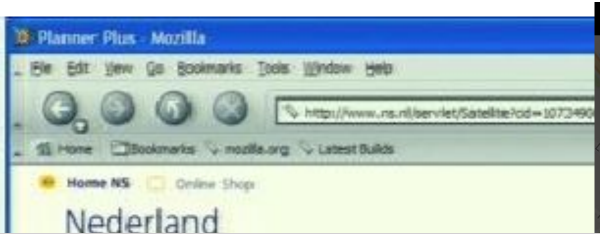
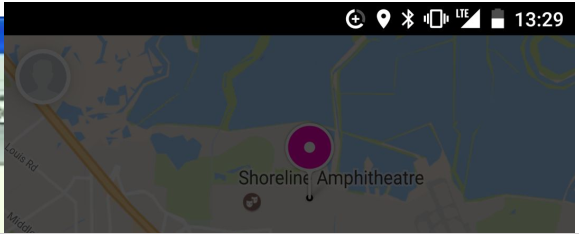
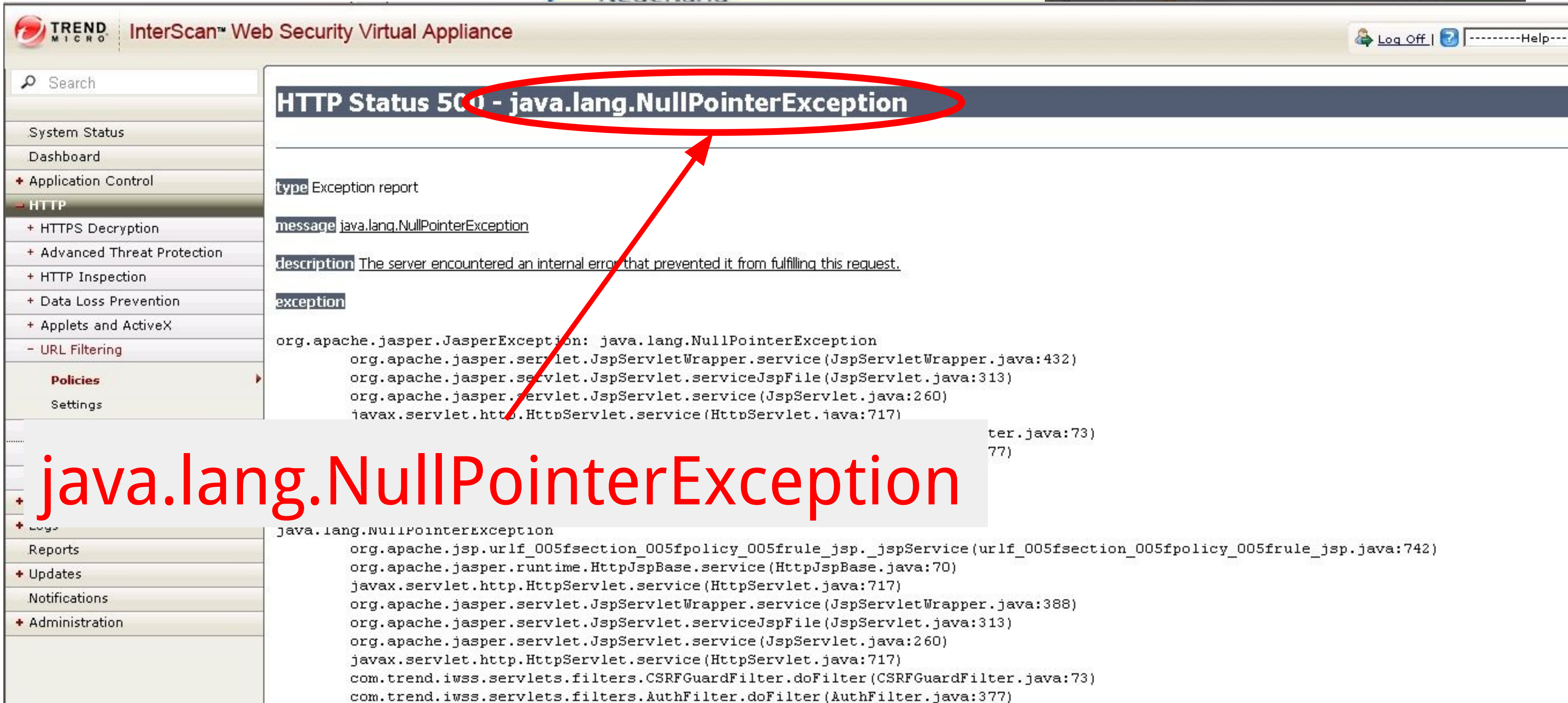
How to implement a type system

CSE 331
University of Washington
Michael Ernst

Motivation


Planner Plus - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://www.ns.nl/servlet/Satellite?cid=1072490
Home Bookmarks mozilla.org Latest Builds
Home NS Online Shop
Nederland





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:

possible
NullPointerException

```
String mNa  e;
mName.equals(...);

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

With Optional:

possible
NoSuchElementException

```
Optional<St  g> omName;
omName.get().equals(...);

if (omName. sPresent()) {
   omName.ge   equals(...);
}
```

possible
NullPointerException

Complex rules for using Optional correctly!

How not to use Optional

Other gu
Stephen
Dalorzo,
Brian Go
Olszewsk
Oleg She

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 nal for the specific purpose of chaining met
5. If an Optiona chain, or has an intermediate result of Opt ex.
