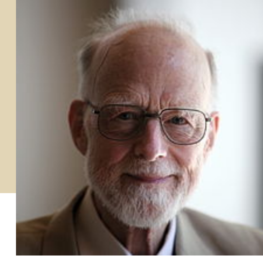
Lecture 16 — NonNull Analysis

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Null References



> “I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. **But I couldn’t resist the temptation to put in a null reference, simply because it was so easy to implement**. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”
>
> *–Sir Tony Hoare*

See “Null References: The Billion Dollar Mistake”, Tony Hoare, 2009 (InfoQ).

Photograph CC Licensed by Rama

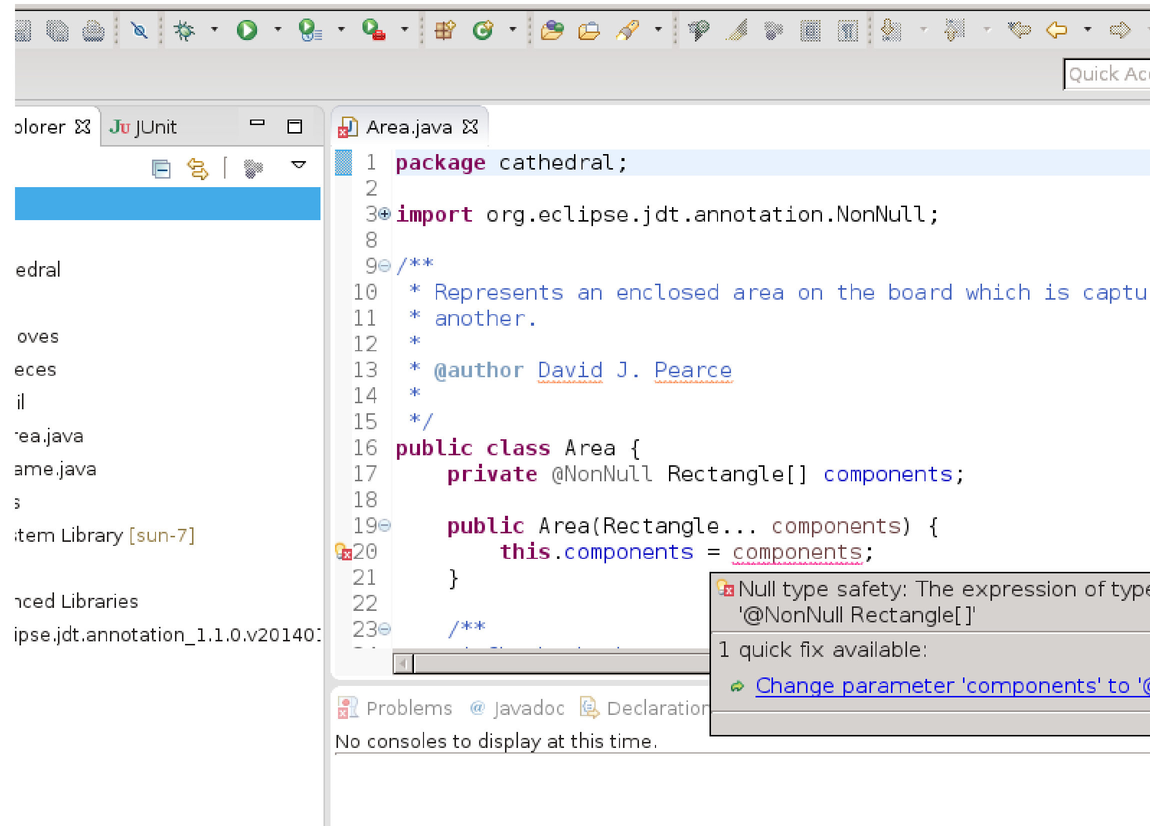
@NonNull Analysis

```
class Rectangle {
  private @NonNull Point p1;
  private @NonNull Point p2;

  Rectangle(@NonNull Point p1, @NonNull Point p2){
    this.p1 = p1;
    this.p2 = p2;
} }
```

