

# Implement Your Own Type System in 45 Minutes

## The Checker Framework



<http://CheckerFramework.org/>

Twitter: @CheckerFrmwrk

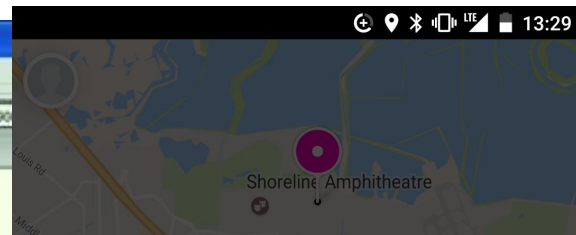
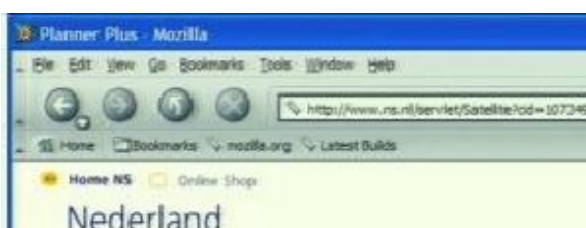
Live demo: <http://CheckerFramework.org/live/>

Werner Dietl, University of Waterloo

Michael Ernst, University of Washington



# Motivation



TREND MICRO InterScan™ Web Security Virtual Appliance

Log Off | Help

Search

- System Status
- Dashboard
- + Application Control
- HTTP
  - + HTTPS Decryption
  - + Advanced Threat Protection
  - + HTTP Inspection
  - + Data Loss Prevention
  - + Applets and ActiveX
  - URL Filtering
- Policies
  - Settings
- Reports
- + Updates
- Notifications
- + Administration

## HTTP Status 500 - java.lang.NullPointerException

**type** Exception report

**message** [java.lang.NullPointerException](#)

**description** The server encountered an internal error that prevented it from fulfilling this request.

**exception**

```
org.apache.jasper.JasperException: java.lang.NullPointerException
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:432)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:313)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:260)
javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
    ter.java:73)
    77)
java.lang.NullPointerException
org.apache.jsp.urlf_005fsection_005fpolicy_005frule_jsp._jspService(urlf_005fsection_005fpolicy_005frule_jsp.java:742)
org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:388)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:313)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:260)
javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
com.trend.iwss.servlets.filters.CSRFGuardFilter.doFilter(CSRFGuardFilter.java:73)
com.trend.iwss.servlets.filters.AuthFilter.doFilter(AuthFilter.java:377)
```

# java.lang.NullPointerException

# Java's type system is too weak

Type checking prevents many errors

```
int i = "hello";
```

Type checking doesn't prevent **enough** errors

```
System.console().readLine();
```



# Java's type system is too weak

Type checking prevents many errors

```
int i = "hello";
```

Type checking doesn't prevent enough errors

NullPointerException

```
System.console().readLine();
```



# Prevent null pointer exceptions

Java 8 introduces the `Optional<T>` type

- Wrapper; content may be *present* or *absent*
- Constructor: `of(T value)`
- Methods: `boolean isPresent()`, `T get()`

```
Optional<String> maidenName;
```



# Optional reminds you to check

## Without Optional:

```
String mName;  
mName.equals(...);  
  
if (mName != null) {  
    mName.equals(...);  
}
```

possible  
NullPointerException

## With Optional:

```
Optional<String> omName;  
omName.get().equals(...);  
  
if (omName.isPresent()) {  
    omName.get().equals(...);  
}
```

possible  
NoSuchElementException

possible  
NullPointerException

**Complex rules for using Optional correctly!**



# How not to use Optional

Stuart Marks's rules:

1. Never, ever, use null for an Optional variable or return value.
2. Never use Optional.get() unless you can prove that the Optional is present.
3. Prefer alternative APIs over Optional.isPresent() and Optional.get().
4. It's generally **Let's enforce the rules with a tool.** Optional for the specific purpose of chaining methods.
5. If an Optional is part of a chain, or has an intermediate result of Optional, use Optional.orElse() or Optional.orElseGet().
6. Avoid using Optional in fields, method parameters, and collections.
7. Don't use an Optional to wrap any collection type (List, Set, Map). Instead, use an empty collection to represent the absence of values.

