

# Exercise

```
void doSomething(List<? extends String> xs) {  
    xs.add(new String("Hello")); // 1  
    Object o = xs.get(0); // 2  
    String s = xs.get(0); // 3  
}  
void foo(){  
    doSomething(new ArrayList<String>()); // 4  
    doSomething(new ArrayList<Integer>()); // 5  
}
```

- Q) Which statements are OK ?

A/B for ok/not ok

1)

2)

3)

4)

5)

# Exercise

```
class Point { int x; int y; }
class ColPoint extends Point { int colour; }
class Aux1{
    void print(List<Point> ps) {
        for(Point p : ps) {
            System.out.println("x=" + p.x + ", y=" + p.y);
        }
    }
    void foo(){
        ArrayList<Point> vp = new ArrayList<Point>();
        ArrayList<ColPoint> vcp = new ArrayList<ColPoint>();
        /*...*/
        print(vp);
        print(vcp);
    }
}
```

OK? (A/B) for Yes/No

# Exercise

```
class Point { int x; int y; }
class ColPoint extends Point { int colour; }
class Aux1{
    void print(List<?> ps) {
        for(Point p : ps) {
            System.out.println("x=" + p.x + ", y=" + p.y);
        }
    }
    void foo(){
        ArrayList<Point> vp = new ArrayList<Point>();
        ArrayList<ColPoint> vcp = new ArrayList<ColPoint>();
        /*...*/
        print(vp);
        print(vcp);
    }
}
```

OK? (A/B) for Yes/No

# Exercise

```
class Point { int x; int y; }
class ColPoint extends Point { int colour; }
class Aux1{
    void print(List<? extends Point> ps) {
        for(Point p : ps) {
            System.out.println("x=" + p.x + ", y=" + p.y);
        }
    }
    void foo(){
        ArrayList<Point> vp = new ArrayList<Point>();
        ArrayList<ColPoint> vcp = new ArrayList<ColPoint>();
        /*...*/
        print(vp);
        print(vcp);
    }
}
```

OK? (A/B) for Yes/No



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



# SWEN221: Software Development

## 16: Reflection

David J. Pearce & Nicholas Cameron & James Noble & Marco  
Servetto

# 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 a mechanism by which a program can find out about the capabilities of its objects at runtime, and manipulate the objects whose capabilities it has discovered*

*-- OOD&P Book*

# What is reflection?

## Reflection in Java

- Java provides `java.lang.Class`

<http://docs.oracle.com/javase/7/docs/api/java/lang/Class.html>

- This represents classes
- Each object associated with unique instance of Class
- Can find out about an object by inspecting its Class object

# Point

```
public class Point {  
    float x;  
    float y;  
}
```

Left click on "Point" (it became gray)  
and then Right click on "Point"

Select "source->generate hashCode() and  
equals()"



# Point

```
public int hashCode() {  
    final int prime = 31;  
    int result = 1;  
    result = prime * result + Float.floatToIntBits(x);  
    result = prime * result + Float.floatToIntBits(y);  
    return result;  
}  
  
public boolean equals(Object obj) {  
    if (this == obj){return true;}  
    if (obj == null){return false;}  
    if (this.getClass() != obj.getClass()){return false;}  
    Point other = (Point) obj;  
    if(Float.floatToIntBits(x) != Float.floatToIntBits(other.x)){  
        return false;}  
    if(Float.floatToIntBits(y) != Float.floatToIntBits(other.y)){  
        return false;}  
    return true;  
}
```

# Point

```
public int hashCode() {  
    final int prime = 31;  
    int result = 1;  
    result = prime * result + Float.floatToIntBits(x);  
    result = prime * result + Float.floatToIntBits(y);  
    return result;  
}  
  
public boolean equals(Object obj) {  
    if (this == obj){return true;}  
    if (obj == null){return false;}  
    if (this.getClass() != obj.getClass()){return false;}  
    Point other = (Point) obj;  
    if(Float.floatToIntBits(x) != Float.floatToIntBits(other.x)){  
        return false;}  
    if(Float.floatToIntBits(y) != Float.floatToIntBits(other.y)){  
        return false;}  
    return true;  
}
```

# Point

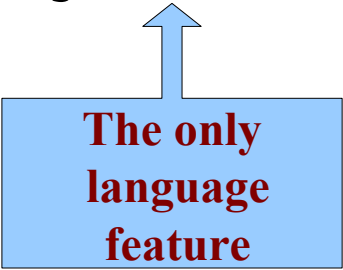
```
if (this.getClass() != obj.getClass()){return false;}
```

