



A UML state machine diagram is positioned in the background. It features several states represented by circles, some of which are double circles indicating initial or final states. Transitions are shown as arrows between these states, including a sequence of transitions at the top left and a complex set of transitions in the center and bottom right.

# FUEGO

## DESIGN STANDARDS FOR JAVA COMPONENTS

PROGRAMMING JAVA COMPONENTS FOR FUEGO 5

February 18, 2005

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>General Guidelines</b>	<b>3</b>
<b>3</b>	<b>Attributes</b>	<b>3</b>
3.1	Accessors . . . . .	4
3.2	Public Attributes . . . . .	6
3.3	Private/Protected Attributes . . . . .	7
<b>4</b>	<b>Methods</b>	<b>8</b>
4.1	Public Methods . . . . .	8
4.2	Private/Protected Methods . . . . .	8
4.3	Get/Set Methods . . . . .	8
<b>5</b>	<b>Java Standard Types</b>	<b>9</b>
<b>6</b>	<b>Using a BeanInfo class</b>	<b>10</b>
<b>A</b>	<b>Source Code</b>	<b>13</b>
A.1	Customer Class Example . . . . .	13
A.2	Order Class Example . . . . .	14
A.3	OrderBeanInfo Class Example . . . . .	15
A.4	Item Class Example . . . . .	16
A.5	ItemBeanInfo Class Example . . . . .	17

## 1 Introduction

Java components are one of the easiest type of components that Studio can introspect. This is due to the fact that Studio is also built in Java. Even though Studio is able to introspect any Java class inside a Java jar file, some guidelines<sup>1</sup> for developing these components are needed to ease the readability of the code that the Studio developers have to write.

## 2 General Guidelines

The following are general guidelines that apply for every Java component being used by Studio:

- A Java component is not required to inherit from any particular base class or interface in order to be introspected and used by Studio.
- If the component will be used as a process instance variable, then the component must implement the *Serializable* Java interface. This holds true even if the component is used as an instance variable of a *FuegoObject* which in turn is used as a process instance variable.
- The two most important features of a Java component are:
  - **Attributes:** Attributes are named members associated with a component's attribute that can be accessed (read/write) from the FBL.
  - **Methods:** The methods a Java component exports are just normal Java methods which can be called from the FBL. By default all of a component's public methods will be exported, but a component can choose to export only a subset of its public methods. See section 6 for more information.

## 3 Attributes

The following section describes how different kinds of attributes are treated by Studio when introspected:

---

<sup>1</sup>The conventions that we will outline, follow those defined for Java Beans.

### 3.1 Accessors

Studio tries to introspect properties using design patterns to locate accessor methods (setters and getters) by looking for methods of the form:

```
public <PropertyType> get<AttributeName>();  
public void set<AttributeName>(<PropertyType> aPropertyType);
```

Where `PropertyType` can be:

- a primitive type
- a user defined class
- an array of any of these two previous kinds

If Studio discovers a matching pair of `get<AttributeName>` and `set<AttributeName>` methods and those methods that take and return the same type, then Studio regards these methods as defining a read-write attribute whose name will be `<AttributeName>`.

For example, the class *Customer* has two methods that access a private attribute named `lastname`.

```
1  ...  
2      private String lastname;  
3  
4      public void setLastname(String aName)  
5      {  
6          lastname = aName;  
7      }  
8  
9      public String getLastname()  
10     {  
11         return lastname;  
12     }  
13  ...
```

When this code is introspected by Studio, you will find that the component *Customer* does not present the methods `getLastname()` nor `setLastname()`. Instead Studio replaces those methods and presents the attribute `lastname`, even though the attribute appears as private in the Java code.

