Incentive Alignment for OSS Library Maintainers

Java Ecosystem Working group

Donald Raab, JCP EC Meeting, October 16, 2024

Progress in the Java Ecosystem

- Java Ecosystem Wiki
  - https://github.com/jcp-org/Java-Ecosystem-JCP-Working-Group/wiki
- JEP 14 - The Tip & Tail Model of Library Development
  - https://openjdk.org/jeps/14
  - This is a great guide for new libraries in active development

Different Libraries, Different Incentives

- Higher level frameworks have greater incentives to baseline on newer JDK versions
  - Pervasive Frameworks move the application market with them and drive adoption of JDK upgrades
    - Spring and SpringBoot are great examples
- Lower level libraries have different incentives
- Community Driven OSS vs. Corporate Sponsored OSS
  - Volunteers vs. Paid developers
- **We need to understand what would incentivize library maintainers to baseline on new versions of the JDK**

Incentive Alignment for Library Maintainers

- Releases are expensive, and so are upgrades
- Libraries are incentivized to test against latest versions of JDK
  - Many Libraries test against multiple versions of the JDK in OpenJDK Quality Outreach Program
  - Testing multiple with JDKs was made less expensive by tool automation
- Upgrades of dependent libraries (e.g., JUnit 4 -> 5 migration) can be very expensive
  - Eclipse Collections had a volunteer contributor from the community take this on
  - Open Rewrite helped with some of the effort, but the biggest challenge was reviewing all of the pull requests… we broke the work up into many pull requests over 6-9 months.
- **Library maintainers need incentives to baseline JDK on newer versions**

Library Incentives Evolve Over Time

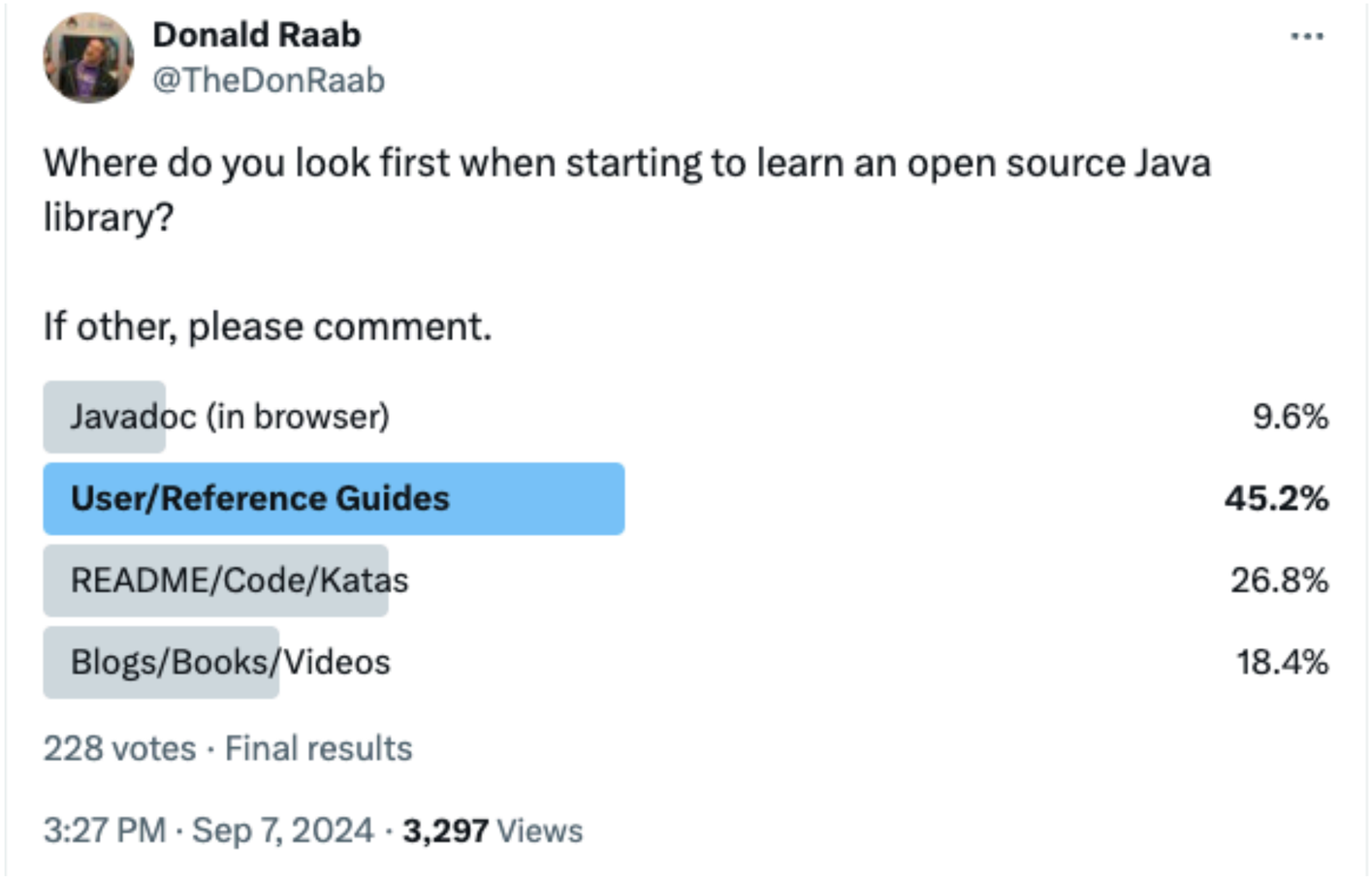
- Newer Libraries
  - Releases are frequent and potentially less stable
  - New language features can fuel productivity
  - Marketing loves freshness and excitement
- Mature Libraries
  - Releases are event based (e.g., bug fixes) and need to be stable
  - Documentation becomes more important as time passes

