

# CS290i - Lecture 7

## Java Reflection and Serialization Recap

Scalable Internet Services and Systems, Winter 2002

Thorsten von Eicken  
Department of Computer Science  
University of California at Santa Barbara

## Motivation

### ■ Java Beans, Java RMI

- Both use Serialization to save/restore objects
- Serialization uses Reflection to inspect objects

### ■ Reflection

- The structure of a class is *reflected* in a set of descriptors
- Allows a Java program to query/"discover" the data type of a class/object

### ■ Serialization

- An object, and transitively all objects referred-to are written to a (*serial*) byte stream such that the stream can be read in and the objects recreated.

2

## Reflection

### ■ Enables Java code to

- discover information about fields, methods, and constructors of loaded classes
- use reflected fields, methods, and constructors to operate on their underlying counterparts

### ■ Create a reflected class from a given instance of a class

### ■ The reflected class allows:

- Get information about the underlying member or constructor
- Get and set field (variable) values
- Invoke methods of the underlying member
- Create new instances of classes

3

## Reflection Examples

### ■ Retrieve a class object:

```
TextField t = new TextField();  
Class c = t.getClass();  
Class s = c.getSuperclass();
```

### ■ Retrieve class name:

```
static void printName(Object o) {  
    Class c = o.getClass();  
    String s = c.getName();  
    System.out.println(s); }  
}
```

4

## Reflection Examples

### ■ Identify class fields:

```
• static void printFieldNames(Object o) {
    Class c = o.getClass();
    Field[] publicFields = c.getFields();
    for (int i = 0; i < publicFields.length; i++) {
        String fieldName = publicFields[i].getName();
        Class typeClass = publicFields[i].getType();
        String fieldType = typeClass.getName();
        System.out.println("Name: " + fieldName +
            ", Type: " + fieldType);
    }
}
```

### ■ Identify methods:

```
• Class c = o.getClass();
  Method[] m = c.getMethods();
• Class[] parameterTypes = m[i].getParameterTypes();
```

5

## Reflection Examples

### ■ Accessing field values

```
• static void printHeight(Rectangle r) {
    Field heightField; Integer heightValue;
    Class c = r.getClass();
    try {
        heightField = c.getField("height");
        heightValue = (Integer) heightField.get(r);
        System.out.println("Height: " + heightValue.toString());
    } catch (NoSuchFieldException e) { System.out.println(e);
    } catch (SecurityException e) { System.out.println(e);
    } catch (IllegalAccessException e) { System.out.println(e);
    }
}
• heightField.set(r, 123);
```

6

## Reflection (cont.)

### ■ Example reflected class: Method

- information about and access to a method of a class/interface
- public String getName()
- public native int getModifiers()
  - ◆ abstract, final, public, private, static, synchronized, etc.
- public Class getReturnType()
- public Class[] getParameterTypes()
- public Class[] getExceptionTypes()
  - ◆ which exceptions are thrown
- public native Object invoke(Object obj, Object args[]) throws IllegalAccessException, IllegalArgumentException, InvocationTargetException
  - ◆ Invokes the method and returns its return value
  - ◆ The return value is automatically wrapped in an object if it has a primitive type

7

## Reflection Examples

### ■ Invoking a method

```
• public static String
  append(String firstWord, String secondWord) {
    String result = null;
    Class c = String.class;
    Class[] parameterTypes = new Class[] {String.class};
    Method concatMethod;
    Object[] arguments = new Object[] {secondWord};
    try {
        concatMethod = c.getMethod("concat", parameterTypes);
        result = (String)concatMethod.invoke(firstWord, arguments);
    } catch (NoSuchMethodException e) { System.out.println(e);
    } catch (IllegalAccessException e) { System.out.println(e);
    } catch (InvocationTargetException e){ System.out.println(e);
    }
    return result;
}
```

8

# Serialization

## ■ Encoding of objects into a stream of bytes

- Recursively encodes the objects reachable from root set
- Complementary reconstruction of the objects from the stream
- Used for: persistence, communication via sockets, RMI, ...

## ■ Example:

- ```
FileOutputStream f = new FileOutputStream("tmp");
ObjectOutputStream s = new ObjectOutputStream(f);
s.writeObject("Today"); s.writeObject(new Date());
s.flush();
```
- ```
FileInputStream in = new FileInputStream("tmp");
ObjectInputStream s = new ObjectInputStream(in);
String today = (String)s.readObject();
Date date = (Date)s.readObject();
```