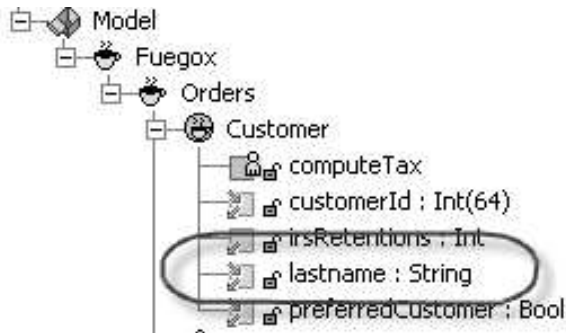


Figure 1: Getters and Setters are replaced by an attribute in Studio

It's important to remember that Studio *will* use the `get<AttributeName>` method to get the attribute's value and the `set<AttributeName>` method to set the attribute's value even though they are not displayed to the user in Studio. Also note that these methods may be located either in the same class or in different classes in the same hierarchy (one may be in a base class and the other may be in a derived class).

If Studio finds only one of these methods, then it regards it as defining either a read-only or a write-only attribute called `<AttributeName>`.

In addition, for boolean properties, Studio allows a getter method to match the following pattern:

```
public boolean is<AttributeName>();
```

This `is<AttributeName>` method may be provided instead of or in addition to a `get<AttributeName>` method. In either case, if the `is<AttributeName>` method is present for a boolean attribute then Studio will use it to read the attribute value. An example of a boolean attribute in our *Customer* class might be:

```
1 ...
2     private boolean preferredCustomer;
3
4     public void setPreferredCustomer(boolean isPreferred)
5     {
6         preferredCustomer = isPreferred;
7     }
8
```

```
9 public boolean isPreferredCustomer()  
10 {  
11     return preferredCustomer;  
12 }  
13 ...
```

Once introspected, Studio will only display an attribute of type *Bool*.

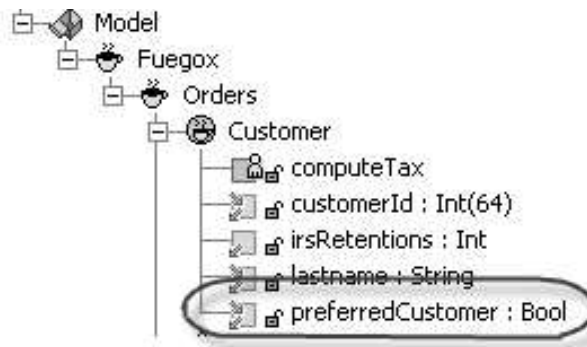


Figure 2: Boolean getters are replaced by an attribute in Studio

All these methods should not have a 'throw exception' declaration in them in order to be treated as getters-setters.

If this component will be used in methods running on the client side, the attribute's type must be a primitive type or one implementing the *Serializable* interface.

### 3.2 Public Attributes

Public attributes that lack any getter or setter methods are displayed in Studio as attributes of the introspected component. For example, the following attribute in the *Customer* class:

```
1 ...  
2     public long customerId;  
3 ...
```

Will be displayed in Studio as an attribute of the *Customer* component:

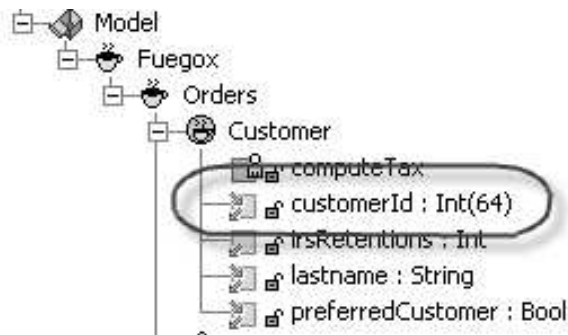


Figure 3: Pubic attributes in Studio

### 3.3 Private/Protected Attributes

Private or protected attributes, are not displayed in Studio and can not be used by Fuego developers. For example, the following attributes in the *Customer* class:

```
1 ...  
2     private String firstname;  
3     protected String telephone;  
4 ...
```

Will not be available in Studio:

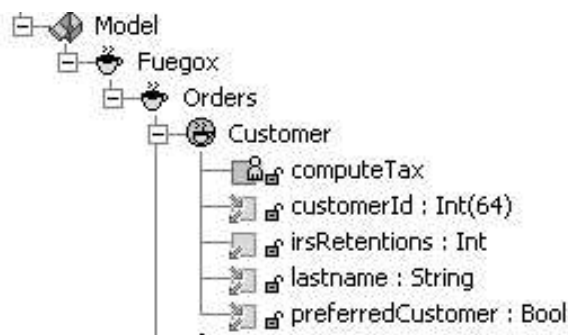


Figure 4: Private/protected attributes are not available in Studio

## 4 Methods

### 4.1 Public Methods

Studio will expose all available public methods of a Java component. These methods can then be used from any FBL code.