Eclipse Collections Upgrade Incentives

- Eclipse Collections is a low level library and driven by a volunteer OSS community
- Some JDK Upgrade Incentives for Eclipse Collections
  - Documentation Improvements
    - Consistent Javadoc/IDE support for Method Categories
    - Show packages in tree view, in addition to flattened list
  - Compatibility, Memory, Performance
    - Project Valhalla with generic specialization for primitives
    - New Collections interfaces supporting large collection sizes

Where do developers start learning OSS?

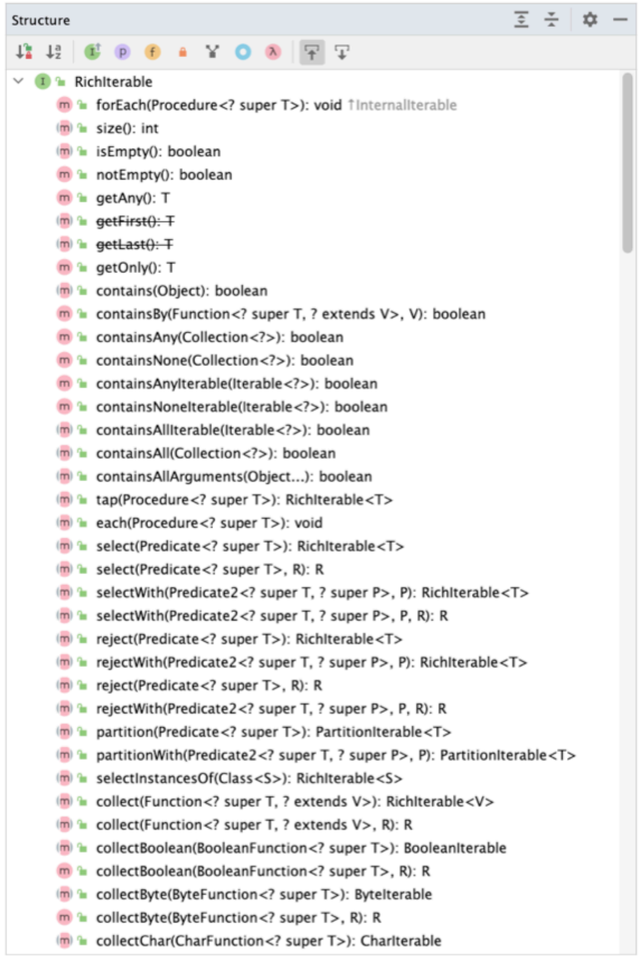
A Twitter/X Poll



Custom Code Folding Regions in IntelliJ

Structure View in IntelliJ with RichIterable

RichIterable (Eclipse Collections 11.x)

