



The long Good-Bye to NullPointerException



Stephan Herrmann





The Road Behind

1965

- › The billion dollar mistake
 - › 1965 Tony Hoare introduced Null references in ALGOL W

2006 – 3.2

- › Bug 110030 – Provide support for **null reference analysis**

2011 – 3.8

- › Bug 186342 – Using **annotations** for null checking
 - › EclipseCon Europe 2011: “Bye-bye NPE”

2013 – 4.3

- › Bug 383368 – syntactic null analysis for **field references**

2014 – 4.4

- › Bug 392099 – Apply null **annotations on types** for null analysis
 - › EclipseCon Europe 2014: “A Deep Dive into the Void”

2015 – 4.5

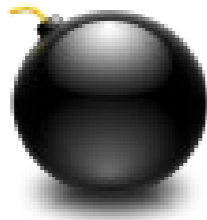
- › Bug 331651 – Support **external null annotations** for libraries
- › Continuous improvement of analysis for:
 - › loops, assert, generics, modules, interfacing with “legacy” code
 - › injection, well-known libraries



The billion dollar mistake

1965

```
String val;  
  
...  
  
val = null;  
  
...  
  
uc = val.toUpperCase();
```





Null Reference Analysis

2006 – 3.2

```
String val;  
  
...  
  
val = x;  
  
...  
  
if (val != null)  
    uc = val.toUpperCase();  
else  
    lc = val.toLowerCase();
```



- As part of flow analysis
- Only local analysis



Using Annotations for Null Checking

- › Inter-procedural analysis
- › Simple “contracts”

2011 – 3.8

```
String meth(@NonNull String val1, @Nullable String val2)
{
    if (someFlag)
        return val1.toUpperCase();
    else
        return val2.toLowerCase();
}

...

s = meth("hello", null);

s = meth(null, "hello");
```



Flow Analysis for Fields?

2013 - 4.3



```
class Test {  
    @Nullable String f;  
  
    String meth() {  
        if (this.f != null) {  
            // some code here  
            return this.f.toUpperCase();  
        }  
        return "<don't know>";  
    }  
}
```

- Unexpected errors
- 3 risks of shared data
 - concurrency
 - aliasing
 - side effects
- Compromise
 - accept minimal risk
 - no sophistication
 - “syntactic analysis”



Null Annotations on Types

```
String meth(@NonNull List<@Nullable Person> val)
{
    return val.get(0).getName();
}
...
```



- › Since Java 8
- › JSR 308
- › @Target(TYPE_USE)
 - › Contracts?
 - › Extended type system

2014 – 4.4



External Null Annotations

```
@NonNull Map<@NonNull String, NonNull Person> val = x;  
String name = val.get("Joe").getName();  
...
```

- › “Legacy” libraries
- › Files: *.eea
- › Command “Annotate”

2015 – 4.5



External Null Annotations

The screenshot shows the Eclipse IDE interface with two tabs: 'Map.class' and 'MapTest.java'. The 'MapTest.java' tab is active, displaying the following code:

```
*  
* @param key the key whose associated value is to be returned  
* @return the value to which the specified key is mapped, or  
*      {@code null} if this map contains no mapping for the key  
* @throws ClassCastException if the key is of an inappropriate type  
*         this map  
* (<a href="Collection.html#optional-restrictions">optional</a>)  
* @throws NullPointerException if the specified key is null and thi  
*         does not permit null keys  
* (<a href="Collection.html#optional-restrictions">optional</a>)  
*/  
get(Object key);
```

A tooltip is visible over the 'get' method signature, showing two annotation proposals:

- @ Annotate as '@NonNull V'
- @ Annotate as '@Nullable V'

The 'get' method signature is also visible: `get (Ljava/lang/Object;)TV; (Ljava/lang/Object;)T0V;`

A dark grey box at the bottom of the tooltip contains the text: **Ctrl+1 – Annotate Class File**, with the subtitle: *Externally add Annotations to a Class File.*

At the bottom of the tooltip, there are two buttons: 'Change external annotations' and 'Press 'Tab' from proposal table o'.

- “Legacy” libraries
- Files: *.eea
- Command “Annotate”

2015-



External Null Annotations

The screenshot shows the Eclipse IDE with a Javadoc popup for the `@org.eclipse.jdt.annotation.Nullable` annotation on the `java.util.Map.get` method. The popup text is as follows:

```
@org.eclipse.jdt.annotation.Nullable V  
java.util.Map.get(java.lang.Object key)  
Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.  
More formally, if this map contains a mapping from a key k to a value v such that (key==null ?
```

- › “Legacy” libraries
- › Files: *.eea
- › Command “Annotate”

2015 -

The screenshot shows the Eclipse IDE with the `@org.eclipse.jdt.annotation.Nullable` annotation on the `java.util.Map.get` method. The text is as follows:

```
@org.eclipse.jdt.annotation.Nullable V java.util.Map.get(  
Returns the value to which the specified key is mapped, or null if
```