Methods can return or accept as parameteres:

- a primitive type
- a user defined type
- an array of any of these 2 previous kinds

Methods can raise or throw any kind of exception. However, these exceptions are not displayed in Studio eventhough they *are* thrown at runtime. Have in mind that any exception thown must be instrospected into Studio as well.

### 4.2 Private/Protected Methods

Private and protected methods are not displayed in Studio.

### 4.3 Get/Set Methods

Methods of the form `get<Name>` or `set<Name>` are represented as attributes in Studio even though they may not be associated to any attribute in the java class. For example, the following method in out *Customer* example:

```
1 ...  
2     public int getIRSRetentions()  
3     {  
4         return 0;  
5     }  
6 ...
```



Will be displayed in Studio as a read-only<sup>2</sup> attribute:

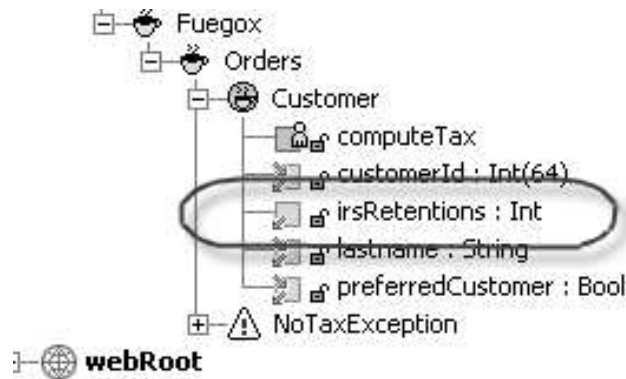


Figure 5: Get/Set methods are displayed as attributes

## 5 Java Standard Types

Java standard types are replaced by Fuego types whenever a method or attribute is introspected. Both types are compatible and whenever a method or attribute expected a certain Java standard type, Fuego developers have to use the corresponding Fuego standard type.

For example, if the method expected a *boolean* java type:

```
1 ...
2 public void setPreferredCustomer(boolean isPreferred)
3 ...
```

In Studio, you will have to provide a Fuego *Bool* type:

```
1 c = Customer()
2 p as Bool = true
3 c.preferredCustomer = p
```

<sup>2</sup>Note that the icon next to the attribute has a single outgoing arrow denoting the attribute as read-only.

## 6 Using a BeanInfo class

A Java component can also **explicitly** specify which properties and methods it supports by providing a class that implements the *BeanInfo* interface. This interface is part of the *java.beans* package contained within JRE's default classes.

The BeanInfo interface provides methods for learning about the properties and methods about a target component. The class extending BeanInfo interface must be named adding *BeanInfo* as a suffix to the component's class name. For example, if we have a component called *Order*, then the bean info class will have *OrderBeanInfo* as the class name. A component's BeanInfo class may choose to expose only part of a bean's behavior.

For example, in our *Orders* class (see appendix for complete source code) we have a public method named `returnsTax()`:

```
1  ...
2      public boolean returnsTax()
3      {
4          return (this.calculateAmount() > 0);
5      }
6  ...
```

However this method is not included in the *OrdersBeanInfo* class description:

```
1  ...
2      public OrderBeanInfo()
3      {
4          super(Order.class);
5          // 'getItems'
6          FuegoMethodDescriptor method = addMethod("getItems");
7          method.setShortDescription("Get all items of the order");
8          // 'calculateAmount'
9          method = addMethod("calculateAmount");
10         method.setShortDescription("Calculate amount of order.");
11         System.err.println("Has catalogued public methods.");
12     }
13  ...
```

When we introspect the *Orders* class, the method `returnsTax()` is

not available (even though it is a public method) since it was not present in the *OrdersBeanInfo* class.

