Java: Reflection

"Concept && Coding" YT Video Notes

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:**

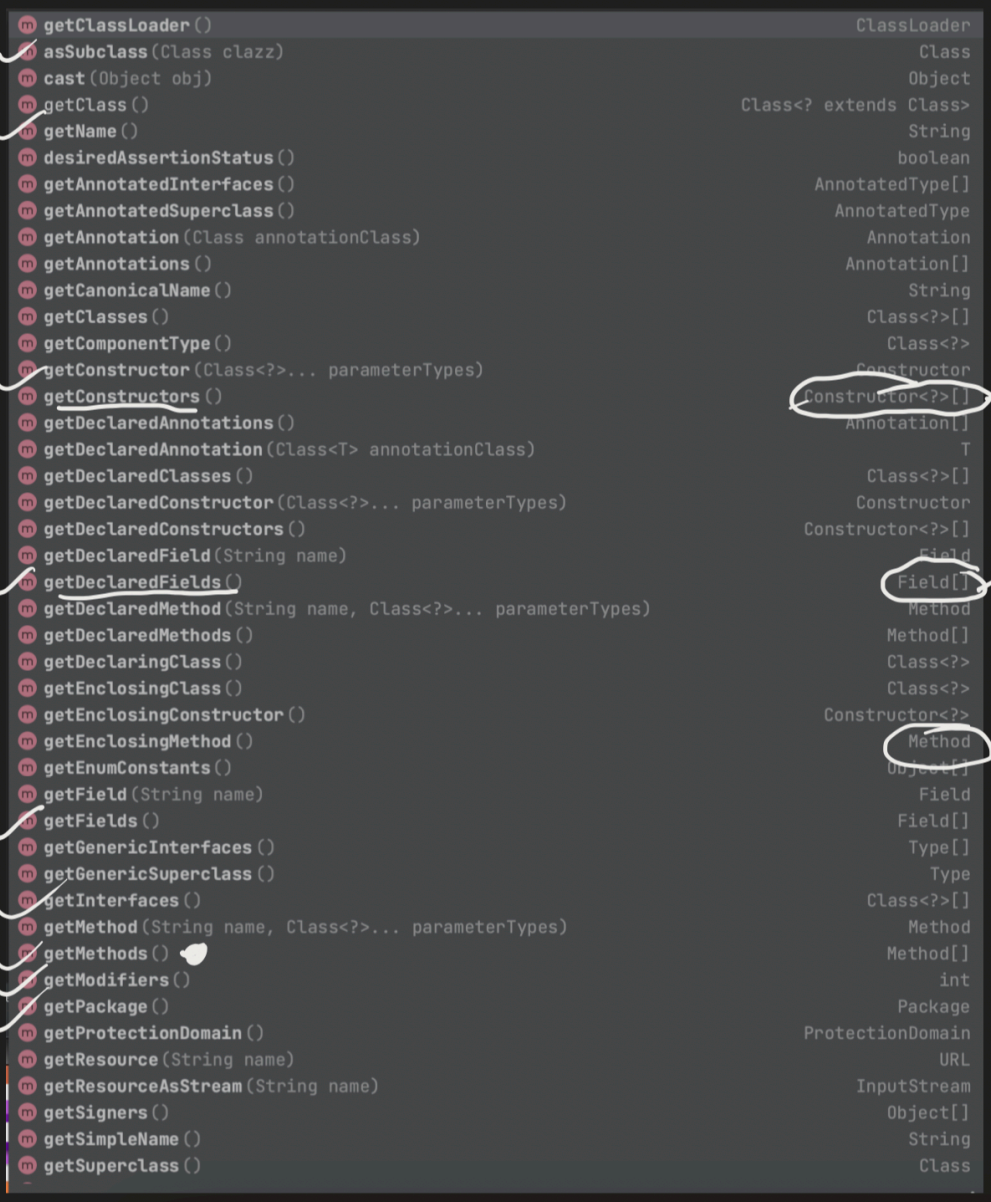
```
public class Eagle {

    public 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

Process finished with exit code 0
```

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



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:

```java
public class Eagle {

    public String breed;
    private boolean canSwim;

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

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

```java
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("*****");

        }
    }
}
```

All public methods it will return

Output:

```
MethodName: fly
ReturnType: void
ClassName: class Eagle
*******
MethodName: wait
ReturnType: void
ClassName: class java.lang.Object
*******
```

```java
public class Eagle {

    public String breed;
    private boolean canSwim;

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

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

+ also w

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

        Class eagleClass = Eagle.class;

        Method[] methods = eagleClass.getDeclaredMethods();
        for(Method method : methods){
         System.out.println("MethodName: " + method.getName());

        }
    }
}
```

All public and private methods it will return within Eagle class only

Output:

```
MethodName: fly
MethodName: eat
```

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

getName()	String
equals(Object obj)	boolean
getReturnType()	Class<?>
getDeclaringClass()	Class<?>
getDefaultValue()	Object
getAnnotatedReturnType()	AnnotatedType
getAnnotation(Class<T> annotationClass)	T
getDeclaredAnnotations()	Annotation[]
getDeclaredAnnotation(Class<T> annotationClass)	T
getExceptionTypes()	Class<?>[]
getGenericExceptionTypes()	Type[]
getGenericParameterTypes()	Type[]
getGenericReturnType()	Type
getModifiers()	int
getParameterAnnotations()	Annotation[][]
getParameterCount()	int
getParameterTypes()	Class<?>[]
getTypeParameters()	TypeVariable<Method>[]
hashCode()	int
invoke(Object obj, Object... args)	Object
isBridge()	boolean
isDefault()	boolean
isSynthetic()	boolean
isVarArgs()	boolean
toGenericString()	String
toString()	String
getAnnotatedExceptionTypes()	AnnotatedType[]
getAnnotatedParameterTypes()	AnnotatedType[]
getAnnotatedReceiverType()	AnnotatedType
getAnnotations()	Annotation[]
getAnnotationsByType(Class<T> annotationClass)	T[]
getDeclaredAnnotationsByType(Class<T> annotationClass)	T[]

Invoking Method using Reflection:

```java
public class Eagle {

    Eagle() {
    }

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

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

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

Output:

```
fly intParam: 1 boolParam: true strParm: hello

Process finished with exit code 0
```

Reflection of Fields:

```java
public class Eagle {

    public String breed;
    private boolean canSwim;

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

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

```java
public static void main(String args[]) {

    Class eagleClass = Eagle.class;

    //Get [redacted] public [redacted] 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
```

```java
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

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

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();
        //Get both public and private fields with this
        Field field = eagleClass.getDeclaredField( name: "breed");
        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();
        //Get both public and private fields with this
        Field field = eagleClass.getDeclaredField( name: "canSwim");
        field.set(eagleObj, true);
    }
}
```

```
Exception in thread "main" java.lang.IllegalAccessException Create breakpoint : Class Main can not access a member of class Eagle with modifiers "private" <1 internal line>
    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:761)
    at Main.main(Main.java:11)
```

Setting the value of private field:

```java
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");
        }
    }
}
```

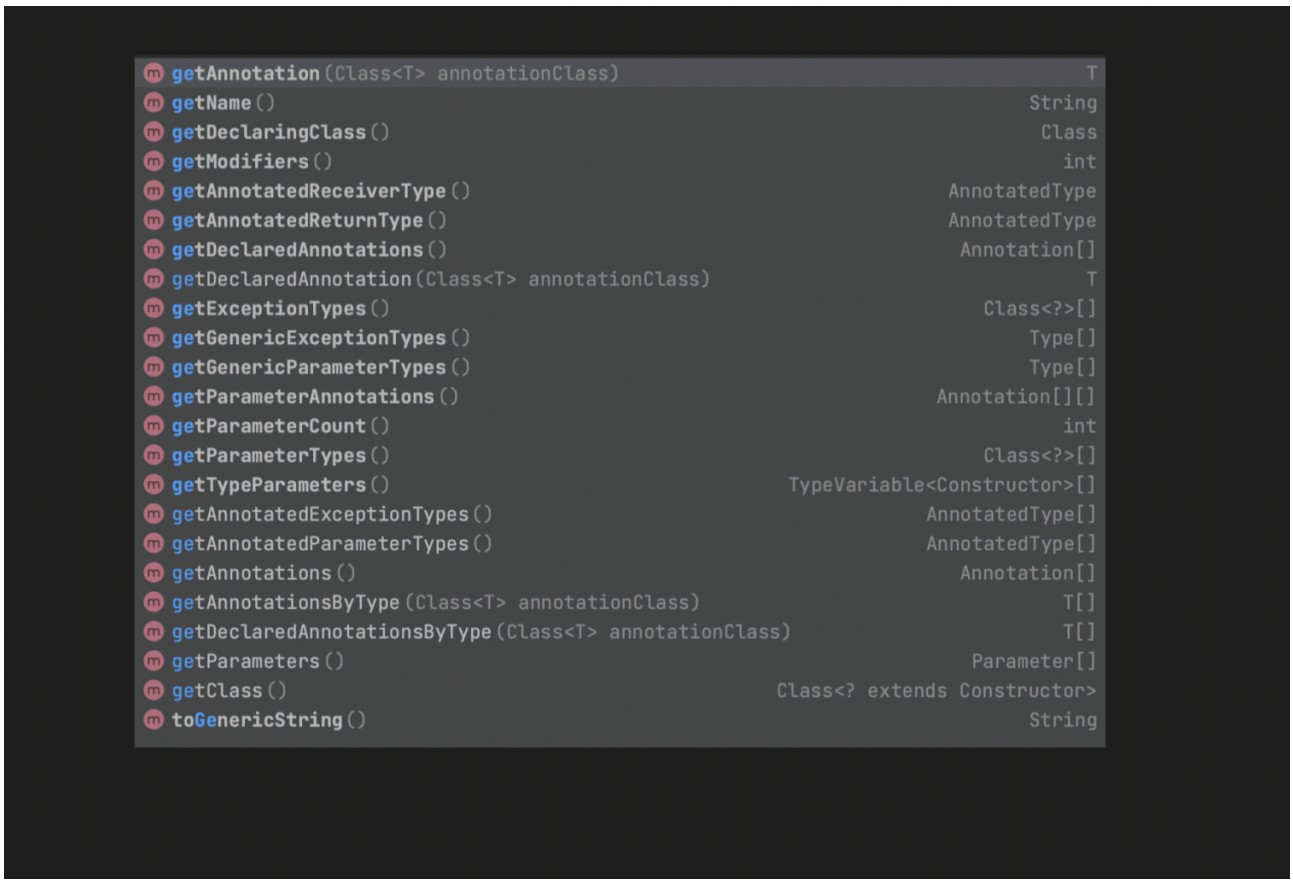
Output:

```
value is set to true

Process finished with exit code 0
```

Reflection of Constructor:

```java
public class Eagle {

    private Eagle() {

        //private constructor
    }

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

```java
public class Main {
    public static void main(String args[]) throws InvocationTargetException, InstantiationException, IllegalAccessException {

        Class eagleClass = Eagle.class;

        //to access private constructor too.
        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();
        }
    }
}
```


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