Java Annotations

Annotations

- Annotations are metadata or *data about data*. An annotation indicates that the declared element should be processed in some special way by a compiler, development tool, deployment tool, or during runtime.

Annotation-based development is certainly one of the latest Java development trends

The Basics

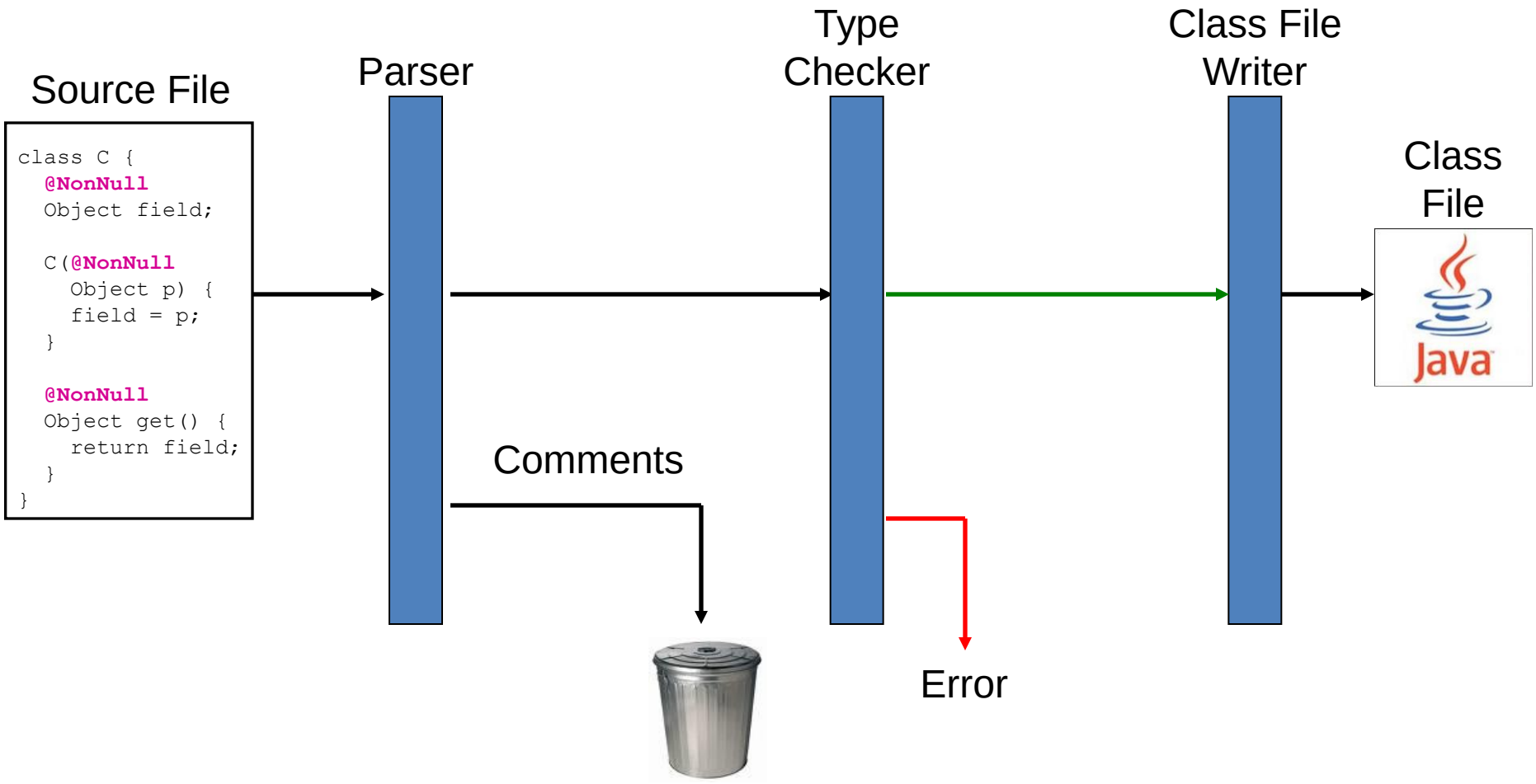
- **Example to Define an Annotation (Annotation type)**

```
public @interface MyAnnotation {
        String doSomething();
    }
```