Other gu  
Stephen  
Dalorzo,  
Brian Go  
Olszewski  
Oleg She



# Which rules to enforce with a tool

Stuart Marks's rules:

1. **Never**, ever, use null for an Optional variable or return value.
2. **Never** use Optional.get() unless you can prove that the Optional is present.
3. *Prefer* alternative APIs over Optional.isPresent() and Optional.get().
4. It's *generally a bad idea* to create an Optional for the specific purpose of chaining methods from it to get a value.
5. If an Optional chain has a nested Optional chain, or has an intermediate result of Optional, it's *probably too complex*.
6. *Avoid* using Optional in fields, method parameters, and collections.
7. **Don't** use an Optional to wrap any collection type (List, Set, Map). Instead, use an empty collection to represent the absence of values.





# Which rules to enforce with a tool

Stuart Marks's rules:

1. **Never**, ever, use null for an Optional variable or return value.
2. **Never** use Optional.get() unless you can prove that the Optional is present.
3. *Prefer* alternative APIs over Optional.isPresent() and Optional.get().
4. It's *generally* *preferred* for the specific purpose of chaining methods.
5. If an Optional is used as an intermediate result of Optional chaining, it's *preferred*.
6. *Avoid* using Optional in fields, method parameters, and collections.
7. **Don't** use an Optional to wrap any collection type (List, Set, Map). Instead, use an empty collection to represent the absence of values.

These are  
*type system* properties.



# Define a type system

	$h \in \text{Heap}$	$= \text{Addr} \rightarrow \text{Obj}$
	$\iota \in \text{Addr}$	$= \text{Set of Addresses} \cup \{\text{null}_a\}$
	$o \in \text{Obj}$	$= \text{rType, Fields}$
	$\text{*T} \in \text{rType}$	$= \text{OwnerAddr ClassId} \langle \overline{\text{rType}} \rangle$
$P \in \text{Program}$	$::= \overline{\text{Class, ClassId, Expr}}$	
$\text{Cls} \in \text{Class}$	$::= \text{class ClassId} \langle \text{TVarId} \langle \text{Type} \rangle \rangle$ $\text{extends ClassId} \langle \text{*Type} \rangle$ $\{ \overline{\text{FieldId} \text{*Type}; \text{Met}} \}$	
$\text{*T} \in \text{*Type}$	$::= \text{*NType} \mid \text{TVarId}$	
$\text{*N} \in \text{*NType}$	$::= \text{OM ClassId} \langle \text{*Type} \rangle$	
