
Reflection

CSE 413, Autumn 2002
Programming Languages

<http://www.cs.washington.edu/education/courses/413/02au/>

Readings and References

- Reading
 - » Chapter 5, Inheritance, Section: Reflection ,*Core Java Volume 1*, by Horstmann and Cornell
- Other References
 - » "The Reflection API", Java tutorial
 - » <http://java.sun.com/docs/books/tutorial/reflect/index.html>

Classes from another viewpoint

- Ordinarily we deal with specific classes that perform specific functions that are known at compile time
 - » The Scanner class is used to read through a source line in the D language and return Token objects
- Sometimes we want to deal with a group of classes in a general sense
 - » one or more classes that perform a particular task but whose names are not known at compile time

Reflection API

- The ability to treat classes as data is provided by the classes in package `java.lang` and `java.lang.reflect`
- The Reflection API is used to build programs that work with classes as data objects
 - » development tools such as debuggers, class browsers, and application builders
 - » programs with dynamic behavior enabled by providing additional class files and one class that knows how to discover and use the added classes

Example: User Interface builder

- A GUI builder may allow the end-user to select a Button from a menu of components,
 - » menu built by scanning a directory for class files
- create the Button object,
 - » object created by invoking the constructor, but we don't know the name of the class until runtime
- then click the Button to request an action.
 - » invoke a method on the newly created object

Example: Application builder

When you expand the two nodes under your message-driven bean's package node, you see something like the tree view in FIGURE 7-2.