Equivalent to

```
//here Object obj  
Class<? extends Point> myClass = this.getClass();  
Class<? extends Object> objClass = obj.getClass();  
if (myClass != objClass) return false;
```

# Class Objects

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



The only  
language  
feature

```
System.out.println(String.class.getName());  
// java.lang.String  
System.out.println(String.class.getSimpleName());  
// String
```

# Class Objects

```
String s1 = new String("X");  
String s2 = new String("Y");  
Integer i1 = new Integer(1);
```

# Class Objects

```
String s1 = new String("X");  
String s2 = new String("Y");  
Integer i1 = new Integer(1);  
Class<? extends String> c1 = s1.getClass();
```

# Class Objects

```
String s1 = new String("X");  
String s2 = new String("Y");  
Integer i1 = new Integer(1);  
Class<? extends String> c1 = s1.getClass();  
Class<String> c2 = s2.getClass();
```

# Class Objects

```
String s1 = new String("X");  
String s2 = new String("Y");  
Integer i1 = new Integer(1);  
Class<? extends String> c1 = s1.getClass();  
// Class<String> c2 = s2.getClass();//compilation error  
Class<String> c2 = String.class;
```



# Class Objects

```
String s1 = new String("X");
String s2 = new String("Y");
Integer i1 = new Integer(1);
Class<? extends String> c1 = s1.getClass();
// Class<String> c2 = s2.getClass();//compilation error
Class<String> c2 = String.class;
Class<? extends Integer> c3 = i1.getClass();
```

# Class Objects

```
String s1 = new String("X");  
String s2 = new String("Y");  
Integer i1 = new Integer(1);  
Class<? extends String> c1 = s1.getClass();  
// Class<String> c2 = s2.getClass(); // compilation error  
Class<String> c2 = String.class;  
Class<? extends Integer> c3 = i1.getClass();  
Class<?> c4 = i1.getClass();
```

# Class Objects

```
String s1 = new String("X");
String s2 = new String("Y");
Integer i1 = new Integer(1);
Class<? extends String> c1 = s1.getClass();
// Class<String> c2 = s2.getClass(); // compilation error
Class<String> c2 = String.class;
Class<? extends Integer> c3 = i1.getClass();
Class<?> c4 = i1.getClass();
assert c1 == c2;
```

# Class Objects

```
String s1 = new String("X");
String s2 = new String("Y");
Integer i1 = new Integer(1);
Class<? extends String> c1 = s1.getClass();
// Class<String> c2 = s2.getClass(); // compilation error
Class<String> c2 = String.class;
Class<? extends Integer> c3 = i1.getClass();
Class<?> c4 = i1.getClass();
assert c1 == c2;
assert c1 != c3;
```

# Class Objects

```
String s1 = new String("X");
String s2 = new String("Y");
Integer i1 = new Integer(1);
Class<? extends String> c1 = s1.getClass();
// Class<String> c2 = s2.getClass();//compilation error
Class<String> c2 = String.class;
Class<? extends Integer> c3 = i1.getClass();
Class<?> c4 = i1.getClass();
assert c1 == c2;
// assert c1 != c3;//compilation error
```

# Class Objects

```
String s1 = new String("X");
String s2 = new String("Y");
Integer i1 = new Integer(1);
Class<? extends String> c1 = s1.getClass();
// Class<String> c2 = s2.getClass();//compilation error
Class<String> c2 = String.class;
Class<? extends Integer> c3 = i1.getClass();
Class<?> c4 = i1.getClass();
assert c1 == c2;
// assert c1 != c3;//compilation error
assert c1 != c4;
```

# Class Objects

```
String s1 = new String("X");
String s2 = new String("Y");
Integer i1 = new Integer(1);
Class<? extends String> c1 = s1.getClass();
// Class<String> c2 = s2.getClass();//compilation error
Class<String> c2 = String.class;
Class<? extends Integer> c3 = i1.getClass();
Class<?> c4 = i1.getClass();
assert c1 == c2;
// assert c1 != c3;//compilation error
assert c1 != c4;
System.out.println("c1 is a " + c1.getName());
```

# Class Objects

```
String s1 = new String("X");
String s2 = new String("Y");
Integer i1 = new Integer(1);
Class<? extends String> c1 = s1.getClass();
// Class<String> c2 = s2.getClass();//compilation error
Class<String> c2 = String.class;
Class<? extends Integer> c3 = i1.getClass();
Class<?> c4 = i1.getClass();
assert c1 == c2;
// assert c1 != c3;//compilation error
assert c1 != c4;
System.out.println("c1 is a " + c1.getName());
// c1 is a java.lang.String
```