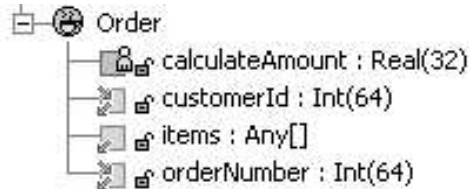


Figure 6: *BeanInfo* classes specify what members to expose

Fuego has developed a wrapper over these classes in order to simplify its usage. It can be seen from the examples included in the appendix below how to use these classes and their methods.

Note that whenever you are introspecting a jar file that contains *BeanInfo* classes, these will not show up in the introspection wizard:

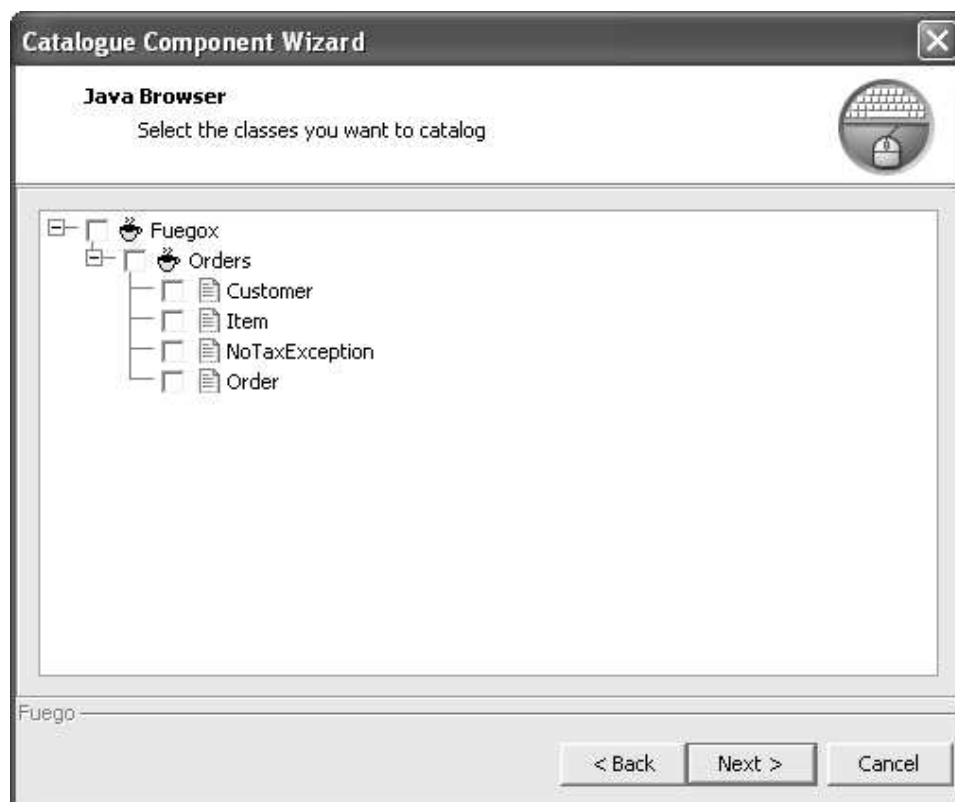


Figure 7: BeanInfo classes don't show up during introspection

## A Source Code

### A.1 Customer Class Example

```
1 package fuegox.orders;
2 import java.util.Vector;
3 import java.util.Enumeration;
4
5 public class Customer
6     implements java.io.Serializable
7 {
8     public long customerId;
9     private String firstname;
10    private String lastname;
11    protected String telephone;
12    private boolean preferredCustomer;
13
14    public Customer()
15    {
16    }
17
18    public void setLastname(String aName)
19    {
20        lastname = aName;
21    }
22
23    public String getLastname()
24    {
25        return lastname;
26    }
27
28    public void setPreferredCustomer(boolean isPreferred)
29    {
30        preferredCustomer = isPreferred;
31    }
32
33    public boolean isPreferredCustomer()
34    {
35        return preferredCustomer;
36    }
37
38    public void computeTax() throws NoTaxException
39    {
40    }
41
42    protected void reportTax()
43    {
```

```
44     }
45
46     public int getIRSRetentions()
47     {
48         return 0;
49     }
50 }
```

