Java Reflection

- **Reflection** is a feature in the Java programming language.
- It allows an executing Java program to examine or "**introspect**" upon itself, and manipulate internal properties of the program.

  For example: it's possible for a Java class to obtain the names of all its members and display them.

- The ability to examine and manipulate a Java class from within itself may not sound like very much, but in other programming languages this feature simply doesn't exist.

  For example: there is no way in a **Pascal, C, or C++** program to obtain information about the functions defined within that program.

Java Reflection (Cont..)

- ✓ One tangible use of reflection is in **JavaBeans**, where software components can be manipulated visually via a builder tool.
- ✓ The tool uses reflection to obtain the properties of Java components (**classes**) as they are dynamically loaded.
- ✓ Reflection allows us to inspect and manipulate classes, fields, methods, and constructors during runtime.
- ✓ It provides a way to work with Java classes dynamically, making it a versatile tool for tasks like introspection, testing, debugging, and even creating flexible frameworks.

Setting Up to Use Reflection

The reflection classes, such as **Method**, are found in **java.lang.reflect.**

There are **three steps** that must be followed to use these classes.

1. The first step is to obtain a **java.lang.Class** object for the class that you want to manipulate.

   java.lang.Class is used to represent **classes** and **interfaces** in a running Java program.

Setting Up to Use Reflection (Cont..)

One way of obtaining a Class object is to say:

Class c = Class.forName("java.lang.String");

to get the Class object for String.

Another approach is to use:

Class c = int.class; or Class c = Integer.TYPE;

to obtain Class information on fundamental types.

The latter approach accesses the predefined TYPE field of the wrapper (such as **Integer**) for the fundamental type.

Setting Up to Use Reflection (Cont..)

2. The second step is to call a method such as **getDeclaredMethods**, to get a list of all the methods declared by the class.

3. The third step is to use the **reflection API** to manipulate the information.

For example, the sequence:

```
Class  c  =  Class.forName("java.lang.String");
Method m[ ]  =  c.getDeclaredMethods();
System.out.println(m[0].toString());
```

will display a **textual** representation of the first method declared in **String**.

Classes defined in **java.lang.reflect**

1. **Array**: It allows us to dynamically create and manipulate arrays.
2. **Constructor**: It provides us information about a constructor.
3. **Field**: It provides us information about a field.
4. **Method**: It provides us information about a method.
5. **Modifier**: It provides us information about the class and member access modifiers.
6. **ReflectPermission**: It allows us the reflection on private or protected members of a class.

Methods of java.lang.reflect

1. **getName**(): It returns the name of the class.
2. **getSuperclass**(): It returns the superclass reference
3. **getInterfaces**(): It returns an array of interfaces implemented by the specified class
4. **getModifiers**(): It returns an int containing flags that describe which modifiers apply for the given class.
5. **getDeclaredMethod**(): It is used to create an object of the method to be invoked.
6. **invoke**(): It is used to invoke a method of the class at runtime.
7. **getDeclaredField(FieldName)**: It is used to get the private field. It returns an object of type Field for the specified field name.

Methods of java.lang.reflect

8. **setAccessible(true):** It allows us to access the field irrespective of the access modifier used with the field.
9. **newInstance():** It is used to create a new instance of the class. The **newInstance**() method of a **Class** class can invoke zero-argument constructor whereas the **newInstance**() method of Constructor class can invoke any number of arguments.

Advantages of Reflection in Java

1. **Extensibility Features:** An application may use external, user-defined classes by creating instances of extensibility objects using their fully-qualified names.
2. **Class Browsers and Visual Development Environments**: A class browser must be ready to enumerate the members of classes. Visual development environments can enjoy making use of type information available in reflection to assist the developer in writing the correct code.
3. **Debuggers and Test Tools:** Debuggers got to be ready to examine private members in classes. Test harnesses can make use of reflection to systematically call a discoverable set of APIs defined on a category, to ensure a high level of code coverage during a test suite.

Disadvantages of Reflection in Java

1. **Performance Overhead:** Certain Java virtual machine optimizations cannot be performed because reflection involves types that are dynamically resolved.
2. **Security Restrictions:** When running under a security manager, reflection requires a runtime permission which may not be present. This is in a crucial consideration for code that has got to run during a restricted security context, like in an Applet.
3. **Exposure of Internals:** The use of reflection can result in unexpected side-effects because reflection allows code to perform operations that would be illegal in non-reflective code, such as accessing private fields and methods, which can render code dysfunctional and should destroy portability. Reflective code breaks abstractions and thus may change behavior with upgrades of the platform.