$u \in \text{OM}$	$::=$	$h, \text{*}\Gamma, e_0 \rightsquigarrow h_0, \iota_0$
$\text{mt} \in \text{Meth}$	$::=$	$\iota_0 \neq \text{null}_a$
$\text{MethSig}$	$::=$	$h_0, \text{*}\Gamma, e_2 \rightsquigarrow h_2, \iota$
$w \in \text{Purity}$	$::=$	$h' = h_2[\iota_0.f := \iota]$
$e \in \text{Expr}$	$::=$	$\text{OS-Upd} \frac{h, \text{*}\Gamma, e_0.f = e_2 \rightsquigarrow h'}{h, \text{*}\Gamma, e_0.f = e_2 \rightsquigarrow h'}$
$\text{*}\Gamma \in \text{*Env}$	$::=$	$\text{Expr.MethId} \langle \text{*Type} \rangle (\text{Expr}) \mid$ $\text{new *Type} \mid (\text{*Type}) \text{Expr}$ $\text{TVarId *NType; ParId *Type}$
		$h, \text{*}\Gamma, e_0 \rightsquigarrow h', \iota_0$ $\iota_0 \neq \text{null}_a$ $\iota = h'(\iota_0) \downarrow_2 (f)$ $\text{OS-Read} \frac{\iota = h'(\iota_0) \downarrow_2 (f)}{h, \text{*}\Gamma, e_0.f \rightsquigarrow h', \iota}$
		$\Gamma \vdash e_0 : N_0 \quad N_0 = u_0 C_0 \langle \_ \rangle$ $T_1 = fType(C_0, f)$ $\Gamma \vdash e_2 : N_0 \triangleright T_1$ $\text{GT-Read} \frac{\Gamma \vdash e_0 : N_0 \quad N_0 = \_ \quad \text{GT-Upd} \frac{u_0 \neq \text{any} \quad rp(u_0, T_1)}{\Gamma \vdash e_0.f = e_2 : N_0 \triangleright T_1}}{\Gamma \vdash e_0.f : N_0 \triangleright fType(C_0, f)}$
$h \vdash \text{*}\Gamma : \text{*}\Gamma$		
$h \vdash \iota_1 : \text{dyn}(\text{*N}, h, \iota_1)$		
$h \vdash \iota_2 : \text{dyn}(\text{*T}, \iota_1, h(\iota_1) \downarrow_1)$		
$\text{*N} = u_N C_N \langle \_ \rangle$		
$u_N = \text{this}_u \Rightarrow \text{*}\Gamma(\text{this})$		
$\text{free}(\text{*T}) \subseteq \text{dom}(C_N)$		
	$\text{DYN} \frac{\left. \begin{array}{l} \implies h \vdash \iota_2 : \text{dyn}(\text{*N} \triangleright \text{*T}, h, \text{*}\Gamma) \\ \text{*T} = \iota' \_ \langle \_ \rangle \quad \iota \vdash \text{*T} \text{*} \langle \iota' C \langle \text{*T} \rangle \rangle \quad \iota \vdash \text{*T} \text{*} \langle \iota' C \langle \text{*T}_a \rangle \rangle \Rightarrow \iota \vdash \text{*T} \text{*} \langle \text{*T}_a \rangle \\ \text{dom}(C) = \bar{X} \quad \text{free}(\text{*T}) \subseteq \bar{X} \circ \bar{X}' \end{array} \right\}}{\text{dyn}(\text{*T}, \iota, \text{*T}, (\bar{X}' \text{*T}'; -)) = \text{*T}[\iota'/\text{this}, \iota'/\text{peer}, \iota'/\text{rep}, \text{any}_a/\text{any}_u, \text{*T}/\bar{X}, \text{*T}'/\bar{X}']}$	