# Class Objects

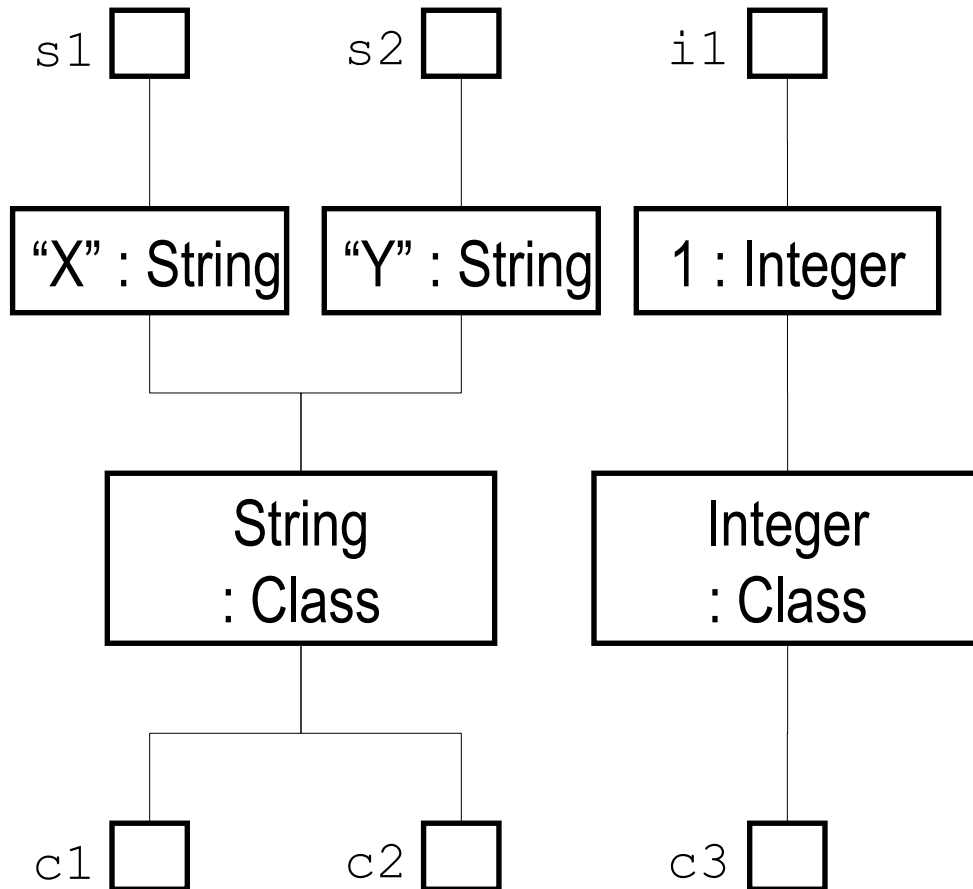
```
String s1 = new String("X");
String s2 = new String("Y");
Integer i1 = new Integer(1);
Class<? extends String> c1 = s1.getClass();
// Class<String> c2 = s2.getClass();//compilation error
Class<String> c2 = String.class;
Class<? extends Integer> c3 = i1.getClass();
Class<?> c4 = i1.getClass();
assert c1 == c2;
// assert c1 != c3;//compilation error
assert c1 != c4;
System.out.println("c1 is a " + c1.getName());
// c1 is a java.lang.String
System.out.println("c3 is a " + c3.getName());
```

# Class Objects

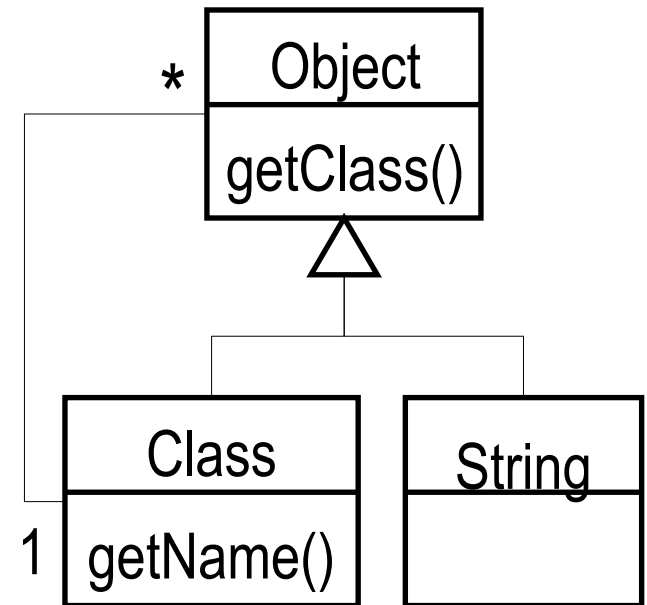
```
String s1 = new String("X");
String s2 = new String("Y");
Integer i1 = new Integer(1);
Class<? extends String> c1 = s1.getClass();
// Class<String> c2 = s2.getClass();//compilation error
Class<String> c2 = String.class;
Class<? extends Integer> c3 = i1.getClass();
Class<?> c4 = i1.getClass();
assert c1 == c2;
// assert c1 != c3;//compilation error
assert c1 != c4;
System.out.println("c1 is a " + c1.getName());
// c1 is a java.lang.String
System.out.println("c3 is a " + c3.getName());
// c3 is a java.lang.Integer
```

