UJO Framework

the revolutionary architecture of the beans

version 0.80

http://ujoframework.org/

Pavel Pone(c), September 2008

History

year 2004

- modified objects from a framework Cayenne
- disadvantage is the poor type control

year 2007

- generic data types Java 5.0
- a core of the application jWorkSheet
- publishing a separate project UJO Framework 0.70 in October 2007

What is it?

- UJO - Unified Java Object
  - uniform architecture objects
  - single common method for a **writing** all attributes
  - single common method for a **reading** all attributes

- Features:
  - type-safe solution for writing the object attributes
  - easy introspection
  - serialization to **XML**, CSV, Resource bundle
  - JTable component support
  - implementation of useful functions:.
    toString (), clone (...), equals (ujo), the text conversion

Vision

- replace the JavaBeans by a Map model
- compilator checks a value data type !
- compilator checks a property key !

Java 5.0

```
Map<String,Object>
     bean = new HashMap();

bean.put("name", "Paul");
bean.put("cash",  10d);

Double cash = (Double)
     bean.get("cash");
```

UJO Framework:

```
Person bean = new Person();


bean.set(Person.NAME,"Paul");
bean.set(Person.CASH, 10d);

Double cash
     = bean.get(Person.CASH);
```

Confrontation JavaBeans & UJO

Information about the property types are located on the one place of UJO object.

Java 5.0

```
class Person extends
             Object {

 private Double cash = 0d;

 public Double getCash() {
   return cash;
 }

 public
 void setCash(Double cash){
   this.cash = cash;
 }
}
```

UJO Framework:

```
class Person extends
      MapUjoExt<Person> {

 public static final
 UjoProperty<Person,Double>
 CASH = newProperty("Cash",
        0d);
}
```

Confrontation JavaBeans & UJO

How to handle UJO attributes?

JavaBeans object

```
class Person extends
             Object {

 ....

 void addCash(double val) {
   double newCash
     = this.cash + val;
   this.cash = newCash;
 }
}
```

UJO object

```
class Person extends
      MapUjoExt<Person> {

 ....

 void addCash(double val) {
   double newCash
     = get(CASH) + val;
   set(CASH, newCash);
 }
}
```

Default property values

- JavaBean properties can have got a **default value**
  { Integer i=10; }
- UJO property can have got a similar **default value**
- through reading of a property is an undefined value **null** replaced by the default value of the UjoProperty so the default value can be restored anytime (set the **null**) !
- use a 'property type' parameter to create a new property with the **null** default value

```
class Person extends MapUjoExt<Person> {

    public static final UjoProperty<Person,Double>
                                CASH = newProperty("Person", 0d);

    /** The method never returns null ! */
    public Double getCash() {
        return get(CASH);
    }
}
```

Chaining of the properties

- UJO properties can be **chained** over more objects
- chaining is type safe in the compilation time
- next sample uses an ADDRESS attribute of a Person class

```
import static Person.*;
import static Address.*;

Person bean = new Person();

bean.set(ADDRESS, new Address();
bean.set(ADDRESS, STREET, "Videnska");
bean.set(ADDRESS, CITY  , "Brno");

String street = bean.get(ADDRESS, STREET);
String city   = bean.get(ADDRESS, CITY);
```

Chaining of setters

- calling of the method **set()** can be **chained** for writing more properties on the one source line

```
import static Person.*;

Person bean = new Person();
bean.set(NAME, "Pavel").set(CASH, 100d);

String name = bean.get(NAME);
double cash = bean.get(CASH);
```

List of UJO objects

- an attribute of UJO can be a List too, however for the data type is dedicated the interface **UjoPropertyList**
- an class **UjoExt** provides for the property some useful methods

```
class Person extends MapUjoExt<Person> {

 public static final MapPropertyList<Person,Address>
    ADDRESSES = newPropertyList("Address", Address.class);

 void test() {
    add(ADDRESSES, new Address());       // list is created
    int  count = getItemCount(ADDRESSES);
    Address ad = get(ADDRESSES, 0);      // a value from pos. 0

    List<Address> adr1 = get (ADDRESSES);
    List<Address> adr2 = list(ADDRESSES); // not null allways

  }
}
```