# Define a type system

1. **Type hierarchy** (subtyping)
2. **Type rules** (what operations are illegal)
3. **Type introduction** (what types for literals, ...)
4. **Dataflow** (run-time tests)

We will define two type systems:

**Nullness** and **Optional**

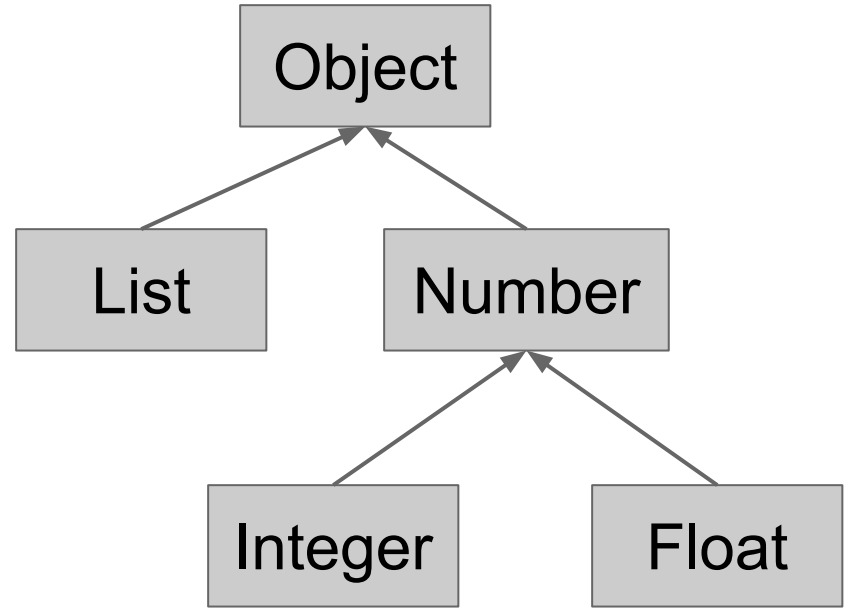
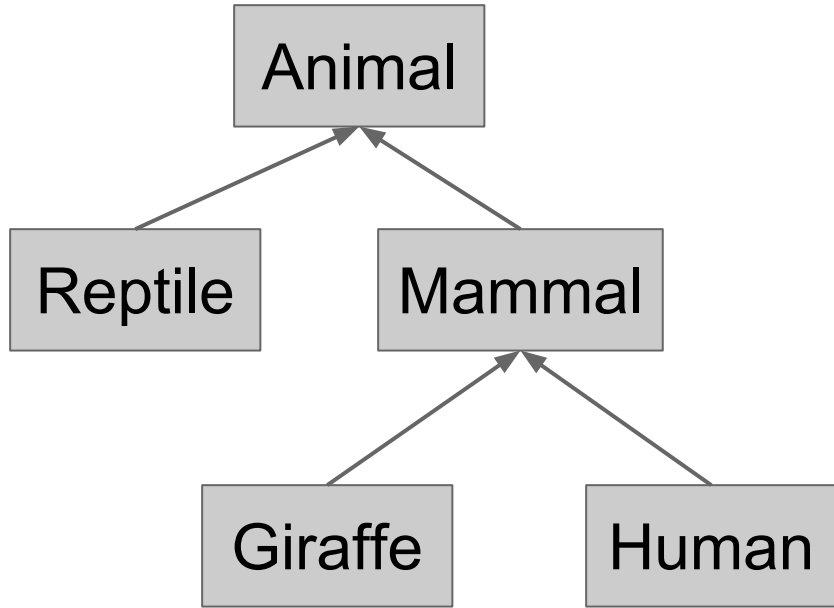


# Define a type system

- 1. Type hierarchy (subtyping)**
2. Type rules (what operations are illegal)
3. Type introduction (what types for literals, ...)
4. Dataflow (run-time tests)



# 1. Type hierarchy

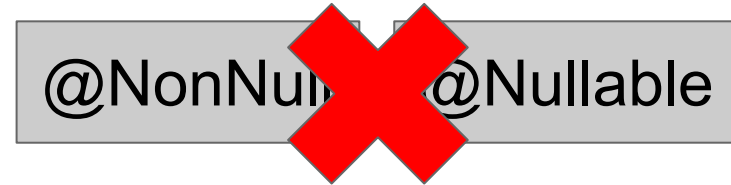
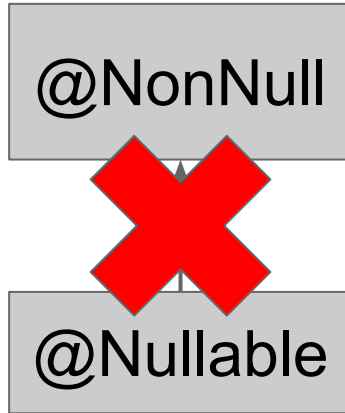
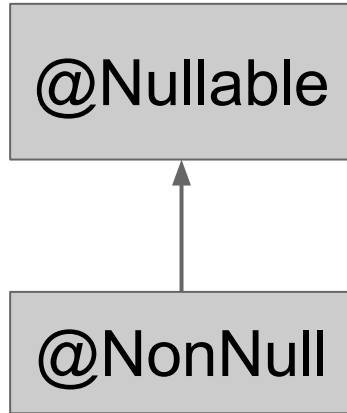


2 pieces of information:

- the types
- their relationships (lower = fewer values, more properties)



