



Victoria University
of Wellington, New Zealand
*Te Whare Wananga o te
Upoko o te Ika a Maui
Aotearoa*



SWEN221: Software Development

19: Reflection

Outline

- Why Reflection
- Get class information
 - Modifier, types, ...
- Discover class members
 - Fields
 - Methods
 - Constructors

Why Reflection?

- How to check if two objects are equal?

```
public class Point {  
    double x, y;  
  
    public boolean equals(Object o) {  
        if (x != o.x) return false;  
        return y == o.y;  
    }  
}
```

- `o` has to belong to the same class as `this`
- Need to know the class information (`metadata`) of `o`

Reflection

- Reflection in Java
 - Java provides `java.lang.Class`
 - <http://docs.oracle.com/javase/7/docs/api/java/lang/Class.html>
 - This represents class information
 - Each object associated with unique instance of `Class`
 - Can find out about an object by inspecting its `Class` field (`o.getClass()`)

Reflection + Metadata

- Reflection gives access to *metadata*
 - That is, data about data
 - In this case, the data describes our classes
 - We can find out:
 - What an object's class is
 - What methods that class has (inc. private + protected)
 - What their parameter / return types are
 - What fields that class has (inc. private + protected)
 - What their types are
 - What interfaces the class implements
 - What class it extends from

The java.lang.Class

- A lot of methods for these purposes
 - `getName () : java.lang.String, ...`
 - `getSimpleName () : String, ...`
 - `getDeclaredMethods ()`
 - `getDeclaredFields ()`
 - ...
- More details
 - <http://docs.oracle.com/javase/7/docs/api/java/lang/Class.html>

Reflection for equals()

- How to check if two objects are equal?

```
public class Point {  
    double x, y;  
  
    public boolean equals(Object o) {  
        if (this == o) return true;  
        if (o == null) return false;  
        if (getClass() != o.getClass()) return false;  
  
        Point p = (Point) o;  
        if (x != p.x) return false;  
        return y == p.y;  
    }  
}
```

What is Reflection?

In computer science, reflection is the ability of a program to examine and possibly modify its high level structure at runtime

-- Wikipedia

Reflection is commonly used by programs which require the ability to examine or modify the runtime behavior of applications running in the Java virtual machine.

-- Oracle Documentation

Class Objects

```
String s1 = new String("X");  
String s2 = new String("Y");  
Integer i1 = new Integer(1);  
Class c1 = s1.getClass();  
Class c2 = s2.getClass();  
Class c3 = i1.getClass();  
  
System.out.println("c1 is a " + c1.getName());  
System.out.println("c3 is a " + c3.getName());  
  
System.out.println(c1 == c2);  
System.out.println(c1 == c3);
```

Output:

```
c1 is a java.lang.String  
c3 is a java.lang.Integer  
true  
false
```

Generic Type in Class

- A class of what?

java.lang

Class Class<T>

java.lang.Object

java.lang.Class<T>

Type Parameters:

T - the type of the class modeled by this Class object. For example,

All Implemented Interfaces:

Serializable, AnnotatedElement, GenericDeclaration, Type

getClass

```
public final Class<?> getClass()
```

Returns the runtime class of this Object. The returned

The actual result type is **Class<? extends |X|>**

Class Objects

```
String s1 = new String("X");  
String s2 = new String("Y");
```

A

```
Class<String> c1 = s1.getClass();
```

B

```
Class<? extends String> c2 = s2.getClass();
```

Which can work?

A

B

Both

Neither

Class Objects

```
String s1 = new String("X");  
String s2 = new String("Y");
```

A

```
Class<String> c1 = s1.getClass();
```

B

```
Class<? extends String> c2 = s2.getClass();
```

Which can work?

A 

B 

getClass

```
public final Class<?> getClass()
```

Returns the runtime class of this Object. The return

The actual result type is `Class<? extends |x|>`

Class Objects

- **.class** syntax to get the class without giving instance

```
System.out.println(String.class == "i".getClass());  
System.out.println(String.class.getName());  
System.out.println("i".getClass().getName());
```

Output:

```
true  
java.lang.String  
java.lang.String
```

Class Objects

- `.class` syntax can be used for primitive types
- `getClass()` cannot be used for primitive types

```
int a = 1;  
Class c1 = a.getClass();    // compile-time error  
Class c2 = int.class;      // correct
```

- Can be used for arrays

```
System.out.println(int[].class.getName());  
System.out.println(Double[][] .class.getName());
```

```
[I  
[[Ljava.lang.Double;
```

Class Objects

- `Integer.class = Integer.getClass()`
- `Integer.class != int.class`
- `Integer.TYPE = int.class`

```
Integer a = 1;  
Class<? extends Integer> c1 = int.class;  
Class<? extends Integer> c2 = a.getClass();  
Class<? extends Integer> c3 = Integer.class;  
System.out.println(c1 == c2);  
System.out.println(c1 == c3);  
System.out.println(c1 == Integer.TYPE);
```

