About Java Reflection

Java Reflection is a one kind of power full mechanism and process of analyzing a particular class at runtime. Reflection API is a set of predefined classes provided by java to perform reflection over a particular class. Reflection is not used in project development , it use only product development.

Such as: Compilers , JVM , Server’s , IDE’s etc.

The java.lang.Class class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

We can learn a lot about a class just by using Java Reflection. Such as : modifiers , method, class , interface , value , data type, constructor etc.

* There are three ways to get the instance of Class class .

Using forName(--) method:

|  |
| --- |
| Test1.java |
| **package** java\_reflection\_example\_part1;  **class** Simple {  }  **public** **class** Test1 {  **public** **static** **void** main(String[] args) **throws** Exception {  Class c = Class.*forName*("java\_reflection\_example\_part1.Simple");  System.***out***.println(c.getName());  }  } |
| **Output:** java\_reflection\_example\_part1.Simple |
| Syntax of forName() method:  firstly we have to create a object.  Class c=Class.forName(“packageName.ClassName”);  Then after using getName() method we get class name. |
| If we get the package name separately then can be use getPackageName() method. |

Using getClass() method:

|  |
| --- |
| Test2.java |
| **package** java\_reflection\_example\_part1;  **class** Simple1 {  }  **public** **class** Test2 {  **public** **static** **void** main(String[] args) {  Simple1 sm = **new** Simple1();  Class c=sm.getClass();  System.***out***.println(c.getName());  System.***out***.println(c.getPackageName());  }  }}  } |
| **Output:**  java\_reflection\_example\_part1.Simple1  java\_reflection\_example\_part1 |
| Syntax of getClass() method:  firstly we have to create a class object.  Simple1 sm=new Simple1();  Class c=classobject.getClass();  Then after using getName() method we get class name. |
| If we get the package name separately then can be use getPackageName() method. |

Using .class file name:

|  |
| --- |
| Test3.java |
| **package** java\_reflection\_example\_part1;  **class** Simple3 {  }  **public** **class** Test3 {  **public** **static** **void** main(String[] args) {  Class c=Simple3.class;  System.***out***.println(c.getName());    }  } |
| **Output:**  java\_reflection\_example\_part1.Simple3 |
| Syntax of .Class() file:  firstly we have to create a class object.  Class c=ClassName.class;  Then after using getName() method we get class name. |
| If we get the package name separately then can be use getPackageName() method. |

|  |
| --- |
| **package** java\_reflection\_example\_part1;  **import** java.lang.reflect.\*;  /\*--  \* in this example we will learn how to get  \* class name , super class , modifier ,  \* interfaces  \* first we declare super class B then  \* declare child Class Employee and use  \* java built in interfaces  \*  \* --\*/  **class** B {  }  **class** Employee **extends** B **implements** java.io.Serializable, Cloneable {  }  **public** **class** Test4 {  **public** **static** **void** main(String[] args) **throws** Exception{  Class c=Class.*forName*(“java\_reflection\_example\_part1.Employee”);    System.***out***.println(“Class Name is :” +c.getName());  System.***out***.println();  System.***out***.println(“Super Class Name is :” +c.getSuperclass().getName());  System.***out***.println();    /\*--  \*interface is a class then we use  \*public Class[] getInterfaces() method  \* --\*/  Class[] cls=c.getInterfaces();  System.***out***.println(“Interfaces is : “);  **for** (Class c1 : cls) {  System.***out***.println(c1.getName());  }  System.***out***.println();    /\*--  \* if we need to know get modifier then  \* use this method  \* public int getModifiers()  \* --\*/  **int** i=c.getModifiers();  System.***out***.println(“Modifier is :” +Modifier.*toString*(i));    }  } |
| **Output is :**  Class Name is :java\_reflection\_example\_part1.Employee  Super Class Name is :java\_reflection\_example\_part1.B  Interfaces is :  java.io.Serializable  java.lang.Cloneable  Modifier is ://no modifiers because we can not use any modifier |