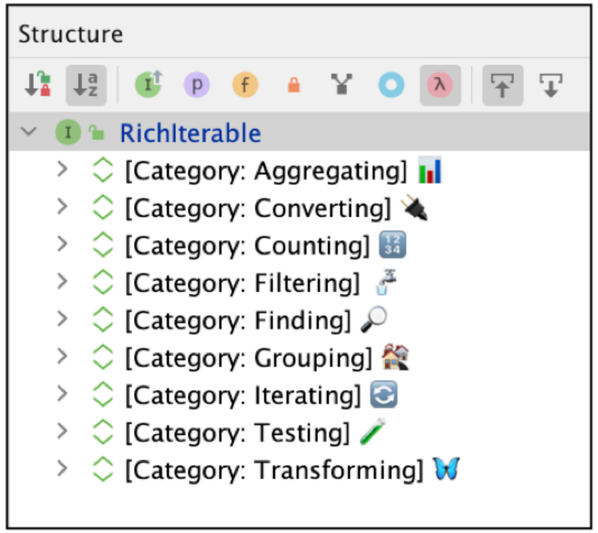


No Method Categories - Order in File

RichIterable (Eclipse Collections 11.x)

No Method Categories - Alpha Order

RichIterable (Eclipse Collections 12.x)



with Method Categories - Alpha Order

Javadoc for RichIterable with Method Categories

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

ALL CLASSES

SEARCH: Search

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

```
public interface RichIterable<T>
extends InternalIterable<T>
```

RichIterable is a read-only interface which extends the InternalIterable interface and adds many internal iterator methods. The basic methods were inspired by the Smalltalk Collection protocol. These methods include select, reject, detect, collect, injectInto, anySatisfy, allSatisfy. The API also includes converter methods to convert a RichIterable to other mutable and immutable collection types. The converter methods typically have a prefix of "to" (e.g. toList, toSet, toBag, toMap, etc.).

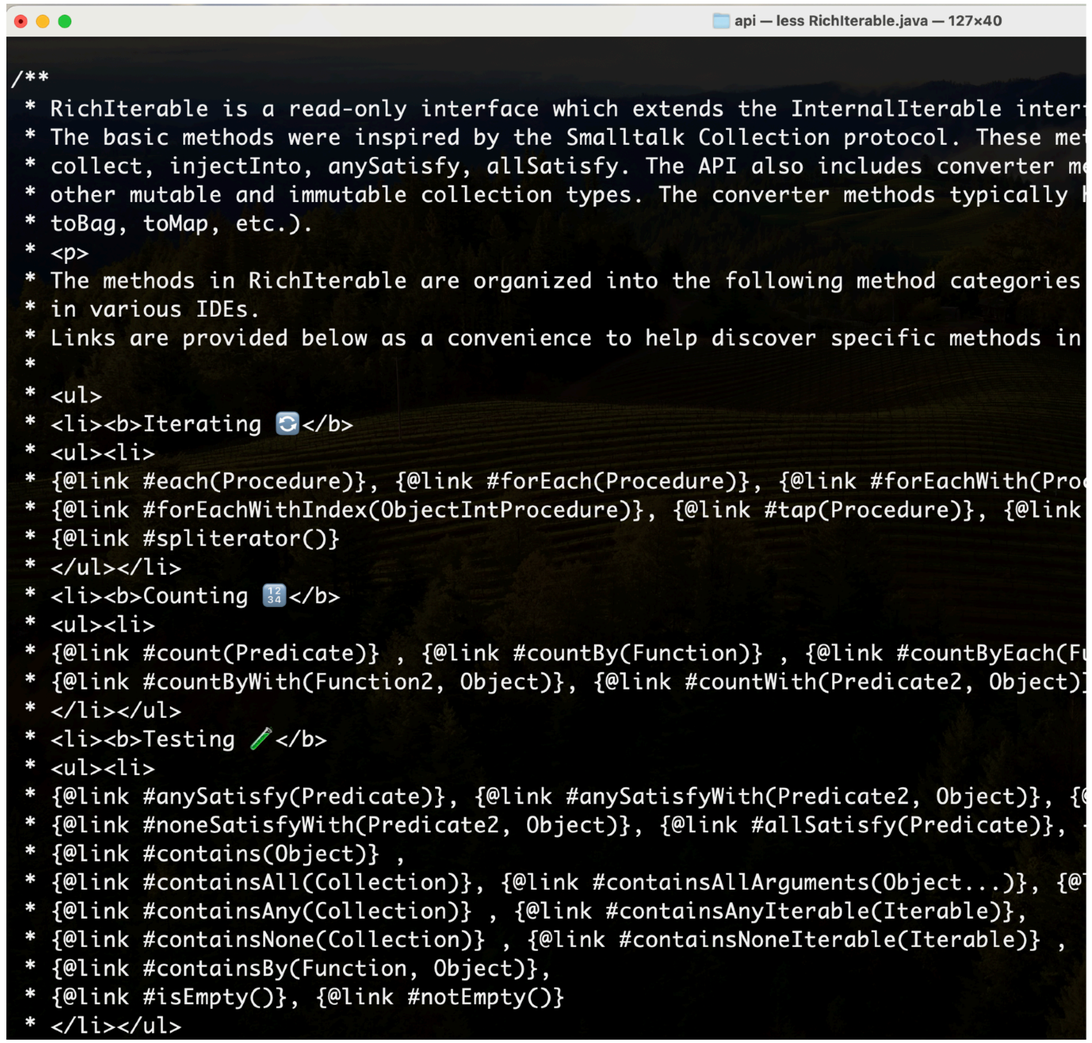
The methods in RichIterable are organized into the following method categories via region comments which are collapsible in various IDEs. Links are provided below as a convenience to help discover specific methods in Javadoc.

