

OCB: An Object / Class Browser for (Persistent) Java

Graham Kirby and Ron Morrison

University of St Andrews, Scotland

With

Malcolm Atkinson, Laurent Daynès

University of Glasgow, Scotland

Mick Jordan, Michael Van De Vanter

Sun Labs

Motivations

Persistent store visualisation

- programmers
- users of persistent applications

Initiated in response to need identified in PJama work

- but not closely tied to that or any persistence model
- may be useful with standard Java systems

Design Goals

General

- simple and consistent user interface
- maximum portability
- quick implementation

Specific

- representations of objects and classes
- access to persistent roots
- navigation through object reference graph
- use as stand-alone application or through API
- customised display styles
 - for specific classes
 - control over display of inherited members

User Interface

Object pane

- names and access modifiers of fields
- values of fields subject to modifiers
 - scalars show values
 - objects show links

Class pane

- names and types of fields
- static fields
- methods and static methods
- constructors
- superclass
- interfaces



Java Object/Class Browser

Browser Instance Class Preferences

Instance: Student

Class Student

fields

address Address

age 19

children Person[]

■ *id* inaccessible

name Josephine Bloggs

number 4712

spouse Student

fields

address Address

age int

children Person[]

■ *id* int

name java.lang.String

number int

spouse Student

static fields

■ *lastId* inaccessible

numberOfPeople 2

methods

addChild void (Person)

getSpouse Person ()

incAge void ()

marry void (Student)

marry void (Person)

static methods

constructors

Student (java.lang.String, int, Address, int)

superclass

Person

interfaces

java.lang.Cloneable



Object

Animal

Person

Student

Java Object/Class Browser

Browser Instance Class Preferences

Instance:

PrivateInfo	inacc
daemon	inacc
doDispatch	inacc
eeetop	inacc
group	inacc
initial_stack_memory	inacc
name	inacc
priority	inacc
single_step	inacc
stillborn	inacc
target	inacc
theQueue	inacc
thread0	inacc

Class java.awt.EventQueueThread

static fields

thread0	java.lang.Thread
MAX_PRIORITY	10
MIN_PRIORITY	1
NORM_PRIORITY	5
activeThread0	inaccessible
threadInkNumber	inaccessible

methods

checkAccess	void ()
countStackFrames	int ()
destroy	void ()
exit	void ()
getName	java.lang.String ()
getPriority	int ()
getThreadGroup	java.lang.ThreadGroup ()
init	void (java.lang.Th...
interrupt	void ()
interrupt0	void ()
isAlive	boolean ()
isDaemon	boolean ()
isInterrupted	boolean ()
isInterrupted	boolean (boolean)
join	void (long) throw:
join	void (long, int) th
join	void () throws jav

Modifiers

- public
- protected
- private
- final
- abstract
- transient
- volatile
- synchronized
- native

Object Thread EventDispatchThread

Browsing Persistent Stores

PJama:

- lists of persistent roots and classes accessible via menu

Other persistent Java systems:

- need to add code to use appropriate persistence API

Pinning

Default behaviour:

- following a link overwrites current window display
 - if an object, its class is displayed
 - if a class, object pane is cleared unless the object is an instance of a subclass
- revisit past states with forward and back arrows

Pinned mode:

- selected using checkbox
- new state displayed in a new window
- existing window remains as it is

The image displays two overlapping windows from the Java Object/Class Browser. The larger window in the foreground is titled "Java Object/Class Browser" and shows details for an instance and its class.

Instance: Student

address	Address
age	23
children	Person[]
id	inaccessible
name	Max Hastings
number	5813
spouse	Student

Class Student

fields

address	Address
age	int
children	Person[]
id	int
name	java.lang.String
number	int

At the bottom of the window is a class hierarchy slider with a "Pin" checkbox and a double arrow icon. The slider shows the hierarchy: Object -> Animal -> Person -> Student. Red vertical markers are positioned at the Animal and Student levels.

The smaller window in the background, titled "Java", shows similar details for another instance of Student:

Instance: Student

address	Address
age	19
children	Person[]
id	inaccessible
name	Josephine Bloggs
number	4712
spouse	Student

It also shows a class hierarchy slider with markers at the Animal and Student levels.

