package com.twitter.product\_mixer.component\_library.selector

private[selector] object DynamicPositionSelector {

sealed trait IndexType

case object RelativeIndices extends IndexType

case object AbsoluteIndices extends IndexType

/\*\*

\* Given an existing `result` seq, inserts candidates from `candidatesToInsertByIndex` into the `result` 1-by-1 with

\* the provided index being the index relative to the `result` if given [[RelativeIndices]] or

\* absolute index if given [[AbsoluteIndices]] (excluding duplicate insertions at an index, see below).

\*

\* Indices below 0 are added to the front and indices > the length are added to the end

\*

\* @note if multiple candidates exist with the same index, they are inserted in the order which they appear and only count

\* as a single element with regards to the absolute index values, see the example below

\*

\* @example when using [[RelativeIndices]] {{{

\* mergeByIndexIntoResult(

\* Seq(

\* 0 -> "a",

\* 0 -> "b",

\* 0 -> "c",

\* 1 -> "e",

\* 2 -> "g",

\* 2 -> "h"),

\* Seq(

\* "D",

\* "F"

\* ),

\* RelativeIndices) == Seq(

\* "a",

\* "b",

\* "c",

\* "D",

\* "e",

\* "F",

\* "g",

\* "h"

\* )

\* }}}

\*

\* @example when using [[AbsoluteIndices]] {{{

\* mergeByIndexIntoResult(

\* Seq(

\* 0 -> "a",

\* 0 -> "b",

\* 1 -> "c",

\* 3 -> "e",

\* 5 -> "g",

\* 6 -> "h"),

\* Seq(

\* "D",

\* "F"

\* ),

\* AbsoluteIndices) == Seq(

\* "a", // index 0, "a" and "b" together only count as 1 element with regards to indexes because they have duplicate insertion points

\* "b", // index 0

\* "c", // index 1

\* "D", // index 2

\* "e", // index 3

\* "F", // index 4

\* "g", // index 5

\* "h" // index 6

\* )

\* }}}

\*/

def mergeByIndexIntoResult[T]( // generic on `T` to simplify unit testing

candidatesToInsertByIndex: Seq[(Int, T)],

result: Seq[T],

indexType: IndexType

): Seq[T] = {

val positionAndCandidateList = candidatesToInsertByIndex.sortWith {

case ((indexLeft: Int, \_), (indexRight: Int, \_)) =>

indexLeft < indexRight // order by desired absolute index ascending

}

// Merge result and positionAndCandidateList into resultUpdated while making sure that the entries

// from the positionAndCandidateList are inserted at the right index.

val resultUpdated = Seq.newBuilder[T]

resultUpdated.sizeHint(result.size + positionAndCandidateList.size)

var currentResultIndex = 0

val inputResultIterator = result.iterator

val positionAndCandidateIterator = positionAndCandidateList.iterator.buffered

var previousInsertPosition: Option[Int] = None

while (inputResultIterator.nonEmpty && positionAndCandidateIterator.nonEmpty) {

positionAndCandidateIterator.head match {

case (nextInsertionPosition, nextCandidateToInsert)

if previousInsertPosition.contains(nextInsertionPosition) =>

// inserting multiple candidates at the same index

resultUpdated += nextCandidateToInsert

// do not increment any indices, but insert the candidate and advance to the next candidate

positionAndCandidateIterator.next()

case (nextInsertionPosition, nextCandidateToInsert)

if currentResultIndex >= nextInsertionPosition =>

// inserting a candidate at a new index

// add candidate to the results

resultUpdated += nextCandidateToInsert

// save the position of the inserted element to handle duplicate index insertions

previousInsertPosition = Some(nextInsertionPosition)

// advance to next candidate

positionAndCandidateIterator.next()

if (indexType == AbsoluteIndices) {

// if the indices are absolute, instead of relative to the original `result` we need to

// count the insertions of candidates into the results towards the `currentResultIndex`

currentResultIndex += 1

}

case \_ =>

// no candidate to insert by index so use the candidates from the result and increment the index

resultUpdated += inputResultIterator.next()

currentResultIndex += 1

}

}

// one of the iterators is empty, so append the remaining candidates in order to the end

resultUpdated ++= positionAndCandidateIterator.map { case (\_, candidate) => candidate }

resultUpdated ++= inputResultIterator

resultUpdated.result()

}

}