Verifying the Option Type with Rely–Guarantee Reasoning

James Yoo, Michael D. Ernst, René Just
University of Washington

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING
PLSE

Key Insights

Γ **The option type**

- Either unsafe or verbose

✅ **Optional Checker: Verifier for the option type**

- *Partial* rely-guarantee reasoning

📋 **Large-scale evaluation**

- Effective: **93%** precision and **100%** recall, **13 real-world defects**

Explicitly Encoding Presence and Absence

`Option`

`Optional<T>`,
`Maybe T, ...`

is either...

`Present`

`Some T`,
`Just T, ...`

`Absent`

`None`,
`Nothing, ...`

Some languages with option types






```
String  format(Optional<Transaction> t) {

    String id = t.get().id;
    String time = t.get().timestamp;
    return "Transaction: " + id + "at: " + time;
}

format(ts);
```

```
Exception in thread  “main” java.util.NoSuchElementException : No value present
— at java.base/java.util.Optional.get(Optional.java:148)
```

How to prevent NoSuchElementException using Optional class in Java [duplicate]

Ask Question

Asked 2 years, 4 months ago Modified 5 months

People also ask

What does NoSuchElementException mean?

How to resolve NoSuchElementException?

When to throw NoSuchElementException?

What is the difference between element not found exception and no such element exception?

Discussions and forums

Avoid NoSuchElementException with Stream - java

Stack Overflow · 4 answers · 9y ago

"Exception java.util.NoSuchElementException: No value present ...