Text handling (1)

The real applications work with a text format:

- edit value in graphical user interface
- serialization from/to the text format like xml, csv, ...
- HTTP request parameters
- debugging

UJO Framework provides an support of text conversion of the UJO objects

Text handling (2)

There are the three ways to a text conversion:

- parent class `SuperUjoExt` supports the most usual Java objects by the method:
  ```
  setText(UjoProperty property,
                           String value)
  ```
- the your implementation (or overwriting) of the method `UjoTextable.writeValueString(...)`
- framework can works with a feature **`PropertyTextable (ValueTextable)`** of an object.

the object constructor with the one parameter type of String means the format, which creates a method toString ().

Text handling (3)

The next code writes and reads a number in a text format

```
import static Person.*;

Person bean = new Person();

bean.setText(CASH, "1.379");
String cash = bean.getText(CASH);

// PropertyTextable test:
new Double(cash).toString().equals(cash);
```

More information you can find in a JavaDoc of PropertyTextable (ValueTextable since 0.85).

XML export (1)

- **six times** higher speed in comparison to XMLEncoder / XMLDecoder.
- deserialization is **about 10% faster** in compare to JAXB 2.1 by using the implementation ArrayUjo.
- serialization is **slower by 44%** in compare to JAXB 2.1

```
Person person = new Person();
person.set(NAME    , "Joseph");
person.add(ADDRESSES, new Address("Brno", "Videnska"));

// Make Serialization:
UjoManagerXML.getInstance().saveXML(writer, person, null,
                            "My Export");

// Make Deserialization:
person = UjoManagerXML.getInstance().parseXML(inputStream,
                            Person.class,
                            "My Import");
```

XML export (2)

- do you need to disable the export of some attributes?
- you may overwrite the method Ujo.readAuthorization()
  this method is used to authorize an action in relation to: property, value, type and context of events.

```
class Person extends MapUjoExt<Person> {
 public static final UjoProperty<Person,String>
        NAME = newProperty("Person", String.class);
   ....

 /** Method disable an export of the NAME attribute */
 public boolean readAuthorization(UjoAction action,
   UjoProperty property, Object value) {
     return action.getType()==UjoActions.ACTION_XML_EXPORT
     && property==NAME
     ? false : super.readAuthorization( -"- );
 }
}
```

XML export (3)

- each property value is exported to a separate element in the XML file
- any UJO property (no List) you can mark for an export to a element **attribute** by a one of the next way:
    - overwrite method `Ujo.readAuthorization()`
    - mark the property by an annotation XmlAttribute

```
class Person extends MapUjoExt<Person> {

 @XmlAttribute
 public static final UjoProperty<Person,String>
        NAME = newProperty("Person", String.class);

}
```

Interfaces Ujo & UjoExt

- all previous information are related to **UjoExt** interface (UJO extended)
- the UjoExt provides more conservative and therefore readable API.
- the UjoExt brings the new possibility of chaining setters
- the UjoExt is supported from UJO Framework 0.80
- the both of the interface facilities are approximately identical
- core of the framework works with an original Ujo interface only
- the main difference is that:

  the **Ujo** have got a part of its key methods
  declared in UjoProperty !

