Reflection API

- Reflection API is a powerful technique to find out the environment of the class as well as to inspect the class itself.
- Reflection API is introduced in Java 1.1.
- The classes of Reflection API are the part of the package java.lang.reflect and methods are the part of java.lang.class.
- It allows the user to get complete information about classes,interfaces,constructors,fields and various methods being used.
- It also provides an easy way to develop java applications which is not possible before java 1.1.
- We can create the methods like event handler, hash code etc and also we can find out objects and classes.
- Avoid using of the reflection API wherever it affects the performance of an application like applet applications.
- We are using reflection API to mapping objects into tables in database at runtime.

Reflection API Methods

getClass(), classname.class, Class.forName("classname with package");	These methods are used to get information about the classes which are declared. Here whenever class is loaded into stack memory java.lang.class object is created for respective class.

```
package ReflectionAPI;
class H {

}
class H2 extends H{

}
public class H1 {
        public static void main(String[] args) {
                H h = new H();
                H1 h1 = new H1();
                H2 h2 = new H2();
                System.out.println("reference
                                        variable h");
                Class c1 = h.getClass();
                Class c2 = h1.getClass();
                Class c3 = h2.getClass();

                System.out.println(c1==c2);
                System.out.println(c2==c3);
                System.out.println(c3==c1);


        }

}
Output :
reference variable h
false
false
false
```

```
package ReflectionAPI;

public class F {
        public static void main(String[] args) {
                Integer i1 = new Integer(90);
                Integer i2 = 30;

                Class c1 = i1.getClass();
                Class c2 = F.class;
                Class c3 = i2.getClass();

                System.out.println(i1==i2);
                System.out.println(c1==c2);
                System.out.println(c1==c3);
        }
}

Output:
false
false
true
```

```
package ReflectionAPI;

public class F {
        public static void main(String[] args)
throws ClassNotFoundException {
                Integer i1 = new Integer(90);
                Integer i2 = 30;

                Class c1 = i1.getClass();
                Class c2 = F.class;
                Class c4 =
                 Class.forName("ReflectionAPI.F");
                Class c3 = i2.getClass();

                System.out.println(i1==i2);
                System.out.println(c1==c2);
                System.out.println(c1==c3);
                System.out.println(c2==c4);
        }

}
Output:
false
false
true
true
```

```
package ReflectionAPI;

public class F1 {
        public static void main(String[] args) {
                G g1 = new G();

                Class c1 = g1.getClass();
                Class c2 = F.class;
                Class c3 = G.class;

                System.out.println(c1==c2);
                System.out.println(c2==c3);
                System.out.println(c3==c1);


        }

}
Output:
false
false
true
```

```
package ReflectionAPI;
class H {

}
class H2 extends H{

}
public class H1 {
        public static void main(String[] args)
throws Exception{
                H h = new H();
        Class c4 =
Class.forName("ReflectionAPI.H");
                Class c5 = H.class;
                Class c6 = h.getClass();


                System.out.println(c4==c5);
                System.out.println(c5==c6);
                System.out.println(c6==c4);


        }

}
```

