Java™

ORACLE

Type annotations in Java 8 и почему это хорошо

Alexandre (Shura) Iline
Java quality architect

Program Agenda

- Quick emotional introduction
  - What's wrong with testing
- Annotations before Java 8
- Type annotations
- Checkers framework
- Custom annotations

What's wrong with testing?

Here's a developer.

He works hard on a feature

Designs the feature

Implements the feature

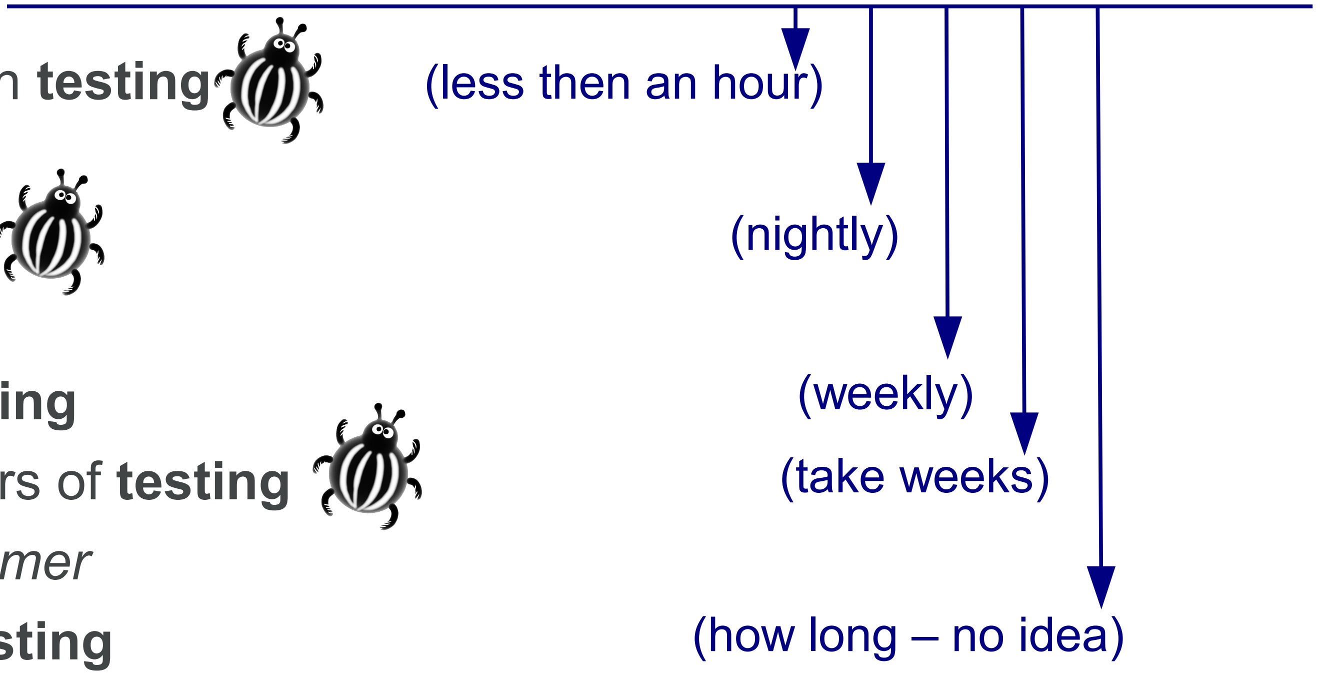
Tests (good developer) ...

Submits the fix ...

With a bug!

Testing

- Fix ready
- Pre-integration **testing**
- *Build*
- More **testing**
- *Promotion*
- Yet more **testing**
  - **Few** layers of **testing**
- *Went to customer*
- Real world **testing**



