Reflection API
—---------------
The process of analyzing all the capabilities of a particular class at runtime is called “Reflection”.

Here, analyzing all the capabilities of a particular class means that reading the details of a particular class like Class Name, access modifiers of the class, superclass details, implemented interfaces details, variables details, methods details, constructors details which we declare inside the class at runtime of our application.

To perform Reflection in Java applications , Java has provided a set of predefined classes called “Reflection API”.

Note: IN general, Reflection API is not very useful in Application Development, it is very much useful in Product Development like Compilers, JVMs, Servers, Frameworks like Spring, Hibernate,.....

Java has provided the following classes as part of the Reflection API in order to read all the details of the classes.

java.lang.Class
java.lang.reflect.Field
java.lang.reflect.Constructor
java.lang.reflect.Method
java.lang.reflect.Modifier
—--
—--

java.lang.Class:
—----------------
java.lang.Class object is able to provide the metadata of a particular class , where the metadata of a particular class includes

1. Class Name
2. Access Modifiers
3. Super class metadata
4. Implemented Interfaces Metadata
5. Variables metadata
6. Methods metadata
7. Constructors Metadata
   —------
   —-------

In Java applications, if we want to provide all the above details , first we have to create a java.lang.Class object for the respective class.

To create java.lang.Class objects in java applications we have to use the following three approaches.

1. By Using Class.forName() method
2. By Using getClass() method from java.lang.Object class.
3. By Using .class file name

1. By Using Class.forName() method:
—----------------------------------
To create java.lang.Class objects in java applications we have to use the following method from java.lang.Class class.

public static Class forName(String className)throws ClassNotFoundException
EX: Class cls = Class.forName("com.durgasoft.core.Employee");

When we execute the above instruction , JVM will perform the following actions.

1. JVM will take the provided class name from forName() method.
2. JVM will search for the respective class .class file [Employee.class] at the Current Location, at the Java predefined Library and at the locations referred by the "classpath" environment variable.
3. If the required .class file is not identified at all the above locations then JVM will raise an exception like java.lang.ClassNotFoundException.
4. If the required .class file is identified at either of the above locations then JVM will load its bytecode to the memory[Method Area].
5. After loading the class bytecode to the memory , JVM will create a java.lang.Class object in Heap Memory with the metadata of the loaded class.
6. In the above context, JVM will return the generated java.lang.Class object reference value from forName() method.

EX:
```
class Employee{

}
public class Main {
```

```
    public static void main(String[] args)throws Exception
{
        Class cls = Class.forName("Employee");
        System.out.println(cls.getName());
    }
}
```

2. By Using getClass() method from java.lang.Object class.

IN Java applications, when we create a particular class object , first JVM must load the respective class bytecode to the memory, automatically, JVM will create java.lang.Class object in the Heap memory.

IN the above context, to get the generated java.lang.Class object we have to use the following method from java.lang.Object .

public Class getClass()

EX:

```
class Employee{

}
public class Main {
    public static void main(String[] args)throws Exception
{
        Employee employee = new Employee();
        Class cls = employee.getClass();
        System.out.println(cls.getName());
    }
}
```

3. By Using .class file name

In Java applications, every .class file is representing a java.lang.Class object of the respective class.

EX: Class cls = Employee.class;

When we create an object for a particular class, automatically the Compiler will add a static variable to the respective class that is “class” , here the added class variable is representing the generated java.lang.Class object.

EX:

```
class Employee{

}
public class Main {
   public static void main(String[] args)throws Exception
{
        Class cls = Employee.class;
        System.out.println(cls.getName());
   }
}
```

If we want to get the name of the class we have to use the following method.

public String getName()

To get Super class details in the form of java.lang.Class object we have to use the following method.

Public Class getSuperclass()

To get the Access Modifiers list we have to use the following method.

public int getModifiers()

If we want to get the access modifiers names then we have to use the following method from java.lang.reflect.Modifier class.

public static String toString(int val)

```
import java.lang.reflect.Modifier;

public class Main {
   public static void main(String[] args)throws Exception
{
```

```
        Class cls =
Class.forName("com.durgasoft.entities.Employee");
        System.out.println("Name         :
"+cls.getName());
        System.out.println("Super class :
"+cls.getSuperclass().getName());
        System.out.println("Access Modifiers : "+
Modifier.toString(cls.getModifiers()));
    }
}
```

Consider the following program.