Differences Ujo & UjoExt

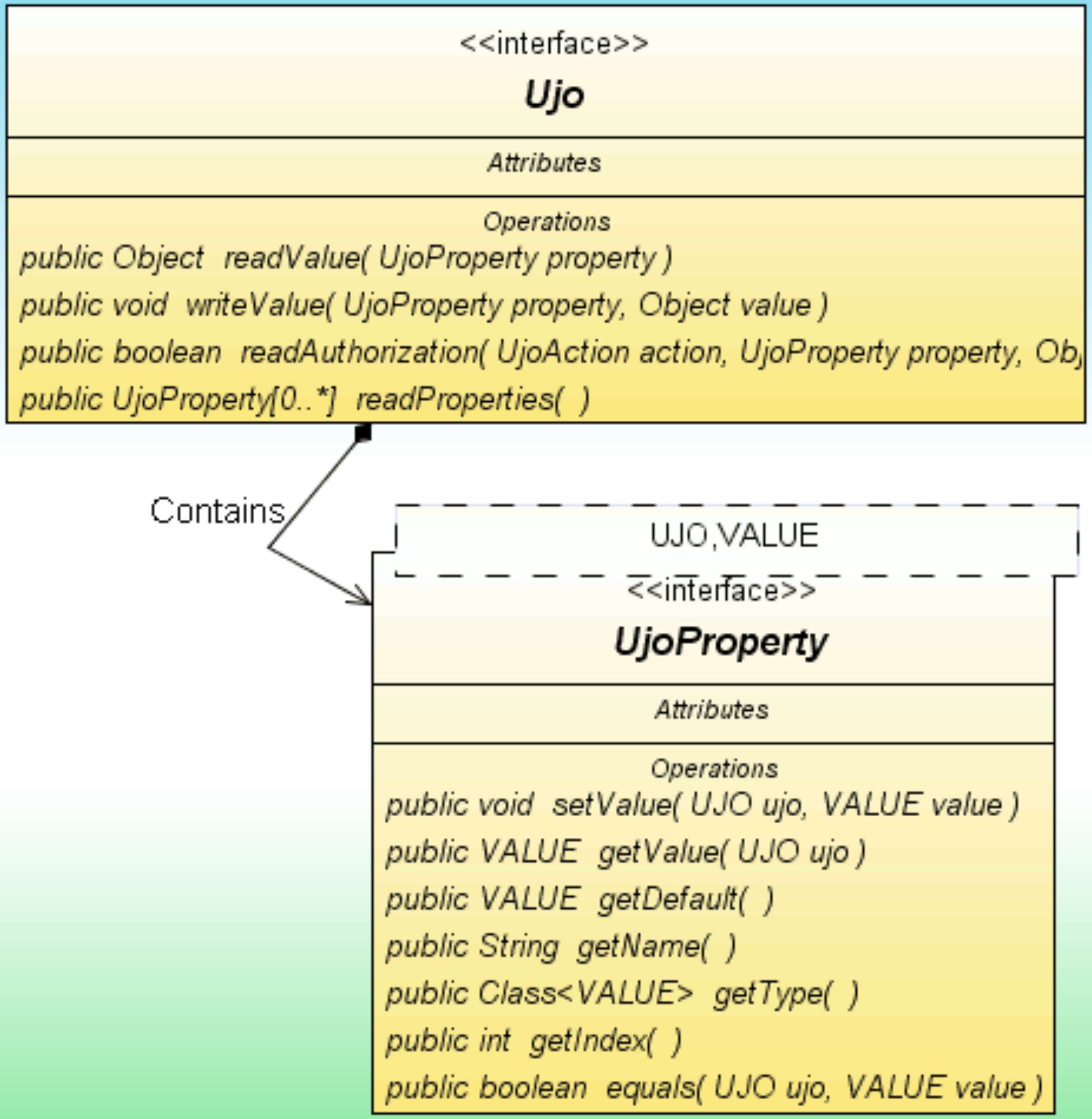
Ujo

- too revolutionary API :-)
- easy implementation
- better type safe feature of child properties
- little risk of collision with another object method interfaces
- methods may be a little faster

UjoExt

- readable API
- laborious implementation
- worse type safe feature of child properties
- a large number of methods bear a higher risk of collision with another interface
- support properties and setters chaining

Basic two interfaces



<<interface>>
Ujo
Attributes
Operations
public Object readValue(UjoProperty property)
public void writeValue(UjoProperty property, Object value)
public boolean readAuthorization(UjoAction action, UjoProperty property, Ob
public UjoProperty[0..*] readProperties( )
Contains
UJO,VALUE
<<interface>>
UjoProperty
Attributes
Operations
public void setValue(UJO ujo, VALUE value)
public VALUE getValue(UJO ujo)
public VALUE getDefault( )
public String getName( )
public Class<VALUE> getType( )
public int getIndex( )
public boolean equals(UJO ujo, VALUE value)

Basic two interfaces

Interface Ujo:

- implementation includes business data
- method for authorization properties
- gateway to an introspection (provides list of UjoProperties)

Interface UjoProperty

- provides property features (meta data)
- contains a default value
- provides type safe methods for reading and writing values
- never contains business data!