# Class Objects

## Instance Diagram



## Class Diagram



# Q) What gets printed?

```
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:



# Q) What gets printed?

```
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
```

# Q) What gets printed?

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

Output:



# Q) What gets printed?

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

Output:

```
o has method: wait  
o has method: equals  
o has method: toString  
o has method: hashCode  
....
```

# Metadata

- Reflection gives access to *metadata*
  - That is, data about data
  - In this case, the metadata 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



# More than just *metadata*

We can *invoke* methods through reflection as well

```
class SimpleClass {  
    public void aSimpleMethod() {  
        System.out.println("Got called");  
    }  
}  
  
...  
Object o = new SimpleClass();  
Class<?> c = o.getClass();  
try {  
    Method m = c.getMethod("aSimpleMethod");  
    m.invoke(o);  
} catch (NoSuchMethodException e) { ... }  
    catch (InvocationTargetException e) { ... }  
    catch (IllegalAccessException e) { ... }
```

# More than just *metadata*

We can *invoke* methods through reflection as well

```
try {  
    Method m = c.getMethod("aSimpleMethod");  
    m.invoke(o);  
}  
catch (NoSuchMethodException e) { throw new Error(e); }  
catch (IllegalAccessException e) { throw new Error(e); }  
catch (InvocationTargetException e){  
    Throwable ee=e.getCause();  
    if(ee instanceof Error){throw (Error)ee;}  
    if(ee instanceof RuntimeException){throw (RuntimeException)ee;}  
    //Otherwise do something...  
    //...  
}
```

# More than just *metadata*

We can *get and set* fields through reflection as well

```
import java.lang.reflect.Field;

public 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) {...}
    }
}
```

# Surprising ... ?

What about private fields? Fine Here

```
import java.lang.reflect.Field;
public 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) {...}
    }
}
```

# Surprising ... ?

What about private fields? Does not work here

```
import java.lang.reflect.Field;
```

```
class Test1 { private int afield = 1;  
    public int getAField() {return this.afield; } }
```

```
public class Test2 {  
    public static void main(String[] args) {  
        Test1 o = new Test1();  
        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.getAField());  
        }  
        catch (NoSuchFieldException e) {throw new Error(e);}  
        catch (IllegalAccessException e) {throw new Error(e);}  
    }  
}
```

# Surprising ... ?

What about private fields? Fine here again!!

```
import java.lang.reflect.Field;
class Test1 { private int afield = 1;
    public int getAField() {return this.afield; } }