External Null Annotations



```
@NonNull Map<@NonNull String, NonNull Person> val = x;  
String name = val.get("Joe").getName();  
...  
...
```

- › “Legacy” libraries
- › Files: *.eea
- › Command “Annotate”

2015 – 4.5



The Road Behind

1965

- › The billion dollar mistake
 - › 1965 Tony Hoare introduced Null references in ALGOL W

2006 – 3.2

- › Bug 110030 – Provide support for **null reference analysis**

2011 – 3.8

- › Bug 186342 – Using **annotations** for null checking
 - › EclipseCon Europe 2011: “Bye-bye NPE”

2013 – 4.3

- › Bug 383368 – syntactic null analysis for **field references**

2014 – 4.4

- › Bug 392099 – Apply null **annotations on types** for null analysis
 - › EclipseCon Europe 2014: “A Deep Dive into the Void”

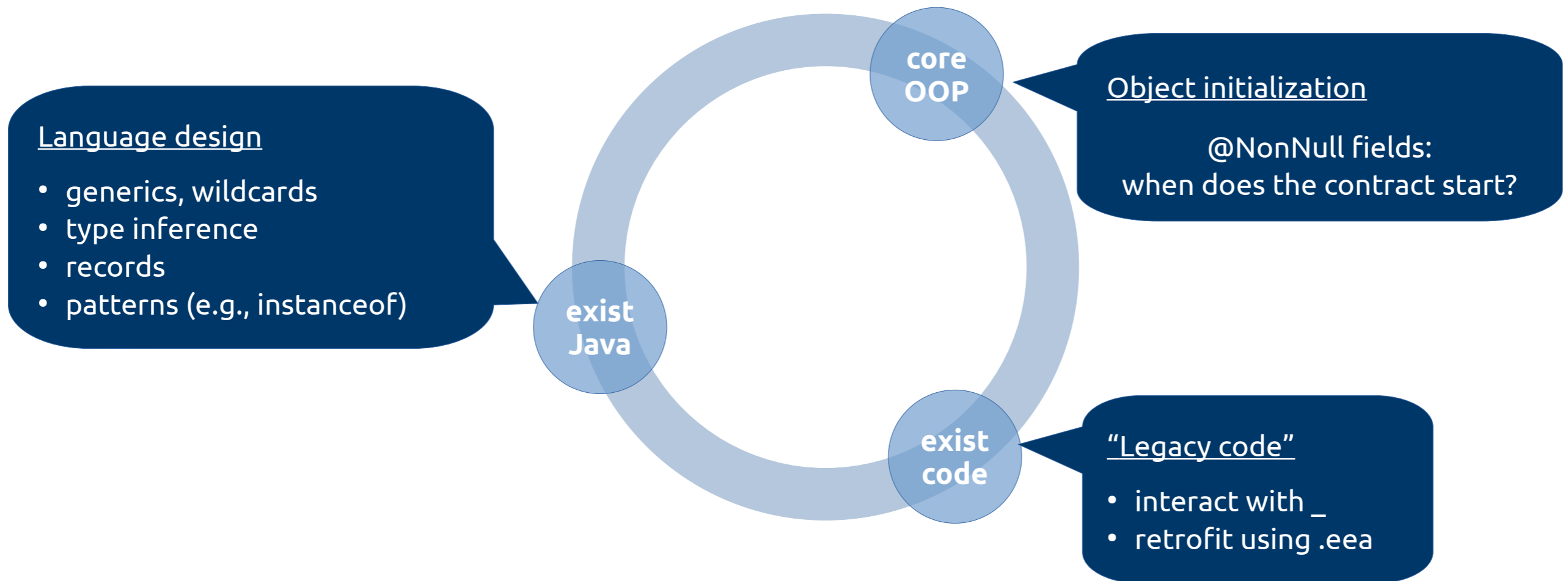
2015 – 4.5

- › Bug 331651 – Support **external null annotations** for libraries



Difficulties

➤ Adding a core concept **interacts** with all that's already there





Generics meet Legacy

- Warn when legacy code can taint null-checked values

The screenshot shows the Eclipse IDE with three tabs: Class2.java, Legacy.java, and Class3.java. The Class3.java tab is active, displaying the following code:

```
@NonNullByDefault
public class Class3 {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add("Joe");
        Legacy.printNames(names);
        for (String name : names) {
            System.out.println(name.toLowerCase());
            System.out.flush();
        }
    }
}
```

A warning tooltip is visible over the code, stating: "Unsafe null type conversion (type annotations): The value of type 'List<@NonNull String>' is made accessible using the less-annotated type 'List<String>'".