## A.2 Order Class Example

```
1 package fuegox.orders;
2
3 import java.util.Vector;
4 import java.util.Enumeration;
5
6 public class Order
7     implements java.io.Serializable
8 {
9     private long customerId_d = 0;
10    private long orderNumber_d = 0;
11    private float amount_d = 0;
12    private Vector items_d = null;
13
14    public Order()
15    {
16        customerId_d = 0;
17        orderNumber_d = 0;
18        amount_d = 0;
19        items_d = new Vector();
20    }
21
22    public void setCustomerId(long ci)
23    {
24        customerId_d = ci;
25    }
26
27    public long getCustomerId()
28    {
29        return customerId_d;
30    }
31
32    public void setOrderNumber(long on)
33    {
34        orderNumber_d = on;
35    }
36 }
```

```
37 public long getOrderNumber()  
38 {  
39     return orderNumber_d;  
40 }  
41  
42 public boolean addItem(Item i)  
43 {  
44     items_d.addElement(i);  
45     return true;  
46 }  
47  
48 public Vector.getItems()  
49 {  
50     return items_d;  
51 }  
52  
53 public float calculateAmount()  
54 {  
55     Enumeration e = items_d.elements();  
56     Item i = null;  
57     amount_d = 0;  
58     while (e.hasMoreElements() == true) {  
59         i = (Item)e.nextElement();  
60         amount_d += i.getQty() * i.getPrice();  
61     }  
62     return amount_d;  
63 }  
64  
65 public boolean returnsTax()  
66 {  
67     return (this.calculateAmount() > 0);  
68 }  
69 }
```

### A.3 OrderBeanInfo Class Example

```
1 package fuegox.orders;  
2  
3 import fuego.components.*;  
4 import java.beans.*;  
5 import java.lang.reflect.Method;  
6  
7 public class OrderBeanInfo  
8     extends FuegoBeanInfo  
9 {  
10     public OrderBeanInfo()
```

```
11 {
12     super(Order.class);
13     // 'getItems'
14     FuegoMethodDescriptor method = addMethod("getItems");
15     method.setShortDescription("Get all items of the order");
16     // 'calculateAmount'
17     method = addMethod("calculateAmount");
18     method.setShortDescription("Calculate amount of order.");
19     System.err.println("Has catalogued public methods.");
20 }
21 }
```

#### A.4 Item Class Example

```
1 package fuegox.orders;
2
3 public class Item
4     implements Java.io.Serializable
5 {
6     private String product_d = null;
7     private int quantity_d = 0;
8     private double price_d = 0.0;
9
10    public Item()
11    {
12        product_d = null;
13        quantity_d = 0;
14        price_d = 0.0;
15    }
16    public void setProduct(String product)
17    {
18        product_d = product;
19    }
20    public String getProduct()
21    {
22        return product_d;
23    }
24    public void setQty(int qty)
25    {
26        quantity_d = qty;
27    }
28    public int getQty()
29    {
30        return quantity_d;
31    }
32    public void setPrice(double p)
```



```
33     {  
34         price_d = p;  
35     }  
36     public double getPrice()  
37     {  
38         return price_d;  
39     }  
40 }
```

### A.5 ItemBeanInfo Class Example

```
1  package fuegox.orders;  
2  
3  import fuego.components.*;  
4  import java.beans.*;  
5  import java.lang.reflect.Method;  
6  
7  public class ItemBeanInfo  
8      extends FuegoBeanInfo  
9  {  
10     public ItemBeanInfo()  
11     {  
12         super(Item.class);  
13     }  
14 }
```