public class Test2 {
    public static void main(String[] args) {
        Test1 o = new Test1();
        Class<?> c = o.getClass();
        try {
            Field f = c.getDeclaredField("afield");
            f.setAccessible(true); // !!!!!!!!
            System.out.println("GOT: " + f.get(o));
            f.set(o, 2);
            System.out.println("NOW: " + o.getAField());
        }
        catch (NoSuchFieldException e) {throw new Error(e);}
        catch (IllegalAccessException e) {throw new Error(e);}
    } }
```

# Looks broken! but is just flexible

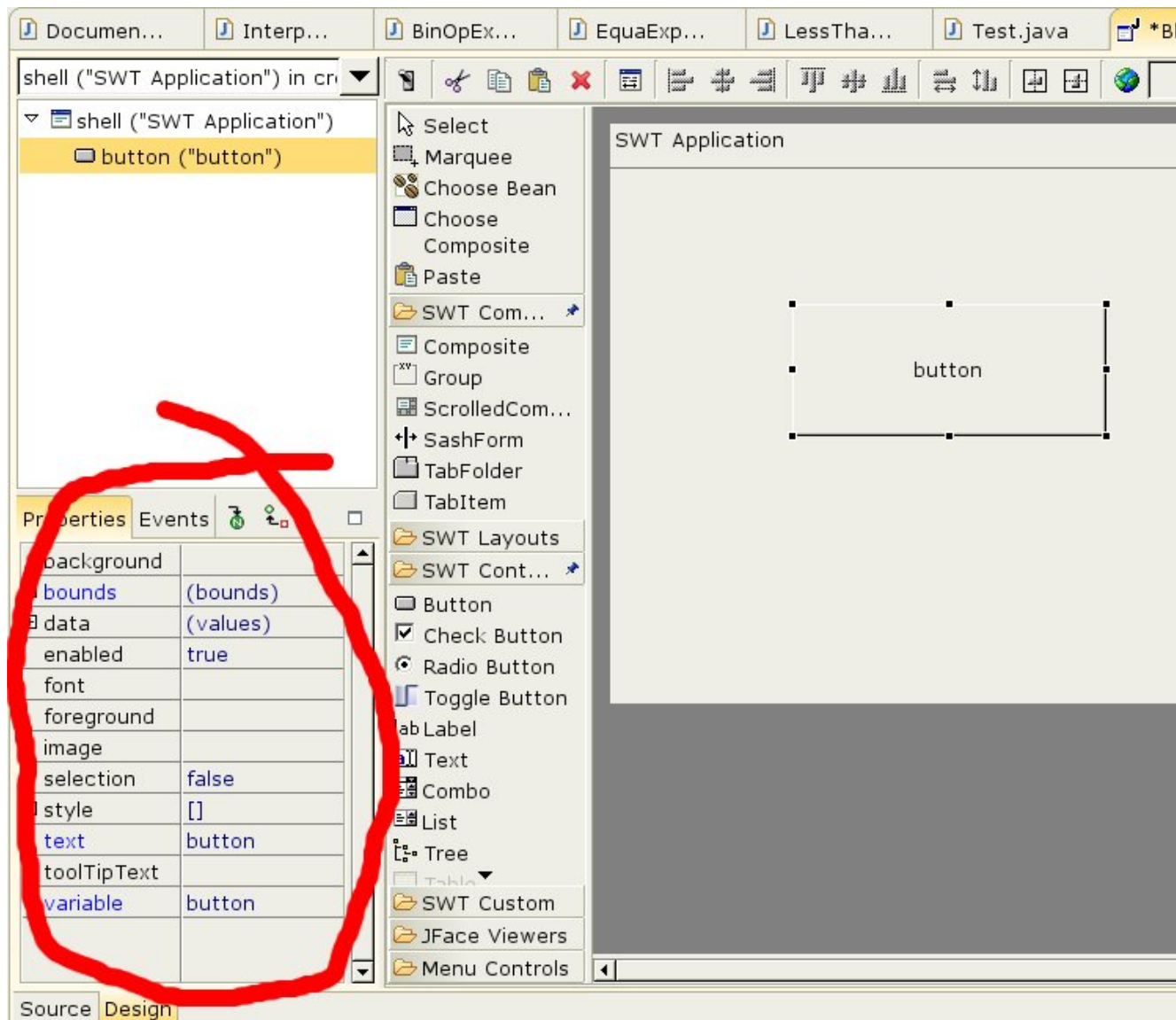
```
public class Test3 {  
    public static void main(String[] args) {  
        System.setSecurityManager(  
            new SecurityManager() {  
                public void checkPermission(Permission p) {  
                    if (p instanceof RuntimePermission  
                        && "setSecurityManager".equals(p.getName()))  
                        throw new SecurityException();  
                    if (p instanceof ReflectPermission  
                        && "suppressAccessChecks".equals(p.getName())) {  
                        StackTraceElement[] st  
                            =Thread.currentThread().getStackTrace();  
                        if(...st...)throw new SecurityException("Haha");  
                    }  
                }  
                // Writing a SecurityManager is harder than that...  
            }  
        });  
        Test2.main(args);  
    }  
}
```

# Why is reflection useful?

- Anonymous objects
  - Objects whose class is not known at compile time
  - Reflection enables us to use anonymous objects
- Example - *GUI builder*
  - A classic use of reflection
  - A *GUI builder* displays objects
    - User can see what attributes it supports
    - E.g. colour, size, position, text, icons etc
  - We want to add new *GUI* components whenever we like



# Example – SWT designer



# Why is reflection useful?

- `Class<?> c=Class.forName("myPName.MyClass");`
  - Obtain a class object from a string!
  - Unlock your imagination

- Example, in Web Assessment Tool:

```
Class<?> c=Class.forName("questions.QuestionW0_0");  
Question q=(Question)c.newInstance();
```

- now I have a Question object, and I can use it without any more reflection.
- Similar patterns are used to do plug-in loading and automatically upgradable programs.