Reflection API

- Introduction
- Class
- Method
- Field
- Constructor

Reflection:

The process of analyzing all the capabilities of a particular class at runtime is called as "Reflection".

Reflection API is a set of predefined classes provided by Java to perform reflection over a particular class.
Reflection API is not useful in projects development, reflection API is useful in products development like Compilers, JVM's, Server's, IDE's, FrameWork's.....
Java has provided the complete predefined library for Reflection API in the form of "java.lang" package and "java.lang.reflect" package.

Java.lang
Class
java.lang.reflect
Field
Method
Constructor
Package
Modifier

---

java.lang.Class:

This class can be used to represent a particular class metadata which includes class name, super class details, implemented interfaces details, access modifiers.....
To get the above class metadata, first we have to create Class Object for the respective class.
There are three ways to create object for java.lang.Class

- ## Using forName(--) Method:

  forName() method is defined in java.lang.Class to load a particular class byte code to the memory and to get metadata of the respective loaded class in the form of Class object.
  public static Class forName(String class_Name) throws ClassNotFoundException
  class c = Class.forName("Employee");

When JVM encounter the above instruction, JVM will perform the following actions.

- JVM will take provided class from forName() method.
- JVM will search for its .class file at current location, at java predefined library and at the locations reoffered by "classpath" environment variable.
- If the required .class file is not available at all the above locations then JVM will rise an exception like "java.lang.ClassNotFoundException".
- If the required .class file is available at either of the above locations then JVM will load its byte code to the memory.
- After loading its byte code to the memory, JVM will collect the respective class metadata and stored in the form of java.lang.Class object.

• Using getClass() Method:

- In Java applications, if we create object for any class then JVM will load the respective class byte code to the memory, JVM will get metadata of the respective class and stored in the form of java.lang.Class object.
- In the above context, to get the generated Class object, we have to use the following method from java.lang.Object class.
  public Class getClass()

  EX:
  Employee e = new Employee();
  Class c = e.getClass();

• Using .class File Name:

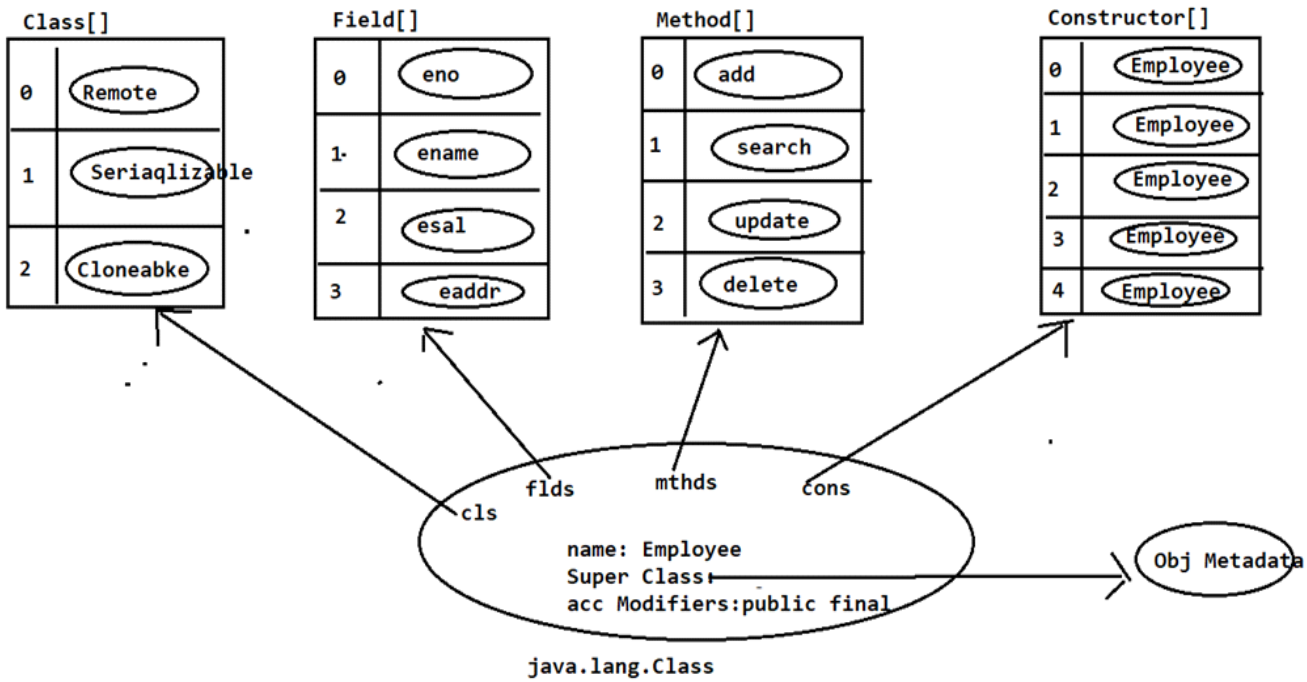
- In Java, every .class file is representing a java.lang.Class Object, it will manage the metadata of the respective class.
  EX: Class c = Employee.class

