Reflection

- Definition and use-cases
- Drawbacks
- ReflectionHelper

Definition

Reflection is not a class, an object, a library or a framework. Reflection code is the code which is able to inspect other code in the same system (or itself).

For example for serialization/deserialization.

Very common use case is the usage with annotations.

For example:

- JUnit 4 uses reflection to look through classes for methods tagged with the @Test annotation, and will then call them when running the unit test,
- Hibernate looks for ORM annotations
- Web frameworks look for @WebServlet annotation

Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine.

Use cases

With help of reflection one can:

- get list of constructors of a class,
- get list of methods of a class,
- get list of fields of a class,
- instantiate a class,
- call a method,
- get and set value of a field,
- get list of annotations,
- get list of superclasses and interfaces.

Class<?> can be obtained from an object of by name.

Drawbacks

Reflection is powerful, but should not be used indiscriminately.

If it is possible to perform an operation without using reflection, then it is preferable to avoid using it.

Performance Overhead

Because reflection involves types that are dynamically resolved, certain Java virtual machine optimizations can not be performed. Consequently, reflective operations have slower performance than their non-reflective counterparts, and should be avoided in sections of code which are called frequently in performance-sensitive applications.

Security Restrictions

Reflection requires a runtime permission which may not be present when running under a security manager. This is in an important consideration for code which has to run in a restricted security context, such as in an Applet.

Exposure of Internals

Since reflection allows code to perform operations that would be illegal in non-reflective code, such as accessing private fields and methods, the use of reflection can result in unexpected side-effects, which may render code dysfunctional and may destroy portability. Reflective code breaks abstractions and therefore may change behavior with upgrades of the platform.