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.

Let's enforce the rules with a tool.

✓

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.

✓

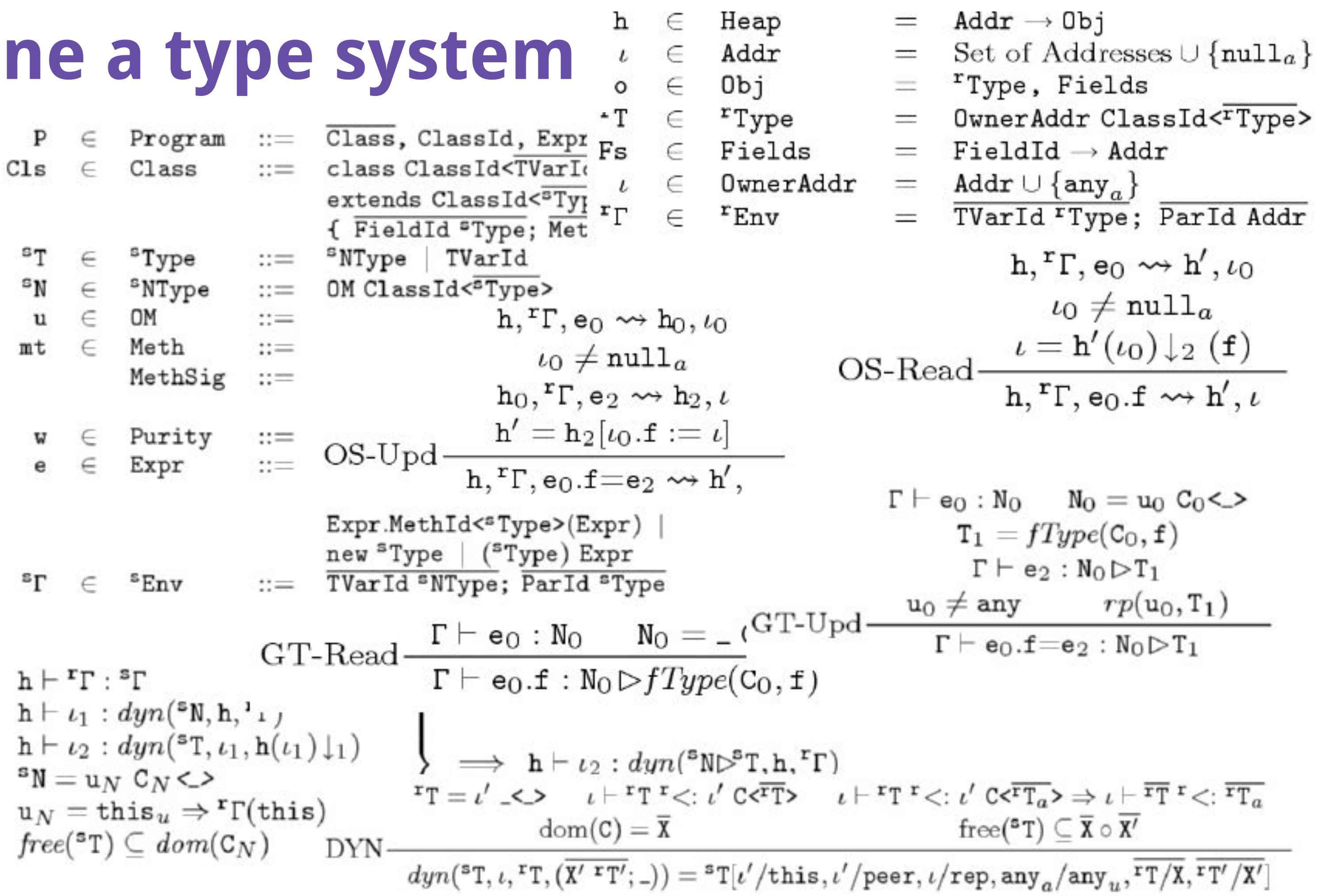
Which rules to enforce with a tool

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* … specific purpose of chaining met…
5. If an Optiona… as an intermediate result of Opt…
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



$\mathtt{h}$	$\in$	Heap	$=$	Addr $\to$ Obj
ι	$\in$	Addr	$=$	Set of Addresses $\cup \{\mathtt{null}_a\}$
$\mathtt{o}$	$\in$	Obj	$=$	^{r}Type, Fields
${}^{r}\mathtt{T}$	$\in$	^{r}Type	$=$	OwnerAddr ClassId<$\overline{{}^{r}\text{Type}}$>
Fs	$\in$	Fields	$=$	FieldId $\to$ Addr
ι	$\in$	OwnerAddr	$=$	Addr $\cup \{\mathtt{any}_a\}$
${}^{r}\Gamma$	$\in$	^{r}Env	$=$	$\overline{\text{TVarId } {}^{r}\text{Type}}$; $\overline{\text{ParId Addr}}$

P	$\in$	Program	::=	$\overline{\text{Class}}$, ClassId, Expr
Cls	$\in$	Class	::=	class ClassId<TVarI
				extends ClassId<^{s}Ty