- To get name of the class, we have to use the following method.
  public String getName()

- To get super class metadata in the form of Class object, we have to use the following method.
  public Class getSuperclass()

- To get implemented interfaces metadata in the form of Class[] we have to use the following method.
  public Class[] getInterfaces()

- To get the specified access modifiers list, we have to use the following method.
  public int getModifiers()



EX:

```
1)  int val = c.getModifiers()
2)  String modifiers = Modifier.toString(val);
3)  import java.lang.reflect.*;
4)  public abstract class Employee implements java.io.Serializable,java.lang.Cloneable
    {
5)  }
6)  class Test {
7)      public static void main(String args[]) throws Exception {
8)
9)          Class c1 = Clas.forName("Employee");
10)         System.out.println(c1.getName());
11)         Employee e = new Employee();
12)
13)         Class c2 = e.getClass();
14)         System.out.println(c2.getName());
15)
16)         Class c3 = Employee.class;
17)         System.out.println(c3.getName());
18)
19)         Class c = Class.forName("Employee");
20)         System.out.println("Class Name  :"+c.getName());
21)         System.out.println("Super Class :"+c.getSuperclass().getName());
22)         Class[] cls = c.getInterfaces();
23)         System.out.println("Interfaces  :");
24)
```

```
25)             for(int i=0; i<cls.length; i++) {
26)                     Class cl = cls[i];
27)                     System.out.println(cl.getName()+" ");
28)             }
29)             System.out.println();
30)             int val = c.getModifiers();
31)             System.out.println("Modifers :"+Modifier.toString(val));
32)     }
33) }
```

java.lang.reflect.Field:

- This class can be used to get the metadata of a particular variable which contains the name of the variable, data type of the variable, access modifiers of the variable and value of the variable.
- If we want to get all the variables metadata of a particular class first we have to get java.lang.Class object. After getting Class object, we have to get all the variables metadata in the form Field[].

- To get all the variables metadata in the form of Field[], we have to use the following two methods.

- **public Field[] getFields()**
  This method will return all the variables metadata which are declared as public in the respective class and in the super class.

- **public Field[] getDeclaredFields()**
  This method will return all the variables metadata which are available in the respective class only irrespective of public declaration.

- To get name of the variable, we have to use the following method.
  public String getName()

- To get datatype of the variables, we have to use the following method.
  public Class getType()

- To get value of the variable, we have to use the following method.
  public Object get(Field f)

- To get access modifiers of a variable, we have to use the following method.
  public int getModifiers()
  public static String toString(int val)

```
1)  import java.lang.reflect.*;
2)  class Employee {
3)        public static String eid = "E-111";
4)        public static String ename = "Durga";
5)        public static String eaddr = "Hyd";
6)  }
7)  class Test {
8)       public static void main(String args[]) throws Exception {
9)              Class c = Employee.class;
10)             Field[] flds = c.getDeclaredFields();
11)             for(int i=0; i<flds.length; i++) {
12)                   Field f = flds[i];
13)                   System.out.println("Field Name:"+f.getName());
14)                   System.out.println("data Type:"+f.getType().getName());
15)                   System.out.println("value:"+f.get(f));
16)                   int val = f.getModifiers();
17)                   System.out.println("Modifiers :"+Modifier.toString(val));
18)                   System.out.println("-------------------------------------");
19)             }
20)       }
21) }
```

java.lang.Method:

- This class can be used to represent the metadata of a particular method, which includes method name, return type, parameter types, access modifiers and exception types

- If we want to get all the methods metadata in the form of method[], first we have to get java.lang.Class object then we have to use either of the following methods.
  public Method[] getMethods()
- It can be used to get all the methods metadata which are declared as public in the respective class and in the super class.

  public Method[] getDecalredMethods()
  It can be used to get all the methods metadata which are available in the respective class irrespective of public declaration.