Output:

```
true
true
true
```

newInstance()	This method is used to create an object without using new keyword to access the instance members of the respective class.

```
package ReflectionAPI;

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

        NewInstance n1 = new NewInstance();
        Class c1 = n1.getClass();
        NewInstance n2 =
                  (NewInstance)c1.newInstance();
        System.out.println(n1);
        System.out.println(c1);
        System.out.println(n2);
        System.out.println(n1 == n2);

}}
Output:
ReflectionAPI.NewInstance@1034bb5
class ReflectionAPI.NewInstance
ReflectionAPI.NewInstance@15f5897
false
```

```
package ReflectionAPI;


public class NewInstance1 {
NewInstance1(){
        System.out.println("Constructor");
        }

NewInstance1(int x){
        System.out.println("Constructor(int x)");
        }

NewInstance1(String s ,int y){
        System.out.println("Constructor(String s
.int y)");
        }

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

NewInstance1 n1 = new NewInstance1();
NewInstance1 n11 = newNewInstance1(12);
NewInstance1 n12 = new
NewInstance1("keerthana",67);


        Class c1 = n1.getClass();
        Class c2 = n11.getClass();
        NewInstance1 n2 =
(NewInstance1)c1.newInstance();
                NewInstance1 n21 =
(NewInstance1)c2.newInstance(10);//CTE
                NewInstance1 n22 =
(NewInstance1)c1.newInstance("thanu",20);//CTE


        }

}
Output:
CTE
```

```
package ReflectionAPI;


public class NewInstance1 {
NewInstance1(){
        System.out.println("Constructor");
        }

NewInstance1(int x){
        System.out.println("Constructor(int x)");
        }

NewInstance1(String s ,int y){
System.out.println("Constructor(String s .int y)");
        }

public static void main(String[] args) throws
Exception{
        NewInstance1 n1 = new NewInstance1();
        NewInstance1 n11 = new NewInstance1(12);
        NewInstance1 n12 =
                new NewInstance1("keerthana",67);


        Class c1 = n1.getClass();
        Class c2 = n11.getClass();
        NewInstance1 n2 =
(NewInstance1)c1.newInstance();

}}
Output:
Constructor
Constructor(int x)
Constructor(String s .int y)
Constructor
```

getDeclaredMethod(method name)
getDeclaredMethods()

- getDeclaredMethod() method is used to get the name of the method of a particular class. It is in Class class and accepts method name as an argument. The return type of this method is Method object type or reference type.
- getDeclaredMethods() is used to get the methods using array with for loop to get the method name.

```
package ReflectionAPI;
import java.lang.reflect.Method;

 class N1 {
        void method1(){
                System.out.println("N1:From
method1");
        }
        void method2(){
                System.out.println("N1:From
method2");
        }}
        public class N extends N1 {
                void method1(){
                        System.out.println("N:From
method1");
                }
                void method2(){
                        System.out.println("N:From
method2");
                }


        public static void main(String[] args)
throws Exception{
                N n = new N();
                Class c1 = N.class;
                N n1 = (N)c1.newInstance();
                N1 n2 = (N1)c1.newInstance();
                Method m1 =
c1.getDeclaredMethod("method1");
                Method m2 =
c1.getDeclaredMethod("method2");
                m1.invoke(n2);
                m2.invoke(n2);

}}
```

Output:

```
N:From method1
N:From method2
```

```
package ReflectionAPI;
import java.lang.reflect.Method;

 class N1 {
        void method1(){
                System.out.println("N1:From
method1");
        }
        void method2(){
                System.out.println("N1:From
method2");
        }}
        public class N extends N1 {
                void method1(){
                        System.out.println("N:From
method1");
                }
                void method2(){
                        System.out.println("N:From
method2");
                }


        public static void main(String[] args)
throws Exception{
                N n = new N();
                Class c1 = N.class;
                Class c2 = N1.class;
                N n1 = (N)c1.newInstance();
                N1 n2 = (N1)c1.newInstance();
                Method m1 =
c2.getDeclaredMethod("method1");
                Method m2 =
c2.getDeclaredMethod("method2");
                m1.invoke(n2);
                m2.invoke(n2);

}}
```

Output:

```
N:From method1
N:From method2
```

```
package ReflectionAPI;
import java.lang.reflect.Method;

public class M9 {
        void method1(int x){
                System.out.println("from method1 "
+ x);
        }
        void method2(String s ,int y){
                System.out.println("from method2 "
+ s + " , " +y );
        }


        public static void main(String[]
args)throws Exception {
                M9 m = new M9();
                Class c1 = M9.class;
                M9 m9 = (M9)c1.newInstance();
                Method m1 =
c1.getDeclaredMethod("method1", int.class);
                Method m2 =
c1.getDeclaredMethod("method2",
String.class,int.class);
                m1.invoke(m9, 100);
                m2.invoke(m9,"Keerthana",200);

        }}
```