Back to the developer

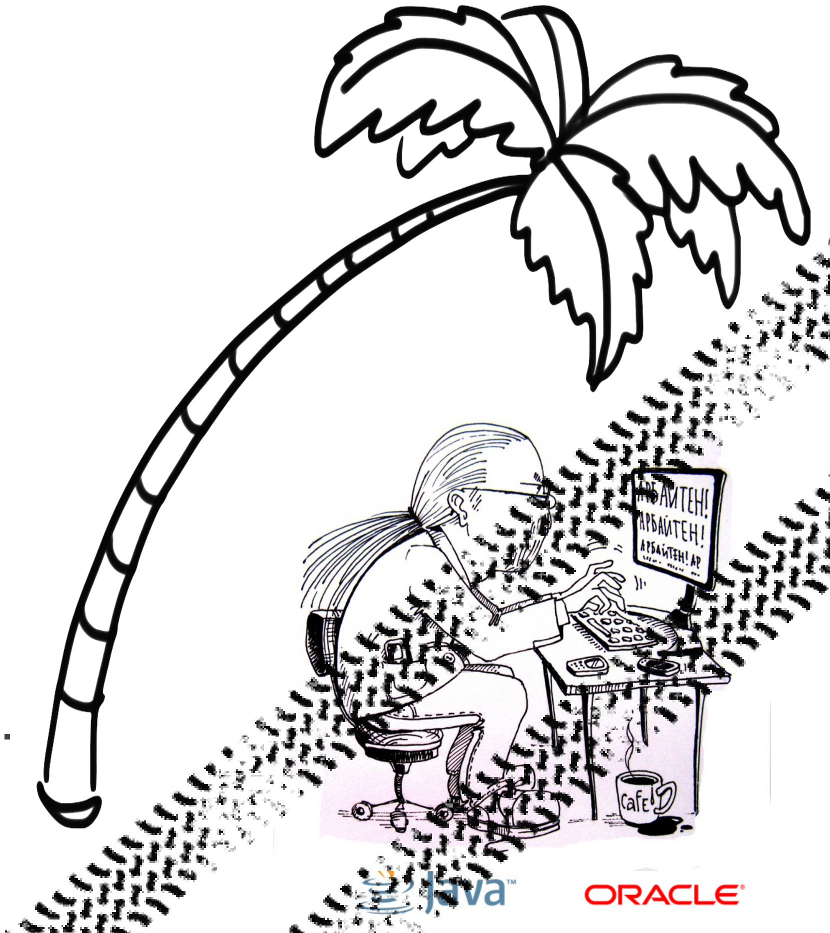
Which developer?

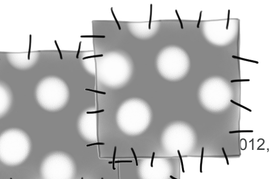
That developer could ...

- be working on something else
- leave for a vacation
- get sick
- leave the company
- transfer the responsibility

Has to switch, re-learn the design, etc. etc.

Often just patch.





Could it be done differently?

- More pre-integration testing.
  - Yes, please! :)
  - Including unit and some functional
  - Still too little
- Static analysis
  - only helps that much
- Reviews
  - better
  - no predictability, though

Fundamental problems with testing

- Too late
  - Comes after the implementation is complete
  - Cost of bug grows
  - Enforces multi-tasking
- Not a silver bullet
  - Next slide

Program testing can be used to show the presence of bugs, but never to show their absence.

[“Structured programming”, Dahl O.J., Dijkstra E.W. and Hoare C.A.R.] (1972)

Formal verification

Formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.

- Testing – **upper** bounds for program quality
  - Passed tests mean nothing
  - Failed tests means something is broken
- Formal verification – **lower** bounds for program quality
  - Guarantees absence of failures of some kind

Formal verification. Applied. Take with caution.

```
boolean isPowerOfTwo(int a){return (a&(a-1))==0;}
```

$$\forall\, 0 < a \in N : a \,\&\, (a-1) = 0 \Leftrightarrow \exists\, n \in N : a = 2^n$$

a > 0 => binary presentation of **a** has a least one 1 bit

$\mathbf{a = (a_1 \ldots a_k)}\overbrace{\mathbf{10\ldots0}}^{m\ >=\ 0}$ $\quad$ $a\text{-}1 = (a_1 \ldots a_k)0\overbrace{1\ldots1}^{m}$ $\quad$ $a\&(a\text{-}1) = (a_1 \ldots a_k)0\overbrace{0\ldots0}^{m}$

$a\&(a\text{-}1) = 0 \Rightarrow a_1,\ldots,a_k = 0 \Rightarrow a = 2^m$

$a = 2^n \Rightarrow m=n,\ a_1,\ldots,a_k = 0 \Rightarrow a\&(a\text{-}1) = 0$

[When] you have given the proof of [a program's] correctness, … [you] can dispense with testing altogether.

[“Software engineering”, Naur P., Randell B.]

