Why Annotations?

Develop Intelligence

- Because you often want to provide data or instructions about your code ("metadata"). For example:
  - "This method is a test method"
  - "This parameter should be nonnegative"
  - "Suppress compile warnings about this method"
- Annotations can include information about annotations too:
  - "These annotations should be included in the Javadoc"
- In fact annotations can be attached to any Java declaration:
  - packages and classes, constructors, methods, fields, etc
- From Java 8, can be attached to any type use – eg method parameters, local variables

Basic Annotations

Develop Intelligence

- Simplest form:

```
@Override
void mySuperMethod() { ... }
```

  - An annotation can include elements, with values supplied at the time of use:

```
@Author(
   name = "Benjamin Franklin",
   date = "3/27/2003"
)
class MyClass() { ... }
```

```
@SuppressWarnings(value =
                          “unchecked")
void myMethod() { ... }
```

Basic Annotations

- Simplest form:

```
@Override
void mySuperMethod() { ... }
```

  - An annotation can include elements, with value
    of use:

```
@Author(
    name = "Benjamin Franklin",
    date = "3/27/2003"
)
class MyClass() { ... }
```

If only one element is being supplied, and its name is “value”, you can omit the name.

```
@SuppressWarnings(value =
                          “unchecked")
void myMethod() { ... }
```

Basic Annotations

- Simplest form:

```
@Override
void mySuperMethod() { ... }
```

  - An annotation can include elements, with values supplied at the time of use:

```
@Author(
    name = "Benjamin Franklin",
    date = "3/27/2003"
)
class MyClass() { ... }
```

```
@SuppressWarnings(
                            “unchecked")
void myMethod() { ... }
```

What are the consumers of annotations?

Develop Intelligence

- javac:
  - `@Override`
  - `@FunctionalInterface`
- JavaDoc processor:
  - `@deprecated, @author, @version, @since`
- `JPA:`
  - `@Entity, @Column, @Table`
- `JUnit:`
  - `@Test, @Before, @After, @Ignore`
- Every other framework…

When are annotations used?

Develop Intelligence

- The declaration of an annotation provides information about when it is going to be used
- Provided by an annotation on the annotation:
  - `@Retention(RetentionPolicy.SOURCE)`
    - Will be used by processors working on the source only
  - `@Retention(RetentionPolicy.CLASS)`
    - Will be used by processors working on the compiled byte code
  - `@Retention(RetentionPolicy.RUNTIME)`
    - Will be available at run time by reflection

Declaring and using an Annotation

Develop Intelligence

An annotation to provide class information to JavaDoc:

Declari

```
@interface ClassPreamble {
   String author();
   String date();
   int currentRevision() d
   String lastModified() d
   String lastModifiedBy()
   String[] reviewers();
}
```

Using the Annotation

```
@ClassPreamble (
   author = "John Doe",
   date = "3/17/2002",
   currentRevision = 6,
   lastModified = "4/12/2004",
   lastModifiedBy = "Jane Doe",
   reviewers = {"Alice", "Bob", "Cindy"}
)
public class Foo {...}
```

Predefined Annotation Types

Develop Intelligence

Used by the language:

- `@Override`
- `@Deprecated`
- `@SuppressWarnings`
- `@SafeVarargs`
- `@FunctionalInterface`

Applied to other annotations:

- `@Retention`
- `@Target`
- `@Documented`
- `@Inherited`
- `@Repeatable`

Finding an Annotation

Develop Intelligence

Obtain the `java.lang.Class` object of the object that might be annotated

```
Class<?> class = targetObject.getClass();
```

Get the elements that might be annotated

```
Method[] methods = clazz.getDeclaredMethods();
```

Look for an annotation, "`MyAnno`", say

```
for (Method m : Methods) {
   MyAnno anno = m.getAnnotation(MyAnno.class))
   if (anno != null) // found annotation, can process it
}
```

A Toy Test Harness

Develop Intelligence

```
public class Foo {
    public static void m1() { }
    public static void m2() { }
    public static void m3() {
        throw new RuntimeException("Boom");
    }
    public static void m4() { }
    public static void m5() { }
    public static void m6() { }
    public static void m7() {
        throw new RuntimeException("Crash");
    }
    public static void m8() { }
}
```

A Toy Test Harness – Declare an Annotation

Develop Intelligence

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

A Toy Test Harness – Applying the Annotation

Develop Intelligence

```
public class Foo {
    @Test public static void m1() { }
    public static void m2() { }
    @Test public static void m3() {
        throw new RuntimeException("Boom");
    }
    public static void m4() { }
    @Test public static void m5() { }
    public static void m6() { }
    @Test public static void m7() {
        throw new RuntimeException("Crash");
    }
    public static void m8() { }
}
```

A Toy Test Harness – Create Annotation Processor

Develop Intelligence

```
public class RunTests {
   public static void main(String[] args) throws Exception {
      int passed = 0, failed = 0;
      for (Method m : Class.forName(args[0]).getMethods()) {
         if (m.isAnnotationPresent(Test.class)) {
            try {
               m.invoke(null);
               passed++;
            } catch (Throwable ex) {
               System.out.printf("Test %s failed: %s %n", m, ex.getCause());
               failed++;
            }
         }
      }
      System.out.printf("Passed: %d, Failed %d%n", passed, failed);
   }
}
```