JBK1017-Reflection

A simple Java program to demonstrate the use of reflection

```
import java.lang.reflect.Method;
import java.lang.reflect.Field;
import java.lang.reflect.Constructor;
// class whose object is to be created
class Test{
   // creating a private field
   private String s;

   // creating a public constructor
   public Test()  {
   s = "GeeksforGeeks";
    }
   // Creating a public method with no
argumentsjkllkkkkkkkkkkkkkkkkkk4z\
   public void method1()  {
      System.out.println("The string is " + s);
   }
   // Creating a public method with int as argument
   public void method2(int n)  {
      System.out.println("The number is " + n);
   }
   // creating a private method
   private void method3() {
      System.out.println("Private method invoked");
   }}
class Demo{
   public static void main(String args[]) throws Exception {
      // Creating object whose property is to be checked
      Test obj = new Test();

      // Creating class object from the object using
      // getclass method
      Class cls = obj.getClass();
      System.out.println("The name of class is " +cls.getName());

      // Getting the constructor of the class through the
      // object of the class
      Constructor constructor = cls.getConstructor();
      System.out.println("The name of constructor is
"+ constructor.getName());
```

```
      System.out.println("The public methods of class are : ");

      // Getting methods of the class through the object
      // of the class by using getMethods
      Method[] methods = cls.getMethods();

      // Printing method names
      for (Method method:methods)
         System.out.println(method.getName());

      // creates object of desired method by providing the
      // method name and parameter class as arguments to
      // the getDeclaredMethod
      Method methodcall1 =
cls.getDeclaredMethod("method2", int.class);

      // invokes the method at runtime
      methodcall1.invoke(obj, 19);

      // creates object of the desired field by providing
      // the name of field as argument to the
      // getDeclaredField method
      Field field = cls.getDeclaredField("s");

      // allows the object to access the field irrespective
      // of the access specifier used with the field
      field.setAccessible(true);

      // takes object and the new value to be assigned
      // to the field as arguments
      field.set(obj, "JAVA");

      // Creates object of desired method by providing the
      // method name as argument to the getDeclaredMethod
      Method methodcall2 = cls.getDeclaredMethod("method1");

      // invokes the method at runtime
      methodcall2.invoke(obj);

      // Creates object of the desired method by providing
      // the name of method as argument to the
      // getDeclaredMethod method
      Method methodcall3 = cls.getDeclaredMethod("method3");
      // allows the object to access the method irrespective
      // of the access specifier used with the method
      methodcall3.setAccessible(true);

      // invokes the method at runtime
      methodcall3.invoke(obj);
   }}
```

Example on reflection ,load the Java class, call its methods or analysis the class at runtime.

```
public class AppTest{
private int counter;
public void printIt(){
                System.out.println("printIt() no param");
        }
        public void printItString(String temp){
                System.out.println("printIt() with param String : " +
temp);
        }
        public void printItInt(int temp){
                System.out.println("printIt() with param int : " +
temp);
        }
        public void setCounter(int counter){
                this.counter = counter;
                System.out.println("setCounter() set counter to : " +
counter);
        }
        public void printCounter(){
                System.out.println("printCounter() : " + this.counter);
        }}
import java.lang.reflect.Method;
public class ReflectApp{
        public static void main(String[] args) {

        //no paramater
        Class noparams[] = {};

        //String parameter
        Class[] paramString = new Class[1];
        paramString[0] = String.class;
```

```
          //int parameter
          Class[] paramInt = new Class[1];
          paramInt[0] = Integer.TYPE;
          try{
                //load the AppTest at runtime
                    Class cls =
Class.forName("com.mkyong.reflection.AppTest");
                    Object obj = cls.newInstance();

                    //call the printIt method
          Method method = cls.getDeclaredMethod("printIt",
noparams);
                    method.invoke(obj, null);

                    //call the printItString method, pass a String param
          method = cls.getDeclaredMethod("printItString",
paramString);
                    method.invoke(obj, new String("mkyong"));

                    //call the printItInt method, pass a int param
                    method = cls.getDeclaredMethod("printItInt",
paramInt);
                    method.invoke(obj, 123);

                    //call the setCounter method, pass a int param
                    method = cls.getDeclaredMethod("setCounter",
paramInt);
                    method.invoke(obj, 999);

                    //call the printCounter method
                    method = cls.getDeclaredMethod("printCounter",
noparams);
                    method.invoke(obj, null);
          }catch(Exception ex){
                    ex.printStackTrace();
          }}}
```