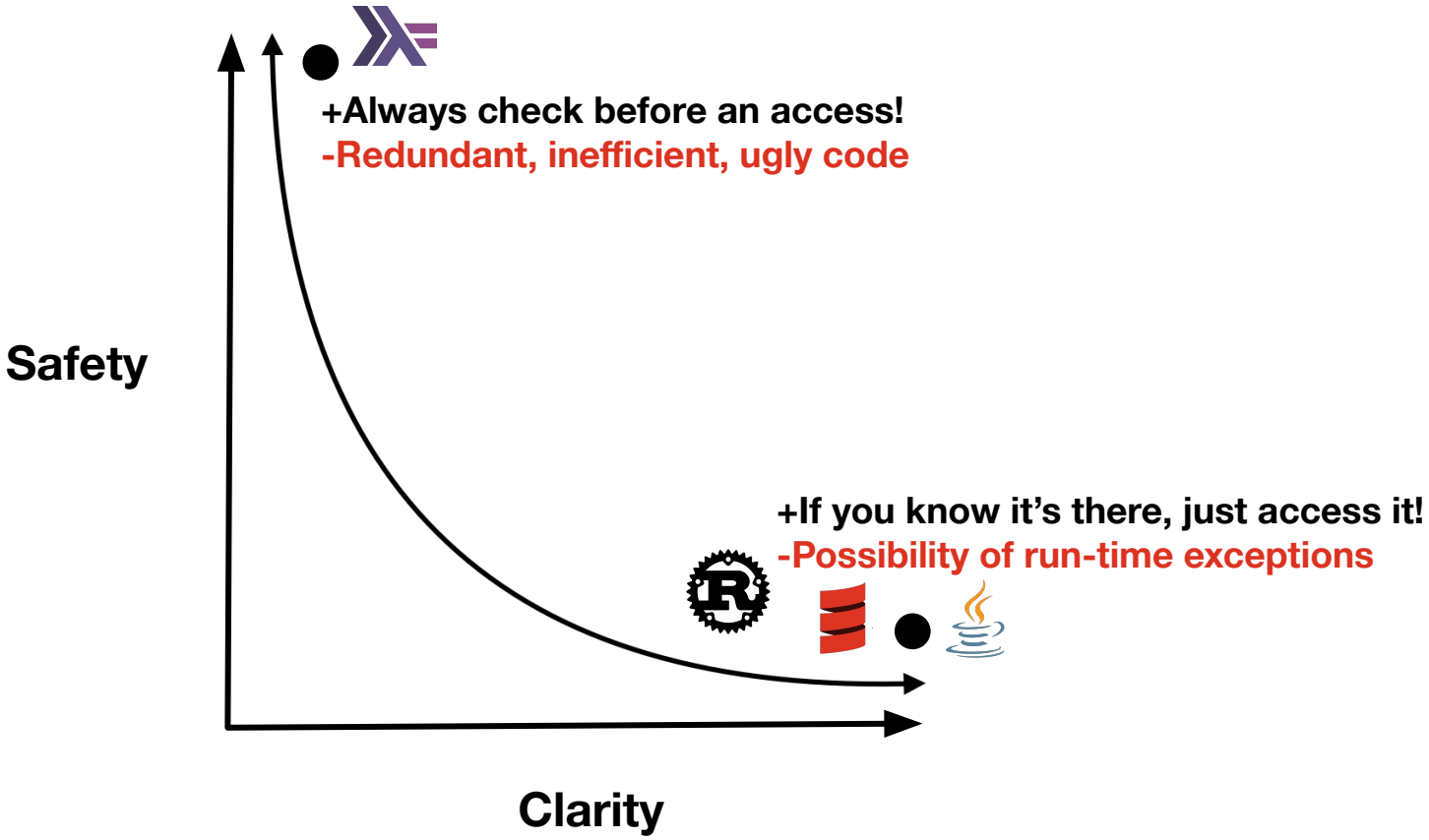
B4X · 6mo ago

Correct way of exception handling, an optional?

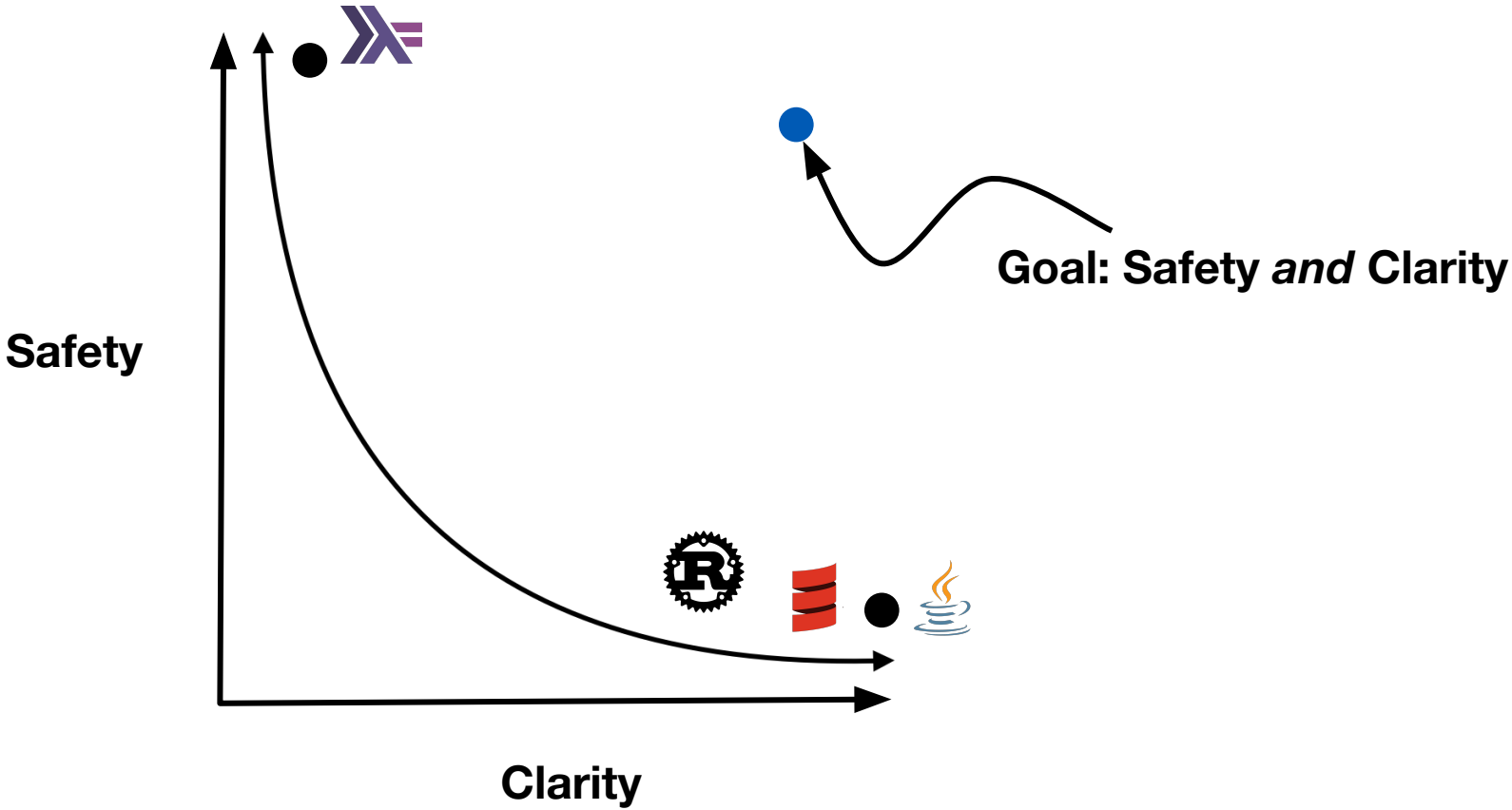
Reddit · r/javahelp · 10+ comments · 1y ago

```
Optional<String> format(Optional<Transaction> t) {
    if (t.isPresent()) {
        String id = t.get().id;
        String time = t.get().timestamp;
        return Optional.of(
            "Transaction: " + id + "at: " + time);
    }
    return Optional.empty();
}
```

