

Java Reflection

1. What is Reflection?

This is used to examine the Classes, Methods, Fields, Interfaces at runtime and also possible to change the behavior of the Class too.

✓For example:

- What all methods present in the class. ✓
- What all fields present in the class. ✓
- What is the return type of the method. ✓
- What is the Modifier of the Class ✓
- What all interfaces class has implemented ✓
- Change the value of the public and private fields of the Class etc.....

2. How to do Reflection of Classes?

To reflect the class, we first need to get an Object of **Class**

(So, lets first understand, **Class** then we will come back to how to reflect the class.)

What is this class **Class**?

-
- Instance of the class **Class** represents classes during runtime.
 - JVM creates one **Class** object for each and every class which is loaded during run time.
 - This **Class** object, has meta data information about the particular class like its method, fields, constructor etc.

How to get the particular class **Class** object? ✓

There are 3 ways:

1. Using **forName()** method

```
//assume that we have one class called Bird
class Bird { }

//get the object of Class for getting the metadata information of Bird class.
Class birdClass = Class.forName( className: "Bird");
```

2. Using `.class`

```
//assume that we have one class called Bird
class Bird { }

//get the object of Class for getting the metadata information of Bird class.
Class birdClass = Bird.class;
```

3. Using `getClass()` method

```
//assume that we have one class called Bird
class Bird { }

Bird birdObj = new Bird();

//get the object of Class for getting the metadata information of Bird class.
Class birdClass = birdObj.getClass();
```

Now lets get back to, [how to do Reflection of Classes](#):

```
public class Eagle {
    private String breed;
    private boolean canSwim;

    public void fly(){
        System.out.println("fly");
    }

    public void eat(){
        System.out.println("eat");
    }
}

public class Main {
    public static void main(String args[]){
        Class eagleClass = Eagle.class;
        System.out.println(eagleClass.getName());
        System.out.println(Modifier.toString(eagleClass.getModifiers()));
    }
}
```

OUTPUT:

```
Eagle
public
```

Methods Available in Class object, all are get, not set methods

m getClassLoader()	ClassLoader
✓ m asSubclass(Class clazz)	Class
m cast(Object obj)	Object
m getClass()	Class<? extends Class>
✓ m getName()	String
m desiredAssertionStatus()	boolean
m getAnnotatedInterfaces()	AnnotatedType[]
m getAnnotatedSuperclass()	AnnotatedType
m getAnnotation(Class annotationClass)	Annotation
m getAnnotations()	Annotation[]
m getCanonicalName()	String
m getClasses()	Class<?>[]
m getComponentType()	Class<?>
✓ m getConstructor(Class<?>... parameterTypes)	Constructor
✓ m getConstructors()	Constructor<?>[]
m getDeclaredAnnotations()	Annotation[]
m getDeclaredAnnotation(Class<T> annotationClass)	T
m getDeclaredClasses()	Class<?>[]
m getDeclaredConstructor(Class<?>... parameterTypes)	Constructor
m getDeclaredConstructors()	Constructor<?>[]
m getDeclaredField(String name)	Field
✓ m getDeclaredFields()	Field[]
m getDeclaredMethod(String name, Class<?>... parameterTypes)	Method
m getDeclaredMethods()	Method[]
m getDeclaringClass()	Class<?>
m getEnclosingClass()	Class<?>
m getEnclosingConstructor()	Constructor<?>
m getEnclosingMethod()	Method
m getEnumConstants()	Object[]
m getField(String name)	Field
✓ m getFields()	Field[]
m getGenericInterfaces()	Type
m getGenericSuperclass()	Type
✓ m getInterfaces()	Class<?>[]
m getMethod(String name, Class<?>... parameterTypes)	Method
✓ m getMethods()	Method[]
✓ m getModifiers()	int
m getPackage()	Package
m getProtectionDomain()	ProtectionDomain
m getResource(String name)	URL
m getResourceAsStream(String name)	InputStream
m getSigners()	Object[]
m getSimpleName()	String
m getSuperclass()	Class

The package "java.lang.reflect" provides classes that can be used to access and manipulate the value like fields, methods, constructor etc.

And these classes are generally returned by above listed get Methods only.

