Casting about in the Dark - Artifact Evaluation for OOPSLA'19

Our paper is an empirical study about how developers use casting in Java. We have manually inspected and categorized 5,000 cast operations. This artifact provides the cast categorization we have used in the paper. Moreover, this artifact provides an Overview, the Getting Started Guide and the Step by Step Instructions on how to extract a sample from our cast dataset.

A note about *consistency*. This artifact differs in *three* ways with the *submitted* paper.

1. The reviewers pointed out that some cast instances were clustered together, when they should be categorized as different patterns. Thus, we had to split some patterns into new ones. In Section Manual Categorization of Casts we describe which are the new patterns, and which ones were removed.
2. Additionally, the reviewers mentioned that it is important to know the distribution of casts per projects. In Section Distribution of Casts we provide the steps to gather the cast distribution. We plan to incorporate these new results into the final version of the paper.
3. Finally, to satisfy the methodology described in Section 2.2 of the paper, we had to effectively analyzed 5,000 casts. This improvement is described in Section Manual Categorization of Casts.

Overview

This artifact is provided through a source code control repository. It can be found online at https://gitlab.com/acuarica/java-cast-oopsla-19-aec. Given the nature of our empirical study, no need to compile source code and just a few dependencies, we decided that a source code control repository would be a more appropriate to distribute the artifact rather than a Virtual Machine (recommended method).

The root folder contains the following files:

- `Makefile`: The Makefile script to run all commands described in the *Step by Step Instructions*.
- `query-results.tar.xz` and `query-results.json`: This is the dataset provided directly by Semmle. The `query-results.tar.xz` can be found online.
- `casts-*.csv`: The cast samples that have been manually inspected.
- `casts.csv`: Consolidated casts from `casts-*.csv`.
- `sample-casts-5000.csv`: A random sample output of casts.
- `import.py`: Script to extract the cast instances into a SQLite database.
- `sample.r`: Script to create a sample from the cast database.
- `analysis.r`: Main script to make tables and plots.

- `dist.r`: Script to make the cast distribution plots.
- `casts.def`: Statistics about patterns.
- `input-patterns.def`: LaTeX include file to include each pattern sorted by frequency.
- `table-casts-patterns.def`: Patterns table where patterns are sorted by frequency.
- `table-patterns.pdf` and `patterns/*.pdf`: Generated plots by `analysis.r`.
- `dist-csv.pdf` and `dist-population.pdf`: Distribution plots generated by `dist.r`.

The `*.def` files are automatically generated files by the `analysis.r` script in LaTeX format to be included directly in the source of the paper.

Getting Started Guide

The scripts uses different tools and languages:

- `git`
- `make`
- `tar`
- `python 2.7`
- `R`

To run the R scripts, the following R packages are needed to be installed:

```
install.packages("DBI")
install.packages("RSQLite")
install.packages("ggplot2")
install.packages("tidyr")
install.packages("plyr")
install.packages("reshape2")
```

The scripts in the following section were tested on macOS (10.14) and Linux (Mint 19.1 64-bit).

Step by Step Instructions

Uncompress cast results

The cast dataset provided by Semmle is a `.tar.xz` compressed file. The following command uncompress this file.

```
make untar
```

Expected output

The `query-results` folder should be created. This folder contains the casts found per source control host and project.

For each project there is an `output.csv` file with the cast results as well as `stderr` and `stdout` files, which are not interesting (these are log files). Note that each `output.csv` file contains a header line, so if the file has one line this means there were not any casts.

The directory names tell you which project the query was run on. For example, `query-results/github/yanzhenjie/NoHttp/1506231257083:1506170287574:2966721477769636017/outpu` contains the results of running your query on the GitHub project `yanzhenjie/NoHttp` (i.e., https://github.com/yanzhenjie/NoHttp). The last component is an internal Semmle identifier is of not interest to us.

Importing all casts into a single database

Once the query results were extracted, they need to be imported in a SQLite database for better manipulation. This is done with the following command.

```
make import
```

Expected output

The database `output.sqlite3` should be created.

Creating sample table

This step shows how to create a random sample from the cast population. Once the sample is created, we have manually annotated each row with the result of the manual inspection.

```
make sample
```

Expected output

The sample file `sample-casts-5000.csv` should be created.

Manual categorization of casts

The manual categorization tables are `casts-5000.csv`, `casts-480.csv`, `casts-47.csv` and `casts-3.csv`. A consolidated table made can be found in `casts.csv`. These files are comma-separated values (CSV) tables.