Finding Out About Methods of a Class

```
import java.lang.reflect.*;

  public class method1 {
    private int f1(
     Object p, int x) throws NullPointerException     {
      if (p == null)
        throw new NullPointerException();
      return x;
    }
    public static void main(String args[]){

        Class cls = Class.forName("method1");
         Method methlist[]
           = cls.getDeclaredMethods();
         for (int i = 0; i < methlist.length;
           i++) {
           Method m = methlist[i];
           System.out.println("name
             = " + m.getName());
           System.out.println("decl class = " +
                     m.getDeclaringClass());
           Class pvec[] = m.getParameterTypes();
           for (int j = 0; j < pvec.length; j++)
             System.out.println("
              param #" + j + " " + pvec[j]);
           Class evec[] = m.getExceptionTypes();
           for (int j = 0; j < evec.length; j++)
             System.out.println("exc #" + j
               + " " + evec[j]);
           System.out.println("return type = " +
                        m.getReturnType());
           System.out.println("-----");
         }}}
```

Obtaining Information About Constructors

```
import java.lang.reflect.*;

  public class constructor1 {
    public constructor1( {
    }

    protected constructor1(int i, double d)  {
    }
    public static void main(String args[]){
        Class cls = Class.forName("constructor1");
        Constructor ctorlist[]= cls.getDeclaredConstructors();
       for (int i = 0; i < ctorlist.length; i++) {
           Constructor ct = ctorlist[i];
           System.out.println("name= " + ct.getName());
           System.out.println("decl class = " + ct.getDeclaringClass());
           Class pvec[] = ct.getParameterTypes();
           for (int j = 0; j < pvec.length; j++)
             System.out.println("param #"+ j + " " + pvec[j]);
           Class evec[] = ct.getExceptionTypes();
           for (int j = 0; j < evec.length; j++)
             System.out.println("exc #" + j + " " + evec[j]);
           System.out.println("-----");
         } }}
```

Finding Out About Class Fields

```
import java.lang.reflect.*;
  public class field1 {
    private double d;
    public static final int i = 37;
    String s = "testing";

    public static void main(String args[]){
        Class cls = Class.forName("field1");

        Field fieldlist[]
          = cls.getDeclaredFields();
        for (int i
          = 0; i < fieldlist.length; i++) {
          Field fld = fieldlist[i];
          System.out.println("name
            = " + fld.getName());
          System.out.println("decl class = " +
                   fld.getDeclaringClass());
          System.out.println("type
            = " + fld.getType());
          int mod = fld.getModifiers();
          System.out.println("modifiers = " +
                  Modifier.toString(mod));
          System.out.println("-----");
        } } }
```

Invoking Methods by Name

```
import java.lang.reflect.*;
  public class method2 {
    public int add(int a, int b){
      return a + b;
    }
    public static void main(String args[]){
        Class cls = Class.forName("method2");
        Class partypes[] = new Class[2];
         partypes[0] = Integer.TYPE;
         partypes[1] = Integer.TYPE;
         Method meth = cls.getMethod(
           "add", partypes);
         method2 methobj = new method2();
         Object arglist[] = new Object[2];
         arglist[0] = new Integer(37);
         arglist[1] = new Integer(47);
         Object retobj
           = meth.invoke(methobj, arglist);
         Integer retval = (Integer)retobj;
         System.out.println(retval.intValue());
      } }
```

Creating New Objects

```
import java.lang.reflect.*;
  public class constructor2 {
    public constructor2(){
    }
    public constructor2(int a, int b){
      System.out.println(
        "a = " + a + " b = " + b);
    }
    public static void main(String args[]){
      try {
        Class cls = Class.forName("constructor2");
        Class partypes[] = new Class[2];
         partypes[0] = Integer.TYPE;
         partypes[1] = Integer.TYPE;
         Constructor ct
           = cls.getConstructor(partypes);
         Object arglist[] = new Object[2];
         arglist[0] = new Integer(37);
         arglist[1] = new Integer(47);
         Object retobj = ct.newInstance(arglist);
      }
      catch (Throwable e) {
        System.err.println(e);
      } } }
```

Changing Values of Fields

```
import java.lang.reflect.*;
  public class field2 {
    public double d;
    public static void main(String args[])  {
      try {
        Class cls = Class.forName("field2");
        Field fld = cls.getField("d");
        field2 f2obj = new field2();
        System.out.println("d = " + f2obj.d);
        fld.setDouble(f2obj, 12.34);
        System.out.println("d = " + f2obj.d);
      }
      catch (Throwable e) {
        System.err.println(e);
      }}}
```

reflection is in creating and manipulating arrays

```
import java.lang.reflect.*;
  public class array2 {
    public static void main(String args[]){
      int dims[] = new int[]{5, 10, 15};
      Object arr
       = Array.newInstance(Integer.TYPE, dims);

      Object arrobj = Array.get(arr, 3);
      Class cls =
       arrobj.getClass().getComponentType();
      System.out.println(cls);
      arrobj = Array.get(arrobj, 5);
      Array.setInt(arrobj, 10, 37);
      int arrcast[][][] = (int[][][])arr;
      System.out.println(arrcast[3][5][10]);
    }  }
```