JAVA: REFLECTION IN DEPTH

By Saurav Saxena.

1. What is Reflection in java.
->This id 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 int he class.
->What all fields present int the class.
->What id the return type of methods.
->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 Class?
-> To reflect the class, we first need to get an Object of 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 it's 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 bject of Class for getting the metadata information of Bird class.
Class birdClass = Class.forName("Bird");
```

2. Using .class .

```
Class birdClass = Bird.class;
```

3. Using getClass() method.

```
Bird obj = new Bird();
Class birdClass = birdObj.getClass();
```

->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
```

->The package 'java.lang.reflect' provides classes that can be used to access and manipulate the values like fields, methods, constructor etc.
->And these classes are generally returned by above list of get methods only.

3. Reflection of Methods.

->

```
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 eagle = Eagle.class;
    //it return all public method of current class as well as parent class public method
too.
    Method[] methods = eagle.getMethods();
    for(Method method : methods){
      System.out.println(method.getName() +" " + method.getReturnType()+" "
method.getDeclaringClass());
      System.out.println("******");
    }
    }
  }
  output:
  fly void Eagle
  ******
  eat void Eagle
  ******
```

eagle.getDeclaredMethods():
->All public and private method it will return within current class only.

4. How to Invoking Method using Reflection.
->

```
  public class Eagle{
    Eagle(){}
    public void fly(int p, boolean b, String str){
      System.out.println("p: "+p+" b: "+b+" str: "+str);
    }
  }
  public class Main{
     public static void main(String[] args){
       Class class = Class.forName("Eagle");
       Object obj = class.newInstance();
       Method fly = class.getMethod("fly",int.class,boolean.class,String.class);
      fly.invoke(obj,1,true,"hello");
     }
  }
  output:
```

5. Reflections Of fields.
->

```
  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 eagle = Eagle.class;
      //it return all public fields of current class as well as parent class public fields
  too.
      Fields[] fields= eagle.getFields();
      for(Field f: fields){
        System.out.println(f.getName() +" " + f.getType()+" "
  Modifiers.toString(f.getModifiers()));
        System.out.println("******");
      }
    }
  }
```

eagle.getDeclaredFields():
->All public and private field it will return within current class only.

6. Setting the value of public fields.
->

```
  public class Main{
    public static void main(String[] args) throws
  NoSuchFieldException,IlleagalAccessException{
      Class eClass = Eagle.class;
      Eagle eObj = new Eagle();
      //get both static and private fields
      //with this
      Field field = eClass.getDeclaredField("breed");
      field.set(eObj,"Eagle brown breed");
      System.out.println(eagleObj.breed);
    }
  }
  output:
  Eagle brown breed
```

7. Setting the value of Private fields.
->[InCorrect way]

```
  //throw exception
  public class Main{
    public static void main(String[] args) throws
  NoSuchFieldException,IlleagalAccessException{
      Class eClass = Eagle.class;
      Eagle eObj = new Eagle();
      //get both static and private fields
      //with this
      Field field = eClass.getDeclaredField("canSwim");
      //field.setAccessible(true);
      field.set(eObj,true);
      System.out.println(eagleObj.breed);
    }
  }
```

8. By using field.setAccessible(true)
-> We can change private field
->it breaks the law of encapsulation and inheritance.

9. Reflection Of Constructor.

->Reflection break singleton

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

10. Why we try to avoid using reflection.

->It's violate the singleton principle or we can say whole oop's concept is going to failed here.

->Reflection is slow operation perform at run time.

->It increase the complexity of program and not easy to understand.