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

import com.twitter.product\_mixer.component\_library.selector.DropSelector.dropDuplicates

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

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.\_

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.model.common.presentation.ModuleCandidateWithDetails

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

object DropDuplicateModuleItemCandidates {

/\*\*

\* Limit the number of module item candidates (for 1 or more modules) from a certain candidate

\* source. See [[DropDuplicateModuleItemCandidates]] for more details.

\*

\* @param candidatePipeline pipelines on which to run the selector

\*

\* @note Scala doesn't allow overloaded methods with default arguments. Users wanting to customize

\* the de-dupe logic should use the default constructor. We could provide multiple

\* constructors but that seemed more confusing (five ways to instantiate the selector) or not

\* necessarily less verbose (if we picked specific use-cases rather than trying to support

\* everything).

\*/

def apply(candidatePipeline: CandidatePipelineIdentifier) = new DropDuplicateModuleItemCandidates(

SpecificPipeline(candidatePipeline),

IdAndClassDuplicationKey,

PickFirstCandidateMerger)

def apply(candidatePipelines: Set[CandidatePipelineIdentifier]) =

new DropDuplicateModuleItemCandidates(

SpecificPipelines(candidatePipelines),

IdAndClassDuplicationKey,

PickFirstCandidateMerger)

}

/\*\*

\* Limit the number of module item candidates (for 1 or more modules) from certain candidate

\* pipelines.

\*

\* This acts like a [[DropDuplicateCandidates]] but for modules in `remainingCandidates`

\* from any of the provided [[candidatePipelines]]. Similar to [[DropDuplicateCandidates]], it

\* keeps only the first instance of a candidate within a module as determined by comparing

\* the contained candidate ID and class type.

\*

\* @param pipelineScope pipeline scope on which to run the selector

\* @param duplicationKey how to generate the key used to identify duplicate candidates (by default use id and class name)

\* @param mergeStrategy how to merge two candidates with the same key (by default pick the first one)

\*

\* For example, if a candidatePipeline returned 5 modules each

\* containing duplicate items in the candidate pool, then the module items in each of the

\* 5 modules will be filtered to the unique items within each module.

\*

\* Another example is if you have 2 modules each with the same items as the other,

\* it won't deduplicate across modules.

\*

\* @note this updates the module in the `remainingCandidates`

\*/

case class DropDuplicateModuleItemCandidates(

override val pipelineScope: CandidateScope,

duplicationKey: DeduplicationKey[\_] = IdAndClassDuplicationKey,

mergeStrategy: CandidateMergeStrategy = PickFirstCandidateMerger)

extends Selector[PipelineQuery] {

override def apply(

query: PipelineQuery,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val remainingCandidatesLimited = remainingCandidates.map {

case module: ModuleCandidateWithDetails if pipelineScope.contains(module) =>

// this applies to all candidates in a module, even if they are from a different

// candidate source, which can happen if items are added to a module during selection

module.copy(candidates = dropDuplicates(

pipelineScope = AllPipelines,

candidates = module.candidates,

duplicationKey = duplicationKey,

mergeStrategy = mergeStrategy))

case candidate => candidate

}

SelectorResult(remainingCandidates = remainingCandidatesLimited, result = result)

}

}