

CS290i - Lecture 7

Beans, beans, beans

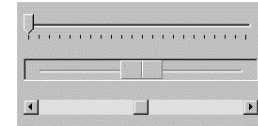
Scalable Internet Services and Systems, Spring 2001

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

Beans

† a Java Bean is:

- † a reusable software component
- † that can be visually manipulated
- † in builder tools



slider beans

† Concepts

- † Similar to “VBXs”, “OCXs”, “controls”, “widgets”, ...
- † Builder tools query components about their properties and behavior (introspection)
- † Visually select components from palettes, panels, or menus
- † Drop them into a form
- † Hook up events to event handlers
- † Design-time vs. run-time



calendar bean

2

Defining Features

- † **Introspection** allows a builder tool to analyze how a bean works
- † **Customization** allows a user to alter the appearance and behavior of a bean
- † Beans can fire **events**, and builder tools can find out about the events they can fire and the events they can handle
- † **Properties** allow beans to be manipulated programatically, and support the customization
- † **Persistence** allows the state of customized beans to be saved and restored

3

Properties design patterns

† Instance variable

- † value can be manipulated with **set** and **get** methods
- † Examples:
 - † Label has a text field that can be set with **setText** and gotten with **getText**
 - † Component.resize has been renamed to **setSize** there is also **getSize**

† Boolean variables

- † instead of a get method, an **is** method is used
- † Example:
 - † TextComponent has a property called **editable**
 - † set with **setEditable**(boolean)
 - † tested with **isEditable**

† **exposed property**: the methods of a property are public

4

Implicit Introspection

† java.beans.Introspector

- † can be used to find out about the properties, events, and methods of a class
- † implicit method or explicit method

† Implicit method

- † Uses design patterns
- † The Introspector creates a list of all public methods in the bean
- † Then searches that list for signatures that match a particular pattern
- † Example:
 - † **public void setFlavor(String f)**
- † Assume:
 - † **Property called "flavor"**
 - † **setFlavor is the write accessor for the property**

5

Implicit Introsp. (cont.)

† Other design patterns..

- † get/set
 - † **public PropertyType getPropertyName()**
 - † **public void setPropertyName(PropertyType v)**
 - † These indicate that the Bean has a property propertyName of class PropertyType.
- † is/get/set
 - † **public boolean isPropertyName()**
 - † **public boolean getPropertyName()**
 - † **public void setPropertyName(boolean v)**
- † Indexed Property Pattern
 - † **public PropertyType[] getPropertyName()**
 - † **public PropertyType getPropertyName(int i)**
 - † **public void setPropertyName(PropertyType[] v)**
 - † **public void setPropertyName(int i, PropertyType v)**

6

Implicit Introsp. (cont.)

- † Public Method Pattern
 - † The introspector creates method descriptors for all of the bean's public methods including all of the public methods of the bean's superclasses.
- † Multicast Event Set Pattern
 - † **public void addEventListener(EventNameListener l)**
 - † **public void removeEventListener(EventNameListener l)**
 - † an event set called eventName
 - † Example:
 - † **public void addActionListener(ActionListener listener)**
 - † **public void removeActionListener(ActionListener listener)**
 - † the bean fires action event sets that will be processed by ActionListeners:

7

Explicit Introspection

† Write explicit code to provide info

† Example

- † Bean b is class Sample
- † Find class SampleBeanInfo
- † Call **SampleBeanInfo.getBeanInfo(b)**, returns BeanInfo object
- † BeanInfo object has get methods to provide info
 - † E.g. **getEventSetDescriptors** returns information about the kinds of events fired by this bean

8

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

9

Reflection Example

- † **Example:**
 - † you load a class and can find out the name (and parameter types) of a method of that class, how would you execute the method?
 - † variable of type MyClass
 - † a String, MyMethod, which is the name of a method in the class
 - † how do you execute that method?

10

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

11

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();
```

12

Enterprise Java Beans

† Architecture for multi-tiered apps

- † Allow app writers to concentrate on “business logic”
- † And write portable applications
- † Standard for app servers to implement

† 3-tier & multi-tier server architectures

- † Instead of 2-tier client-server
- † User interface
- † Business logic
- † System services

† Middleware services

- † Transaction monitors, state management
- † Messaging , resource pooling
- † Object request brokers, more...

13

EJBs

† 3 pieces:

- † EJB components (“EJBs”)
- † EJB containers
- † EJB server

† EJB components

- † Implement business logic

† EJB container

- † Provide services to EJB components, e.g. transaction and resource management
 - † E.g. component can tell its container to abort transaction
- † Typically multiple components per container

14