Method Chaining Redux: An Empirical Study of Method Chaining in Java, Kotlin, and Python

Ali M. Keshk and Robert Dyer
What is Method Chaining?

**without method chaining**

```java
1 JSONObject obj = new JSONObject();
2 obj.put("Conference", "MSR");
3 obj.put("Year", "2023");
4 String json = obj.toJSONString();
```
What is Method Chaining?

without method chaining

```java
1  JSONObject obj = new JSONObject();
2  obj.put("Conference", "MSR");
3  obj.put("Year", "2023");
4  String json = obj.toJSONString();
```
What is Method Chaining?

without method chaining

1. `JSONObject obj = new JSONObject();`
2. `obj.put("Conference", "MSR");`
3. `obj.put("Year", "2023");`
4. `String json = obj.toString();`
What is Method Chaining?

without method chaining

```java
1 JSONObject obj = new JSONObject();
2 obj.put("Conference", "MSR");
3 obj.put("Year", "2023");
4 String json = obj.toJSONString();
```

with method chaining

```java
1 String json = new JSONObject()
2 .put("Conference", "MSR")
3 .put("Year", "2023")
4 .toString();
```
What is Method Chaining?

Definition: We define a method chain as a sequence of one or more method invocations joined by the '.' symbol.

without method chaining

```java
JSONObject obj = new JSONObject();
obj.put("Conference", "MSR");
obj.put("Year", "2023");
String json = obj.toJSONString();
```

with method chaining

```java
String json = new JSONObject()
    .put("Conference", "MSR")
    .put("Year", "2023")
    .toJSONString();
```
Motivation/Impact on APIs

• Language implementers could optimize their compilers to account for method chaining.

• High method chain use can lead API developers to write fluent designs.

• There is a debate on whether method chaining is good or bad practice.
Previous Empirical Study

Nakamaru et al. MSR’20

- Do developers use method chaining in Java?

Nakamaru et al. MSR’20

- Do developers use method chaining in Java?
- Finding: use of method chains in Java is **popular** and **increasing**

Previous Empirical Study

(a) $f_n$ in 2010 and 2018

(b) $r$

Research Questions

RQ1 Replication

RQ2 Extend to larger Java dataset?

RQ3 Generalize to Kotlin?

RQ4 Generalize to Python?

RQ5 Can we support the language extensions proposed by the prior study?
### Boa Datasets

<table>
<thead>
<tr>
<th>Boa Dataset</th>
<th>Number of Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Original</td>
<td>&quot;2021 Method Chains&quot;</td>
</tr>
<tr>
<td>Java Full</td>
<td>&quot;2022 Jan/Java&quot;</td>
</tr>
<tr>
<td>Kotlin</td>
<td>&quot;2021 Aug/Kotlin&quot;</td>
</tr>
<tr>
<td>Python</td>
<td>&quot;2022 Feb/Python&quot;</td>
</tr>
</tbody>
</table>

- Filtered older years with less than 250 projects
- Filtered so all data ends 31 Dec 2020 to avoid partial years
- Duplicate files across projects are filtered out using the hash value of each file's AST
1. `new C();` // not a method chain
2. `super();` // not a method chain

3. // length: 1
4. `o.m();` // explicit receiver
5. `m();` // implicit 'this' receiver
6. `new C().m();` // constructor not included
7. `super.m();` // super is not a call here

8. // length: 2
9. `o.m().n();`

10. // two chains, each length: 1
11. `m(n());`
12. `m().f.n();`
\[ f_n = \frac{m_n}{m_1} \]

\[ r = \frac{\sum_{n \geq 2} n \cdot m_n}{\sum_{n \geq 1} n \cdot m_n} \]

\[ U_n: \text{percent of all projects containing at least one chain whose length is } \geq n \]

**Testing code:** file path including case-insensitive string "test" anywhere
RQ1: Replication

Previous Study’s Dataset

Our Dataset
RQ2: Larger Java Dataset

Increasing trend continues

Prior Study

This Study

% of Chains

Year

% of Chains

Year

% of Chains

2010
2011
2012
2013
2014
2015
2016
2017
2018
RQ2: Larger Java Dataset

Prior Study

This Study

Increasing trend continues

Trend decreased in the past!
RQ2: Larger Java Dataset

Prior Study

This Study
RQ2: Larger Java Dataset

This Study
• 1/3 of extra long chains (n in top 5%) come from the following Java APIs:
  
  • java.lang.StringBuilder
  • org.springframework.security...builders.HttpSecurity
  • com.google.common.collect.ImmutableMap
  • java.lang.StringBuffer
  • javax.swing.GroupLayout
RQ3: Kotlin

Java

Kotlin

% of Chains

Year

% of Chains

Year


RQ3: Kotlin
$U_n$ values by language

<table>
<thead>
<tr>
<th>$n$ (minimum chain length)</th>
<th>Java</th>
<th>Kotlin</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0%</td>
<td>25%</td>
<td>5%</td>
</tr>
<tr>
<td>9</td>
<td>10%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>41</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>42</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

RQ4: Python
RQ4: Python

![Graph showing the percentage of chains over years. The graph includes two categories: non-test and test. The y-axis represents the percentage of chains ranging from 0.00 to 0.40. The x-axis represents the years from 2005 to 2020. The non-test category is represented by blue bars, and the test category is represented by orange bars. The graph shows a consistent increase in the percentage of chains over the years.]
RQ4: Python
RQ5: Language Extensions

NullExceptionAvoidance

\[ m1() \text{?} \cdot m2() \cdot m3() \]

RepeatedReceiver

\[ o.m1(); o.m2(); \]

DownCast

\[ m1().asC().m2().m3() \]

ConditionalExecution

\[ \text{if } (o.m1()) o.m2() \]

51.5% of 499K Kotlin projects

> 19% of chains

< 0.1% of chains

< 0.1% of chains
Kotlin similar to Java, but not growing

Python overall less use, but more in non-tests...

Java results similar to prior study

Language feature support

NullExceptionAvoidance
51.5% of 499k Kotlin projects

RepeatedReceiver
> 19% of chains

...smaller chains