(1969)

Only the <u>true formal</u> verification is too expensive

How is that all related to type annotations?

Hold on.

Annotations in Java Before 8

Annotation

`@FromDictionary`

an-no-ta-tion: a critical or explanatory note added to a text

- the statement above is from a dictionary
- the statement above is an annotation

In Java (and other programming languages)

Information about a software program that is not a part of the program itself

Annotations in Java. Examples

```
@Stateless @LocalBean public class GalleryFacade {
  @EJB private GalleryEAO galleryEAO;

  @TransactionAttribute(SUPPORTS)
    public Gallery findById(Long id) { ... }

  @TransactionAttribute(REQUIRED)
  public void create(String name) { … }
```

Annotations in Java

- Introduced in Java 5
- Built-in
  - @Override
  - @Deprecated
  - @SupressWarning
- Custom
- Used extensively
  - JavaEE
  - Test harnesses

@

JSRs

- JSR-175: A Metadata Facility for the Java^{TM} Programming Language

A metadata facility for the Java^{TM} Programming Language would allow classes, interfaces, fields, and methods to be marked as having particular attributes.

- JSR-250: Common Annotations for the Java^{TM} Platform

This JSR will develop annotations for common semantic concepts in the J2SE and J2EE platforms that apply across a variety of individual technologies.

- JSR-269: Pluggable Annotations Processing API

Provide an API to allow the processing of JSR 175 annotations (metadata); this will require modeling elements of the Java(TM) programming language as well as processing-specific functionality.

Before Java 8

- Declarations only
  - Class
  - Method
  - Field
  - Parameter
  - Variable

```
@A public class Test {
  @B private int a = 0;
  @C public void m(@D Object o) {
    @E int a = 1;
    ...
  }
}
```

Before Java 8

- `@Target`
  - `ANNOTATION_TYPE, CONSTRUCTOR, FIELD, LOCAL_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE`
- `@Retention`
  - `SOURCE, CLASS, RUNTIME`
- Fields, default values
  - `@Test(`**`timeout=100`**`)`
  - Primitive, String, Class, enum, array of the above
- No inheritance

Typical application programmer will never have to define an annotation type, but it is not hard to do so.

[Java language guide.

http://docs.oracle.com/javase/1.5.0/docs/guide/language/]

Yet done often for a purpose of testing.

Custom annotations

- Define a custom annotation type
- Apply to the code
- Use in runtime
  - Reflection
- Use in compile-time
  - Annotation processor

Custom annotation

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Property {
    String value();
    boolean waitable() default false;
}
```

```
@Property("Text")
public String getText() {...}
```

Custom annotation. Runtime.

- java.lang.Class
  - `getAnnotation(Class<A>), getAnnotations(), getDeclaredAnnotations()`
- java.lang.reflect.Method
  - `getAnnotation(Class<A>), getDeclaredAnnotations(), getParameterAnnotation()`
- java.lang.reflect.Field
  - `getAnnotation(Class<A>), getDeclaredAnnotations()`

Custom annotation. Runtime.

```
private void addAnnotationProps(Object bean) {
  for (Method m : bean.getClass().getMethods()) {
    if (m.isAnnotationPresent(Property.class)) {
      String name = m.getAnnotation(Property.class).value();
      //put method result into a map
    }
  }
}
```

Custom annotation. Compile time.

- `javax.annotation.processing.Processor`
  - `Set<String> getSupportedAnnotationTypes()`
  - `boolean process(Set<? extends TypeElement>, RoundEnvironment)`
- `javax.annotation.processing.RoundEnvironment`
  - `Set<? extends java.lang.model.element.Element> getElementsAnnotatedWith(Class<A>)`
- `javac … -processor <annotation processor class> …`

Custom annotations. Compile time.