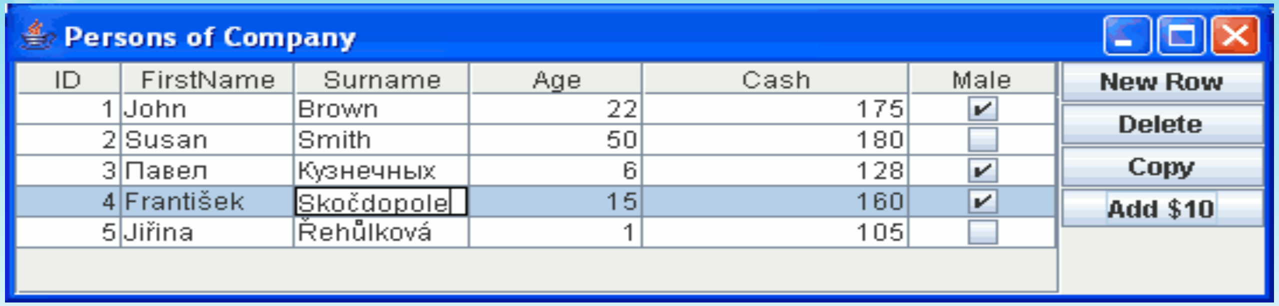
Basic Ujo implementations

There are some abstract classes for an easy implementation with a different features:

- **MapUjo** - is suitable for simple implementation with sufficient power for common applications, it is based on the object HashMap
- **ArrayUjo** - the high performance is implemented by the object array
- **PojoUjo** - implementation calls related methods of a JavaBean directly using a Java reflection
- **XxxUjoExt<UjoImpl>** - for each implementation exists an extended class with ends the **Ext**

