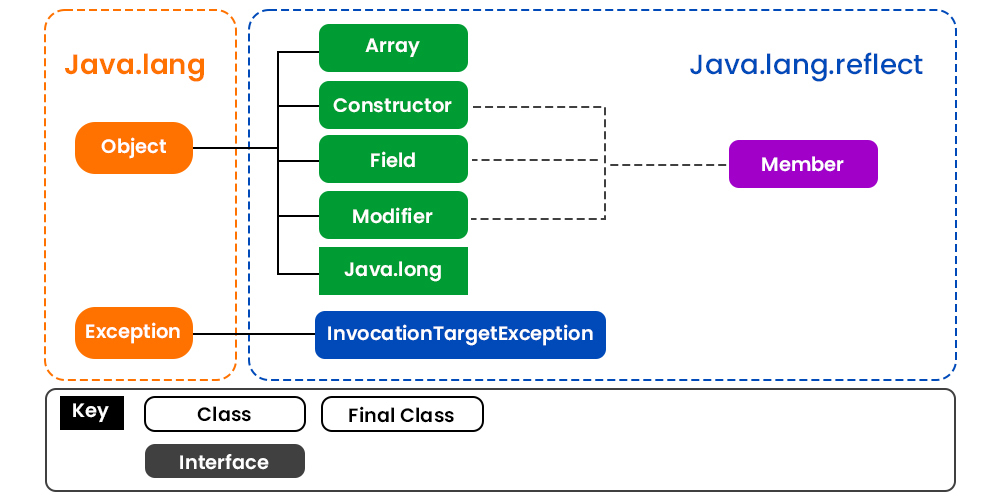
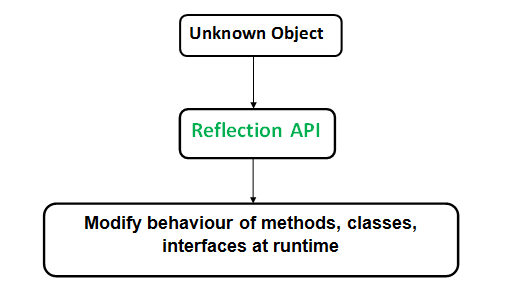
**Reflection in Java**

Reflection is an API that is used to examine or modify the behavior of methods, classes, and interfaces at runtime. The required classes for reflection are provided under **java.lang.reflect** package which is essential in order to understand reflection. So we are illustrating the package with visual aids to have a better understanding as follows:



* Reflection gives us information about the class to which an object belongs and also the methods of that class that can be executed by using the object.
* Through reflection, we can invoke methods at runtime irrespective of the access specifier used with them.



Reflection can be used to get information about class, constructors, and methods as depicted below in tabular format as shown:

|  |  |
| --- | --- |
| **Class** | The getClass() method is used to get the name of the class to which an object belongs. |
| **Constructors** | The getConstructors() method is used to get the public constructors of the class to which an object belongs. |
| **Methods** | The getMethods() method is used to get the public methods of the class to which an object belongs. |

We can invoke a method through reflection if we know its name and parameter types. We use two methods for this purpose as described below before moving ahead as follows:

1. getDeclaredMethod()
2. invoke()

**Method 1:** getDeclaredMethod():Itcreates an object of the method to be invoked.

**Syntax:** The syntax for this method

Class.getDeclaredMethod(name, parametertype)

**Parameters:**

* Name of a method whose object is to be created
* An array of Class objects

**Method 2:**invoke(): It invokes a method of the class at runtime we use the following method.

**Syntax:**

Method.invoke(Object, parameter)

***Tip:****If the method of the class doesn’t accept any parameter then null is passed as an argument.*

***Note:****Through reflection, we can access the private variables and methods of a class with the help of its class object and invoke the method by using the object as discussed above. We use below two methods for this purpose.*

**Method 3:** Class.getDeclaredField(FieldName): Used to get the private field. Returns an object of type Field for the specified field name.

**Method 4:**Field.setAccessible(true):Allows to access the field irrespective of the access modifier used with the field.

**Important observations Drawn From Reflection API**

* **Extensibility Features:** An application may make use of external, user-defined classes by creating instances of extensibility objects using their fully-qualified names.
* **Debugging and testing tools**: Debuggers use the property of reflection to examine private members of classes.
* **Performance Overhead:**Reflective operations have slower performance than their non-reflective counterparts, and should be avoided in sections of code that are called frequently in performance-sensitive applications.
* **Exposure of Internals:**Reflective code breaks abstractions and therefore may change behavior with upgrades of the platform.

**Annotations in Java**

The Annotations in Java are a form of the metadata that can be added to Java code to provide information about the code to compiler and runtime environment.

* **Built-in Annotations**: These are the annotations provided by Java, such as, **@Override**, **@Deprecated** and **@SuppressWarnings**.
* **Custom Annotations**: To Creating custom annotations to add metadata to our own code.
* **Annotation Processors**: The Processing annotations at compile time to generate additional code or perform specific tasks.

**Use Cases:**

* **Code Organization:** Adding information about code’s intended use, relationships or constraints.
* **Frameworks and Libraries**: The Many Java frameworks use annotations for the configuration and behavior customization.
* **Documentation**: The Generating documentation or providing hints to the tools and IDEs.

Ex:

@Component(value=”emp”)

public class Employee{

}

How to create custom annotations:

===========================

We have to use some set of predefined annotations like

@Target

@Retention

Annotations are written using @interface keyword.

public @interface CustomAnnotation{

String name() default “king is my name”;

}

Note: Annotations can be created and used for different levels.

Class level , method level , field level

@CustomAnnotation // class level

Ex: public class Demo {

@CustomAnnotation

int eid;

@CustomAnnotation(name=”javeed”) // method level

public void m1(@CustomAnnotation String name){

}

}