Output:

```
from method1 100
from method2 Keerthana , 200
```

```
package ReflectionAPI;
import java.lang.reflect.Method;

 class M10 {
        void method1(int x){
                System.out.println("from method1 "
+ x);
        }}

        public class M9 extends M10{

        void method2(String s ,int y){
                System.out.println("from method2 "
+ s + " , " +y );
        }


        public static void main(String[]
args)throws Exception {
                M9 m = new M9();
                Class c1 = M10.class;
                Class c2 = M9.class;
                M10 m10 = (M10)c1.newInstance();
                Method m1 =
c1.getDeclaredMethod("method1", int.class);
                Method m2 =
c2.getDeclaredMethod("method2",
String.class,int.class);
                m1.invoke(m10, 100);
                m2.invoke(m,"Keerthana",200);

        }}
```

Output:

```
from method1 100
from method2 Keerthana , 200
```

```
package ReflectionAPI;
import java.util.Scanner;
import java.lang.reflect.Method;

class M12{
        void method1(){
                System.out.println(" M12:from
method1");
        }
        void method2(){
                System.out.println("M12:from
method2");
        }
}

public class M11 extends M12 {
        void method1(){
                System.out.println("M11:from
method1");
        }
        void method2(){
                System.out.println("M11:" +
                                "from method2");
        }


        public static void main(String[] args)
throws Exception
        {
                System.out.println("enter the
method name");
                Scanner sc = new
Scanner(System.in);
                Class c1 = M11.class;
                M11 m11 = (M11)c1.newInstance();
                String s1 = sc.next();
                Method m1 =
c1.getDeclaredMethod(s1);
                m1.invoke(m11);

                System.out.println("--------");


                System.out.println("enter the
method name");
                Scanner sc1 = new
Scanner(System.in);
                M12 m12 = (M12)c1.newInstance();
                String s2 = sc1.next();
                Method m2 =
c1.getDeclaredMethod(s2);
                m2.invoke(m12);



        }

}
```

Output:

```
enter the method name
method1
M11:from method1
--------
enter the method name
method1
M11:from method1
```

```
package ReflectionAPI;
import java.lang.reflect.Method;
import java.util.Scanner;




abstract class M14{
        void method1(){
                System.out.println("M14: method1");
        }
        void method2(){
                System.out.println("M14:method2");
        }
}
 class M15 {
        void method3(){
                System.out.println("M15:method3");
        }
        void method4(){
                System.out.println("M15:method3");
        }
}


public class M13  {

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

                Scanner sc = new
Scanner(System.in);
                System.out.println("enter the class
name");

                String className = sc.next();
                System.out.println("enter the
method name");
                String methodName = sc.next();
        Class c1 = Class.forName("ReflectionAPI " +
"." + className);
//
                Object obj =c1.newInstance();
                Method m1[] =
c1.getDeclaredMethods();
                for(Method m : m1){
                System.out.println(m);




        }

}}
```

Output:

```
enter the class name
M14
enter the method name
metod1
from method1
```

getParameterTypes() : This method is used to get the parameters passed in the methods which is declared in the class.

```
package ReflectionAPI;

import java.lang.reflect.Method;

public class Parameter {
        void method1(int x,int y){
                System.out.println("from
method1 " + x + "," + y );

        }
        void method2(String s , double d){
                System.out.println("from
method2 " + s + "," +d) ;

        }
        static boolean method3(boolean b){
                System.out.println("from
method3 " + b);
                return b;
        }


        public static void main(String[]
args)throws Exception {
                Class c1 = Parameter. class;
                Parameter p1 =
(Parameter)c1.newInstance();
                Method m1[] =
c1.getDeclaredMethods();
                for(Method m : m1){

        System.out.println("methodName : " +
m);

        System.out.println("***************"
);

        System.out.println("Parameter types
: " +m.getParameterTypes());
                }
                System.out.println("--------
-------");
                Method m2 =
c1.getDeclaredMethod("method1", int.class,
int.class);
                Method m3 =
c1.getDeclaredMethod("method2",
String.class,double.class);
                m2.invoke(p1, 100,400);
                m3.invoke(p1,"Thanu"
,300.00);



        }

}
```

getModifiers() :This method is used to get the access specifiers of a class and its return type is int.

```
package ReflectionAPI;

import java.lang.reflect.Modifier;

class A2{

}
abstract class A3 extends A2{

}
abstract class A4 extends A3{

}
final class A5 extends A4{

}

public class Modifiers {
        public static void main(String[] args) {
                Class c1 = A2.class;
                Class c2 = A3.class;
                Class c3 = A4.class;
                Class c4 = A5.class;

        System.out.println(Modifier.toString(c1.getModifier
s()));

        System.out.println(Modifier.toString(c2.getModifier
s()));

        System.out.println(Modifier.toString(c3.getModifier
s()));

        System.out.println(Modifier.toString(c4.getModifier
s()));
        }
}
```