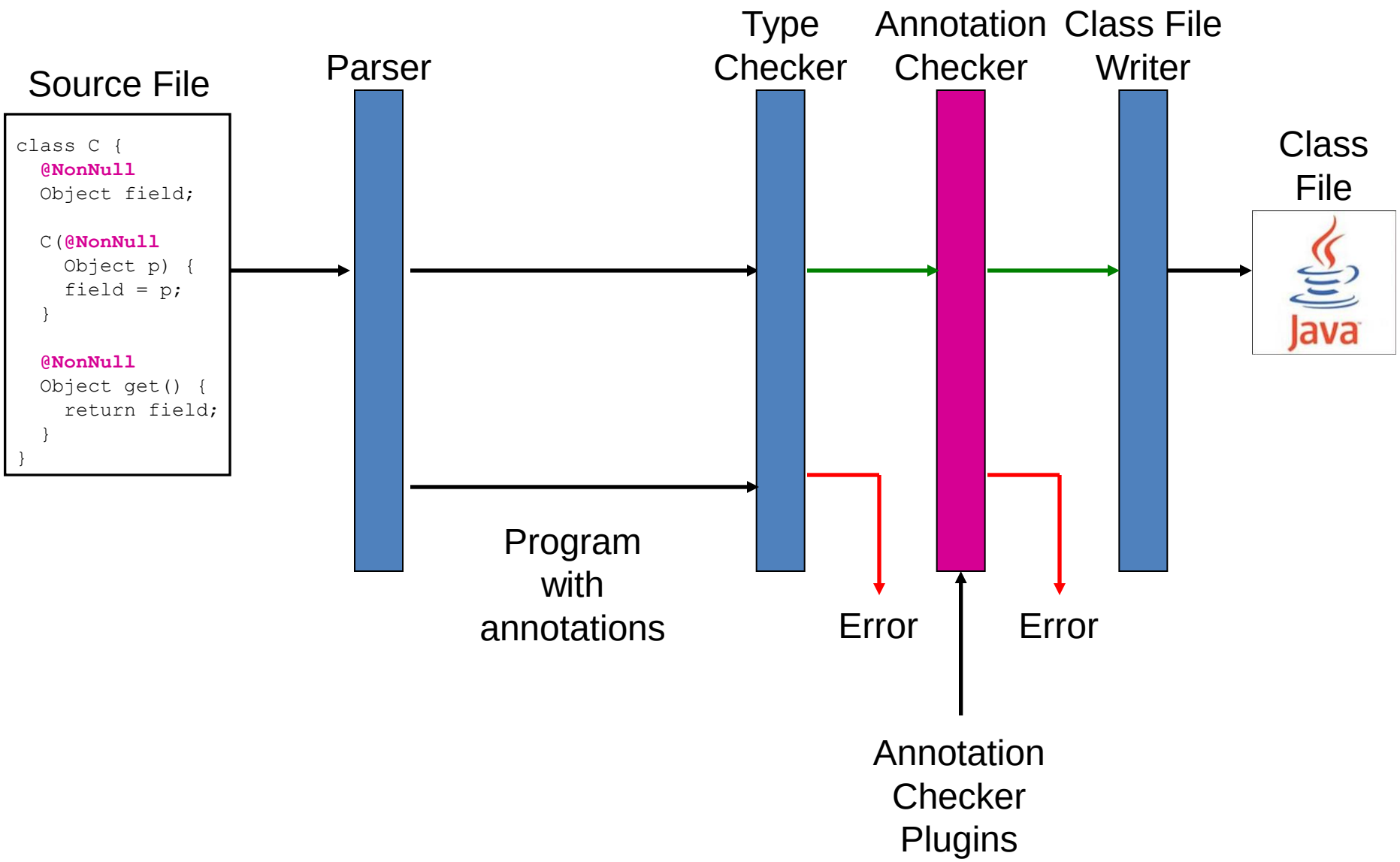
- **Example to Annotate Your Code (Annotation)**

```
@MyAnnotation (doSomething="What to do")
public void mymethod() {
        .......
    }
```

Structure of Java Compiler



Structure of Java Compiler



Annotation Types

- **Marker**
- **Single-Element**
- **Full-value or multi-value**

Marker

Marker annotations take no parameters. They are used to mark a Java element to be processed in a particular way.

- **Example:**

```
public @interface MyAnnotation {
}
```

- **Usage:**

```
@MyAnnotation
    public void mymethod() {
                ....
    }
```

Single-Element

- Single-element, or single-value type, annotations provide a single piece of data only. This can be represented with a data=value pair or, simply with the value (a shortcut syntax) only, within parenthesis.

- **Example:**

```
public @interface MyAnnotation {
        String doSomething();
 }
```

- **Usage:**

```
@MyAnnotation ("What to do")
 public void mymethod() {
        ....
 }
```

Full-value or multi-value

- Full-value type annotations have multiple data members.

- **Example:**

```
public @interface MyAnnotation {
    String doSomething();
     int count;
     String date();
 }
```

- **Usage:**

```
@MyAnnotation (doSomething=
        "What to do",
         count=1,
   date="09-09-2005")
public void mymethod() {
        ....
}
```

The Built-In Annotations

- Java defines multiple built-in annotations. For example,
- From java.lang.annotation
    - @Retention,
    - @Documented,
    - @Target,
    - and @Inherited.
- Included in java.lang.
  - @Override,
  - @Deprecated,
  - and @SuppressWarnings.

Override

- Override is a Marker Annotation type that can be applied to a method to indicate to the Compiler that the method overrides a method in a Superclass. This Annotation type guards the programmer against making a mistake when overriding a method. For eg The syntax ---@Override

- Example Program:

```
class Parent {
   public float calculate (float a, float b) {
      return a * b;
   }
}
```

Whenever you want to override a method, declare the Override annotation type before the metho d:

```
public class Child extends Parent {
   @Override
   public int calculate (int a, int b) {
      return (a + 1) * b;
   }
}
```

The Deprecated annotation

- This annotation indicates that when a deprecated program element is used, the compiler should warn you about it. Example 2 shows you the deprecated annotation.
- The syntax --- @Deprecated

- Example :

```
public class DeprecatedTest {
  @Deprecated
  public void serve() {

  }

}
```

If you use or override a deprecated method, you will get a warning at compile time.

```
public class DeprecatedTest2 {
        public static void main(String[] args) {
           DeprecatedTest test = new DeprecatedTest();
           test.serve();
        }
}
```

The Suppresswarnings annotation

- SuppressWarnings is used to suppress compiler warnings. You can apply @SuppressWarnings to types, constructors, methods, fields, parameters, and local variables.
- The syntax --- @SuppressWarnings
- **Eg:**

```
import java.util.Date;

public class Main {
   @SuppressWarnings(value={"deprecation"})
   public static void main(String[] args) {
      Date date = new Date(2009, 9, 30);

      System.out.println("date = " + date);
   }
}
```

What can be annotated?

Annotatable program elements:

- package
- class, including
  - interface
  - enum
- method
- field
- only at compile time
  - local variable
  - formal parameter