				{ $\overline{\text{FieldId } {}^{s}\text{Type};}$ Met
^{s}T	$\in$	^{s}Type	::=	^{s}NType | TVarId
^{s}N	$\in$	^{s}NType	::=	OM ClassId<^{s}Type>
u	$\in$	OM	::=	
mt	$\in$	Meth	::=	
		MethSig	::=	
w	$\in$	Purity	::=	
e	$\in$	Expr	::=	
				Expr.MethId<^{s}Type>(Expr) |
				new ^{s}Type | (^{s}Type) Expr
${}^{s}\Gamma$	$\in$	^{s}Env	::=	$\overline{\text{TVarId } {}^{s}\text{NType}}$; $\overline{\text{ParId } {}^{s}\text{Type}}$

$$\text{OS-Upd}\;\frac{\begin{array}{c}\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}_0, \iota_0 \\ \iota_0 \neq \mathtt{null}_a \\ \mathtt{h}_0, {}^{r}\Gamma, \mathtt{e}_2 \rightsquigarrow \mathtt{h}_2, \iota \\ \mathtt{h}' = \mathtt{h}_2[\iota_0.\mathtt{f} := \iota]\end{array}}{\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 \rightsquigarrow \mathtt{h}',}$$

$$\text{OS-Read}\;\frac{\begin{array}{c}\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}', \iota_0 \\ \iota_0 \neq \mathtt{null}_a \\ \iota = \mathtt{h}'(\iota_0)\downarrow_2(\mathtt{f})\end{array}}{\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0.\mathtt{f} \rightsquigarrow \mathtt{h}', \iota}$$

