1. Java Annotations (<http://tutorials.jenkov.com/java/annotations.html>)

Java annotations are used to provide meta data for your Java code. Being meta data, Java annotations do not directly affect the execution of your code, although some types of annotations can actually be used for that purpose.

Java annotations were added to Java from Java 5. This text covers Java annotations as they look in Java 6. As far as I know, Java annotations have not changed in Java 7, so this text should be valid for Java 7 programmers too.

## Java Annotation Purposes

Java annotations are typically used for the following purposes:

* Compiler instructions
* Build-time instructions
* Runtime instructions

Java has 3 built-in annotations that you can use to give instructions to the Java compiler. These annotations are explained in more detail later in this text.

Java annotations can be be used at build-time, when you build your software project. The build process includes generating source code, compiling the source, generating XML files (e.g. deployment descriptors), packaging the compiled code and files into a JAR file etc. Building the software is typically done by an automatic build tool like Apache Ant or Apache Maven. Build tools may scan your Java code for specific annotations and generate source code or other files based on these annotations.

Normally, Java annotations are not present in your Java code after compilation. It is possible, however, to define your own annotations that are available at runtime. These annotations can then be accessed via[**Java Reflection**](http://tutorials.jenkov.com/java-reflection/index.html), and used to give instructions to your program, or some third party API.

1. Java Reflection (http://tutorials.jenkov.com/java-reflection/index.html)

Java Reflection makes it possible to inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. at compile time. It is also possible to instantiate new objects, invoke methods and get/set field values using reflection.

Java Reflection is quite powerful and can be very useful. For instance, when mapping objects to tables in a database at runtime, like [Butterfly Persistence](http://butterfly.jenkov.com/) does. Or, when mapping the statements in a script language to method calls on real objects at runtime, like [Butterfly Container](http://butterfly.jenkov.com/) does when parsing its configuration scripts.

There are already numerous Java Reflection Tutorials on the internet. However, most of them, including Sun's own Java Reflection tutorial, only scratch the surface of Java Reflection and its possibilities.

This tutorial will get into Java reflection in more depth than most of the tutorials I have seen. It will explain the basics of Java Reflection including how to work with arrays, annotations, generics and dynamic proxies, and do dynamic class loading and reloading. It will also show you how to do more specific tasks, like reading all getter methods of a class, or accessing private fields and methods of a class. This tutorial will also clear up some of the confusion out there about what Generics information is available at runtime. Some people claim that all Generics information is lost at runtime. This is not true.

This tutorial describes the version of Java Reflection found in Java 6.

https://www.youtube.com/watch?v=sUED3FDmQNk&list=PLzS3AYzXBoj\_JOP9UU2HYM\_a8xVuLub7K&index=1

1. Why do we need Reflection.

* We can load class from property file using reflection Class.forname(“nameofClass”). No need to compile any code and runtime we can load any class.
* This can be used to inspect class at runtime. Like methods field etc.