Difference between getField() and getDeclaredField() Methods:

Understood the getField() and getDeclaredField() then follow this example:

|  |  |
| --- | --- |
| **package** java\_reflection\_example\_part2;  **public** **class** Phone {  **public** **static** **long** *phone*=172546789;  } | This is a normal java class.  Here we declare one variable which type is long.  Modifier is public and situation is static. |

|  |  |
| --- | --- |
| **public** **class** Employee **extends** Phone {  **public** **static** **int** *eno*=101;  **static** String *ename*="wornoz";  **private** **static** **int** *salary*=12345;  **public** **static** **final** String ***eaddr***="Dhaka";  } | This is a normal java class.  And Phone class extends this class. |

Now we make a TestClass:

|  |  |
| --- | --- |
| getField() | getDeclaredField() |
| **package** java\_reflection\_example\_part2;  **import** java.lang.reflect.\*;  **public** **class** TestClass {  **public** **static** **void** main(String[] args) **throws** Exception {  Employee e=**new** Employee();  Class c=e.getClass();    Field[] flds=c.getFields();  **for** (Field f : flds) {    System.***out***.println("Field Name is :" +f.getName());  System.***out***.println("Data Type is :" +f.getType().getName());  **int** modi=f.getModifiers();  System.***out***.println("Modifier is :" +Modifier.*toString*(modi));  System.***out***.println("Value is :" +f.get(f));  System.***out***.println("---------------------------------");  }  }  } | **package** java\_reflection\_example\_part2;  **import** java.lang.reflect.\*;  **public** **class** TestClass {  **public** **static** **void** main(String[] args) **throws** Exception {  Employee e=**new** Employee();  Class c=e.getClass();    Field[] flds=c.getDeclaredFields();  **for** (Field f : flds) {    System.***out***.println("Field Name is :" +f.getName());  System.***out***.println("Data Type is :" +f.getType().getName());  **int** modi=f.getModifiers();  System.***out***.println("Modifier is :" +Modifier.*toString*(modi));  System.***out***.println("Value is :" +f.get(f));  System.***out***.println("---------------------------------");  }  }  } |
| Field Name is :eno  Data Type is :int  Modifier is :public static  Value is :101  ---------------------------------  Field Name is :eaddr  Data Type is :java.lang.String  Modifier is :public static final  Value is :Dhaka  ---------------------------------  Field Name is :phone  Data Type is :long  Modifier is :public static  Value is :172546789  --------------------------------- | Field Name is :eno  Data Type is :int  Modifier is :public static  Value is :101  ---------------------------------  Field Name is :ename  Data Type is :java.lang.String  Modifier is :static  Value is :wornoz  ---------------------------------  Field Name is :eaddr  Data Type is :java.lang.String  Modifier is :public static final  Value is :Dhaka  --------------------------------- |
| If we using this method then we will get parent class and child class value. But we get only public value. | If we use this method, we get the values of the class of the object we created (child class). We can not get parent class value.  If we use private modifier , we can show only data type , modifier . but we can not show private value.  If we want to show private value , we get java.lang.IllegalAccessException error. |

Now we learn about method :

|  |  |
| --- | --- |
| **package** java\_reflection\_example\_method\_part3;  **public** **class** Employee {  **public** **void** add(**int** eno, String ename, String eaddr) **throws** ClassNotFoundException {  }  **public** String search() **throws** ArithmeticException {  **return** "success";  }    **public** **void** delete(**int** eno) {    }  } | This is a java class . we declare some method. Method return type , parameter , exception.  Now we found the everything all this method what we use here. |