$$\text{GT-Upd}\;\frac{\begin{array}{c}\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \mathtt{u}_0\ \mathtt{C}_0\texttt{<\_>} \\ \mathtt{T}_1 = \mathit{fType}(\mathtt{C}_0, \mathtt{f}) \\ \Gamma \vdash \mathtt{e}_2 : \mathtt{N}_0 \rhd \mathtt{T}_1 \\ \mathtt{u}_0 \neq \mathtt{any} \qquad \mathit{rp}(\mathtt{u}_0, \mathtt{T}_1)\end{array}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 : \mathtt{N}_0 \rhd \mathtt{T}_1}$$

$$\text{GT-Read}\;\frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \_}{\Gamma \vdash \mathtt{e}_0.\mathtt{f} : \mathtt{N}_0 \rhd \mathit{fType}(\mathtt{C}_0, \mathtt{f})}$$

$$\left.\begin{array}{l}\mathtt{h} \vdash {}^{r}\Gamma : {}^{s}\Gamma \\ \mathtt{h} \vdash \iota_1 : \mathit{dyn}({}^{s}\mathtt{N}, \mathtt{h}, {}^{r} \\ \mathtt{h} \vdash \iota_2 : \mathit{dyn}({}^{s}\mathtt{T}, \iota_1, \mathtt{h}(\iota_1)\downarrow_1) \\ {}^{s}\mathtt{N} = \mathtt{u}_N\ \mathtt{C}_N\texttt{<>} \\ \mathtt{u}_N = \mathtt{this}_u \Rightarrow {}^{r}\Gamma(\mathtt{this}) \\ \mathit{free}({}^{s}\mathtt{T}) \subseteq \mathit{dom}(\mathtt{C}_N)\end{array}\right\} \Longrightarrow \mathtt{h} \vdash \iota_2 : \mathit{dyn}({}^{s}\mathtt{N} \rhd {}^{s}\mathtt{T}, \mathtt{h}, {}^{r}\Gamma)$$

$$\text{DYN}\;\frac{\begin{array}{c}{}^{r}\mathtt{T} = \iota'\ \texttt{\_<>} \qquad \iota \vdash {}^{r}\mathtt{T}\ {}^{r}\!\!<:\ \iota'\ \mathtt{C}\texttt{<}\overline{{}^{r}\mathtt{T}}\texttt{>} \qquad \iota \vdash {}^{r}\mathtt{T}\ {}^{r}\!\!<:\ \iota'\ \mathtt{C}\texttt{<}\overline{{}^{r}\mathtt{T}_a}\texttt{>} \Rightarrow \iota \vdash \overline{{}^{r}\mathtt{T}}\ {}^{r}\!\!<:\ \overline{{}^{r}\mathtt{T}_a} \\ \mathrm{dom}(\mathtt{C}) = \overline{\mathtt{X}} \qquad\qquad \mathrm{free}({}^{s}\mathtt{T}) \subseteq \overline{\mathtt{X}} \circ \overline{\mathtt{X}'}\end{array}}{\mathit{dyn}({}^{s}\mathtt{T}, \iota, {}^{r}\mathtt{T}, (\overline{\mathtt{X}'\ {}^{r}\mathtt{T}'}; \_)) = {}^{s}\mathtt{T}[\iota'/\mathtt{this}, \iota'/\mathtt{peer}, \iota/\mathtt{rep}, \mathtt{any}_a/\mathtt{any}_u, \overline{{}^{r}\mathtt{T}/\mathtt{X}}, \overline{{}^{r}\mathtt{T}'/\mathtt{X}'}]}$$

