UNIVERSITÄT SALZBURG

Java Reflection

Andreas Naderlinger , 2008

Meta-Programming

- The discipline of writing programs that represent and manipulate other programs or themselves. [Czarnecki and Eisenecker]

- object-language: language of the program that is manipulated.

- meta-language: the language in which the metaprogram is written. (e.g. EBNF, XSL, …)

Reflection

- meta-language = object-language
- weak form of meta-programming:
  - structures are read-only
  - values can be changed
- *meta-information*
  - keeps knowledge of program structure
    - the name of the class, the name of parent classes, fields , methods, …

Run-time vs. Compile-time

- Discover and modify source code constructs during execution
- Convert a string matching a symbolic name into a reference or invocation
- Execute a string as if it were a source code statement

Applications

- IDE (Eclipse, ...)
- Debugger

- Dynamic Computing (Plug-ins)
- Serialization
- Dynamically generate GUIs from XML descriptions
- …

Reflection in Java

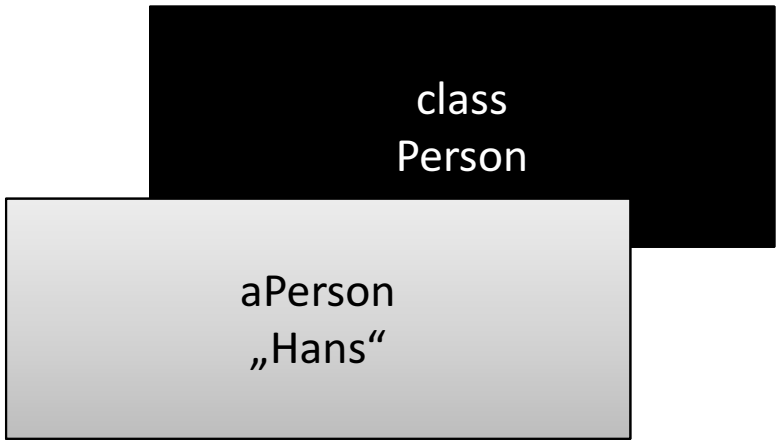
- to inspect classes, interfaces, fields and methods at runtime, without knowing their names at compile time
- to set fields known by name
- invoke methods known by name
- to get an instance (an object) of a class known by name

- java.lang.Class
- java.lang.reflect.* (since 1.1)

java.lang.Class

```
public class Person{
  String name;
  //...
}

Person aPerson
   = new Person(„Hans“);
```



- instances of the class *Class* represent classes and interfaces in a running Java application [javadoc]; they are created by the class-loader.
- Class provides meta-information

Class

- how to get the Class object?

```
Class personClass = Person.class;
Class personClass = aPerson.getClass();
Class personClass = Class.forName(„Person“);
```

- for primitive types

```
Class c = Double.TYPE;
```

java.lang.reflect

- Field
- Method
- Constructor
- …

Examples - Fields

```
Field[] fields = aClass.getFields();
```

```
Person p = new Person("Hans");
Class c = p.getClass();
Field f = c.getField("name");
Object value = f.get(p); //current value
f.set(p, "Lukas"); //set new value
```

what about private members?

```
aClass.getDeclaredFields()
```

```
f.setAccessible(true);
```

Examples – Methods, etc.

```
Method[] methods = aClass.getMethods();


Method m = c.getMethod(“getName”);
Object name = m.invoke(person);


Package package = aClass.getPackage();
aClass.isInterface();
Class[] interfaces = aClass.getInterfaces();
```

Examples - Instances

```
Class c = Class.forName(“Person”);

Object o = c.newInstance();
```

Many things can go wrong

- ClassNotFoundException
- IllegalAccessException
- InstantiationException
- NoSuchFieldException
- NoSuchMethodException
- InvocationTargetException
- ...

Drawbacks

- performance
- try/catch
- security restrictions
- difficult to read
- difficult to debug
- unexpected side-effects

→ “**With great power comes great responsibility!**”

;-)

Links

- java.sun.com/docs/books/tutorial/reflect

- https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/package-summary.html