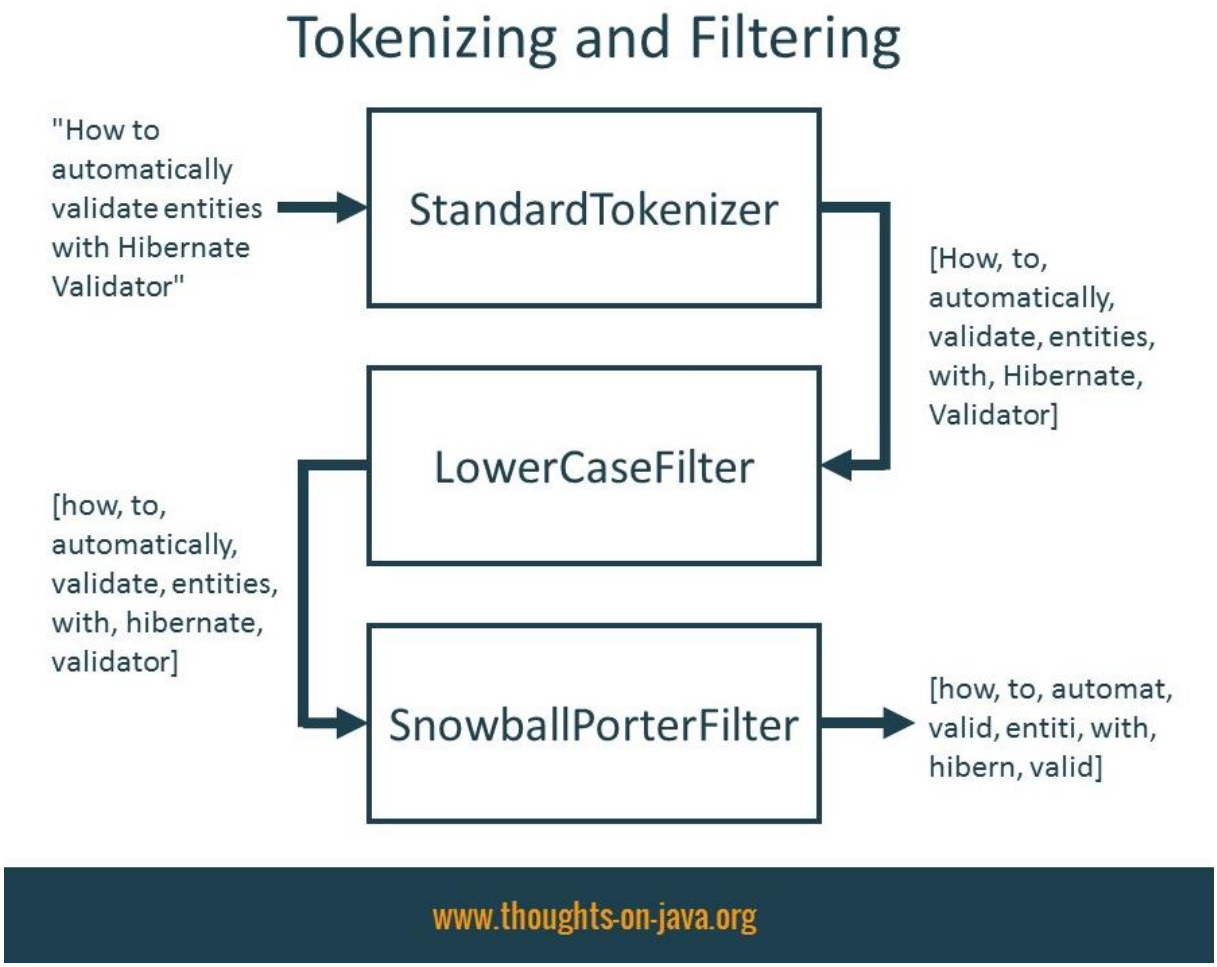
After that is done, you can apply *Filter* to the tokens to ignore case and add stemming.

In this example, I use the *LowerCaseFilterFactory* that transforms all tokens to lower case.

The *SnowballPorterFilterFactory* is more interesting. It creates a *Filter* that performs the stemming. As you can see in the code snippet, the *@TokenFilterDef* of the *SnowballPorterFilterFactory* requires an additional *@Parameter* annotation that provides the *language* that shall be used by the stemming algorithm. Almost all of my tweets are English so I set it to *English*.

```
@AnalyzerDef(
        name = "textanalyzer",
        tokenizer = @TokenizerDef(factory =
StandardTokenizerFactory.class),
        filters = {
                @TokenFilterDef(
                        factory = LowerCaseFilterFactory.class),
                @TokenFilterDef(
                        factory = SnowballPorterFilterFactory.class,
                        params = { @Parameter(name = "language",
                                value = "English") })
        }
)
```

That's all you need to do to define the *Analyzer*. The following graphic summarizes the effect of the configured *Tokenizer* and *Filter* steps.



Use a custom Analyzer

You can now reference the *@AnalyzerDef* by its name in an *@Analyzer* annotation to use it for an entity or an entity attribute. In the following code snippet, I assign the analyzer to the *message* attribute of the *Tweet* entity.

```
@Indexed
@Entity
public class Tweet {


	@Column
	@Field(analyzer = @Analyzer(definition =
“textanalyzer”))
	private String message;


	...
}
```

Hibernate Search applies the *textanalyzer* when it indexes the message attribute. It also applies it transparently when you use an entity attribute with a defined analyzer in a full-text query. That makes it easy to use and allows you to change an *Analyzer* without adapting your business code. But be careful, when you change an *Analyzer* for an existing database. I requires you to reindex your existing data.

```
FullTextEntityManager fullTextEm =
Search.getFullTextEntityManager(em);

QueryBuilder tweetQb =
fullTextEm.getSearchFactory().buildQueryBuilder().forEntity(
Tweet.class).get();

Query fullTextQuery =
tweetQb.keyword().onField(Tweet_.message.getName()).mat
ching(searchTerm).createQuery();

List<Tweet> results =
fullTextEm.createFullTextQuery(fullTextQuery,
Tweet.class).getResultList();
```