✓

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)

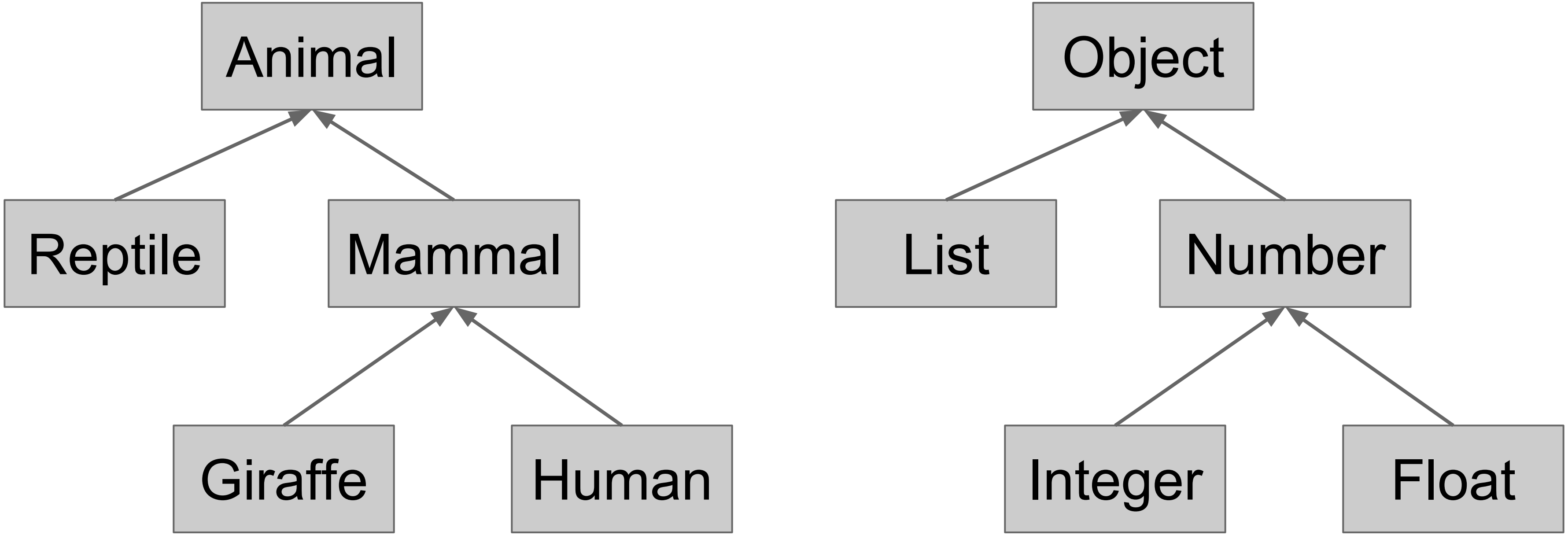
We will define two type systems:
Nullness and Optional

✓

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

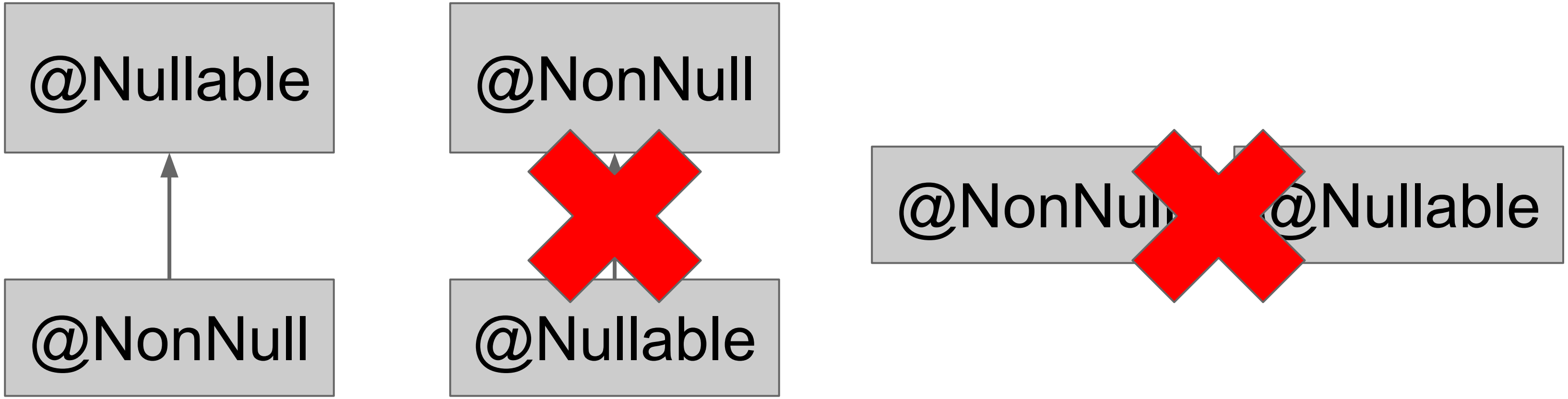


2 pieces of information:

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

✓

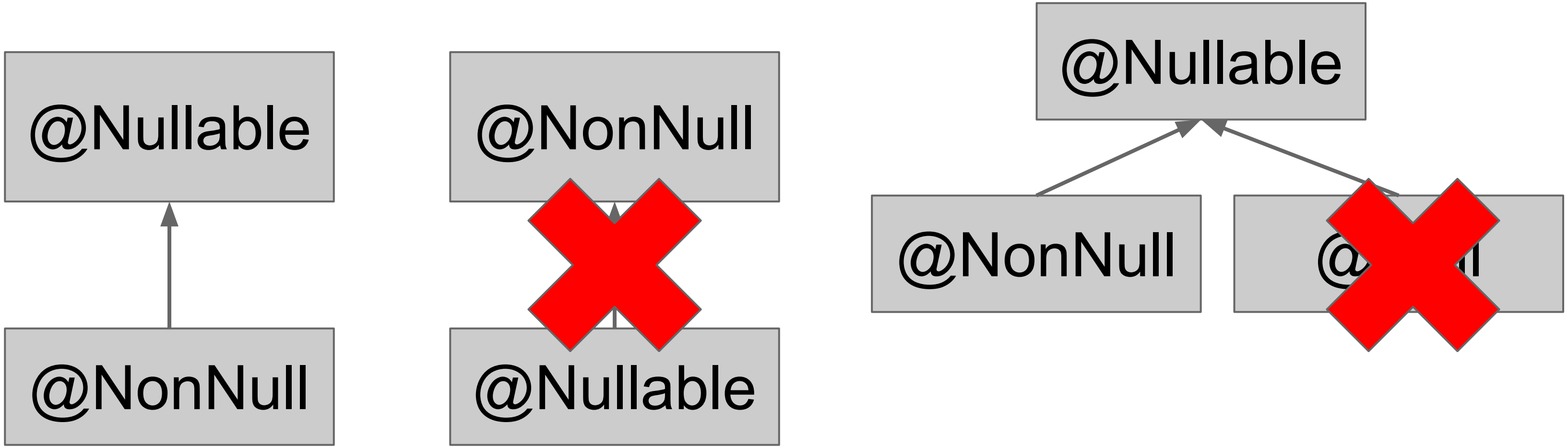
Type hierarchy for nullness



2 pieces of information:

- the types
- their relationships

Type hierarchy for nullness

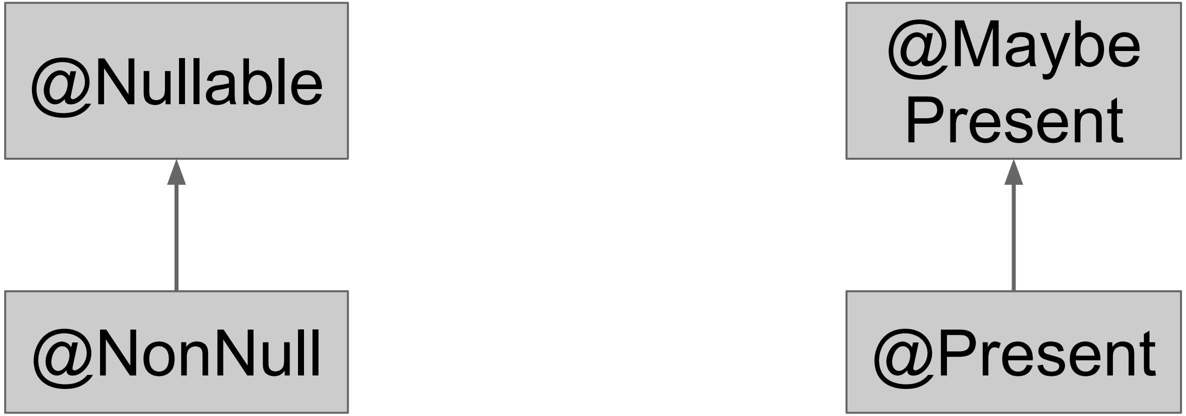


2 pieces of information:

- the types
- their relationships

Type hierarchy for Optional

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

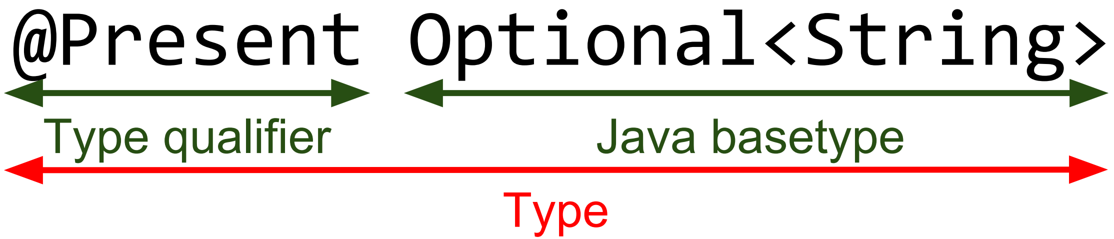


2 pieces of information:

- the types
- their relationships

✓

Type = type qualifier + Java basetype

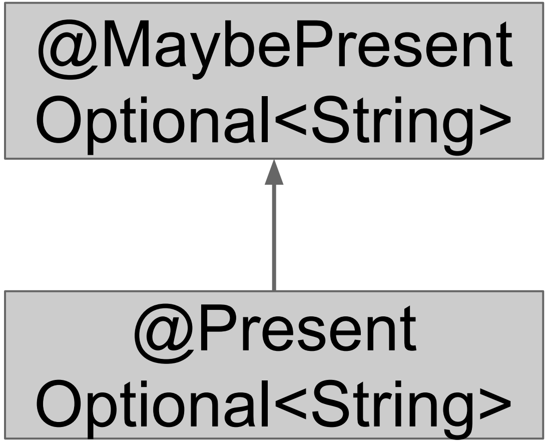


```
@Present Optional<String> maidenName;
```