Samples of usage

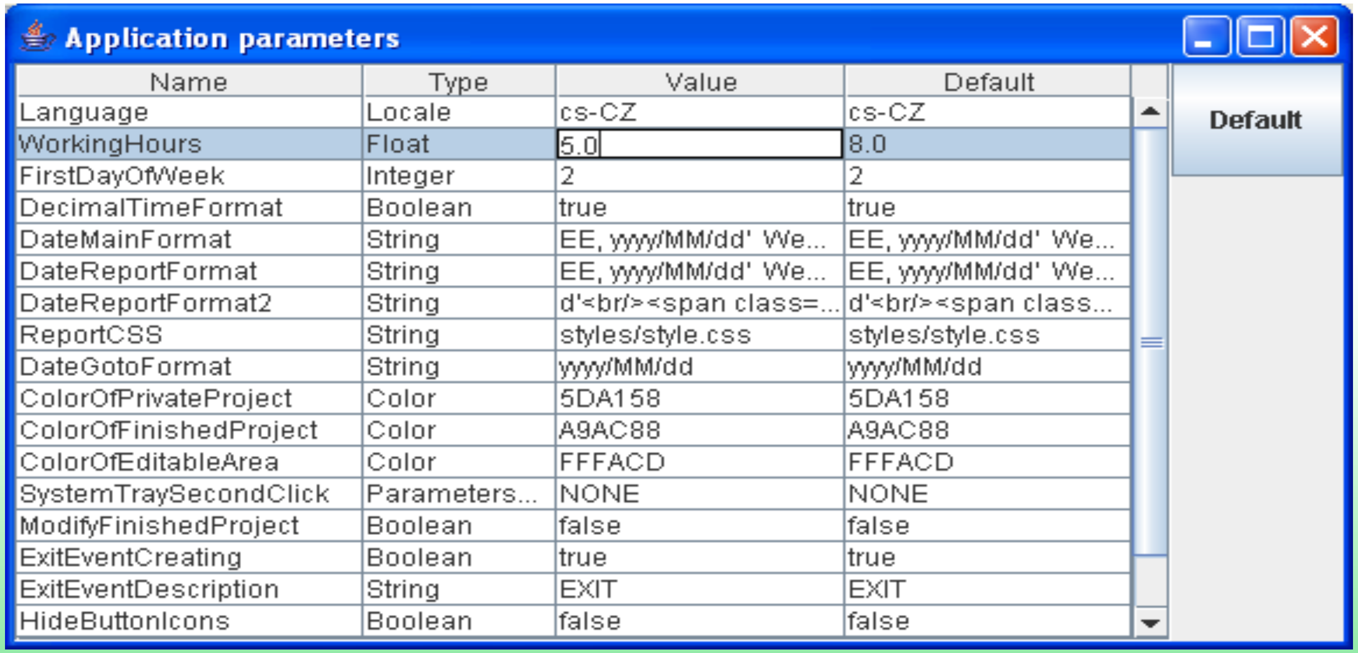
- Persistent table to a XML file



Persons of Company

ID	FirstName	Surname	Age	Cash	Male
1	John	Brown	22	175	✓
2	Susan	Smith	50	180	☐
3	Павел	Кузнечных	6	128	✓
4	František	Skočdopole	15	160	✓
5	Jiřina	Řehůlková	1	105	☐

  - **99 rows** of source code only
  - this is a link to a source code
- Maintenance of the application parameters (link)



Application parameters

Name	Type	Value	Default
Language	Locale	cs-CZ	cs-CZ
WorkingHours	Float	5.0	8.0
FirstDayOfWeek	Integer	2	2
DecimalTimeFormat	Boolean	true	true
DateMainFormat	String	EE, yyyy/MM/dd' We...	EE, yyyy/MM/dd' We...
DateReportFormat	String	EE, yyyy/MM/dd' We...	EE, yyyy/MM/dd' We...
DateReportFormat2	String	d'<br/><span class=...	d'<br/><span class...
ReportCSS	String	styles/style.css	styles/style.css
DateGotoFormat	String	yyyy/MM/dd	yyyy/MM/dd
ColorOfPrivateProject	Color	5DA158	5DA158
ColorOfFinishedProject	Color	A9AC88	A9AC88
ColorOfEditableArea	Color	FFFACD	FFFACD
SystemTraySecondClick	Parameters...	NONE	NONE
ModifyFinishedProject	Boolean	false	false
ExitEventCreating	Boolean	true	true
ExitEventDescription	String	EXIT	EXIT
HideButtonIcons	Boolean	false	false

The benefits and disadvantages

- an easy introspection
- accessible characteristics of UJO properties include a default value
- property list handling (no values)
- the object itself authorises the use of their properties
- an easy implementation of a **generic** functions, e.g:
  - proxy for a class with the common parent
  - generic property listener, ...
- JTable component support
- small framework size (50 kB)
- an open source

- non-traditional architecture
- weak reference
- limited direct support of J2EE services

Further development

- a maintainance of the core only
- rather the development of modules
  - UJO dependency injection (?)
  - data binding (?)
  - ORM support (?)
  - many other directions for a development
- limited time for development

Conclusion

- 4,500 downloads of the **jWorkSheet** per 10 months
- very small size of the application, good performance
- positive feedback and reviews
- access to a home page from large companies

- 210 downloads of the **UJO Framework** per 10 months
- practically seamless core
- easy available some useful methods
- quick development of the the jWorkSheet application

I welcome comments, questions and notice of any errors (documentation, software)

Thank you for your attention

UJO Framework homepage: http://ujoframework.org/
Link to presentation:http://ujoframework.org/presentation/

Pavel Pone(c), September 2008