More on GUIs and Reflection
May 1, 2003

UI Hall of Shame

- Application directory dialog (not system)
  - inconsistent
- Requires typing a path name
  - no browse option
  - What if you have many directories?
- Instead, want recognition over recall

Tips

- Don’t make the user look stupid
- The goal of all software users is to be more effective

UI Hall of Fame

- MS Publisher
- Modified file dialog
- Recognition based
  -Browsable names
  -Browsable content
  -Preview can be turned off
- Optional search
- Design Suggestion:
  - Add entry box for directly typing in name with auto-completion of filenames
Tips

• User interfaces that directly wrap underlying systems are often bad
• Users don’t understand Boolean – think Google

A User Interface for Project 5

• We’ll develop a user interface for Project 5
  – So you can see how it works
• This is not part of Project 6
  – Please do not turn in code with a GUI!

Reflection

• Remember project 1?
  – Used reflection to find module class
    • http://webserver:port/HelloWorld/Foo
• We’ll cover some basics of reflection
  – Highly condensed version of
    • Reflection: Java Technology’s Secret Weapon, by Odendahl (on web page)

What is Reflection?

• Makes classes, methods, and fields objects that can be manipulated at run time
  – Can determine fields and methods of class
  – Can instantiate class given a String containing its name
  – Can invoke methods given a String with name
  – Can create classes at runtime

What Reflection Isn’t

• Doesn’t add any power to the language
  – Given access to all the source code
• Not the solution to every problem
  – Use sparingly, if at all

java.lang.Class

• Object of type Class represents a class
  – Useful for
    • Making instances of a class
    • Getting information about fields/methods
  – Most uses of reflection start with a Class
• Primitive types also have a Class
  – e.g., int.class
Getting Some Class

- Use Object method `getClass()`
  - `Class c = "hello".getClass();`
- Use class literal
  - `Class c = String.class;`
- Use the class name
  - `Class c = Class.forName("java.lang.String")`

Making Objects

- Class object for no-arg constructor
  ```java
  Class c;
  Foo f = (Foo) c.newInstance();
  ```
- Constructor object otherwise
  ```java
  Class c;
  Class[] cArg = { String.class };  // build constructor
  Constructor cons = c.getConstructor(cArg);  // get constructor
  Object[] consArg = { "hello" };  // build arguments
  Foo f = (Foo) cons.newInstance(consArg);  // invoke method
  ```

Working with Fields

- Can get Field objects from Class
  - Can also get all fields in Class
  ```java
  Class c = x.getClass();  // get class of obj x
  Field f = c.getField("foo");  // get field
  ...(Type-of-foo) f.get(x); ...
  ...f.set(x, value);...
  ```

Invoking Methods

- Get from Class object
  - Invoke just like constructor
  ```java
  Class c = x.getClass();
  Class[] cArg = { String.class };  // build constructor
  Method m = c.getMethod("bar", cArg);  // get method
  Object[] mArg = { "hello" };  // build arguments
  Foo f = (m-result-type) m.invoke(x, mArg);  // invoke method
  ```

Putting It All Together

- Example from JavaOne slides:
  ```java
  public static void main(String[] args) throws Exception {
      Field f = System.class.getField("out");
      PrintStream out = (PrintStream) f.get(null);
      Class[] paramTypes = { String.class };
      Method m = PrintStream.class.getMethod("println", paramTypes);
      String[] params = (String[]) Array.newInstance(String.class, 1);
      Array.set(params, 0, "Hello, world!");
      m.invoke(out, params);
  }
  ```

Some Better Uses of Reflection

- Project 1 (ignoring setArg)
  ```java
  if (a.equals("Proxy"))
      new Proxy().run(arg);
  else if (a.equals("HelloWorld"))
      new HelloWorld().run(arg);
  ...
  MiniServlet m = Class.forName(a).newInstance();
  m.run();
  ```
Some Better Uses of Reflection

- Handling multiple code versions

```java
Method m;
try {
    m = Foo.class.getMethod("newMethod", new Class[0]);
    // Use method m
} catch (Exception e) {
    // Work around since m doesn’t exist
}
```

Proxy

- Let’s you dynamically create a new class
  - Useful when you need an object that implements a particular interface
  - Alternative: dynamically generate bytecode

- Let IF be an interface
  - Can create Proxy object with all methods of IF
  - Method dispatched on Proxy objects gets passed to your handler

Creating a Dynamic Proxy Object

```java
Object f = Proxy.newProxyInstance(Foo.class.getClassLoader(), new Class[] { Foo.class }, handler);
```

- First arg is class loader
- Second arg lists interfaces to implement
- The cool part:
  - f instanceof Foo will succeed
  - (Foo) f will succeed

InvocationHandler

```java
interface InvocationHandler {
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable;
}
```

- Last parameter of newProxyInstance
  - Receives calls to proxy methods