- **Iterating** 🔄
  - each(Procedure), forEach(Procedure), InternalIterable.forEachWith(Procedure2, Object), InternalIterable.forEachWithIndex(ObjectIntProcedure), tap(Procedure), asLazy(), Iterable.iterator(), Iterable.spliterator()
- **Counting** 🔢
  - count(Predicate), countBy(Function), countByEach(Function), countByWith(Function2, Object), countWith(Predicate2, Object), size()
- **Testing** 🧪
  - anySatisfy(Predicate), anySatisfyWith(Predicate2, Object), noneSatisfy(Predicate), noneSatisfyWith(Predicate2, Object), allSatisfy(Predicate), allSatisfyWith(Predicate2, Object), contains(Object), containsAll(Collection), containsAllArguments(Object...), containsAllIterable(Iterable), containsAny(Collection), containsAnyIterable(Iterable), containsNone(Collection), containsNoneIterable(Iterable), containsBy(Function, Object), isEmpty(), notEmpty()
- **Finding** 🔎
  - detect(Predicate), detectIfNone(Predicate, Function0), detectWith(Predicate2, Object), detectWithIfNone(Predicate2, Object, Function0), detectOptional(Predicate), detectWithOptional(Predicate2, Object), getAny(), getFirst(), getLast(), getOnly(), min(), max(), minBy(Function), minByOptional(Function), minOptional() maxBy(Function), maxByOptional(Function), maxOptional()
- **Filtering** 🚰
  - select(Predicate), selectInstancesOf(Class), selectWith(Predicate2, Object), reject(Predicate), rejectWith(Predicate2, Object), partition(Predicate), partitionWith(Predicate2, Object)
- **Transforming** 🦋
  - collect(Function), collectIf(Predicate, Function), collectWith(Function2, Object), collectBoolean(BooleanFunction), collectByte(ByteFunction), collectChar(CharFunction), collectDouble(DoubleFunction), collectFloat(FloatFunction), collectInt(IntFunction), collectLong(LongFunction), collectShort(ShortFunction), flatCollect(Function), flatCollectWith(Function2, Object), flatCollectBoolean(Function, MutableBooleanCollection), flatCollectByte(Function, MutableByteCollection), flatCollectChar(Function, MutableCharCollection), flatCollectDouble(Function, MutableDoubleCollection), flatCollectFloat(Function, MutableFloatCollection), flatCollectInt(Function, MutableIntCollection), flatCollectLong(Function, MutableLongCollection), flatCollectShort(Function, MutableShortCollection), zip(Iterable), zipWithIndex(Collection)
- **Grouping** 🏘️
  - chunk(int), groupBy(Function), groupByAndCollect(Function, Function, MutableMultimap), groupByEach(Function), groupByUniqueKey(Function)
- **Aggregating** 📊
  - aggregateBy(Function, Function0, Function2), aggregateInPlaceBy(Function, Function0, Procedure2), injectInto(Object, Function2), injectIntoDouble(double, DoubleObjectToDoubleFunction), injectIntoFloat(float, FloatObjectToFloatFunction), injectIntoInt(int, IntObjectToIntFunction), injectIntoLong(long, LongObjectToLongFunction), reduce(BinaryOperator), reduceBy(Function, Function2), reduceInPlace(Collector), sumByDouble(Function, DoubleFunction), sumByFloat(Function, FloatFunction), sumByInt(Function, IntFunction), sumByLong(Function, LongFunction), sumOfDouble(DoubleFunction), sumOfFloat(FloatFunction), sumOfInt(IntFunction), sumOfLong(LongFunction), summarizeDouble(DoubleFunction), summarizeFloat(FloatFunction), summarizeInt(IntFunction), summarizeLong(LongFunction)
- **Converting** 🔌
  - into(Collection), appendString(Appendable), makeString(), toString(), toArray(), toBag(), toBiMap(Function, Function), toList(), toMap(Function, Function), toSet(), toSortedBag(), toSortedBagBy(Function), toSortedList(), toSortedListBy(Function), toSortedMap(Function, Function), toSortedMapBy(Function, Function, Function), toSortedSet(), toSortedSetBy(Function), toImmutableBag(), toImmutableBiMap(Function, Function), toImmutableList(), toImmutableMap(Function, Function), toImmutableSet(), toImmutableSortedBag(), toImmutableSortedBagBy(Function), toImmutableSortedList(), toImmutableSortedListBy(Function), toImmutableSortedSet(), toImmutableSortedSetBy(Function)