Employee.java

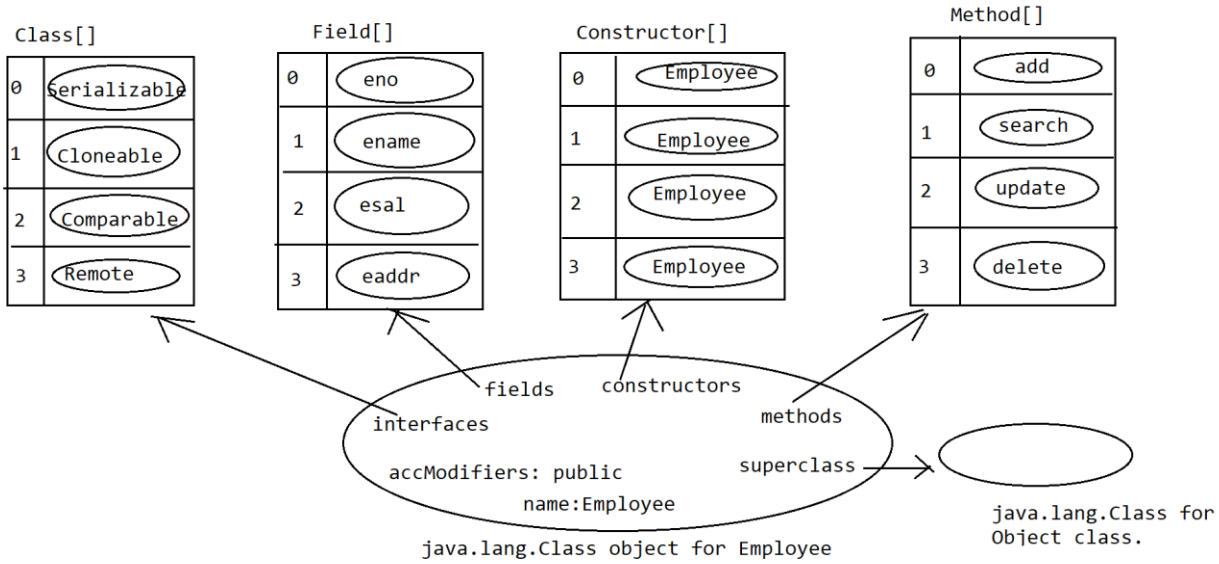
```
public class Employee implements java.io.Serializable, java.lang.Cloneable,
java.lang.Comparable, java.rmi.Remote{

      int eno;
      String ename;
      float esal;
      String eaddr;

      public Employee(int eno){
      }
      public Employee(int eno, String ename){
      }
      public Employee(int eno, String ename, float esal){
      }
      public Employee(int eno, String ename, float esal, String eaddr){
      }

      public void add(int eno, String ename, float esal, String eaddr){
      }
      public void search(int eno){
      }
      public void update(int eno, String ename, float esal, String eaddr){
      }
      public void delete(int eno){
      }
}
```

If we create a java.lang.Class object for the above class then the Class object is able to manage the details like below.



java.lang.reflect.Field:
—----------

java.lang.reflect.Field class object is able to provide metadata of a particular variable or field of a particular class.

If we want to get the metadata of all the variables which are declared in a particular class then we have to use the following methods from java.lang.Class.

public Field[] getFields():
It is able to get all the variables metadata which are declared as public in the respective class and from its superclass.

public Field[] getDeclaredFields():
It is able to get all the variables metadata irrespective of the access modifiers from the respective class only, not from its superclass.

To get Variable Name we have to use the following method from Field class.
public String getName()

To get the Data type of a particular variable we have to use the following method.
public Class getType()

To get the Access modifiers of a particular variable we have to use the following method.

public int getModifiers()

Note: If we pass the generated integer value from the above method to a static toString() method then we are able to get the list access modifiers in the form of a String.

If the variable is a static variable then we are able to get the variable value by using the following method.

public xxx get(Field f)

If the variable is a private variable then it is possible to access by using Reflection API, that is by using the following method.

public void setAccessible(boolean b)
If b value is true then it will be accessible outside of the respective class.
If b value is false then it will not be accessible outside of the class.