- To get method name, we have to use the following method.
  public String getName()

- To get method return type, we have to use the following method.
  public Class getReturnType()

- To get parameter data types, we have to use the following method.
  public Class[] getParameterTypes()

- To get Exception types which are specified along with "throws" keyword then we have to use the following method.
  public Class[] getExceptionTypes()
- To get Access modifier of a particular class, we have to use the following method.
  public int getModifers()
  public static String toString(int value)

```
1)  import java.lang.reflect.*;
2)  class Employee {
3)        public void create(String eid, String ename, String eaddr) throws
4)                                        ClassNotFoundException, NullPointerException {

5)        }
6)        public void search(String eid)throws ClassCastException {}
7)        public void delete(String eid)throws ClassCastException,
8)                                                        ClassNotFoundException {}
9)  }
10) class Test {
11)       public static void main(String args[]) throws Exception {
12)               Class c = Employee.class;
13)               Method[] mthds = c.getDeclaredMethods();
14)
15)               for(int i=0; i<mthds.length; i++) {
16)                       Method m = mthds[i];
17)                       System.out.println("Method Name   :"+m.getName());
18)                       System.out.println("Return Tyep :"+m.getreturnType());
19)                       int val = m.getModifiers();
20)                       System.out.println("Modifiers :"+modifier.toString(val));
21)                       Class[] cls = m.getParameterTypes();
22)                       System.out.println("Parametes   :");
23)
24)                       for(int j=0; j<cls.length; j++) {
25)                               Class cl = cls[j];
26)                               System.out.println(cl.getName()+"  ");
27)                       }
28)
29)                       System.out.println();
30)                       Class[] cls1 = m.getExceptionTypes();
31)                       System.out.println("Exception Types   :");
32)
33)                       for(int j=0; j<cls1.length; j++) {
34)                               Class cl1 = cls1[j];
35)                               System.out.println(cl1.getName()+"  ");
36)                       }
37)
```

```
38)                     System.out.println();
39)                     System.out.println("----------------------------");
40)             }
41)         }
42) }
```

java.lang.reflect.Constructor:

This class can be used to get metadata of a particular constructor.

To get all the constructors metadata of a particular class first we have to get Class object then we have to use the following methods.
public Constructor[] getConstructors()

It will return only public constructors details from the respective class.
public Constructor[] getDeclaredConstrctors()

It will return all the constructors metadata of the respective class irrespective of their public declaration.

Constructor class has provided the following methods to get constructor data.

1) **public String getName()**
2) **public Class[] getParameterTypes()**
3) **public Class[] getExceptionTypes()**
4) **public int getModifiers()**
5) **public static String toString(int val)**

Program is same as java.lang.reflect.Method program with the above methods

```
1)  import java.lang.reflect.*;
2)  class Employee {
3)      public  Employee(String eid, String ename, String eaddr) throws
4)                                      ClassNotFoundException, NullPointerException {
5)      }
6)      public Employee(String eid, String ename) throws ClassCastException {}
7)      public Employee(String eid) throws ClassCastException,
8)                                                      ClassNotFoundException {}
9)  }
10) class Test {
11)     public static void main(String args[]) throws Exception {
12)             Class c = Employee.class;
13)             Constructor[] con = c.getDeclaredConstructors();
14)             for(int i=0; i<con.length; i++) {
15)                 Constructor constructor = con[i];
16)                 System.out.println("Constructor Name:"constructor.getName());
17)                 int val = constructor.getModifiers();
```

```
18)                 System.out.println("Modifiers:"+Modifier.toString(val));
19)                 Class[] cls = constructor.getParameterTypes();
20)                 System.out.println("Parametes   :");
21)                 for(int j=0; j<cls.length; j++) {
22)                         Class cl = cls[j];
23)                         System.out.println(cl.getName()+"  ");
24)             }
25)             System.out.println();
26)             Class[] cls1 = constructor.getExceptionTypes();
27)             System.out.println("Exception Types   :");
28)             for(int j=0; j<cls1.length; j++) {
29)                 Class cl1 = cls1[j];
30)                 System.out.println(cl1.getName()+"  ");
31)             }
32)             System.out.println();
33)             System.out.println("----------------------------");
34)         }
35)     }
36) }
```