# Type hierarchy for nullness

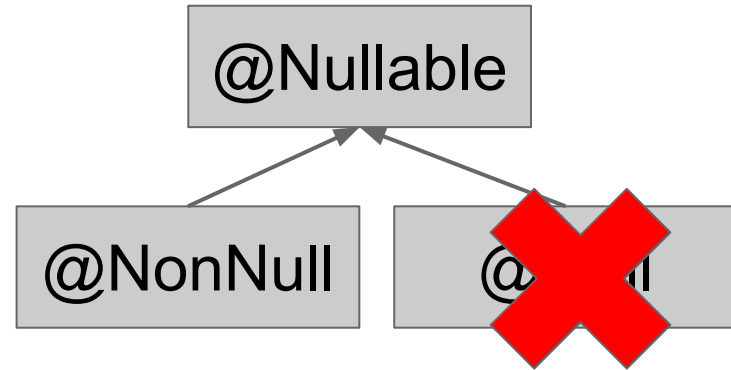
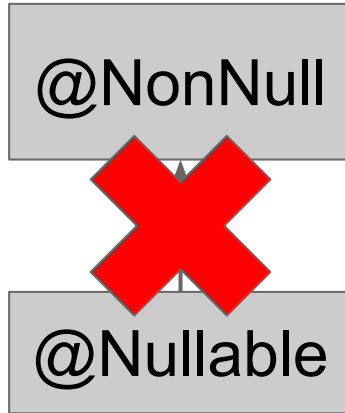
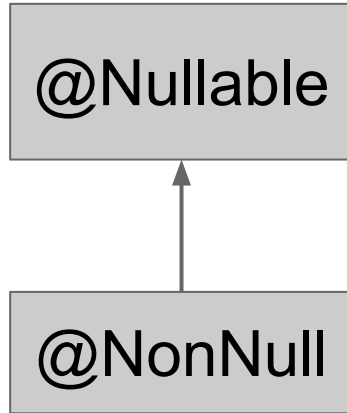


2 pieces of information:

- the types
- their relationships



# Type hierarchy for nullness



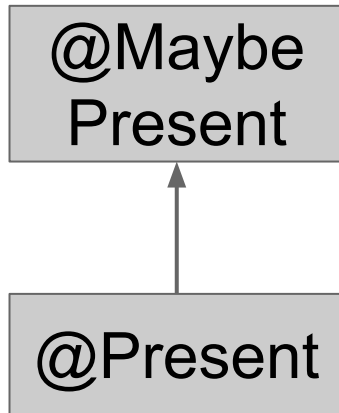
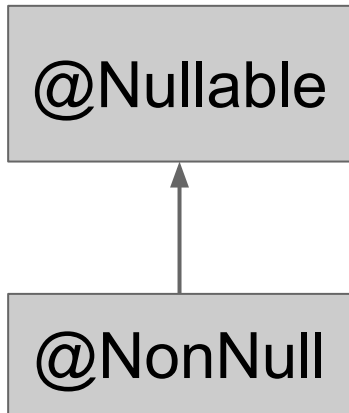
2 pieces of information:

- the types
- their relationships



# Type hierarchy for Optional

“Never use `Optional.get()` unless you can prove that the `Optional` is present.”



2 pieces of information:

- the types
- their relationships





# Type = type qualifier + Java basetype

`@Present Optional<String> maidenName;`

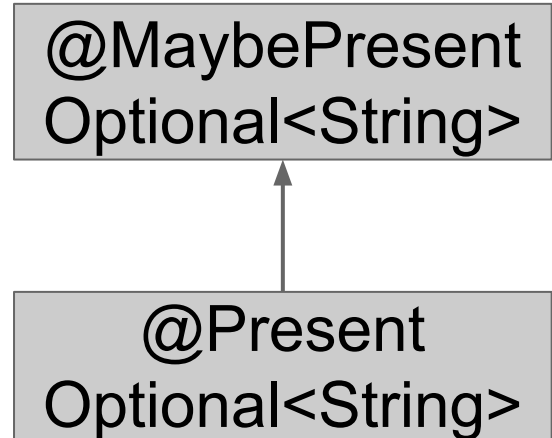
Type qualifier      Java basetype

Type

Default qualifier = @MaybePresent

- @MaybePresent Optional<String>
- Optional<String>

} equivalent



# Type = type qualifier + Java basetype

`@Present Optional<String> maidenName;`

Type qualifier      Java basetype

Type

Default qualifier = @MaybePresent

- @MaybePresent Optional<String>
- Optional<String>

} equivalent

@MaybePresent  
Optional<String>

@Present  
Optional<String>



# Define a type system

1. Type hierarchy (subtyping)
- 2. Type rules (what operations are illegal)**
3. Type introduction (what types for literals, ...)
4. Dataflow (run-time tests)



## 2. Type rules

To prevent null pointer exceptions:

- `expr.field`  
`expr.getValue()`  
receiver must be non-null
- `synchronized (expr) { ... }`  
monitor must be non-null
- ...



