**What is reflection?**

Reflection is like a class manipulator that makes it possible inspect the behavior of 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, Java Reflection can be used to map properties in JSON files to getter / setter methods in Java objects, like [Jackson, GSON, Boon etc.](http://tutorials.jenkov.com/java-json/index.html) does. Or, Reflection can be used to map the column names of a [JDBC](http://tutorials.jenkov.com/jdbc/index.html) ResultSet to getter / setter methods in a Java object.

This tutorial will get into Java reflection in depth. 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 Java Reflection tasks, like reading all getter methods of a class, or accessing private fields and methods of a class.

This Java Reflection 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 8.

**Where reflection is used?**

* If you want to get the definition of a protected or final member, remove the protection and manipulate it as if it had been declared mutable! Reflection changes the definition of OOPs and its normal operation like defy the definition of Encapsulation. It makes the program very, very dangerous.
* Using reflection to find out what super types a class has, what annotations it has, what members it has, to invoke accessible methods and constructors, read and update accessible fields
* Used DI (Dependency Injection), Basically Eclipse Intellisense is using reflection, Debuging. And this pretty much explains when you would use it.

**Is it good to use reflection in an application?**

* One useful real-world use of reflection is when writing a framework that has to interoperate with user-defined classes, where the framework author doesn't know what the members (or even the classes) will be. Reflection allows them to deal with any class without knowing it in advance. For instance, I don't think it would be possible to write a complex aspect-oriented library without reflection.
* As another example, JUnit used to use a trivial bit of reflection: it enumerates all methods in your class, assumes that all those called testXXX are test methods, and executes only those. But this can now be done better with annotations instead, and in fact JUnit 4 has largely moved to annotations instead.
* Not a good idea because it can take some private field. Security problem

**Why reflection is slower?**

Reflection is much slower than just calling methods by their name, because it has to inspect the metadata in the bytecode instead of just using precompiled addresses and constants.

 The compiler can do no optimization whatsoever as it can have no real idea about what you are doing. This probably goes for the JIT as well

 Everything being invoked/created has to be *discovered* (i.e. classes looked up by name, methods looked at for matches etc)

 Arguments need to be dressed up via boxing/unboxing, packing into arrays, Exceptions wrapped in InvocationTargetExceptions and re-thrown etc.

**What is java.lang.Class class and what is the role of that class? How to get an object of Class class?**

Java provides a class with name **Class** in java.lang package. Instances of the class Class represent classes and interfaces in a running Java application. The primitive Java types (boolean, byte, char, short, int, long, float, and double), and the keyword void are also represented as Class objects. It has no public constructor. Class objects are constructed automatically by the Java Virtual Machine([JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)). It is a final class, so we cannot extend it.

Get the access to the current class. This is the entry point without the entry point of the Class class you cannot do anything.

Simply loads a class, including running its static initializers. It has no public constructor. Class objects are constructed automatically by the Java Virtual Machine([JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)). It is a final class, so we cannot extend it.

**List out some commonly used methods of Class class?**

getClassLoader(), getName(), .**forName**

**How to determine the type of Class class object?**

Three way

1. When you call the Class.forName and the name of the parameter. It creates Class object for the class passed as a String argument(className). Since class Class doesn’t contain any constructor, there is static **factory** method present in class Class, which is Class.forName() , used for creating object of class Class.
2. .class syntax When we write .class after a class name, it references the Class object that represents the given class. It is mostly used with primitive data types and only when we know the name of class. The class name for which Class object is to be created is determined at compile-time.
3. .getClass() method This method is present in [Object](https://www.geeksforgeeks.org/object-class-in-java/) class. It return the run-time class of this(obj) object.

**Explain the use of Class.forName()?**

Loads the class dynamically (runtime). Get the class object from string input.

**How to create an object of a class without knowing the name of the class at compile time?**

**Marker interface?**

**Serializable, Cloneable**

It is an empty interface (no field or methods). Examples of marker interface are Serializable, Clonnable and Remote interface. All these interfaces are empty interfaces.

**Different ways of creating an Object in java?**

There are four different ways to create objects in java:

**A**. Using new keyword  
This is the most common way to create an object in java. Almost 99% of objects are created in this way.

MyObject object = new MyObject();

**B**. Using Class.forName()  
If we know the name of the class & if it has a public default constructor we can create an object in this way.

MyObject object = (MyObject) Class.forName("subin.rnd.MyObject").newInstance();

**C**. Using clone()  
The clone() can be used to create a copy of an existing object.

MyObject anotherObject = new MyObject();

MyObject object = (MyObject) anotherObject.clone();

**D**. Using object deserialization  
Object deserialization is nothing but creating an object from its serialized form.

ObjectInputStream inStream = new ObjectInputStream(anInputStream );

MyObject object = (MyObject) inStream.readObject();

Using constructor, using reflection, Clonning, Deserialization (Ananymous class?)

**Difference between classNotFoundException and no NoClassDefFoundError?**

**ClassNotFoundException** is an exception that occurs when you try to load a class at run time using Class.forName() or loadClass() methods and mentioned classes are not found in the classpath. Class is not required to be presented in compile time and runtime. Most of the time this exception will occur when you try to run application without updating classpath with JAR files. This exception is a **checked Exception** derived from **java.lang.Exception** class and you need to provide **explicit handling** for it. This exception also occurs when you have two class loaders and if a ClassLoader tries to access a class which is loaded by another classloader in Java. You must be wondering that what actually is classloader in Java. **Java ClassLoader** is a part of Java Runtime Environment that dynamically loads Java classes in JVM(Java Virtual Machine). The Java Runtime System does not need to know about files and files system because of classloaders.

Thrown when an application tries to load in a class through its string name using:

* The forName method in class Class.
* The findSystemClass method in class ClassLoader .
* The loadClass method in class ClassLoader.

but no definition for the class with the specified name could be found.

This exception has been retrofitted to conform to the general purpose exception-chaining mechanism. The "optional exception that was raised while loading the class" that may be provided at construction time and accessed via the [getException()](https://docs.oracle.com/javase/7/docs/api/java/lang/ClassNotFoundException.html" \l "getException()) method is now known as the *cause*, and may be accessed via the [Throwable.getCause()](https://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html" \l "getCause()) method, as well as the aforementioned "legacy method."

**NoClassDefFoundError** is an error that occurs when a particular class is present at compile time, but was missing at run time.

It is error which is derived from **LinkageError**. Linkage error occurs when a class has some dependencies on another class and latter class changes after compilation of former class. NoClassFoundError is the result of **implicit loading** of class because of calling a method or accessing a variable from that class. This error is more difficult to debug and find the reason why this error occurred. So in this case you should always check the classes which are dependent on this class.

**ClassNotFoundException** and **NoClassDefFoundError** occur when a particular class is not found at runtime. However, they occur at different scenarios.

In the case of **ClassNotFoundException** when theclassis not found in runtime then the exeption is thrown. The class is gonna be search in

**NoClassDefFoundError** Is thrown when the class is present in compile time but due for some reason is class not found at runtime.

**What is Singleton class?**

Singleton class is when making a single object of the class the way you can achieve this is. This is used of private constructor to initialize Singleton class.

**How to break singleton Dp?**

# There are mainly 3 concepts which can break singleton property of a class from Reflection, Serialization and Cloning?

# Reflection by instantiation of the singleton class breaks the singleton pattern design

# Suppose you serialize an object of a singleton class. Then if you de-serialize that object it will create a new instance and hence break the singleton pattern.

# By creating a clone of a singleton object, then it will create a copy that is there are two instances of a singleton class breaking the singleton pattern design.