A) false
false
true

B) false
true
true

C) true
false
true

Class Objects

- `Integer.class = Integer.getClass()`
- `Integer.class != int.class`
- `Integer.TYPE = int.class`

```
Integer a = 1;  
Class<? extends Integer> c1 = int.class;  
Class<? extends Integer> c2 = a.getClass();  
Class<? extends Integer> c3 = Integer.class;  
System.out.println(c1 == c2);  
System.out.println(c1 == c3);  
System.out.println(c1 == Integer.TYPE);
```

A) false
false
true



B) false
true
true



C) true
false
true



Get Class from Name

- If the fully-qualified name is known
- Static method `forName()`

```
// c1: java.lang.String
```

```
Class c1 = Class.forName("java.lang.String");
```

```
// c2: double[]
```

```
Class c2 = Class.forName("[D");
```

```
// c3: Integer[][]
```

```
Class c3 = Class.forName("[[Ljava.lang.Integer;");
```

Examine Class Modifiers & Types

```
Class c = List.class;  
int mod = c.getModifiers();  
System.out.println(mod);  
System.out.println(Modifier.toString(mod));  
TypeVariable[] tvs = c.getTypeParameters();  
for (TypeVariable tv : tvs) {  
    System.out.println(tv.getName());  
}
```

1537

```
public abstract interface  
E
```

Interaction with Objects

- `isInstance (Object)`
- `cast (Object)`

```
Object o = 1;
Class<Integer> c = Integer.class;
if (c.isInstance(o)) { // o instanceof Integer
    int i = c.cast(o); // int i = (int) o;
    System.out.println("i = " + i);
}
```

Discover Class Members

- **Member interface**
 - `java.lang.reflect.Member`
- **Fields**
 - `java.lang.reflect.Field`
- **Methods**
 - `java.lang.reflect.Method`
- **Constructors**
 - `java.lang.reflect.Constructor`

Discover Class Members

Class Methods for Locating Fields

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

- Pay attention to singular/plural
- **Singular** can access inherited members
- **Declared** can access private members

Discover Class Members

Class Methods for Locating Methods

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

Class Methods for Locating Constructors

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

Get All Declared Methods

```
class SimpleClass {  
    public void aSimpleMethod() { ... }  
    public void anotherSimpleMethod() { ... }  
    private void privateMethod() { ... }  
}  
  
Object o = new SimpleClass();  
Class c = o.getClass();  
Method[] ms = c.getDeclaredMethods();  
for(Method m : ms) {  
    System.out.println("o has method: " + m.getName());  
}
```

Output:

```
o has method: aSimpleMethod  
o has method: anotherSimpleMethod  
o has method: privateMethod  
  
...
```

Field

- Get information (type, modifier, ...)
- Basically the same as a class
- Set/Get field values

```
Class Point { public int x; public int y; }
```

```
Class c = Point.class;
```

```
Field xField = c.getField("x");
```

```
Field yField = c.getField("y");
```

```
Point p1 = new Point();
```

```
p1.x = 1; p1.y = 2;
```

```
Point p2 = new Point();
```

```
xField.set(p2, xField.get(p1)); // p2.x = p1.x
```

```
yField.set(p2, yField.get(p1)); // p2.y = p1.y
```


Set/Get Field Values

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

class Test {
    public int afield = 1;

    public static void main(String[] args) {
        Test o = new Test();
        Class c = o.getClass();
        try {
            Field f = c.getField("afield");
            System.out.println("GOT: " + f.get(o));
            f.set(o, 2);
            System.out.println("NOW: " + o.afield);
        } catch (NoSuchFieldException e) {...}
        catch (IllegalAccessException e) {...}
    }
}
```

Set/Get Field Values

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

class Test {
    public int afield = 1;

    public static void main(String[] args) {
        Test o = new Test();
        Class c = o.getClass();
        try {
            Field f = c.getField("afield");
            System.out.println("GOT: " + f.get(o));
            f.set(o, 2);
            System.out.println("NOW: " + o.afield);
        } catch (NoSuchFieldException e) {...}
        catch (IllegalAccessException e) {...}
    }
}
```