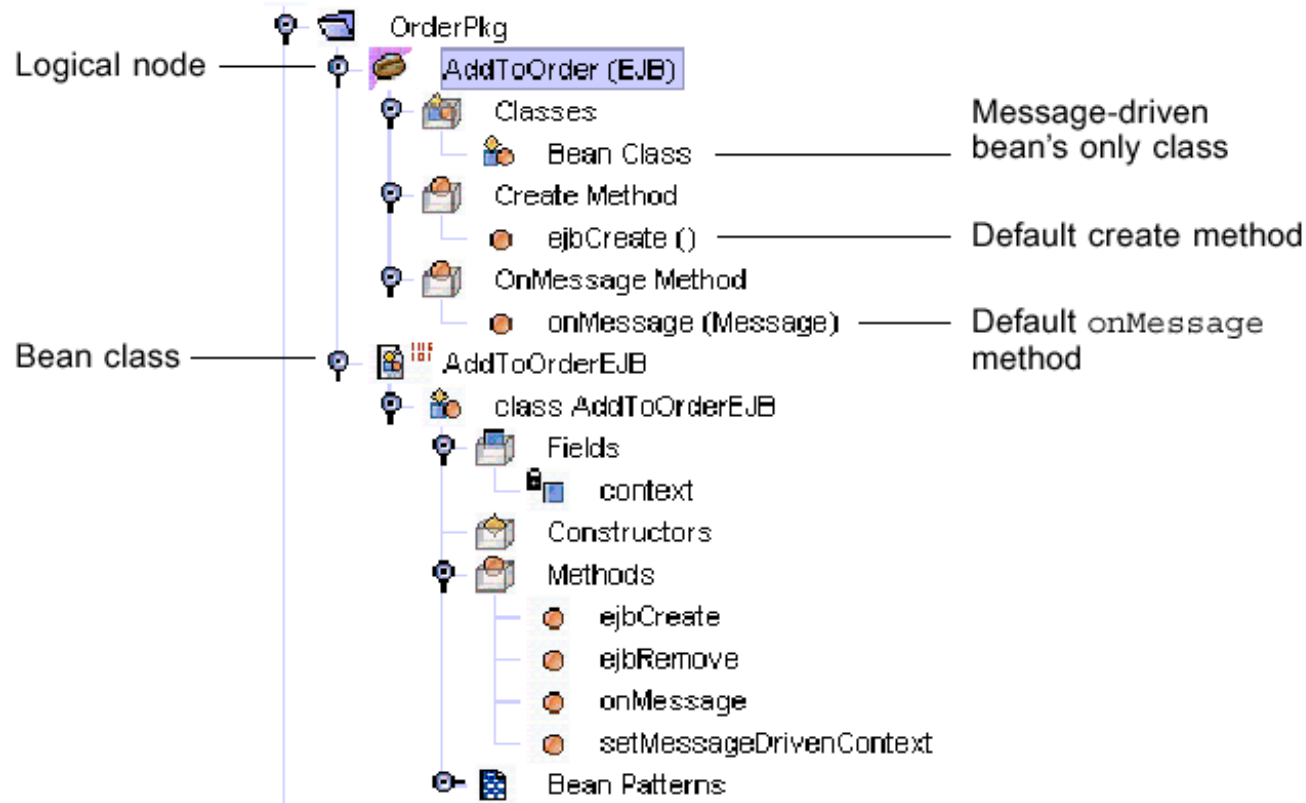


FIGURE 7-2 Explorer's Detailed View of a Typical Message-Driven Bean

The Class class

- Java runtime system maintains information about each class in your program while it is running
- The information is maintained in objects of type `java.lang.Class`
 - » Class is a class, just like String, Scanner, Integer, etc.
 - » Objects of type Class store information about every class in your program

All types are represented with Class

- Instances of the class `Class` represent classes and interfaces in a running Java application.
- Every array also belongs to a class that is reflected as a `Class` object that is shared by all arrays with the same element type and number of dimensions.
- The primitive Java types (boolean, byte, char, short, int, long, float, and double), and the keyword `void` are represented as `Class` objects.

Some methods

- **Object** class

- » **Class getClass()**

Returns the runtime class of an object.

- **Class** class

- » **String getName()**

Returns the name of the entity (class, interface, array class, primitive type, or void) represented by this Class object, as a String.

- » **Class getSuperclass()**

Returns the Class representing the superclass of the entity (class, interface, primitive type or void) represented by this Class.

Example: print class hierarchy

```
public class SampleSuper {
    public static void main(String[] args) {
        Object o = System.out; // object to analyze
        Class subclass = o.getClass();
        Class superclass = subclass.getSuperclass();
        System.out.println(subclass.getName());
        while (superclass != null) {
            String className = superclass.getName();
            System.out.println(className);
            subclass = superclass;
            superclass = subclass.getSuperclass();
        }
    }
}
```

```
java.io.PrintStream
java.io.FilterOutputStream
java.io.OutputStream
java.lang.Object
```

More Class information is available

- **Class [] getInterfaces ()**
 - » Determines the interfaces implemented by the class or interface represented by this object.
- **Constructor [] getConstructors ()**
 - » Returns an array containing Constructor objects reflecting all the public constructors of the class represented by this Class object.
- **Field [] getFields ()**
 - » Returns an array containing Field objects reflecting all the accessible public fields of the class or interface represented by this Class object.
- **Method [] getMethods ()**
 - » Returns an array containing Method objects reflecting all the public member methods of the class or interface represented by this Class object

ReflectionTest.java

Create new object from class name

- **static Class.forName(String className)**
 - » Returns the Class object associated with the class or interface with the given string name.
- **Object newInstance()**
 - » Creates a new instance of the class represented by this Class object.
 - » Calls the default constructor (the zero-argument constructor)

```
String className = "java.util.Random";  
Class classDefinition = Class.forName(className);  
Object object = classDefinition.newInstance();
```

SampleNoArg.java

A note of caution

- The reflection capability is very handy for certain high-level applications or particular functions in a normal application
 - » eg, building a menu item list or options list
- Don't use it when other tools more natural to the Java programming language would suffice
 - » Specifically, provide callback objects by defining interfaces and implementing them in one or more classes. Do not use the Method objects to create elaborate C-style function pointers.