<http://javarevisited.blogspot.sg/2012/05/how-to-access-private-field-and-method.html>

How to access private field and method using Reflection in Java

Reflection in Java is very powerful feature and allows you to **access private method and fields** which is not possible by any other means in Java and because of this feature of reflection many code coverage tool, static analysis tool and Java IDE like Eclipse and Netbeans has been so helpful. In last article we have seen details about private keyword in Java and learned  [why we should always make fields and method private in Java](http://javarevisited.blogspot.com/2012/03/private-in-java-why-should-you-always.html). There we have mentioned that private fields and methods are only accessible in the class they are declared but with reflection you can call private method and access private fields outside the class. In this article we will see simple example of **accessing private field using reflection** and invoking private method using reflection in Java.

## Accessing private fields in Java using reflection

[invoke private method and field in java with reflection code example](http://4.bp.blogspot.com/-k_kFU8dhOkU/Tc08HSf42KI/AAAAAAAAAJg/PVEBybW2qYY/s1600/java_logo_50_50.jpg)In order to access private field using reflection, you need to know the name of field than by calling getDeclaredFields(String name) you will get a java.lang.reflect.Field instance representing that field. remember using **getDclaredFields() method and not** **getFields()** method which returns all non private fields both from sub class and super class. while getDeclaredFields() returns both private and non private fields declared in the class. Once you get the field reference you need to make it accessible by calling Field.setAccessible(true) because you are going to access private field. Now you can get value or private field by calling Field.get(String field\_name).if you don't call setAccessible(true) and try to access private field using reflection you will get Exception as shown in below example

java.lang.**IllegalAccessException**: **Class** test.ReflectionTest can not access a member of **class** test.Person with modifiers "private"  
  
        at sun.reflect.Reflection.ensureMemberAccess(Reflection.java:65)  
        at java.lang.reflect.[**Field**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/reflect/Field.html).doSecurityCheck(**Field**.java:960)  
        at java.lang.reflect.**Field**.getFieldAccessor(**Field**.java:896)  
        at java.lang.reflect.**Field**.get(**Field**.java:358)  
        at test.ReflectionTest.main(ReflectionTest.java:31)

## Calling private methods in Java using reflection

In our last Java tutorial on Reflection we have seen [how to call a method by its String name](http://javarevisited.blogspot.com/2012/04/how-to-invoke-method-by-name-in-java.html) and we will use that information here for invoking private method. Calling private method using reflection is similar to accessing private fields reflectively. Use getDeclaredMethods(String name, Class.. parameter) to get declared private method. pass all the argument type needed by method or nothing if method doesn't accept any argument. This will give you instance of java.lang.reflect.Method which can than be used to call private method using reflection, as shown in code example.

**Code example of accessing private field and method using reflection**

**package test;**

**import** java.lang.reflect.Field;  
**import** java.lang.reflect.InvocationTargetException;  
**import** java.lang.reflect.Method;  
**import** java.util.Arrays;  
**import** java.util.logging.Level;  
**import** java.util.logging.Logger;  
  
**public** **class** ReflectionTest {  
  
    **public** **static** **void** main(**String** args[]) **throws** [**ClassNotFoundException**](http://javarevisited.blogspot.com/2011/08/classnotfoundexception-in-java-example.html) {  
        
        **Class**<Person> person = ([**Class**](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html)<Person>) **Class**.forName("test.Person");  
        
        *//getFields() does not return private field*  
        **System**.out.println("Fields : " + **Arrays**.toString(person.getFields()));  
        
        *//getDeclaredFields() return both private and non private fields using reflection*  
        **System**.out.println("Declared Fields : " + **Arrays**.toString(person.getDeclaredFields()));         
              
        *//getDeclaredMethods() return both private and non private methods using reflection*  
        **System**.out.println("Declared methods : " + **Arrays**.toString(person.getDeclaredMethods()));  
        
        **try** {  
            
            *//accessing value of private field using reflection in Java*  
            Person privateRyan = **new** Person("John" , "8989736353");  
            **Field** privateField = person.getDeclaredField("phone");  
            
            *//this call allows private fields to be accessed via reflection*  
            privateField.setAccessible(**true**);  
            
            *//getting value of private field using reflection*  
            **String** value = (**String**) privateField.get(privateRyan);             
            
            *//print value of private field using reflection*  
            [**System**](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).out.println("private field: " + privateField + " value: " + value);  
            
            
            *//accessing private method using reflection*  
            **Method** privateMethod = person.getDeclaredMethod("call");  
            
            *//making private method accessible using reflection*  
            privateMethod.setAccessible(**true**);  
            
            *//calling private method using reflection in java*  
            privateMethod.invoke(privateRyan);  
            
            
        } **catch** (**InvocationTargetException** ex) {  
            **Logger**.getLogger(ReflectionTest.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
        } **catch** (**NoSuchMethodException** ex) {  
            **Logger**.getLogger(ReflectionTest.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
        } **catch** (**IllegalArgumentException** ex) {  
            **Logger**.getLogger(ReflectionTest.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
        } **catch** (**IllegalAccessException** ex) {  
            **Logger**.getLogger(ReflectionTest.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
        } **catch** (**NoSuchFieldException** ex) {  
            **Logger**.getLogger(ReflectionTest.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
        } **catch** (**SecurityException** ex) {  
            **Logger**.getLogger(ReflectionTest.**class**.getName()).log(**Level**.SEVERE, **null**, ex);  
        }  
        
        
    }  
}  
  
**class** Person{  
    **public** **String** name;  
    **private** **String** phone;  
    
    **public** Person(**String** name, **String** phone){  
        **this**.name = name;  
        **this**.phone = phone;  
    }  
    
    **private** **void** call(){  
        **System**.out.println("Calling " + **this**.name +" at " + **this**.phone);  
    }  
    
    **public** **String** getName(){  
        **return** name;  
    }  
}  
  
Output:  
Fields : [**public** java.lang.**String** test.Person.name]  
Declared Fields : [**public** java.lang.**String** test.Person.name, **private** java.lang.**String** test.Person.phone]  
Declared methods : [**public** java.lang.**String** test.Person.getName(), **private** **void** test.Person.call()]  
**private** field: **private** java.lang.**String** test.Person.phone value: 8989736353  
Calling John at 8989736353