Output :

```
abstract
abstract
final
```

Output :

```
methodName : void
ReflectionAPI.Parameter.method1(int, int)
***************
Parameter types : [int, int]
methodName : void
ReflectionAPI.Parameter.method2(java.lang.St
ring, double)
***************
Parameter types : [class java.lang.String,
double]
methodName : static boolean
ReflectionAPI.Parameter.method3(boolean)
***************
Parameter types : [boolean]
methodName : public static void
ReflectionAPI.Parameter.main(java.lang.Strin
g[]) throws java.lang.Exception
***************
Parameter types : [class
[Ljava.lang.String;]
---------------
from method1 100,400
from method2 Thanu,300.0
```

getDeclaredConstructors() :
This method is used to get the constructors declared in the class with class name.

```
package ReflectionAPI;

import java.lang.reflect.Constructor;

class Cons1{
	Cons1(){
		System.out.println("default
Constructor");
	}
	Cons1(int x){

	System.out.println("parameterized
constructor");
	}

}


public class Cons extends Cons1 {
	Cons(){
		System.out.println("Cons :
default constructor");
	}
	Cons(String s){
		System.out.println("Cons :
parameterized constructor");
	}
	public static void main(String[]
args) {
		Class c1 = Cons. class;
		Class c2 = Cons1.class;

		Constructor c[] =
c1.getDeclaredConstructors();
		for (Constructor ctr : c){

	System.out.println(ctr);
		}
		System.out.println("---------
---");

		Constructor ct[] =
c2.getDeclaredConstructors();
		for(Constructor ctr1 : ct){

	System.out.println(ctr1);
		}
	}

}
```

Output :

```
ReflectionAPI. Cons()
ReflectionAPI. Cons(java.lang.String)
------------
ReflectionAPI.Cons1()
ReflectionAPI.Cons1(int)
```

getPackage() :
From this method we can get the package name in which the class is declared. Its return type is char[] and return type is null. Syntax

```
public static char[] getPackage() {
		return null;
	}
```

```
package toString;

class A
{

}

public class Client1
{
	public static void main(String[] args) {
		A obj1 = new A();
		System.out.println(obj1);
		System.out.println(obj1.toString());
		Class c2 = Client1.class;
	}

	public static char[] getPackage() {
		return null;
	}
}


package ReflectionAPI;
import toString.Client1;

public class Pack1 {
	public static void main(String[] args) throws
Exception{

		Class c1 = Pack1.class;
		Client1 c2 = new Client1();
		System.out.println(c1.getPackage());
		System.out.println(
Client1.getPackage());//CTE
		System.out.println(c2.getPackage());//CTE
	}

}
```

Output :

```
package ReflectionAPI
Exception in thread "main" java.lang.NullPointerException
```

getDeclaredFields(), getName(), getType()
from these methods we can get the type ,name and reference of the field s those are declared in the class.

```
package ReflectionAPI;

import java.lang.reflect.Field;

public class FieldsFd {
        static int i;
        final String s = "Keerthana";
        private byte b;
        protected float f;
        public static void main(String[] args) {



                Class c1 = FieldsFd  .class;
                Field fd[] = c1.getDeclaredFields();
                        for(Field f : fd){
                        System.out.println("References : " +f);
                        System.out.println("Field Name :"  +f.getName());
                        System.out.println("Field type : " +f.getType());
                        System.out.println("----------------------");
                }

        }}
```

```
Output:
References : static int ReflectionAPI.FieldsFd.i
Field Name :i
Field type : int
----------------------
References : final java.lang.String ReflectionAPI.FieldsFd.s
Field Name :s
Field type : class java.lang.String
----------------------
References : private byte ReflectionAPI.FieldsFd.b
Field Name :b
Field type : byte
----------------------
References : protected float ReflectionAPI.FieldsFd.f
Field Name :f
Field type : float
----------------------
```