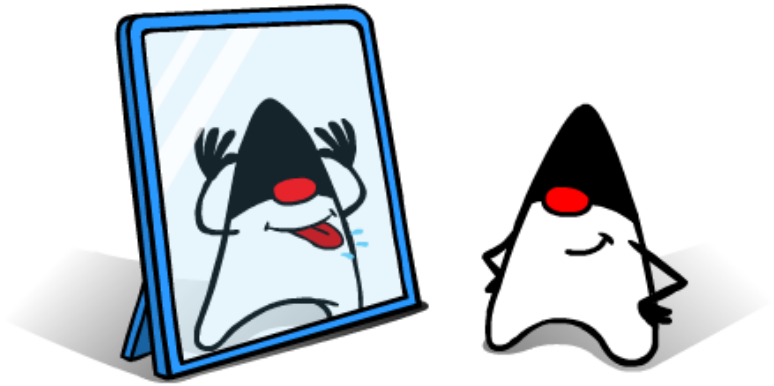
Java reflection



alberto ferrari – university of parma

reflection

- metaprogramming is a programming technique in which computer programs have the ability to treat programs as their data
  - a program can be designed to read, generate, analyse or transform other programs, and even modify itself while running
  - the language in which the metaprogram is written is called the metalanguage
  - the ability of a programming language to be its own metalanguage is called reflection(1)
- reflection is the ability of a computer program to **examine**, **introspect**, and **modify** its own structure and behavior at **runtime**

(1) http://www.giulioangiani.com/programming/metaprogramming

uses of reflection

- reflection is used by programs to **examine or modify the runtime behavior** of applications
- is a **powerful** technique and can enable applications to perform operations which would **otherwise** be **impossible**
- an application may make use of external, user-defined classes by **creating instances** of objects using their names
- **Class Browsers** and Visual Development Environments
  - a class browser needs to be able to **enumerate the members** of classes
- **Debuggers** and Test Tools
  - debuggers need to be able to **examine private members** on classes

ORACLE Java Documentation

drawbacks

- reflection is **powerful**, but should not be used **indiscriminately**
- *if it is possible to perform an operation without using reflection, then it is preferable to avoid using it*
- performance overhead
  - reflective operations have slower performance than their non-reflective counterparts
- security restrictions
  - reflection requires a runtime permission which may not be present when running under a security manager
- exposure of internals
  - reflection allows code to perform operations that would be **illegal** in non-reflective code, such as accessing private fields and methods

Java reflection

- for every type of object, the Java virtual machine instantiates an immutable instance of **java.lang.Class** which provides methods to examine the runtime properties of the object including its members and type information
- Class also provides the ability to create new classes and objects
- it is the entry point for all of the Reflection APIs

Class methods

Class Methods for Locating Fields

Class API	List of members?	Inherited members?	Private members?
getDeclaredField()	no	no	yes
getField()	no	yes	no
getDeclaredFields()	yes	no	yes
getFields()	yes	yes	no

Class Methods for Locating Methods

Class API	List of members?	Inherited members?	Private members?
getDeclaredMethod()	no	no	yes
getMethod()	no	yes	no
getDeclaredMethods()	yes	no	yes
getMethods()	yes	yes	no

Class Methods for Locating Constructors

Class API	List of members?	Inherited members?	Private members?
getDeclaredConstructor()	no	N/A[1]	yes
getConstructor()	no	N/A[1]	no
getDeclaredConstructors()	yes	N/A[1]	yes
getConstructors()	yes	N/A[1]	no

example: user defined Dummy class

```
public class Dummy {
    private String foo;
    protected int bar;
    public char baz;
    public Dummy() {
        foo = "You can't modify me, I'm private :)";
        bar = 1;
        baz = 'x';
    }
    public String getFoo() {
        return "foo value: "+foo;
    }
    protected void incBar() {
        bar++;
    }
    private char setAndGetBaz(char c) {
        baz=c;
        return baz;
    }
}
```

typical Java programmer

```
public static void main(String[] args) {

    String result;
    Dummy d;                          // Object declaration
    d = new Dummy();                  // Object instantiation
    result = d.getFoo();              // Call object method
    System.out.println(result);

}
```

```
> foo value: You can't modify me, I'm private :)
```

hacking (white hat)

```
import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
...
try {

    Class<?> D = Class.forName("Dummy");  // Get the class
    Dummy dwh;                            // Object declaration
    dwh = (Dummy) D.newInstance();        // Object instantiation
    Method method = D.getMethod("getFoo", null);// Get method
    result = (String) method.invoke(dwh, null); // Call method
    System.out.println(result);

} catch (Exception e) {
    e.printStackTrace();
}
```

```
> foo value: You can't modify me, I'm private :)
```

hacking

```
Dummy d = new Dummy;
System.out.println("class name = " +
     d.getClass().getSimpleName());  // Get class name
System.out.println("--- Fields ---");
// Get all declared fields
for (Field s : d.getClass().getDeclaredFields()) {
    System.out.println(s);
}
System.out.println("--- Methods ---");
Method[] methods = d.getClass().getDeclaredMethods();
// Get all declared methods
for (Method m : methods) {
    System.out.println(m);
}
```

```
> class name = Dummy
> --- Fields ---
> private java.lang.String Dummy.foo
> protected int Dummy.bar
> public char Dummy.baz
> --- Methods ---
> protected void Dummy.incBar()
> private char Dummy.setAndGetBaz(char)
> public java.lang.String Dummy.getFoo()
```

hacking (black hat)

```
Dummy obj = new Dummy();
Field objField;
try {
    objField = obj.getClass().getDeclaredField("foo");
    objField.setAccessible(true); // set field foo accessible
    // without previous line:java.lang.IllegalAccessException:
    // Class Main can not access a member of class Dummy
    // with modifiers "private"
    System.out.println("The value of foo is:  " +
            objField.get(obj)); // get filed value
    objField.set( obj, " ... modified :(" );// set field value
} catch (Exception e) {
    System.out.println("Exception: " + e);
}
System.out.println(obj.getFoo());
```

```
> The value of foo is:  You can't modify me, I'm private :)
> foo value:  ... modified :(
```

references

- **Oracle – The reflection API** - https://docs.oracle.com/javase/tutorial/reflect/