EX:
Employee.java

```
package com.durgasoft.entities;

public class Employee {

   public static int eno = 111;
   protected static String ename = "Durga";
   volatile static float esal = 50000.0f;
   private static String eaddr = "Hyd";

}
```

Main.java

```
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;

public class Main {
```

```
   public static void main(String[] args)throws Exception
{
        Class cls =
Class.forName("com.durgasoft.entities.Employee");
        Field[] flds = cls.getDeclaredFields();
        for(Field fld: flds) {
            fld.setAccessible(true);
            System.out.println("Filed Name         :
"+fld.getName());
            System.out.println("Field Data Type
:"+fld.getType());
            System.out.println("Access Modifier    :
"+Modifier.toString(fld.getModifiers()));
            System.out.println("Filed Value        :
"+fld.get(fld));
            System.out.println("-------------------------
-------------------");
        }
    }
}
```

java.lang.reflect.Constructor:
—------------------------------
java.lang.reflect.Constructor class objects are able to provide metadata of the constructors, where the constructors metadata includes Constructor name, access modifiers, parameter Types, Throws Exception Types….,

To get the Constructors metadata of a particular class , first we have to create a java.lang.Class object , then we have to use the following methods.

public Constructor[] getConstructors():
It is able to provide metadata of all the public constructors from the respective class.

public Constructor[] getDeclaredConstructors():
It is able to provide all the constructors of the respective class irrespective of the public declaration.

To get name of the constructor:

public String getName()

To get the Access Modifier of the constructor:
public int getModifiers()

Note: If we pass the return value of getModifiers() method as parameter to the static toString() method from Modieir class then we are able to get the list access modifiers which we provided to the constructor.

To get all the parameter types from a Constructor:
public Class[] getParameterTypes()

To get all the Exception Types which we provided along with constructor:
public Class[] getExceptionTypes()

EX:
—--
Employee.java

```
package com.durgasoft.entities;

import java.io.NotSerializableException;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.sql.SQLException;

public class Employee {
   public Employee(int eno)throws RemoteException,
NotSerializableException {
   }
   public Employee(int eno, String ename)throws
CloneNotSupportedException, SQLException{
   }
   protected Employee(int eno, String ename, float
essal)throws RemoteException, InterruptedException{

   }
   private Employee(int eno, String ename, float esal,
String eaddr)throws SQLException, ClassCastException{
```

```
    }
}
```

Main.java

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;

public class Main {
   public static void main(String[] args)throws Exception {
       Class cls = Class.forName("com.durgasoft.entities.Employee");
       Constructor[] constructors = cls.getDeclaredConstructors();
       for(Constructor constructor: constructors){
           System.out.println("Name        : "+constructor.getName());
           System.out.println("Access Modifiers :
"+Modifier.toString(constructor.getModifiers()));
           Class[] paramTypes = constructor.getParameterTypes();
           System.out.print("Parameter Types    : ");
           for(Class param: paramTypes){
               System.out.print(param.getName()+"  ");
           }
           System.out.println();
           Class[] exceptionTypes = constructor.getExceptionTypes();
           System.out.print("Exception Types   : ");
           for(Class exception: exceptionTypes){
               System.out.print(exception.getName()+"  ");
           }
           System.out.println();
           System.out.println("-----------------------------------------
--");
       }
   }
}
```

```
C:\Java\jdk-17\bin\java.exe "-javaagent:C:\IntelliJ IDEA Community Edition
2022.3.2\lib\idea_rt.jar=50941:C:\IntelliJ IDEA Community Edition
2022.3.2\bin" -Dfile.encoding=UTF-8 -classpath D:\java6\INTELLIJ-
APPS\app41\out\production\app41 Main
Name        : com.durgasoft.entities.Employee
Access Modifiers : private
Parameter Types    : int  java.lang.String  float  java.lang.String
Exception Types   : java.sql.SQLException  java.lang.ClassCastException
-----------------------------------------
```

```
Name        : com.durgasoft.entities.Employee
Access Modifiers : protected
Parameter Types    : int  java.lang.String  float
Exception Types   : java.rmi.RemoteException  java.lang.InterruptedException
-------------------------------------------
Name        : com.durgasoft.entities.Employee
Access Modifiers : public
Parameter Types    : int  java.lang.String
Exception Types   : java.lang.CloneNotSupportedException
java.sql.SQLException
-------------------------------------------
Name        : com.durgasoft.entities.Employee
Access Modifiers : public
Parameter Types    : int
Exception Types   : java.rmi.RemoteException
java.io.NotSerializableException
-------------------------------------------

