ANNOTATIONS
Annotations

- Annotation is metadata associated with the actual information.
- As humans, we use annotations all the time!
  - Highlighting or underlining passages.
- In software engineering:
  - Markup languages like XML and HTML provide annotations.
    - Makes some portions of the information syntactically distinct from others.
    - Provides desirable visual presentation.
    - Also provides semantic information that is machine-readable.
      - Has important applications in “big data” fields.
• In source control
  • Git and other version control systems use annotation functions
    • e.g., “blame”
  • Each line in the code has additional metadata
    • of the last contributor who edited that line

• In Java
  • Java is a somewhat special case
  • Annotations provide special metadata to the source code itself
  • Classes, methods, variables, parameters and packages may be annotated
• Remember these in your Java code?
  • @Override, @author

• … or in JUnit?
  • @Before, @After, @Test

• The “@” symbol is used in Java and Javadoc to mark the beginning of an annotation
  • they provide information *about the program*
WHY ARE ANNOTATIONS HELPFUL?

- Information for the compiler
  - detect errors
  - suppress warnings

- Compile-time and deployment-time processing
  - for IDEs and other tools
  - generate code, XML files, etc.

- Runtime processing
  - some annotations are used at runtime
DETAILED ANNOTATIONS

• Annotations can be more detailed than just a keyword after @
• They can have field names and data

```java
@Author(
    name = "Benjamin Franklin",
    date = "3/27/2003"
)

class MyClass() {
    @SuppressWarnings(value = "unchecked")
    void myMethod() { ... }
}
```
WHERE CAN WE USE ANNOTATIONS?

- Declarations of classes, fields, methods, etc.
- Java 8 also has type annotations
  - Class instance creation expression
    ```java
    new @Interned MyObject();
    ```
  - Type casting
    ```java
    myString = (@NonNull String) str;
    ```
  - The “implements” clause
    ```java
    class UnmodifiableList<T> implements @Readonly List<@Readonly T> { ... }
    ```
  - Thrown exception declaration:
    ```java
    void monitorTemperature() throws @Critical TemperatureException { ... }
    ```
WHY SHOULD WE USE ANNOTATIONS?

• Software tools love to use them
  • JUnit
  • Javadoc
• Web-related Tools:
  • Java Persistence API (JPA): describes the management of relational data in applications
  • Application Servers
HOW TO LOOK UP JAVA ANNOTATIONS?

• The information about annotations is quite scattered all over the official Java API
• Examples:
  • http://docs.oracle.com/javaee/8/api/javax/annotation/package-summary.html
  • http://docs.oracle.com/javaee/8/api/javax/faces/bean/package-summary.html
• Or … we can simply go look them up in various “cheat sheets” publicly available!
# AN EXAMPLE CHEAT SHEET FOR JAVA ANNOTATIONS

## Java EE 7 Annotations

### Alternatives for management

<table>
<thead>
<tr>
<th>CDI</th>
<th>JSF</th>
<th>EJB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDI: java.inject</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDI: java.enterprise.context</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@ApplicationScoped</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@SessionScoped</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@ConversationScoped</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@RequestScoped</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@Dependent</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### CDI: java.enterprise.inject

| TM |
| @Named(value="") |
| @Singleton |

### JSF management: java.faces.bean

| T |
| @ManagedBean(name="", exposeTo="") |
| @CustomScoped(value="") |

### Default injection: java.ejb

| T |
| @EJB(name="", beanInterface=interface.class, mappedName="", lookUp="") |
| @TestEJB |

### Resource injection: java.annotation

| TM |
| @Resource(name="", type=Class.class, authenticationType=AuthenticationType.CREDENTIALS, containerClass=CONTAINER, application=APPLICATION) |
| @GlobalResource("", mappedName="") |

### EJB Types: java.ejb

| T |
| @Stateless(name="Class.Name") |
| @Stateful(name="Class.Name") |
| @Singleton(name="Class.Name") |
| @LocalClass(name="Class.Name") |
| @RemoteClass(name="Class.Name") |
| @LocalBean |
| @Asynchronous |
| @Lock(type=LockType.WRITE.READ) |
| @ConcurrencyManagement(name="CONTAINER.BEAN") |
| @DependsOn(String) |

### Transaction: java.ejb

| TM |
| @TransactionManagement(name="CONTAINER.BEAN") |
| @TransactionAttribute(name="TransactionAttribute") |

### EJB Lifecycle: java.ejb

| H |
| @PostConstruct |
| @PreDestroy |

### Interceptors: java.interceptor

| T |
| @Interceptor(name="") |

### Security: java.annotation.security

| T |
| @RunAs(String) |
| @RolesAllowed(String) |
| @PermitAll |

## Java EE 7 Annotations Cheat Sheet

**Version 1.2 © 2005, 2011 Phillip Maier**

**Version 1.5 ©2013-06-27** by Chris Remmick, based on Java EE 7 API Doc: EJB 3.2, JSF 2.2, JPA 2.1

---

CSE 219, Stony Brook University
@Deprecated

• indicates that the marked element is deprecated and should no longer be used

• The compiler generates a warning whenever a program uses a method, class, or field with the @Deprecated annotation.

```
// Javadoc comment
/**
 * @deprecated
 * explanation of why it was deprecated
 */
@Deprecated
static void deprecatedMethod() {
    ... }
```
PREDEFINED ANNOTATIONS IN JAVA

@Override

• informs the compiler that the element is meant to override an element declared in a superclass

    // mark method as a superclass method
    // that has been overridden

    @Override
    int overriddenMethod() { ... }
@SuppressWarnings

- tells the compiler to suppress specific warnings that it would otherwise generate

```java
// use a deprecated method and tell
// compiler not to generate a warning
@ SuppressWarnings("deprecation")
void useDeprecatedMethod() {
    // deprecation warning suppressed
    objectOne.deprecatedMethod();
}
```
• Define the annotation type:

```java
@interface ClassPreamble {
    String author();
    String date();
    int currentRevision() default 1;
    String lastModified() default "N/A";
    String lastModifiedBy() default "N/A";
    // Note use of array
    String[] reviewers();
}
```
And then you can start using it:

```java
@ClassPreamble (  
    author = "John Doe",
    date = "3/17/2002",
    currentRevision = 6,
    lastModified = "4/12/2004",
    lastModifiedBy = "Jane Doe",
    // Note array notation
    reviewers = {"Alice", "Bob", "Cindy"}
)

public class Generation3List extends List2 {
    // class code goes here
}
```
• To make the information in `@ClassPreamble` appear in Javadoc-generated documentation, when you define the annotation:

```java
// import this to use @Documented
import java.lang.annotation.*;

@Documented
@interface ClassPreamble {
    // Annotation element definitions
}
```