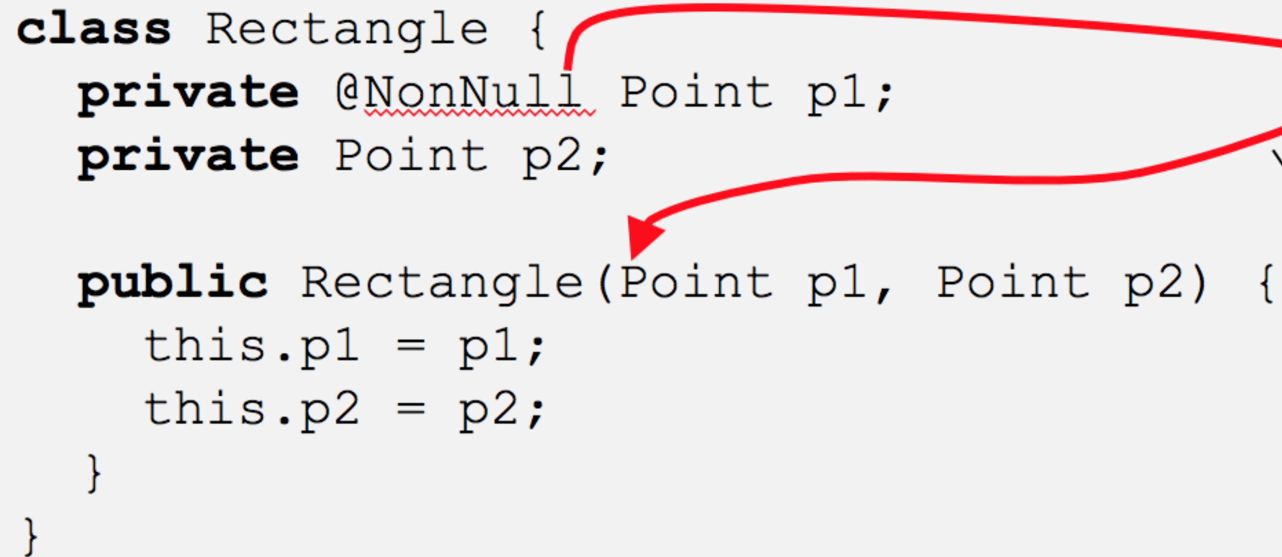
- `@NonNull` indicates variable **cannot** hold null
- `@Nullable` indicates variable **can** hold null
- Annotate program to show it is **free** of null dereferences

Eclipse @NonNull Analysis



- Built-in @NonNull Analysis is available!
  - **Awkward** to enable
  - Has a few “**quirks**”

Using @NonNull Annotations



When starting from **scratch** ...

... adding **one** `@NonNull` Annotation ...

... often leads to **another**!

@NonNull, @Nullable and “Default”

```
public String toString(Point p) {
   return p.toString();
}
```

- Meaning of Annotations:
  - `@NonNull` — Cannot be null
  - `@Nullable` — Can be null
  - **No annotation** — Can be either (this is “default”)

- Hence, system is **unsound** (i.e. unsafe)
  - Only if some unannotated reference types exist
  - Supports **legacy code**, but means above **compiles**

Subtyping

```
public @Nullable Point f(@NonNull Point p) {
    return p;  // Will non-null check OK
}
```

```
public @NonNull Point g(@Nullable Point p) {
    return p;  // Fails to non-null check
}
```

- `@NonNull` is a **subtype** of `@Nullable`
- `@NonNull T` holds **subset** of values in `@Nullable T`
- But, `@Nullable` can hold **`null`** where `@NonNull` cannot
- Subtyping **denoted** as `@NonNull` $\leq$ `@Nullable`

Inheritance (Parameters)

```
class Parent {
  public boolean contains(@Nullable Point p1) { ... }
}
class Child extends Parent {
  public boolean contains(@NonNull Point p1) { ... }
}
```

```
class Parent {
  public boolean contains(@NonNull Point p1) { ... }
}
class Child extends Parent {
  public boolean contains(@Nullable Point p1) { ... }
}
```

- *Which way around makes* ***sense****?*

Inheritance (Returns)

```
class Parent {
  public @Nullable String toString() { ... }
}
class Child extends Parent {
  public @NonNull String toString() { ... }
}
```

```
class Parent {
  public @NonNull String toString() { ... }
}
class Child extends Parent {
  public @Nullable String toString() { ... }
}
```

- *Which way around makes* **sense***?*

Array Annotations

```
@NonNull Object @NonNull [] items;
```

- Every **object** in `items` is `@NonNull`
- The `items` **reference** is `@NonNull`

Limitations

```
class BoundedList {
  private @Nullable Object @NonNull [] items;
  private int length;

  public BoundedList(int size) {
    items = new Object[size];
  }
  public void add(@NonNull Object x) {
    if(length < items.length;) {
      items[length++] = x;
  } }

  public @NonNull Object get(int i) {
     if(i >= 0 && i < length) { return items[i]; }
     throw new ArrayIndexOutOfBoundsException();
} }
```

- Does not pass non-null checking ... why?

Assertions

```
class BoundedList {
  ...
  public @NonNull Object get(int i) {
    if(i >= 0 && i < length) {
      Object item = items[i];
      assert item != null; // hint to non-null checker
      return item;
    }
    throw new ArrayIndexOutOfBoundsException();
} }
```

- Passes non-null checking now!
- Assertion acts as **hint** for non-null checker
- However, assertions can be used **unsafely**

Unsafe Constructors ... ?!

```
class Parent {
  public Parent() { f(null); }
  public void f(String x) {}
}

class Child extends Parent {
  private @NonNull List<String> items;
  public Child() {
    items = new ArrayList<String>();
  }
  public void f(String x) { items.add(x); }
}
```

- Here lurks a **null-dereference error** ... how?