The Problems view shows one item:

Description	Resource	Path	Location	Type
Unsafe null type conversion (type annotations): T	Class3.java	/NullVsLegacy/src/	line 13	Java Pro

The Console view shows the following output:

```
<terminated> Class3 [Java Application] /home/java/jdk1.8.0_162/bin/java (Feb 6, 2020, 9:34:26 PM)
joe
Exception in thread "main" java.lang.NullPointerException
    at snippets.Class3.main(Class3.java:15)
```

2020 – 4.15



Advances concerning **External Annotations**



What “3rd Party” Code?

- › Originally: overlays for jars
- › Other code that cannot be annotated:
 - › Generated code (if you don't own the code generator)!
- › Solution
 - › *Every* classpath entry can refer to external annotations
 - › EEA can be superimposed even on sources

2021 – 4.19



What “3rd Party” Code?

- › Originally
- › Other code
 - › Generated
- › Solution
 - › Every
 - › EEA ca

```
File Edit Source Refactor Navigate Search Project Run Window Help
Package x
NullTest
JRE System Library
src
  bar
    Bar.java
  src-gen
    foo
      Foo.java
Referenced Library
annotations
  foo
    Foo.eea

Bar.java x
@NonNullByDefault
public class Bar {
    void m(Foo f) {
        printAll(f.getFoos());
    }
}

void printAll(Collection<Foo> f) {
    for (Foo foo : f) {
        System.out.println(foo.toString());
    }
}

Foo.java x
public interface Foo {
    public Collection<Foo> getFoos();
}

@NonNull Collection<@NonNull Foo> foo.Foo.getFoos()
```

2021 – 4.19



What “3rd Party” Code?

- > Ori
- > Oth
- >
- > Sol
- >
- >

The screenshot shows the 'Properties for NullTest' dialog in Eclipse. The 'Java Build Path' tab is active, showing the 'Source folders on build path' section. The following items are listed:

- NullTest/src
- NullTest/src-gen
- Included: (All)
- Excluded: (None)
- Native library location: (None)
- External annotations: /NullTest/annotations** (highlighted with a red circle)
- Ignore optional compile problems: No
- Contains test sources: No

At the bottom of the dialog, the 'Default output folder' is set to 'NullTest/bin'. The 'Apply and Close' button is highlighted in blue.

2021 – 4.19



How do clients see my code?

- › Internally, .eea have become part of generated sources
- › Clients of those classes should see the same API!
- › PDE supports new directive in MANIFEST.MF
 - › Ensure .eea are included in deployed jar (build.properties)
 - › **Eclipse-ExportExternalAnnotations**: true
 - › PDE will do the rest behind the scenes
 - Resolved elements of **Plug-in Dependencies** will have proper **annotationpath**
- › Useful for
 - › Plug-in projects with ...
 - › ... generated source ...
 - › ... superimposed with .eea

2022 – 4.24



How to Manage .eea?

- › Text files
 - › Initially expected **inside** each project using a legacy library
- › Should each project maintain its own set of .eea?
 - › Shareable as jar files / artifacts
- › Brute force:
 - › Search all classpath locations for .eea
- › Bad impact on IDE performance

see also: lastnpe.org
addresses such issues by extending m2e



The challenge for EEA in the IDE

- › JDT should precisely know where to find .eea
 - › But now .eea are **artifacts** needing **dependency** management
- › Dependency management is handled by your **build system**
- › JDT doesn't know *any* build system
 - › But JDT knows about **classpath containers**
 - Plug-in Dependencies
 - Maven Dependencies
 - ...
- › **Solution**
 - › Specify annotation location relative to a classpath container:
 - › `annotationpath=org.eclipse.pde.core.requiredPlugins/org.example.annotations`
 - › If annotation artifact is in your dependencies* then JDT will find it for eea lookup

2022 – 4.24



IDE vs. CI Builds

› IDE “knows” about .eea

› Annotation path is configured via **.classpath**

- .classpath may depend on Eclipse-specifics (like classpath containers)

› Read .eea:

- compiler
- hover

› Write .eea:

- **Ctrl+1** Annotate

› Build tools don't know about **.classpath**

› Add .eea artifacts to your dependencies

› Catch all: -annotationpath CLASSPATH

```
<plugin>
  <groupId>org.eclipse.tycho</groupId>
  <artifactId>tycho-compiler-plugin</artifactId>
  <configuration>
    <useProjectSettings>true</useProjectSettings>
    <compilerArgs>
      <compilerArg>-annotationpath</compilerArg>
      <compilerArg>CLASSPATH</compilerArg>
    </compilerArgs>
  </configuration>
</plugin>
```



Summary

- › Did I promise too much in 2011?
 - › Yes
- › Is treatment of null a reason to abandon Java?
 - › No
- › Is it possible to create provably NPE-free code?
 - › Yes, but only in green field, clean room development.
- › Which TYPE_USE annotations?
 - › Not “JSR 305”!
 - › `org.checkerframework.checkers.nullness` ?
 - › `org.eclipse.jdt.annotation_2.2.x` ?
 - › `org.jspecify` ?