# Type rules for Optional

@MaybePresent

@Present



“Never use `Optional.get()` unless you can prove that the `Optional` is present.”

Only call `Optional.get()` on a receiver of type `@Present Optional`.

example call:

```
myOptional.get()
```

```
class Optional<T> {  
    T get() { ... }  
}
```

example call:

```
a.equals(b)
```



# Type rules for Optional

@MaybePresent

@Present



“Never use `Optional.get()` unless you can prove that the `Optional` is present.”

Only call `Optional.get()` on a receiver of type `@Present Optional`.

example call:

```
myOptional.get()
```

```
class Optional<T> {  
    T get(Optional<T> this) { ... }  
}
```



# Type rules for Optional

@MaybePresent

@Present



“Never use `Optional.get()` unless you can prove that the `Optional` is present.”

Only call `Optional.get()` on a receiver of type `@Present Optional`.

example call:

```
myOptional.get()
```

```
class Optional<T> {  
    T get(@Present Optional<T> this) {...}  
}
```



# Type rules for Optional

@MaybePresent

@Present



“Never use `Optional.get()` unless you can prove that the `Optional` is present.”

Only call `Optional.get()` on a receiver of type `@Present Optional`.

example call:

```
myOptional.get()
```

```
class Optional<T> {  
    T get(@Present Optional<T> this) {...}  
    T orElseThrow(@Present O... this, ...) {...}  
}
```





# Define a type system

1. Type hierarchy (subtyping)
2. Type rules (what operations are illegal)
- 3. Type introduction (what types for literals...)**
4. Dataflow (run-time tests)



# Type introduction rules

For Nullness type system:

- `null` : `@Nullable`
- `"Hello World"` : `@NonNull`



# Type introduction for Optional

@MaybePresent

@Present



“Never use Optional.get() unless you can prove that the Optional is present.”

```
Optional<T> of(T value) {...}
```

```
Optional<T> ofNullable(T value){...}
```



# Type introduction for Optional

@MaybePresent

@Present



“Never use Optional.get() unless you can prove that the Optional is present.”

```
@Present Optional<T> of(T value) {...}
```

```
Optional<T> ofNullable(@Nullable T value){...}
```



# Define a type system

1. Type hierarchy (subtyping)
2. Type rules (what operations are illegal)
3. Type introduction (what types for literals, ...)
- 4. Dataflow (run-time tests)**



# Flow-sensitive type refinement

After an operation, give an expression a more specific type

```
@Nullable Object x;
```

```
if (x != null) {
```

```
... x is @NonNull here
```

```
}
```

```
... x is @Nullable again
```

```
@Nullable Object y;
```

```
y = new SomeType();
```

```
... y is @NonNull here
```

```
y = unknownValue;
```

```
... y is @Nullable again
```

# Type refinement for Optional

@MaybePresent

@Present



“Never use `Optional.get()` unless you can prove that the `Optional` is present.”

After `receiver.isPresent()` returns true,  
the receiver's type is `@Present`

```
@MaybePresent Optional<String> x;
```

```
if (x.isPresent()) {
```

```
...
```

x is @Present here

```
}
```

```
...
```

x is @MaybePresent again



# Now, let's implement it

Follow the instructions in the  
Checker Framework Manual

<https://checkerframework.org/manual/#creating-a-checker>





# You can use the Optional Checker

Distributed with the Checker Framework

Checks 6 of the 7 rules for using Optional



# More at CodeOne 2018

Preventing Errors Before They Happen:

The Checker Framework

TUT4991, Tue Oct 23, 8:45 – 10:45

Moscone West - Room 2009

Using Type Annotations to Improve Your Code

BOF4992, Tue Oct 23, 19:30 – 20:15

Moscone West - Room 2009



# Pluggable type-checking improves code

Checker Framework for creating type checkers

- Featureful, effective, easy to use, scalable

Prevent bugs at compile time

Create custom type-checkers

Improve your code!

<http://CheckerFramework.org/>