```
public class PropProcessor extends AbstractProcessor {
  ...
  @Override
  public boolean process(Set<? extends TypeElement> set,
                          RoundEnvironment re) {
    for (Element e :
         re.getElementsAnnotatedWith(Property.class)) {
      Property p = el.getAnnotation(Property.class);
      //generate the waiting code
    }
  }
}
```

How is that all related to verification?

Just a little bit longer.

Annotations in Java 8

Annotations in Java 8

- Could be used on any use of a type
  - more on next slides ...
- `@Target`
  - `TYPE_PARAMETER, TYPE_USE`
- Repeating annotations

JSR 308: Annotations on Java Types

This JSR extends the Java annotation syntax to permit annotations on any occurrence of a type. Previously, annotations could not be placed on generic type arguments, type casts, etc.

JSR 308: Annotations on Java Types (1)

- **method receivers**

```
public int size() @Readonly { ... }
```

- **generic type arguments**

```
Map<@NonNull String, @NonEmpty List<@Readonly Document>>
 files;
```

- **arrays**

```
Document[][@Readonly] docs2 =
                      new Document[2] [@Readonly 12];
```

JSR 308: Annotations on Java Types (2)

- **typecasts**

```
myString = (@NonNull String)myObject;
```

- **type tests**

```
boolean isNonNull = myString instanceof @NonNull String;
```

- **object creation**

```
new @NonEmpty @Readonly List(myNonEmptyStringSet)
```

JSR 308: Annotations on Java Types (3)

- **type parameter bounds**

```
<T extends @A Object, U extends @C Cloneable>
```

- **class inheritance**

```
class UnmodifiableList implements @Readonly List<@Readonly T> { ... }
```

- **throws clauses**

```
void monitorTemperature() throws @Critical TemperatureException { ... }
```

Annotations in Java 8. More examples

```
@NonNull MyClass @Length(2)[] @ReadOnly[]


class Person {
...
  void setValue(@Mutable Person this, String firstName) {
    this.firstName = firstName;
  }
...
}
```

Still, how is that all related to verification?

Next slide.

Annotations in Java 8

- Permitted on every type use
- Are analyse-able at compile time
- The whole compilation tree is available for analysis

Hence …

- A lot of verification could be done at compile time

JSR 305: Annotations for Software Defect Detection

This JSR will work to develop standard annotations (such as @NonNull) that can be applied to Java programs to assist tools that detect software defects.

- Nullness
- Check return value
- Taint
- Concurrency
- Internationalization

Checkers framework

Checkers framework

The Checker Framework enhances Java’s type system to make it more powerful and useful. This lets software developers detect and prevent errors in their Java programs.

- Works now – with JDK 7
- 14 checkers.
- Allows to build custom checkers

http://types.cs.washington.edu/checker-framework/

Checkers framework.

- Works for JDK 7
  - substituting javac
  - generating valid byte-code
  - only affects compile-time
- Eclipse plugin