Type qualifier

Java basetype

Type



@MaybePresent Optional<String>

@Present Optional<String>

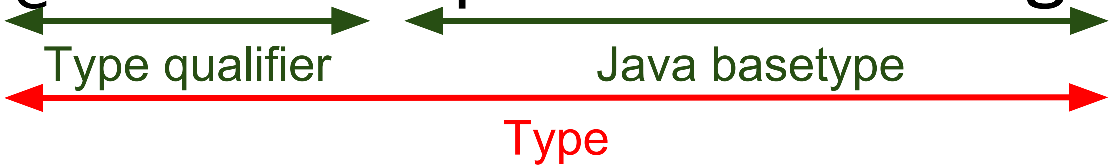
Default qualifier = @MaybePresent

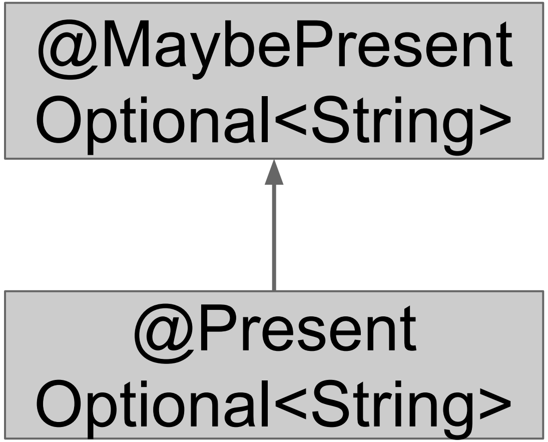
- `@MaybePresent Optional<String>`
- `Optional<String>`

equivalent

Type = type qualifier + Java basetype

```
@Present Optional<String> maidenName;
```





@MaybePresent Optional<String>

@Present Optional<String>

Default qualifier = @MaybePresent

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

equivalent

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)

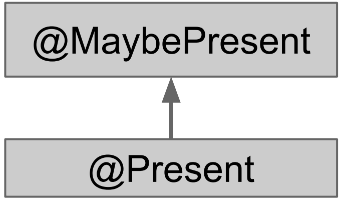
2. Type rules

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



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

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

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

example call:

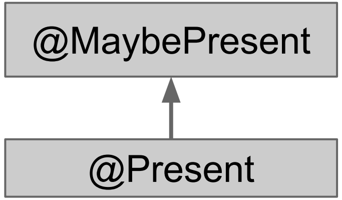
`myOptional.get()`

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`.

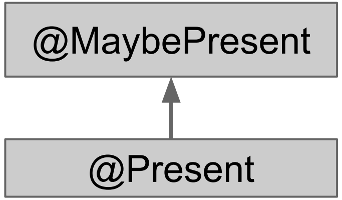