The Problem with the Option type: Safety vs. Clarity



The Problem with the Option type: Safety vs. Clarity



Key Insights

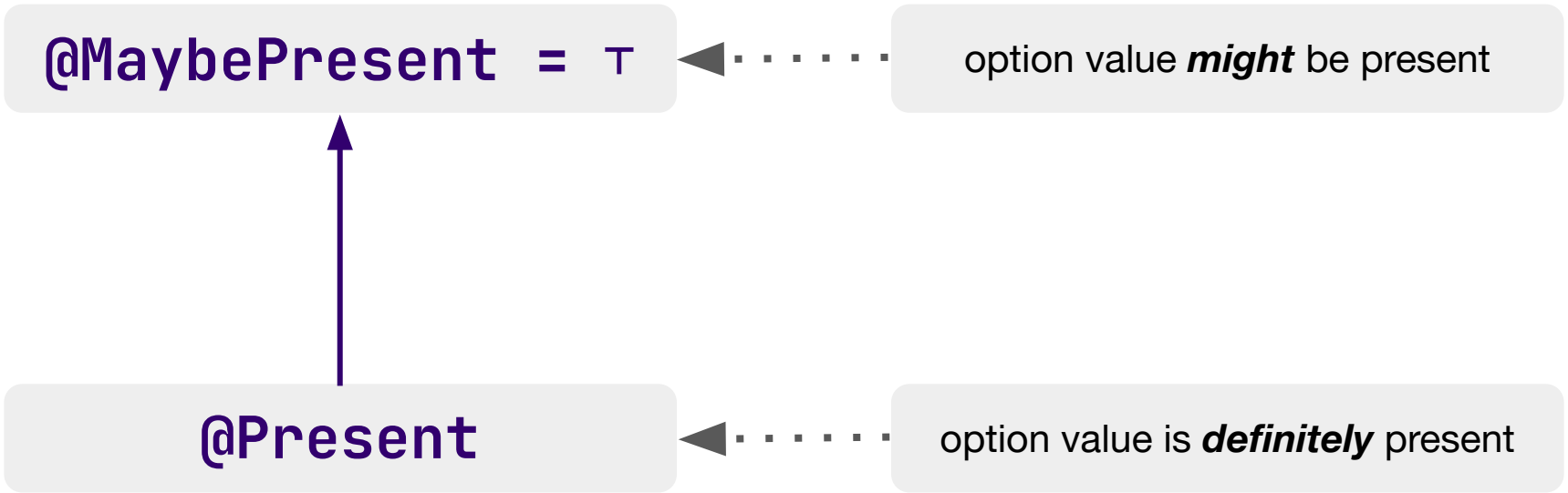
- **The option type**
  - Either unsafe or verbose
- **Optional Checker: Verifier for the option type**

The Optional Checker

Open-source static analysis tool for `Optional`

- **Safety** - eliminates `NoSuchElementException`
  - The first for option types based on a sound theory
  - **via *partial* rely-guarantee reasoning** in type systems
- **Clarity -** enforcement of best practices for option types

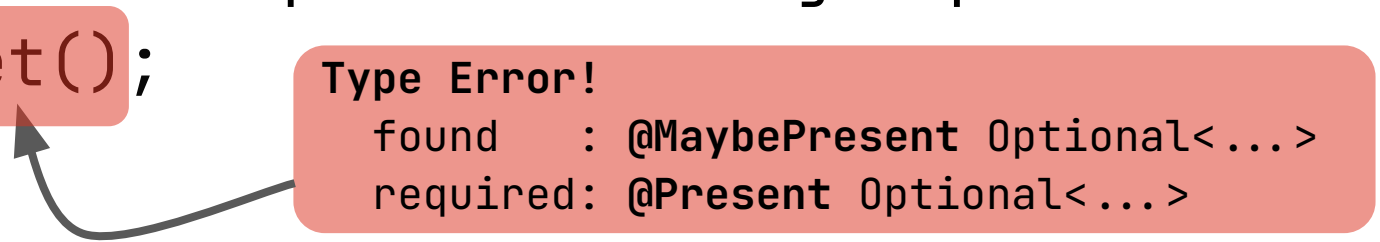
A Type System for Option Values



Preventing NoSuchElementException

- get(@Present Optional<T> this)
- orElseThrow(@Present Optional<T> this)
- 8 annotations in the JDK

