Reflection: Manipulate

see Sun's tutorial

Constructing Objects

String className;

className = ...;

Object o = new (className); // DOES NOT WORK

can use default constructor (if available):

```
o = Class.forName(className).newInstance();
```

Primitive types ?

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

public class Test {

      public static void main(String[] args) {

            Class c = int.class;

            Method methods = c.getMethods();

            Class superClass = c.getSuperclass();

            System.out.println(c + " " + superClass + " " +
              methods.length);

      }

}   // prints:

int null0
```

Constructing Objects with parameters

```
Class c = Class.forName("java.awt.Rectangle");

Class[] paramTypes = new Class[]{int.class,int.class};

Constructor cons = c.getConstructor(paramTypes);

Object[] args = new Object[]{new Integer(12), new Integer
    (34)};

Object o = cons.newInstance(args);

// o is a rectangle with width 12 and height 34
```

Field values

```
static void printField(Object o, String fName) {
    Class c = o.getClass();
    Field field = c.getField(fName);
    Object value = field.get(o);
    System.out.println(fName + " = " + value);
}
// not shown: possible Exceptions raised
// also: "auto-boxing" of primitives
```

Modifying fields

```
static void setField(Object o, String fName, Object newValue) {
    try {
        Class c = o.getClass();
        Field field = c.getField(fName);
        Object value = field.set(o,newValue);
    } catch (NoSuchFieldException e) {
        System.err.println("No such field: " + fName + " " + o);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Invoke a method

```
Class c = String.class;
Class[] paramTypes = new Class[] {String.class};
Method m = c.get.Method(“concat”,paramTypes);
Object[] args = new Object[] { “ everybody”};
Object result = m.invoke(“Hello “, args);
// result will be “Hello everybody”
```

Inspecting Arrays

isArray() test in Class

```
static void printComponentType(Object array) {
    Class arrayClass = array.getClass();
    Class componentClass = arrayClass.getComponentType();
    System.out.println( arrayClass.getName() + " " +
                        componentClass.getName());
}
```

Creating Arrays

```
static Object growArray(Object source) {
    int n = Array.getLength(source);
    Class arrayClass = source.getClass();
    Class componentClass = arrayClass.getComponentType();
    Object result = Array.newInstance(componentClass,2*n);
    System.arraycopy(source,0,result,0,n);
    return result;
}
```

Multi-dimensional arrays

supply an integer array for the dimensions instead of just a size:

```
int[][] dims = new int[] {5,10};
String[][] s = (String[][]) Array.newInstance(String.class, dims);
```

is equivalent to: `String[][] s = new String[5][10];`

```
int[] dims = {12};
int[][] twoDimA = (int[][])  Array.newInstance(int[].class, dims);
```

is equivalent to: `int[][] twoDimA = new int[12][];`

Getting and setting array entries

```
public static void copyArray(Object source, Object dest) {
    for(int i = 0; i<Array.getLength(source); i++) {
        Object o = Array.get(source, i);
        Array.set(dest, i, o);
    }
}
// also available: specialised methods for primitive types
   like getInt, setInt, ...
```

faster, do not have to generate wrapper objects

Class summary

- Two classes in java.lang
  - Object
  - Class
- Rest in java.lang.reflect (must import explicitly)
  - Constructor
  - Field
  - Method
  - Modifier
  - Array