example call:

```
myOptional.get()
```

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

✓

Type rules for Optional



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

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

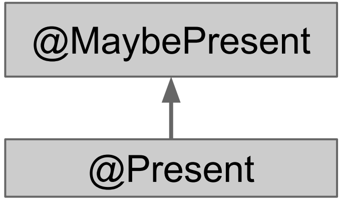
example call:

```
myOptional.get()
```

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

✓

Type rules for Optional



“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)

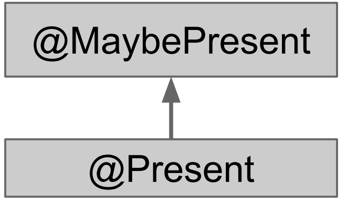
Type introduction rules

For Nullness type system:

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

✓

Type introduction for Optional



@MaybePresent

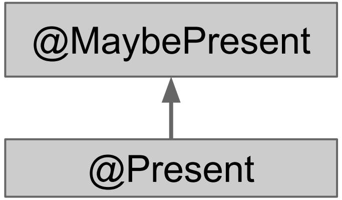
@Present

“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



“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

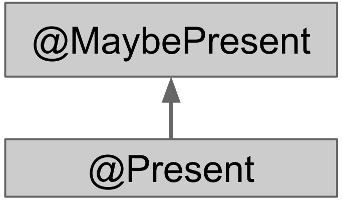
```
@Nullable Object y;
y = new SomeType();
...
```

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`

✓

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/`