```
$ <checkers>/binary/javac -processor <a checker>
<other parameters>
```

```
$ java -jar <checkers>/binary/javac.jar -processor <a
checker> <other parameters>
```

Checkers framework.

- `@TypeQualifier`
- `“Inheritance”`
  - `@SubtypeOf(Class<? extends Annotation>)`
- Automatic type refinement
- Default qualifiers
  - `@ImplicitFor(<class names or classes or types>)`
  - `checkers.types.TreeAnnotator`

Checkers framework. Type checkers.

- Nullness
- IGJ (Immutability Generics Java)
- Lock
- Property file
- Units
- Typestate

@

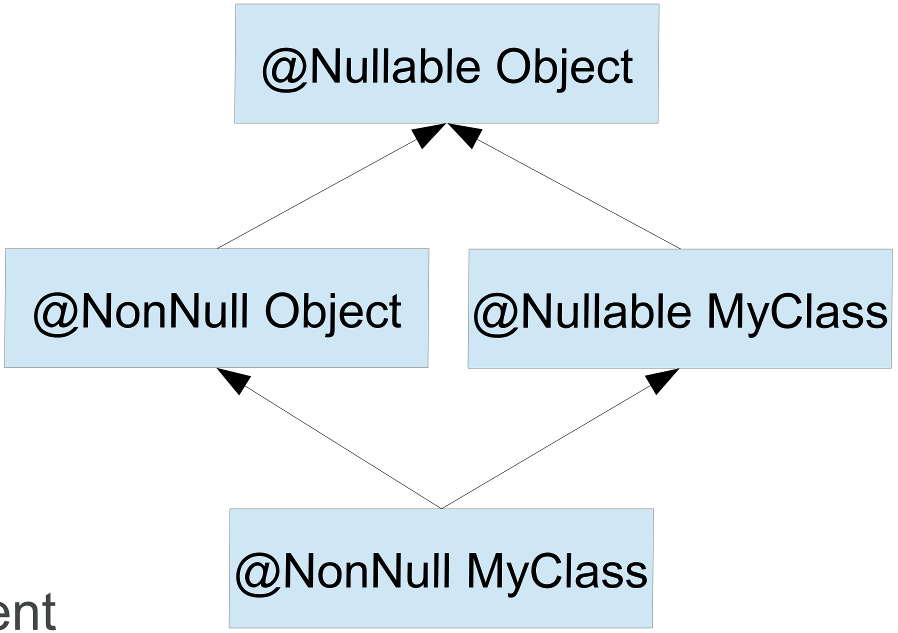
Nullness checker

If the Nullness checker issues no warnings for a given program, then running that program will never throw a null pointer exception.

- @Nullable, @NonNull, @PolyNull
- @Raw, @NonRaw, @PolyRaw
- @NonNullOnEntry, @Pure, @AssertNonNullAfter, @AssertNonNullIfTrue, @AssertNonNullIfFalse, @AssertParametersNonNull

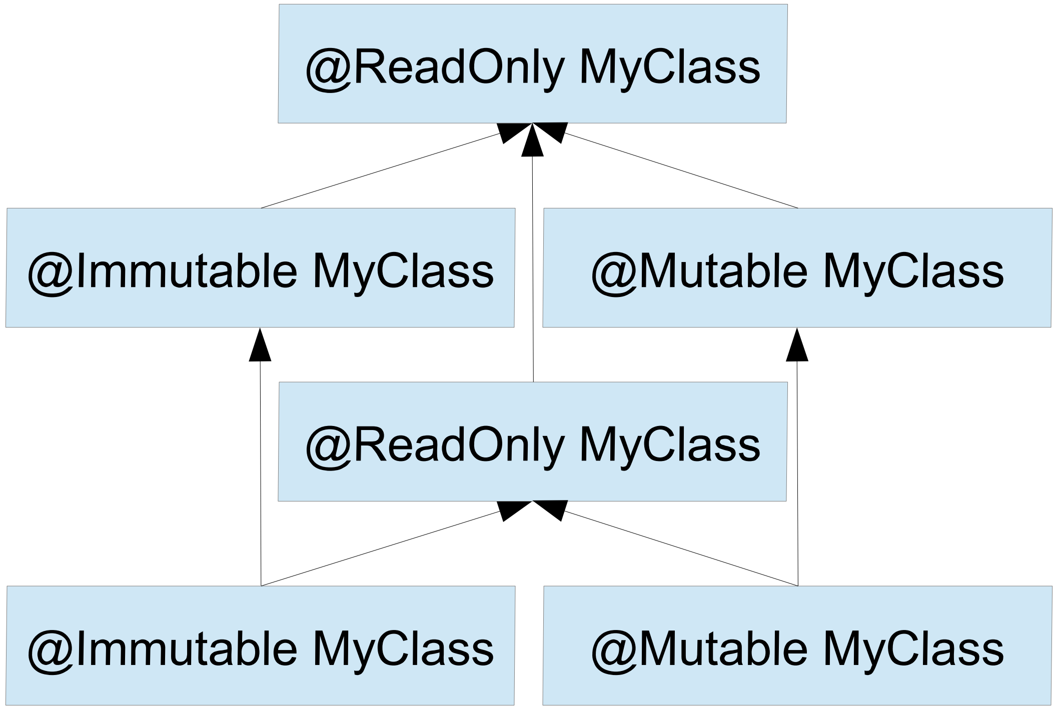
Nullness checker

```
@Nullable Object   obj;
@NonNull  Object nnobj;
...
nnobj.toString();
~~obj.toString();~~
~~nnobj = obj;~~
obj = nnobj;
~~if (nnobj == null)~~ ...
if (obj != null) {
  nnobj = obj; //type refinement
}
```



IGJ checkers

- @Immutable
- @Mutable
- @ReadOnly
- @Assignable
- @AssignFields
- @I



Lock checker

If the Lock checker issues no warnings for a given program, then the program holds the appropriate lock every time that it accesses a variable.

- Inspired by “Java Concurrency In Practice” (?)
- @GuardedBy – only allowed to access if a particular lock is held
- @Holding – only allowed to be called if a particular lock is held

Lock checker

