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

import com.twitter.product\_mixer.core.functional\_component.common.CandidateScope

import com.twitter.product\_mixer.core.functional\_component.common.SpecificPipeline

import com.twitter.product\_mixer.core.functional\_component.common.SpecificPipelines

import com.twitter.product\_mixer.core.functional\_component.selector.Selector

import com.twitter.product\_mixer.core.functional\_component.selector.SelectorResult

import com.twitter.product\_mixer.core.model.common.identifier.CandidatePipelineIdentifier

import com.twitter.product\_mixer.core.model.common.presentation.CandidateWithDetails

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

/\*\*

\* Predicate which will be applied to each candidate. True indicates that the candidate will be

\* kept.

\*/

trait ShouldKeepCandidate {

def apply(candidateWithDetails: CandidateWithDetails): Boolean

}

object DropFilteredCandidates {

def apply(candidatePipeline: CandidatePipelineIdentifier, filter: ShouldKeepCandidate) =

new DropFilteredCandidates(SpecificPipeline(candidatePipeline), filter)

def apply(candidatePipelines: Set[CandidatePipelineIdentifier], filter: ShouldKeepCandidate) =

new DropFilteredCandidates(SpecificPipelines(candidatePipelines), filter)

}

/\*\*

\* Limit candidates from certain candidates sources to those which satisfy the provided predicate.

\*/

case class DropFilteredCandidates(

override val pipelineScope: CandidateScope,

filter: ShouldKeepCandidate)

extends Selector[PipelineQuery] {

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val candidatesUpdated = remainingCandidates.filter { candidate =>

if (pipelineScope.contains(candidate)) filter.apply(candidate)

else true

}

SelectorResult(remainingCandidates = candidatesUpdated, result = result)

}

}