Reflection of Methods

```

public class Eagle {
    public String breed;
    private boolean canSwim;
    public void fly() {
        System.out.println("fly");
    }
    private void eat() {
        System.out.println("eat");
    }
}

public class Main {
    public static void main(String args[]) {
        Class eagleClass = Eagle.class;
        Method[] methods = eagleClass.getMethods();
        for (Method method: methods) {
            System.out.println("Method name: " + method.getName());
            System.out.println("Return Type: " + method.getReturnType());
            System.out.println("Class Name: " + method.getDeclaringClass());
            System.out.println("*****");
        }
    }
}

```

```

public class Eagle {
    public String breed;
    private boolean canSwim;
    public void fly() {
        System.out.println("fly");
    }
    private void eat() {
        System.out.println("eat");
    }
}

public class Main {
    public static void main(String args[]) {
        Class eagleClass = Eagle.class;
        Method[] methods = eagleClass.getDeclaredMethods();
        for (Method method : methods) {
            System.out.println("Method name: " + method.getName());
        }
    }
}

```

Output:

```

Method name: fly ✓
Method name: eat ✓

```

Methods Available in Class object, all are get, not set methods

```

String getName();
Object getClass();
Class<?> getSuperclass();
Class<?> getDeclaringClass();
Type getGenericReturnType();
Annotation[] getAnnotations();
Annotation annotation(Class<?> annotationClass);
Annotation[] getDeclaredAnnotations();
Annotation annotation(Class<?> annotationClass);
Annotation[] getDeclaredAnnotations();
Annotation annotation(Class<?> annotationClass);
Annotation[] getAnnotationsByType(Class<?> annotationClass);
Annotation annotation(Class<?> annotationClass);
Annotation[] getDeclaredAnnotationsByType(Class<?> annotationClass);

```

Invoking Method using Reflection:

```

public class Eagle {
    public Eagle() {
    }

    public void fly(int paramInt, boolean boolParam, String strParam) {
        System.out.println("fly paramInt: " + paramInt + " boolParam: " + boolParam + " strParam: " + strParam);
    }
}

public class Main {
    public static void main(String args[]) throws InstantiationException, IllegalAccessException, ClassNotFoundException {
        ① Class eagleClass = Class.forName( className: "Eagle");
        Object eagleObject = eagleClass.newInstance();

        ② Method flyMethod = eagleClass.getMethod( name: "fly", ...parameterTypes: int.class, boolean.class, String.class);
        ③ flyMethod.invoke(eagleObject, ...args: 1, true, "hello");
    }
}

```

Output:

```

fly paramInt: 1 boolParam: true strParam: hello
Process finished with exit code 0

```

Reflection of Fields:

```
public class Eagle {  
    public String breed;  
    private boolean canSwim;  
  
    public void fly() {  
        System.out.println("fly");  
    }  
  
    private void eat() {  
        System.out.println("eat");  
    }  
}
```

```
① Class eagleClass = Eagle.class;  
② //Get public fields with this  
Field[] fields = eagleClass.getFields();  
for (Field field : fields) {  
    System.out.println("fieldName: " + field.getName());  
    System.out.println("Type: " + field.getType());  
    System.out.println("Modifier: " + Modifier.toString(field.getModifiers()));  
    System.out.println("*****");  
}  
}  
}
```

Output:
fieldName: breed ✓
Type: class java.lang.String ✓
Modifier: public ✓

Process finished with exit code 0

```
① public static void main(String args[]) {  
    Class eagleClass = Eagle.class;  
    //Get both public and private fields with this  
Field[] fields = eagleClass.getDeclaredFields();  
for (Field field : fields) {  
    System.out.println("fieldName: " + field.getName());  
    System.out.println("Type: " + field.getType());  
    System.out.println("Modifier: " + Modifier.toString(field.getModifiers()));  
    System.out.println("*****");  
}  
}  
}
```

Output:
fieldName: breed
Type: class java.lang.String
Modifier: public

fieldName: canSwim
Type: boolean
Modifier: private

Process finished with exit code 0

Get method supported

<code>m get(Object obj)</code>	<code>Object</code>
<code>m getName()</code>	<code>String</code>
<code>m getModifiers()</code>	<code>int</code>
<code>m getType()</code>	<code>Class<?></code>
<code>m getAnnotation(Class<T> annotationClass)</code>	<code>T</code>
<code>m getAnnotatedType()</code>	<code>AnnotatedType</code>
<code>m getAnnotationsByType(Class<T> annotationClass)</code>	<code>T[]</code>
<code>m getAnnotations()</code>	<code>Annotation[]</code>
<code>m getBoolean(Object obj)</code>	<code>boolean</code>
<code>m getByte(Object obj)</code>	<code>byte</code>
<code>m getChar(Object obj)</code>	<code>char</code>
<code>m getDeclaredAnnotations()</code>	<code>Annotation[]</code>
<code>m getDeclaredAnnotation(Class<T> annotationClass)</code>	<code>T</code>
<code>m getDeclaringClass()</code>	<code>Class<?></code>
<code>m getDouble(Object obj)</code>	<code>double</code>
<code>m getFloat(Object obj)</code>	<code>float</code>
<code>m getGenericType()</code>	<code>Type</code>
<code>m getInt(Object obj)</code>	<code>int</code>
<code>m getLong(Object obj)</code>	<code>long</code>
<code>m getShort(Object obj)</code>	<code>short</code>
<code>m getDeclaredAnnotationsByType(Class<T> annotationClass)</code>	<code>T[]</code>
<code>m getClass()</code>	<code>Class<? extends Field></code>



