JAVA REFLECTION

JAVA LOOKING AT JAVA

One of the unusual capabilities of Java is that a program can examine itself

- You can determine the class of an object
- You can find out all about a class: its access modifiers, superclass, fields, constructors, and methods
- You can find out what is there in an interface
- Even if you don't know the names of things when you write the program, you can:
  - Create an instance of a class
  - Get and set instance variables
  - Invoke a method on an object
  - Create and manipulate arrays

WHAT IS REFLECTION FOR?

In “normal” programs you don’t need reflection

You *do* need reflection if you are working with programs that process programs

Typical examples:

- A class browser
- A debugger
- A GUI builder
- An IDE, such Netbeans or eclipse
- A program to grade student programs

USING REFLECTION TO ANALYZE THE CAPABILITIES OF CLASSES

The three classes Field, Method, and Constructor in the java.lang.reflect package describe the fields, methods, and constructors of a class, respectively

All three classes have a method called getName() that returns the name of the item. The Field class has a method getType() that returns an object, again of type Class, that describes the field type

The Method and Constructor classes have methods to report the types of the parameters, and the Method class also reports the return type

JAVA.LANG.CLASS CLASS

- The java.lang.Class class performs mainly two tasks
  - provides methods to get the metadata of a class at run time.
  - provides methods to examine and change the run time behavior of a class.

COMMONLY USED METHODS OF CLASS CLASS:

Method	Description
1) public String getName()	returns the class name
2) public static Class forName(String className)throws ClassNotFoundException	loads the class and returns the reference of Class class.
3) public Object newInstance()throws InstantiationException,IllegalAccessException	creates new instance.
4) public boolean isInterface()	checks if it is interface.
5) public boolean isArray()	checks if it is array.
6) public boolean isPrimitive()	checks if it is primitive.
7) public Class getSuperclass()	returns the superclass class reference.

COMMONLY USED METHODS OF CLASS CLASS:

8) public Field[] getDeclaredFields()throws SecurityException	returns the total number of fields of this class.
9) public Method[] getDeclaredMethods()throws SecurityException	returns the total number of methods of this class.
10) public Constructor[] getDeclaredConstructors()throws SecurityException	returns the total number of constructors of this class.
11) public Method getDeclaredMethod(String name,Class[] parameterTypes)throws NoSuchMethodException,SecurityException	returns the method class instance.

HOW TO GET THE OBJECT OF CLASS CLASS?

- There are 3 ways to get the instance of Class class. They are as follows:
  - forName() method of Class class
  - getClass() method of Object class
  - the .class syntax

1. FORNAME() METHOD OF CLASS CLASS

- is used to load the class dynamically.
- returns the instance of Class class.
- It should be used if you know the fully qualified name of class.

1. FORNAME() METHOD OF CLASS CLASS

```
class Simple{}

public class Test{
 public static void main(String args[]) throws Exception
 {
  Class c=Class.forName("Simple");
  System.out.println(c.getName());
 }
}
```

```
Simple
Press any key to continue . . .
```

2. GETCLASS() METHOD OF OBJECT CLASS

- It returns the instance of Class class. It should be used if you know the type. Moreover, it can be used with primitives.

```
class Simple{}

class Test1{
  void printName(Object obj){
  Class c=obj.getClass();
  System.out.println(c.getName());
  }
  public static void main(String args[]){
   Simple s=new Simple();

   Test1 t=new Test1();
   t.printName(s);
 }
}
```

```
Simple
Press any key to continue . . .
```

3. THE .CLASS SYNTAX

- If a type is available, but there is no instance, then it is possible to obtain a Class by appending ".class" to the name of the type.

```
class Test3{
  public static void main(String args[]){
   Class c = boolean.class;
   System.out.println(c.getName());

   Class c2 = Test.class;
   System.out.println(c2.getName());
 }
}
```

```
boolean
Test
Press any key to continue . . .
```

DETERMINING THE CLASS OBJECT

- The following methods of Class class are used to determine the class object:

- 1) public boolean isInterface(): determines if the specified Class object represents an interface type.
- 2) public boolean isArray(): determines if this Class object represents an array class.
- 3) public boolean isPrimitive(): determines if the specified Class object represents a primitive type.

DETERMINING THE CLASS OBJECT

```
class Simple{}
interface My{}

class Test4{
 public static void main(String args[]){
  try{
   Class c=Class.forName("Simple");
   System.out.println(c.isInterface());

   Class c2=Class.forName("My");
   System.out.println(c2.isInterface());

  }catch(Exception e){System.out.println(e);}

 }
}
```

```
false
true
Press any key to continue . . .
```

CONT..

All three of these classes also have a method called getModifiers() that returns an integer, with various bits turned on and off, that describes the modifiers used, such as public and static

Use methods like isPublic(), isPrivate(), or isFinal() in the Modifier class to tell whether a method or constructor was public, private, or final.

You can also use the Modifier.toString() method to print the modifiers

CONT..

The getFields(), getMethods(), and getConstructors() methods of the Class class return arrays of the public fields, methods, and constructors that the class supports. This includes public members of superclasses

The getdeclaredFields(), geTDeclaredMethods(), and geTDeclaredConstructors() methods of the Class class return arrays consisting of all fields, operations, and constructors that are declared in the class. This includes private and protected members, but not members of superclasses

ADVANTAGE AND DISADVANTAGE OF REFLECTION

- Advantages:
- Inspection of interfaces, classes, methods, and fields during runtime is possible using reflection, even without using their names during the compile time.
- It helps in the creation of Visual Development Environments
- Disadvantages:
- Using reflection, one can break the principles of encapsulation. It is possible to access the private methods and fields of a class using reflection. Thus, reflection may leak important data to the outside world, which is dangerous.
- For example, if one access the private members of a class and sets null value to it, then the other user of the same class can get the NullReferenceException, and this behaviour is not expected.