Java Reflection

Programs about Programs

Java looking at Java

- One of the capabilities of Java is that a program can examine itself
  - You can determine the class of an object
  - You can find out all about a class: its access modifiers, superclass, fields, constructors, and methods
  - You can find out what is in an interface
  - Even if you don't know the names of things when you write the program, you can:
    - Create an instance of a class
    - Get and set instance variables
    - Invoke a method on an object
    - Create and manipulate arrays

What is reflection for?

- In “normal” programs you don’t need reflection
- You *do* need reflection if you are working with programs that process programs
- Typical examples:
  - A class browser
  - A debugger
  - A GUI builder
  - An IDE, such as Netbeans or Eclipse
  - A program to grade student programs

The Class class

- To find out about a class, first get its Class object
  - If you have an object obj, you can get its class object with
    Class c = obj.getClass();
  - You can get the class object for the superclass of a Class c with
    Class sup = c.getSuperclass();
  - If you know the name of a class (say, Button) at compile time, you can get its class object with
    Class c = Button.class;
  - If you know the name of a class at run time (in a String variable str), you can get its class object with
    Class c = class.forName(str);

Getting the class name

- If you have a class object c, you can get the name of the class with c.getName()
- getName returns the fully qualified name; that is,

```
Class c = Button.class;
String s = c.getName();
System.out.println(s);
```

  will print

```
java.awt.Button
```

- Class Class and its methods are in java.lang, which is always imported and available

Getting all the superclasses

- getSuperclass() returns a Class object (or null if you call it on Object, which has no superclass)

```
static void printSuperclasses(Object o) {
    Class subclass = o.getClass();
    Class superclass = subclass.getSuperclass();
    while (superclass != null) {
        String className = superclass.getName();
        System.out.println(className);
        subclass = superclass;
        superclass = subclass.getSuperclass();
    }}
```

Examining classes and interfaces

- The class Class represents both classes and interfaces
- To determine if a given Class object *c* is an interface, use *c*.isInterface()
- To find out more about a class object, use:
  - getModifiers()
  - getFields() // "fields" == "instance variables"
  - getConstructors()
  - getMethods()
  - isArray()

Getting Fields

- public Field[] getFields() throws SecurityException
  - Returns an array of *public* Fields (including inherited fields).
  - Both locally defined and inherited instance variables are returned, but *not* static variables.
- public Field getField(String name)
  throws NoSuchFieldException, SecurityException
  - Returns the named *public* Field
  - If no immediate field is found, the superclasses and interfaces are searched recursively

Methods

- public Method[] getMethods()
  throws SecurityException
  - Returns an array of Method objects
  - These are the *public member* methods of the class or interface, including inherited methods

- public Method getMethod(String name,
  Class… parameterTypes)
  throws NoSuchMethodException, SecurityException

```
import java.lang.reflect.Method;
import java.lang.Class;

Method findInheritedMethod(Class classType,
String theMethodName, Class … methodParamTypes)
{
  Method inheritedMethod = null;
  while(classType != null)  {
       try  {
         inheritedMethod =  classType.getDeclaredMethod(theMethodName,
methodParamTypes);
        classType = null;
     }
     catch (NoSuchMethodException ex)  {
     classType = classType.getSuperclass( );
     }
     return inheritedMethod ;
}
```