Setting the value of Public field:

```
public class Main {
    public static void main(String args[]) throws NoSuchFieldException, IllegalAccessException {
        Class eagleClass = Eagle.class;
        Eagle eagleObj = new Eagle();
        //Set both public and private fields with this
        Field field = eagleClass.getDeclaredField("name");
        field.set(eagleObj, "eagleBrownBreed");
        System.out.println(eagleObj.breed);
    }
}
```

Output: eagleBrownBreed



Setting the value of private field: (incorrect way)

```
public class Main {
    public static void main(String args[]) throws NoSuchFieldException, IllegalAccessException {
        Class eagleClass = Eagle.class;
        Eagle eagleObj = new Eagle();
        //Set both public and private fields with this
        Field field = eagleClass.getDeclaredField("canSoar");
        field.set(eagleObj, true);
    }
}
```

```
Exception in thread "main" java.lang.reflect.AccessibleObject$InvocationTargetException: Class Main can not access a member of class Eagle with modifiers <private> at java.lang.reflect.AccessibleObject.slowCheckMemberAccess(AccessibleObject.java:296)
at java.lang.reflect.AccessibleObject.checkAccess(AccessibleObject.java:288)
at java.lang.reflect.Field.set(Field.java:703)
at Main.main(Main.java:12)
```

Setting the value of private field: ✓ ✓

```
public class Main {
    public static void main(String args[]) throws NoSuchFieldException, IllegalAccessException {
        Class eagleClass = Eagle.class;
        Eagle eagleObj = new Eagle();
        //Get both public and private fields with this
        Field field = eagleClass.getDeclaredField(name: "canSwim");
        field.setAccessible(true);
        field.set(eagleObj, true);
        if(field.getBoolean(eagleObj)){
            System.out.println("value is set to true");
        }
    }
}
```

Output:

```
value is set to true
Process finished with exit code 0
```

Reflection of Constructor:

```
public class Eagle {  
    private Eagle() {  
        //private constructor  
    }  
    public void fly() {  
        System.out.println("fly");  
    }  
  
public class Main {  
    public static void main(String args[]) throws InvocationTargetException, InstantiationException, IllegalAccessException {  
        ① Class eagleClass = Eagle.class;  
        //to access private constructor too.  
        ② Constructor[] eagleConstructorList = eagleClass.getDeclaredConstructors();  
        for(Constructor eagleConstructor : eagleConstructorList){  
            System.out.println("Modifier: " + Modifier.toString(eagleConstructor.getModifiers()));  
  
            eagleConstructor.setAccessible(true);  
            Eagle eagleObject = (Eagle) eagleConstructor.newInstance();  
            eagleObject.fly();  
        }  
    }  
}
```

```
m getAnnotation(Class<T> annotationClass) T
m getName() String
m getDeclaringClass() Class
m getModifiers() int
m getAnnotatedReceiverType() AnnotatedType
m getAnnotatedReturnType() AnnotatedType
m getDeclaredAnnotations() Annotation[]
m getDeclaredAnnotation(Class<T> annotationClass) T
m getExceptionTypes() Class<?>[]
m getGenericExceptionTypes() Type[]
m getGenericParameterTypes() Type[]
m getParameterAnnotations() Annotation[][][]
m getParameterCount() int
m getParameterTypes() Class<?>[]
m getTypeParameters() TypeVariable<Constructor>[]
m getAnnotatedExceptionTypes() AnnotatedType[]
m getAnnotatedParameterTypes() AnnotatedType[]
m getAnnotations() Annotation[]
m getAnnotationsByType(Class<T> annotationClass) T[]
m getDeclaredAnnotationsByType(Class<T> annotationClass) T[]
m getParameters() Parameter[]
m getClass() Class<? extends Constructor>
m toGenericString() String
```