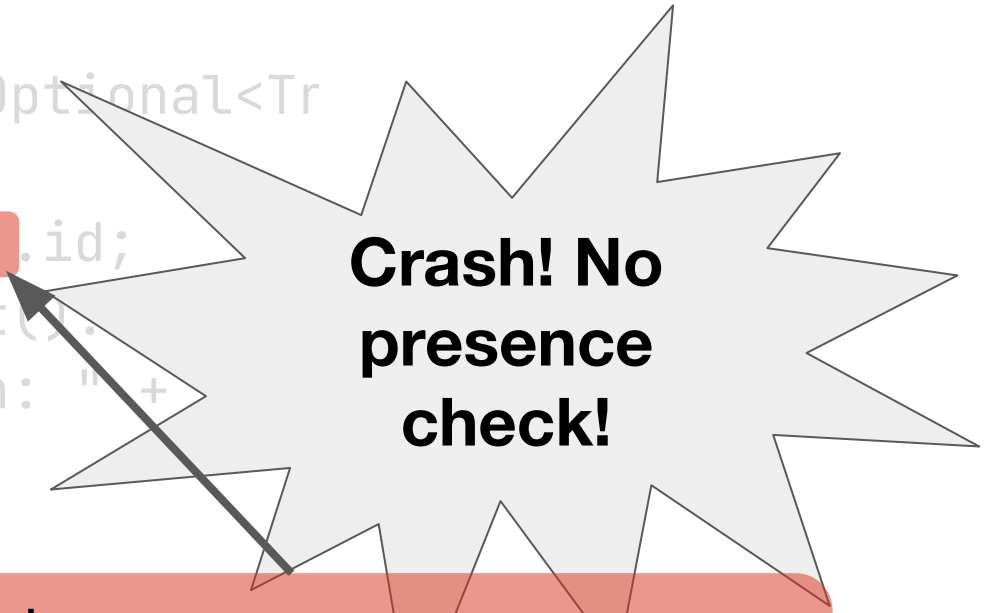
```
@MaybePresent Optional<String> optId = …
optId.get();
```



```
String  format(Optional<Tr

    String id = t.get().id;
    String time = t.get().
    return "Transaction: " +
}

format(ts);
```




```
Type Error!
  found    : @MaybePresent Optional<...>
  required: @Present Optional<...>
```

```
Exception in thread  “main” java.util.NoSuchElementException : No value present
— at java.base/java.util.Optional.get(Optional.java:148)
```

```
if (folder.isPresent()) {
    . . .
    copy(folder);
    . . .
}

copy(@Present Optional<File> folder) {

        Utils.copyDir(folder.get());

}
```

```
// userIds is non-empty
List<Integer> userIds = ...


Optional<Integer> maxUserId =
    userIds.stream()
    .max(Integer::compareTo);
```

```
Type Error!
  found    : @MaybePresent Optional<...>
  required: @Present Optional<...>
```

```
maxUserId.get();
```

userIds	@NonEmpty List<Integer>

```
// userIds is non-empty
List<Integer> userIds = ...
```

userIds	@NonEmpty List<Integer>
maxUserId	@Present Optional

```
Optional<Integer> maxUserId =
    userIds.stream()
    .max(Integer::compareTo);
```

