package com.twitter.product\_mixer.component\_library.model.candidate

import com.twitter.product\_mixer.core.model.common.UniversalNoun

trait BaseArticleCandidate extends UniversalNoun[Int]

/\*\*

\* Canonical ArticleCandidate model. Always prefer this version over all other variants.

\*

\* @note Any additional fields should be added as a [[com.twitter.product\_mixer.core.feature.Feature]]

\* on the candidate's [[com.twitter.product\_mixer.core.feature.featuremap.FeatureMap]]. If the

\* features come from the candidate source itself (as opposed to hydrated via a

\* [[com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.CandidateFeatureHydrator]]),

\* then [[com.twitter.product\_mixer.core.pipeline.candidate.CandidatePipelineConfig.featuresFromCandidateSourceTransformers]]

\* can be used to extract features from the candidate source response.

\*

\* @note This class should always remain `final`. If for any reason the `final` modifier is removed,

\* the equals() implementation must be updated in order to handle class inheritor equality

\* (see note on the equals method below)

\*/

final class ArticleCandidate private (

override val id: Int)

extends BaseArticleCandidate {

/\*\*

\* @inheritdoc

\*/

override def canEqual(that: Any): Boolean = that.isInstanceOf[ArticleCandidate]

/\*\*

\* High performance implementation of equals method that leverages:

\* - Referential equality short circuit

\* - Cached hashcode equality short circuit

\* - Field values are only checked if the hashCodes are equal to handle the unlikely case

\* of a hashCode collision

\* - Removal of check for `that` being an equals-compatible descendant since this class is final

\*

\* @note `candidate.canEqual(this)` is not necessary because this class is final

\* @see [[http://www.artima.com/pins1ed/object-equality.html Programming in Scala,

\* Chapter 28]] for discussion and design.

\*/

override def equals(that: Any): Boolean =

that match {

case candidate: ArticleCandidate =>

(

(this eq candidate)

|| ((hashCode == candidate.hashCode)

&& (id == candidate.id))

)

case \_ =>

false

}

/\*\*

\* Leverage domain-specific constraints (see notes below) to safely construct and cache the

\* hashCode as a val, such that it is instantiated once on object construction. This prevents the

\* need to recompute the hashCode on each hashCode() invocation, which is the behavior of the

\* Scala compiler case class-generated hashCode() since it cannot make assumptions regarding field

\* object mutability and hashCode implementations.

\*

\* @note Caching the hashCode is only safe if all of the fields used to construct the hashCode

\* are immutable. This includes:

\* - Inability to mutate the object reference on for an existing instantiated candidate

\* (i.e. each field is a val)

\* - Inability to mutate the field object instance itself (i.e. each field is an immutable

\* - Inability to mutate the field object instance itself (i.e. each field is an immutable

\* data structure), assuming stable hashCode implementations for these objects

\*

\* @note In order for the hashCode to be consistent with object equality, `##` must be used for

\* boxed numeric types and null. As such, always prefer `.##` over `.hashCode()`.

\*/

override val hashCode: Int = id.##

}

object ArticleCandidate {

def apply(id: Int): ArticleCandidate = new ArticleCandidate(id)

}