GOT: 1

NOW: 2

Set/Get Field Values

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

class Test {
    private int afield = 1;

    public static void main(String[] args) {
        Test o = new Test();
        Class c = o.getClass();
        try {
            Field f = c.getField("afield");
            System.out.println("GOT: " + f.get(o));
            f.set(o, 2);
            System.out.println("NOW: " + o.afield);
        } catch (NoSuchFieldException e) {...}
        catch (IllegalAccessException e) {...}
    }
}
```

Set/Get Field Values

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

class Test {
    private int afield = 1;

    public static void main(String[] args) {
        Test o = new Test();
        Class c = o.getClass();
        try {
            Field f = c.getField("afield");
            System.out.println("GOT: " + f.get(o));
            f.set(o, 2);
            System.out.println("NOW: " + o.afield);
        } catch (NoSuchFieldException e) {...}
        catch (IllegalAccessException e) {...}
    }
}
```

java.lang.NoSuchFieldException: aField

Set/Get Field Values

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

class Test {
    private int afield = 1;

    public static void main(String[] args) {
        Test o = new Test();
        Class c = o.getClass();
        try {
            Field f = c.getDeclaredField("afield");
            System.out.println("GOT: " + f.get(o));
            f.set(o, 2);
            System.out.println("NOW: " + o.afield);
        } catch (NoSuchFieldException e) {...}
        catch (IllegalAccessException e) {...}
    }
}
```

GOT: 1

NOW: 2

Method

- Get information
 - `getReturnType()`
 - `getParameterTypes()`
 - `getExceptionTypes()`
 - ...
- Get modifier

```
Class c = String.class;  
Method m = c.getDeclaredMethod("length");  
System.out.println(Modifier.toString(m.getModifiers()));
```

```
public
```

Invoke a Method

```
class SimpleClass {
    public void aSimpleMethod() {
        System.out.println("Got called");
    }
}

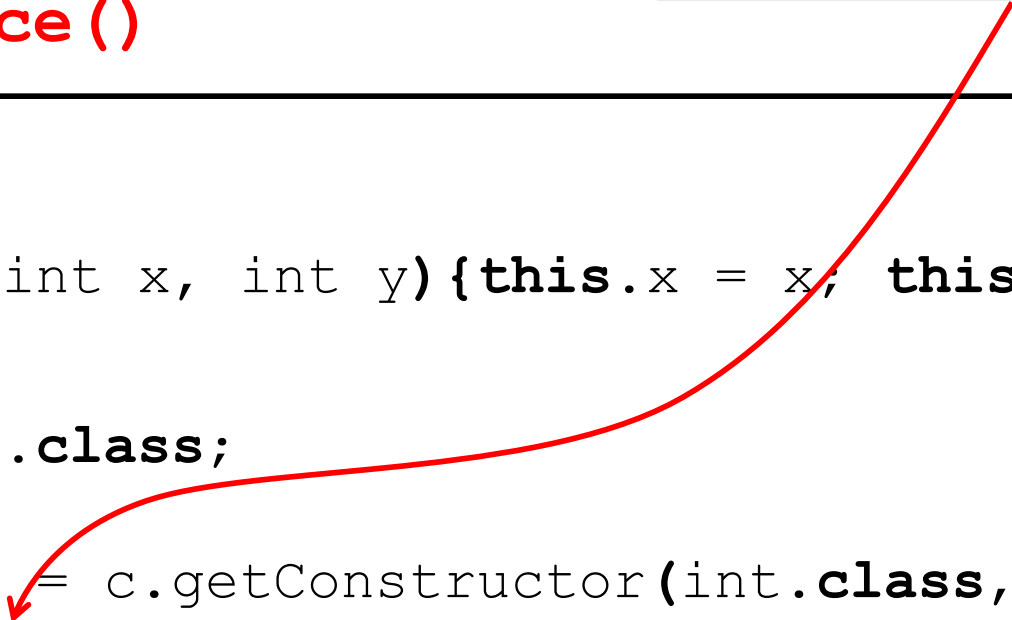
public class SimpleClassTest {
    public static void main() {
        SimpleClass o = new SimpleClass();
        Class c = o.getClass();
        try {
            Method m = c.getMethod("aSimpleMethod");
            m.invoke(o);    // o.aSimpleMethod()
        } catch (NoSuchMethodException e) { ... }
        catch (InvocationTargetException e) { ... }
        catch (IllegalAccessException e) { ... }
    }
}
```

Constructor

- `getParameterTypes()`
- `getModifiers()`
- **`newInstance()`**

Explicitly cast!

```
Class Point {  
    int x; int y;  
    public Point(int x, int y) { this.x = x; this.y = y; }  
}  
  
Class c = Point.class;  
try {  
    Constructor cs = c.getConstructor(int.class, int.class);  
    Point p1 = (Point) cs.newInstance(1,2);  
} catch (NoSuchMethodException e) { ... }  
  catch (InvocationTargetException e) { ... }  
  catch (IllegalAccessException e) { ... }  
  catch (InstantiationException e) { ... }
```



Summary

- Reflection – get metadata of a class
- Get a class object
- Class information: modifier, types, ...
- Class members
 - Field – modifier, types, set/get value
 - Method – modifier, types, invoke
 - Constructor – modifier, types, new instance