Javadoc created by hand to match structure in IntelliJ with Custom Code Folding Regions

Enhance Scanning in Plain Text / GitHub

Emojis can help break huge walls of text in large files



Code | Blame | 2555 lines (2352 loc) · 97.9 KB

```
 92  /**
 93   * RichIterable is a read-only interface which extends the InternalIterable interface and
 94   * The basic methods were inspired by the Smalltalk Collection protocol. These methods in
 95   * collect, injectInto, anySatisfy, allSatisfy. The API also includes converter methods t
 96   * other mutable and immutable collection types. The converter methods typically have a p
 97   * toBag, toMap, etc.).
 98   * <p>
 99   * The methods in RichIterable are organized into the following method categories via reg
100   * in various IDEs.
101   * Links are provided below as a convenience to help discover specific methods in Javadoc
102   *
103   * <ul>
104   * <li><b>Iterating 🔁</b>
105   * <ul><li>
106   * {@link #each(Procedure)}, {@link #forEach(Procedure)}, {@link #forEachWith(Procedure2,
107   * {@link #forEachWithIndex(ObjectIntProcedure)}, {@link #tap(Procedure)}, {@link #asLazy
108   * {@link #spliterator()}
109   * </ul></li>
110   * <li><b>Counting 🔢</b>
111   * <ul><li>
112   * {@link #count(Predicate)} , {@link #countBy(Function)} , {@link #countByEach(Function)
113   * {@link #countByWith(Function2, Object)}, {@link #countWith(Predicate2, Object)} , {@li
114   * </li></ul>
115   * <li><b>Testing 🧪</b>
116   * <ul><li>
117   * {@link #anySatisfy(Predicate)}, {@link #anySatisfyWith(Predicate2, Object)}, {@link #n
118   * {@link #noneSatisfyWith(Predicate2, Object)}, {@link #allSatisfy(Predicate)}, {@link #
```

Useful links to understand EC incentives

- Two blogs in Java Tutorials and Tips in Java Annotated Monthly
  - https://blog.jetbrains.com/idea/2024/10/java-annotated-monthly-october-2024/
- Blog about 64-bit collections in Smalltalk and Java
  - https://donraab.medium.com/10-billion-integers-walk-into-an-array-37097386c964?source=friends_link&sk=1fd9c1da8ac281c6b66bb41a17a0125e