Each row represents a cast instance. This table contains 6 columns. The *castid* and *repoid* columns represent internal IDs to uniquely identify each cast instance and each project. The *target* and *source* columns indicate the source and target types used in the cast. The last two columns—*link* and *value*—are the link to the source code file in lgtm.com and the result of the manual inspection.

We had to sample more than 5000 casts. The CSV table mentioned above contains 5,530 casts (rows). This is because we found 526 links that were not accessible during our analysis, making manual code inspection impossible. Inaccessible links can be found because some projects were removed from the lgtm platform. We also found 1 cast that was clearly a bug, a downcast using the wrong cast operand. Thus, we had to resample the cast instances until we reach 5,000 manually inspected casts. When resampling, we took care of inspecting *different* cast instances, i.e., we have discarded duplicated casts. We found 3 duplicated casts when resampling.

The manual inspection (*value* column) is a string with the following format:

```
(#(%variant)(:%args)?)+,@(src|test|gen)|?BrokenLink|?Duplicated|?Bug
```

where `%variant` is a declared variant of a pattern (see table below) and `%args` is a free string (optional) used to make a comment on a cast. The `@` symbol indicates the scope of the cast: `src` is used to indicate that the cast appears in application/library code, `test` for test code, and `gen` for generated code. The markers `?BrokenLink`, `?Duplicated`, and `?Bug` indicate whether the link of the cast is broken, the link was duplicated, or the cast is a bug respectively.

The full list of possible `%variant`s is given by the *Variant* column. The pattern for which a variant belongs to is given by the *Pattern* column.

Pattern	Variant
Typecase	GuardByInstanceOf, GuardByTypeTag, GuardByClassLiteral
Equals	Equals
OperandStack	OperandStack
Family	Family
Factory	Factory, GetOrCreateByClassLiteral
Deserialization	Deserialization
Composite	Composite
NewDynamicInstance	NewDynamicInstance
Stash	LookupById, Tag, StaticResource
CovariantReturnType	CovariantReturnType, Clone
Redundant	Redundant
VariableSupertype	VariableSupertype
UseRawType	UseRawType
RemoveWildcard	RemoveWildcard
KnownReturnType	KnownReturnType
ObjectAsArray	ObjectAsArray
AccessSuperclassField	AccessSuperclassField
SelectOverload	SelectOverload
ReflectiveAccessibility	ReflectiveAccessibility
CovariantGeneric	CovariantGeneric
SoleSubclassImplementation	SoleSubclassImplementation
FluentAPI	FluentAPI

Pattern	Variant
ImplicitIntersectionType	ImplicitIntersectionType
GenericArray	GenericArray, MatchBoxedType
UnoccupiedTypeParameter	UnoccupiedTypeParameter

For example, the value `#LookupById,@test` indicates that the cast exhibits the *LookupById* variant, in the *Stash* pattern, and the cast appears in test code. As an example with arguments, the value `#GuardByInstanceOf:single,@src` indicates the *GuardByInstanceOf* variant, in the *Typecase* pattern, marked as *single*, and the cast appears in application or library code.

In the submitted paper we had 23 usage patterns. We now have 25 usage patterns. This is because of we have split *IncompleteGenericType* into *UseRawType* and *RemoveWildcard*, and we extracted the *Equals* patterns from *Typecase*.

The script to process the results of the manual inspection is `analysis.r`. To run the analysis, run the following command:

```
make analysis
```

Expected output

This script creates the `table-patterns.pdf` and `patterns/*.pdf` plots, and `casts.def`, `input-patterns.def`, and `table-casts-patterns.def` to be included in source paper. The `table-casts-patterns.def` is the main table shown in Section 4 of the paper. The plots were asked by the reviewers to show how the sample is distributed in Application/Library code, Test code, and Generated code.

Distribution of casts

As mentioned in the Overview, the reviewers asked us to show the distribution of casts across projects. In particular, they want to know whether casts are mostly clustered within a small number of projects or are distributed relatively evenly across projects. To that end, we provide a plot whose x axis is the number of casts (in log scale) and whose y axis is density of projects with x or fewer casts.

The following command does so.

```
make dist
```

Expected output

The `dist-population.pdf` and `dist-csv.pdf` plots should be created. The `dist-population.pdf` uses the total population of casts, and `dist-csv.pdf` uses the cast sample we manually inspected.