package com.twitter.tweetypie.matching

object TokenSequence {

/\*\*

\* Is `suffix` a suffix of `s`, starting at `offset` in `s`?

\*/

def hasSuffixAt(s: CharSequence, suffix: CharSequence, offset: Int): Boolean =

if (offset == 0 && (s.eq(suffix) || s == suffix)) {

true

} else if (suffix.length != (s.length - offset)) {

false

} else {

@annotation.tailrec

def go(i: Int): Boolean =

if (i == suffix.length) true

else if (suffix.charAt(i) == s.charAt(offset + i)) go(i + 1)

else false

go(0)

}

/\*\*

\* Do two [[CharSequence]]s contain the same characters?

\*

\* [[CharSequence]] equality is not sufficient because

\* [[CharSequence]]s of different types may not consider other

\* [[CharSequence]]s containing the same characters equivalent.

\*/

def sameCharacters(s1: CharSequence, s2: CharSequence): Boolean =

hasSuffixAt(s1, s2, 0)

/\*\*

\* This method implements the product definition of a token matching a

\* keyword. That definition is:

\*

\* - The token contains the same characters as the keyword.

\* - The token contains the same characters as the keyword after

\* dropping a leading '#' or '@' from the token.

\*

\* The intention is that a keyword matches an identical hashtag, but

\* if the keyword itself is a hashtag, it only matches the hashtag

\* form.

\*

\* The tokenization process should rule out tokens or keywords that

\* start with multiple '#' characters, even though this implementation

\* allows for e.g. token "##a" to match "#a".

\*/

def tokenMatches(token: CharSequence, keyword: CharSequence): Boolean =

if (sameCharacters(token, keyword)) true

else if (token.length == 0) false

else {

val tokenStart = token.charAt(0)

(tokenStart == '#' || tokenStart == '@') && hasSuffixAt(token, keyword, 1)

}

}

/\*\*

\* A sequence of normalized tokens. The sequence depends on the locale

\* in which the text was parsed and the version of the penguin library

\* that was used at tokenization time.

\*/

case class TokenSequence private[matching] (toIndexedSeq: IndexedSeq[CharSequence]) {

import TokenSequence.tokenMatches

private def apply(i: Int): CharSequence = toIndexedSeq(i)

def isEmpty: Boolean = toIndexedSeq.isEmpty

def nonEmpty: Boolean = toIndexedSeq.nonEmpty

/\*\*

\* Does the supplied sequence of keywords match a consecutive sequence

\* of tokens within this sequence?

\*/

def containsKeywordSequence(keywords: TokenSequence): Boolean = {

val finalIndex = toIndexedSeq.length - keywords.toIndexedSeq.length

@annotation.tailrec

def matchesAt(offset: Int, i: Int): Boolean =

if (i >= keywords.toIndexedSeq.length) true

else if (tokenMatches(this(i + offset), keywords(i))) matchesAt(offset, i + 1)

else false

@annotation.tailrec

def search(offset: Int): Boolean =

if (offset > finalIndex) false

else if (matchesAt(offset, 0)) true

else search(offset + 1)

search(0)

}

}