|  |  |
| --- | --- |
| **package** java\_reflection\_example\_method\_part3;  **import** java.lang.reflect.\*;  **public** **class** TestClass {  **public** **static** **void** main(String[] args) {  Class c = Employee.**class**;  Method[] mthd = c.getDeclaredMethods();  **for** (Method m1 : mthd) {  System.***out***.println("Method Name is :" + m1.getName());  System.***out***.println("Method Return Type is :" + m1.getReturnType().getName());  **int** modi = m1.getModifiers();  System.***out***.println("Modifier is :" + Modifier.*toString*(modi));  Class[] parameter = m1.getParameterTypes();  System.***out***.println("Parameter Type is :");  **for** (Class p1 : parameter) {  System.***out***.println(p1.getName());  }  Class[] except = m1.getExceptionTypes();  System.***out***.println("Exception Type is : ");  **for** (Class e1 : except) {  System.***out***.println(e1.getName());  }  System.***out***.println("---------------------------------");  }  }  } | Here we using .class file method.  If we want to get all the methods metadata in the form of method[],first we have to get java.lang.Class object then we have to use either of the following methods.  Public Method[] getMethods()  It can be used to get all the methods metadata which are declared as public in the respective class and in the super class.  public Method[] getDecalredMethods()  if we get method return type then use  getReturnType() method and getName() method.  If we get method parameter type then use : getParameterType() method and getName() method.  If we get method exception type then use : getExceptionType() method and getName() method. |
| Output is:  Method Name is :add  Method Return Type is :void  Modifier is :private  Parameter Type is :  int  java.lang.String  java.lang.String  Exception Type is :  java.lang.ClassNotFoundException  ---------------------------------  Method Name is :delete  Method Return Type is :void  Modifier is :public  Parameter Type is :  int  Exception Type is :  ---------------------------------  Method Name is :search  Method Return Type is :java.lang.String  Modifier is :public  Parameter Type is :  Exception Type is :  java.lang.ArithmeticException  --------------------------------- | Using getDeclaredMethod() we can get public or private method own class.  If we get parent class public method than we will do override method and use annotation @Override. |

Now we learn about Constructor :

|  |
| --- |
| **package** java\_reflection\_example\_constructor\_part4;  **public** **class** Employee {  **public** Employee(**int** eno, String ename, String eaddr) **throws** ClassCastException, ArithmeticException {  }  **public** Employee(**int** eno, String ename) **throws** java.rmi.RemoteException, java.sql.SQLException {  }  **public** Employee(**int** eno) **throws** InterruptedException {  }  } |

|  |
| --- |
| **package** java\_reflection\_example\_constructor\_part4;  **import** java.lang.reflect.\*;  **public** **class** TestClass {  **public** **static** **void** main(String[] args) {  Class c = Employee.**class**;  Constructor[] con = c.getDeclaredConstructors();  **for** (Constructor c1 : con) {  System.***out***.println("Constructor Name is :" + c1.getName());  **int** modi = c1.getModifiers();  System.***out***.println("Modifier is :" + Modifier.*toString*(modi));  Class[] cls = c1.getParameterTypes();  System.***out***.println("Parameter Type is :");  **for** (Class c2 : cls) {  System.***out***.println(c2.getName() + " ");  }  Class[] cls2 = c1.getExceptionTypes();  System.***out***.println("Exception Type is :");  **for** (Class c3 : cls2) {  System.***out***.println(c3.getName());  }  System.***out***.println("--------------------------------------------");  }  }  } |
| Parameter Type is :  int  java.lang.String  java.lang.String  Exception Type is :  java.lang.ClassCastException  java.lang.ArithmeticException  ----------------------------------------------  Constructor Name is :java\_reflection\_example\_constructor\_part4.Employee  Modifier is :public  Parameter Type is :  int  java.lang.String  Exception Type is :  java.rmi.RemoteException  java.sql.SQLException  ----------------------------------------------  Constructor Name is :java\_reflection\_example\_constructor\_part4.Employee  Modifier is :public  Parameter Type is :  int  Exception Type is :  java.lang.InterruptedException  ---------------------------------------------- |
| To get all the constructors metadata of a particular class first we have to get Class object then we have to use the following methods.  Public Constructor[] getConstructor()  It will return only public constructor details from the respective class  Public Constructor[] getDeclaredConstructors()  It will return all the constructor metadata which are public.  Same as method. |