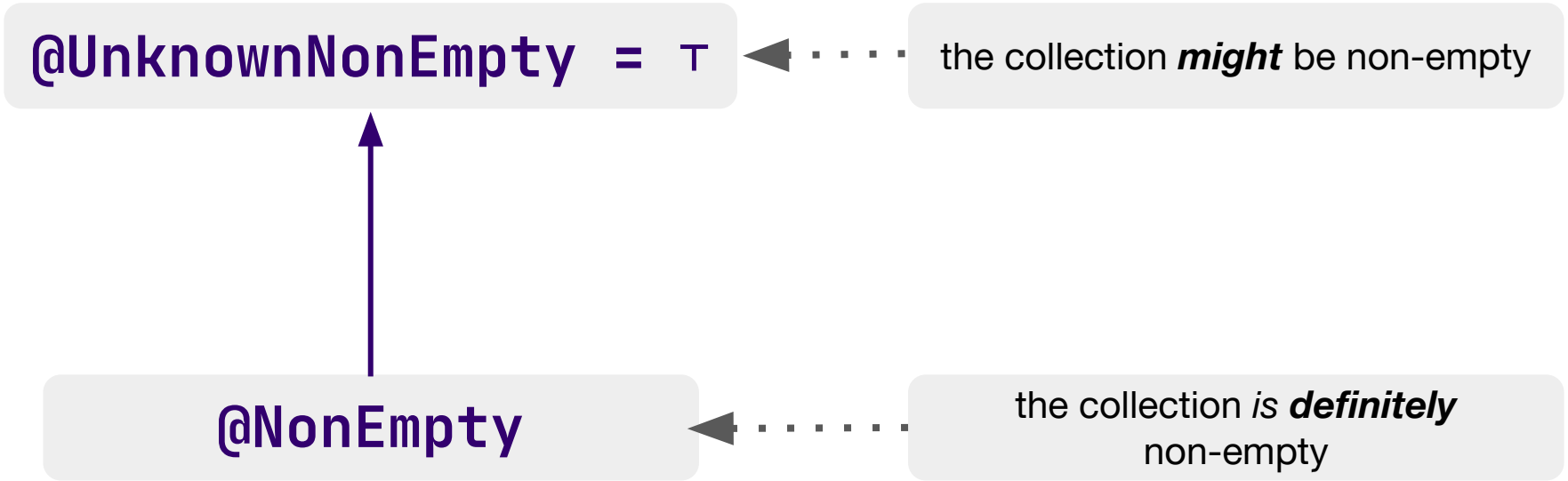
Type Error!

found : @MaybePresent Optional<...>
required: @Present Optional<...>

***Partial* rely-guarantee**

```
maxUserId.get();
```

A Type System for Non-Empty Collections



A Type System for Non-Empty Collections

Encodes rules like

- A non-empty `List` yields a non-empty
  - `Stream`
  - `Iterator`
- Map over a list does not affect emptiness
- Non-exceptional `List.add()` yields a non-empty list

```
// userIds is non-empty
List<Integer> userIds = ...


Optional<Integer> maxUserId =
    userIds.stream()
    .max(Integer::compareTo);
```

```
Type Error!
  found    : @MaybePresent Optional<...>
  required: @Present Optional<...>
```

```
maxUserId.get();
```

Solution: Verify *All* Option Values *and* Collections

Sound guarantees for option values and collections!

- Poor programmer experience
- Increased false positive rate
- Irrelevant errors and warnings for collections

🔑 Key Idea: *Partial* Rely–Guarantee Reasoning

Soundly verify only what's **relevant** (*partial verification*)

- Verify ***all*** **option values** and ***some*** **collections**
- More efficient than demand-driven approaches

Relevant program constructs

- Terminal operations resulting in `Optional<T>`
- Programmer-written `@NonEmpty` annotations
- And their immediate dependents

Key Insights

The option type

- Either unsafe or verbose

Optional Checker: Verifier for the option type

- *Partial* rely-guarantee reasoning
- Ensures safety and clarity

Large-scale evaluation

Evaluation

Dataset

- 1M SLOC of open-source Java programs

Tools

- Optional Checker
- IntelliJ IDEA IDE
- SpotBugs
- Error Prone

Evaluation - Results

Optional Checker

- 🏆 **13** real-world defects
- 🏆 **93% precision** and **100% recall**
- **6** programmer-written annotations in **1M SLOC**

Prior Tools

- 69% precision and 85% recall (IntelliJ)
- 🏆 **0** programmer-written annotations
- SpotBugs and Error Prone missed all defects

Real-World Defect Only Detected by Optional Checker

```
private Optional<String> prefix = ...

Function<..., ...> build() {
   ...
   if (prefix.isPresent()) {
     return m -> prefix.get();
   }
   ...
}
```

Real-World Defect Only Detected by Optional Checker

```
private Optional<String> prefix = ...

Function<..., ...> build() {
  ...
  if (prefix.isPresent()) {
    return m -> prefix.get();
  }
  ...
}
```

```
1: fn = builder.build();
2: builder
     .setPrefix(
       Optional.empty());
3: fn.apply(...);
```

```
Type Error!..>
  found    : @MaybePresent Optional<.
  required: @Present Optional<...>
```

Key Insights

Code

Paper

Γ The option type

- Either unsafe or verbose

✅ Optional Checker: Verifier for the option type

- *Partial* rely-guarantee reasoning
- Ensures safety and clarity

📋 Large-scale evaluation

- Effective: **93%** precision and **100%** recall, **13 real-world defects**
- Low usage burden: **6 annotations** per **1M SLOC**
- Outperforms all previous tools