```
@GuardedBy("MyClass.myLock") Object myMethod() { ... }

@GuardedBy("MyClass.myLock") Object x = myMethod();

@GuardedBy("MyClass.myLock") Object y = x;

~~Object z = x;~~
~~x.toString();~~

synchronized(MyClass.myLock) {
  y.toString();
}
```

Lock checker

```
void helper1(@GuardedBy("MyClass.myLock") Object a) {
  ~~a.toString();~~
  synchronized(MyClass.myLock) {
    a.toString();
  }
}

@Holding("MyClass.myLock")
void helper2(@GuardedBy("MyClass.myLock") Object b) {
  b.toString();
}
```

Lock checker.

@GuardedBy and @Holding parameter syntax.

this : the object in whose class the field is defined.
class-name.this : for inner classes.
itself : the object to which the field refers.
field-name : the (instance or static) field.
class-name.field-name : the static field.
method-name() : the object is returned by calling the named method.
class-name.class : the Class object for the specified class.

Property file checker

The property file checker ensures that a property file or resource bundle (both of which act like maps from keys to values) is only accessed with valid keys.

- @PropertyKey
  - No need to annotate the literals
- @Localized
  - a localized message
- @LocalizableKey
  - a key to a localizable message

Units checker

- @Area: @mm2, @m2, @inch2 ...
- @Current: @A
- @Length: @m, @mm, @inch
- @Luminance: @cd
- @Mass: @kg, @pound
- @Speed: @mPERs, @kmPERh
- @Substance: @mol
- @Temperature: @C, @K
- @Time: @s, @min, @h

Units checker

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@TypeQualifier
@SubtypeOf( { Time.class } )
@UnitsMultiple(quantity=s.class, prefix=Prefix.nano)
@Target(ElementType.TYPE_USE,
ElementType.TYPE_PARAMETER)
public @interface ns {}
```

Typestate checker

```
@State public static @interface Opened { ... }
@State public static @interface Closed { ... }
class Stream {
    void open(@Closed(after=Opened.class) this);
    void close(@Opened(after=Closed.class) this);
    void int read(@Opened this);
}
@Opened Stream stream1 = ...; @Closed Stream stream2 = ...;
stream1.read();
~~stream2.read();~~
stream2.open(); stream2.read();
```

Custom checkers

@CreditCard. Usage

```
public class Account {
  private final @CreditCard String cardNumber;
  public Account(@CreditCard String number) {
    this.cardNumber = number;
  }
  public @CreditCard String getCardNumber() {
    return cardNumber;
  }
}
```

@CreditCard. Annotation

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE_USE,
    ElementType.TYPE_PARAMETER})
@TypeQualifier
@SubtypeOf(Unqualified.class)
public @interface CreditCard {}
```

@CreditCard. Checker and annotator

```
@TypeQualifiers(CreditCard.class)
@SuppressWarningsKey("credit.card")
public class CreditCardChecker extends BaseTypeChecker {
...
}
public class CreditCardAnnotator extends TreeAnnotator {
  public Void visitLiteral(LiteralTree tree,
      AnnotatedTypeMirror type) {
    ...
  }
}
```

@CreditCard. Usage

```
@SuppressWarnings("credit.card")
@CreditCard String convert(String input) {
  if(checkLuhn(input))
    return input;
  else
    throw IllegalArgumentException("...");
}
new Account("4111111111111111");
```

~~new Account("4111111111111110");~~

@CreditCard

- A card number in an account is **always** validated
- That is guaranteed at **compile time**
- You **do not need to test** with invalid numbers
- You **do need to test**
  - All `@SuppressWarnings(`**`"credit.card"`**`)`
  - `checkLuhn(String cardNum)`
-

More real life examples.

```
class SafeMath {
      public static double sqrt(@Positive double value)
{…}
}
@Positive p;
@Negative n;
sqrt(p);
sqrt(p – n);
sqrt(n*n);
```
~~sqrt(n);~~
~~sqrt(p + n);~~

Is this a “formal verification”?

- It is a “verification”, for sure.

Conclusion

- Formal verification vs testing
- Annotations before Java 8
- Annotations in Java 8
- Checkers framework
- Custom annotations

@

Q&A

ORACLE

Type annotations in Java 8 и почему это хорошо

Alexandre (Shura) Iline
Java quality architect

MAKE THE
FUTURE
JAVA

Java™

ORACLE®