Intra- versus Inter-procedural Analysis

Intraprocedural. *Analysis limited to exploring single method at a time. Assumes all valid information flow between two functions is valid.*

Interprocedural. *Analysis explores entire program at one time. Considers actual information flow between functions.*

- **Intraprocedural**:
  - Less precise, but more efficient
  - Requires programmer-supplied information at method boundaries (e.g. types, annotations, etc)
- **Interprocedural**:
  - More precise, but less efficient
  - Can infer information at function boundaries automatically

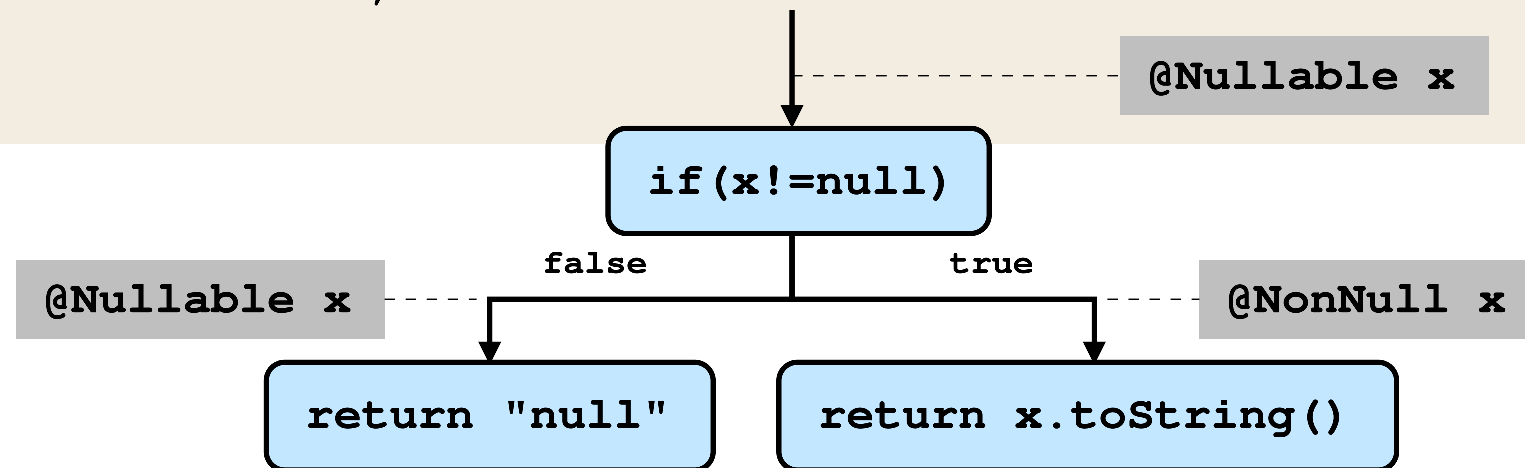
Flow Analysis

> **Data-flow analysis** *is a technique for gathering information about the possible set of values calculated at various points in a computer program*
>
> *–Wikipedia*

- `@NonNull` Checking:
  - Example of **intraprocedural analysis**.
  - Also example of **flow analysis**.
  - Like type checking, methods **checked in isolation**.
  - Life definite assignment, checks over **Control-Flow Graph**.

@NonNull Checking

```
String toString(@Nullable Integer x) {
  if(x != null) { return x.toString();
  } else {
     return ``null'';
} }
```



- NonNull Checker determines null information for **each variable at each point**

Syntactic @NonNull Analysis?

```
class Example {
  private @Nullable String field;

  public void f(@NonNull String item) { ... }

  public void g() {
    if(this.field != null) {
       f(this.field);
    }
  }
}
```

- This does not non-null check ... **why?**

Syntactic @NonNull Analysis?

```
class Example {
  private @Nullable String field;

  public void f(@NonNull String item) { ... }

  public void g() {
    String myField = this.field;
    if(myField != null) {
      f(myField);
    }
  }
}
```

- Now it works ... !!

Syntactic @NonNull Analysis?

```
class NonNullAdaptorList {
  private List<Object> items = ...;

  public @NonNull Object get(int i) {
    if(items.get(i) != null) {
       return items.get(i);
    } else { return ""; }
  }
}
```

- This also **does not** non-null check ... **why?**

Further Reading

- *Java Bytecode Verification for @NonNull Types*. Chris Male, David J. Pearce, Alex Potanin and Constantine Dymnikov. In *Proc. Compiler Construction (CC)*, 2008

- *Non-null references by default in Java: Alleviating the nullity annotation burden*, Chalin and James. In *Proc. ECOOP*, 2007.

- *Declaring and checking non-null types in an object- oriented language*, Fähndrich and Leino. In *Proc. OOPSLA*, 2003.