Process finished with exit code 0
```

java.lang.reflect.Method:
—-------------------------
java.lang.reflect.Method class object is able to provide the metadata of a particular method , where the metadata of a particular method includes the access modifiers list, return type, Method Name, Parameter Types, throws Exception types.

If we want to get methods metadata of a particular class then we have to create java.lang.Class object first, after the java.lang.Class object we have to get all the methods metadata in the form of java.lang.reflect.Method[] by using the following methods from java.lang.Class.

public Method[] getMethods():
It is able to provide all methods metadata which are declared with public from the respective class and from the super class.

public Method[] getDeclaredMethods():
It is able to provide all methods metadata irrespective of the access modifiers which we provided from the respective class only, not from its superclass.

To get method name:

public String getName()

To get Return Type:
public Class getReturnType()

To get Access Modifiers:
public int getModifiers()

Note: If we pass the return value of getModifiers() method as parameter to the static toString() method from java.lang.reflect.Modifier class then we are able to get the access modifiers list as a string.

To get all parameter Types:
public Class[] getParameterTypes()

To get Exception Types:
public Class[] getExceptionTypes()

EX:
Employee.java

```
package com.durgasoft.entities;

import java.io.NotSerializableException;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.sql.SQLException;

public class Employee {
   public void add(int eno, String ename, float esal, String eaddr)throws
SQLException, ClassCastException{

   }
   protected String search(int eno)throws InterruptedException,
CloneNotSupportedException{
       return "";
   }
   void update(int eno, String ename, float esal, String eaddr)throws
ClassNotFoundException, NullPointerException{

   }
   private void delete(int eno)throws SQLException, IllegalAccessException{

   }
}
```

Main.java

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;

public class Main {
   public static void main(String[] args)throws Exception {
        Class cls = Class.forName("com.durgasoft.entities.Employee");
        Method[] methods = cls.getDeclaredMethods();
        for (Method method: methods){
            System.out.println("Name              : "+method.getName());
            System.out.println("Return Type       :
"+method.getReturnType().getName());
            System.out.println("Access Modifiers :
"+Modifier.toString(method.getModifiers()));
            System.out.print("Parameter Types   : ");
            Class[] paramTypes = method.getParameterTypes();
            for(Class paramType: paramTypes){
                System.out.print(paramType.getName()+"  ");
            }
            System.out.println();

            Class[] exceptionTypes = method.getExceptionTypes();
            System.out.print("Exception Types    : ");
            for (Class exceptionType: exceptionTypes){
                System.out.print(exceptionType.getName()+"  ");
            }
            System.out.println();
            System.out.println("-----------------------------------------
----------");
        }

   }
}
```

```
C:\Java\jdk-17\bin\java.exe "-javaagent:C:\IntelliJ IDEA Community Edition
2022.3.2\lib\idea_rt.jar=51028:C:\IntelliJ IDEA Community Edition
2022.3.2\bin" -Dfile.encoding=UTF-8 -classpath D:\java6\INTELLIJ-
APPS\app41\out\production\app41 Main
Name              : add
Return Type       : void
Access Modifiers : public
Parameter Types   : int  java.lang.String  float  java.lang.String
```

```
Exception Types    : java.sql.SQLException  java.lang.ClassCastException
-------------------------------------------------------
Name             : update
Return Type      : void
Access Modifiers :
Parameter Types  : int  java.lang.String  float  java.lang.String
Exception Types    : java.lang.ClassNotFoundException
java.lang.NullPointerException
-------------------------------------------------------
Name             : delete
Return Type      : void
Access Modifiers : private
Parameter Types  : int
Exception Types    : java.sql.SQLException  java.lang.IllegalAccessException
-------------------------------------------------------
Name             : search
Return Type      : java.lang.String
Access Modifiers : protected
Parameter Types  : int
Exception Types    : java.lang.InterruptedException
java.lang.CloneNotSupportedException
-------------------------------------------------------

Process finished with exit code 0
```