Annotations in Java

A Java *annotation* provides information about a program but is not part of the program itself. An annotation is a form of "syntactic metadata" about a program. Here are three annotations that are built into Java:

Three basic annotations

- @Override Place this before a method. If the method does *not* override an inherited method, a compile-time error occurs. We always place this annotation on an overriding method to check for typos or other mistakes, as in writing the function name as tostring instead of toString.
- @Deprecated To *deprecate* means to make obsolete, to disapprove of, to depreciate. When deprecating a method or class, the Java maintainers are telling you that it is better not to use that item, probably because it does not perform as one wants and a better alternative exists. You can continue to use the deprecated item, but beware. The compiler issues a warning if a deprecated item is used. The warning can be ignored.
- @SuppressWarnings You can put this on a method that gives a compile-time warning to have it suppressed. There are cases, particularly with generics, where warnings occur and one knows there is no problem. So you can suppress those warnings.

Other built-in annotations

Below are 7 other built-in annotations. Discussion of them is outside the scope of JavaHyperText. You can find out more of them here: https://en.wikipedia.org/wiki/Java_annotation.

@Retention	@Documented	@Target	@Inherited
@SafeVarargs	@FunctionalInterface	@Repeatable	

More on annotations

The three basic annotations illustrate a main reason for annotations: to help the compiler give you (or hide) information about your program. Using @Override can help uncover mistakes at the earliest possible moment. @Deprecated helps you see that perhaps it's better to look for another alternative. Use @SuppressWarnings to remove unwanted warnings the compiler gives you.

In its simplest form, an annotation is written as the at-sign @ followed by an identifier, as in @Override. Annotations can have parameters. Annotations come before the entity they annotate —a class, method, variable declaration, parameter declaration, or package. We see them mostly on methods. There are ways to have annotations embedded into the .class files that are generated from a program and then have them processed by the program. That's beyond the scope of JavaHyperText.

Good examples of annotations are those developed for JUnit 4 testing —find them under the JavaHyperText entries "Annotation" and "JUnit testing".

Custom annotations

You can declare your own annotations! Few individuals do. Custom annotations are meant more for large groups to declare, like the team that developed JUnit 4.

To the right is a declaration of an annotation called Author —it is like an interface declaration but the word *interface* is preceded by @. Here is a use of this annotation:

```
public @interface Author {
    String first();
    String last();
}
```

```
@Author(first= "John", last= "Doe")
Book book= new Book();
```

Pop the Author declaration into Eclipse or DrJava, add a declaration of class Book, place the annotated declaration somewhere, and you'll see that it compiles.

Of course, we haven't told you how to process this annotation so something useful happens. That's beyond the scope of JavaHyperText.