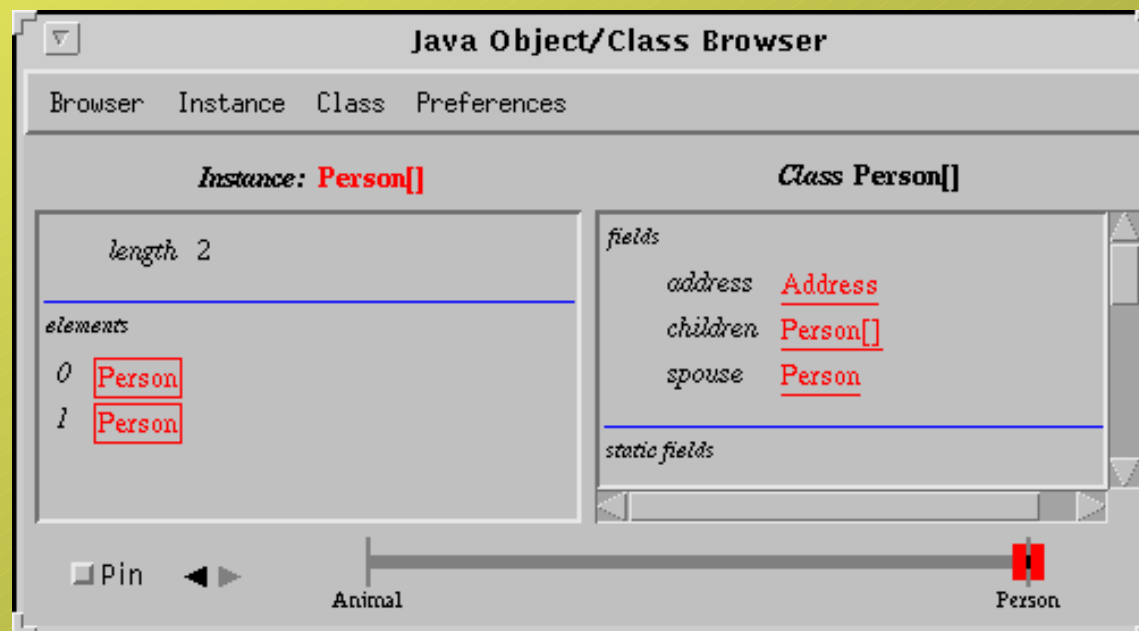
Arrays

Object pane

- number of elements and list of element values

Class pane

- base element class



Customisation

Customised display methods can be set for particular classes

- method returns a string for given instance of the class
- OCB API provides method to register such methods
- alternatively can override *toString* method in class definition
 - if OCB use is planned at time of defining classes

Java Object/Class Browser

Browser Instance Class Preferences

Instance: Student

address 34 High Street, Kirkcaldy
children Person[]
number 4712
spouse Student

Class Student

fields

address Address
children Person[]
number int
spouse Student

static fields

Pin

Object Animal Person Student

The screenshot shows the Java Object/Class Browser window. The title bar reads 'Java Object/Class Browser'. Below the title bar is a menu bar with 'Browser', 'Instance', 'Class', and 'Preferences'. The main area is split into two panes. The left pane, titled 'Instance: Student', shows the instance's state: 'address' is '34 High Street, Kirkcaldy', 'children' is 'Person[]', 'number' is '4712', and 'spouse' is 'Student'. The right pane, titled 'Class Student', shows the class's fields: 'address' is 'Address', 'children' is 'Person[]', 'number' is 'int', and 'spouse' is 'Student'. Below the panes is a class hierarchy diagram with a 'Pin' button on the left and a horizontal line with markers for 'Object', 'Animal', 'Person', and 'Student'. The 'Person' and 'Student' markers have red vertical bars above them.

Viewing Inherited Members

For clarity may want to hide some class members

- hide members defined in a superclass if they're not currently relevant
 - e.g. *clone* and *hashCode* methods defined in *Object*
- hide members defined in the most specific class to use a superclass view
 - e.g. view a *Student* as though it were a *Person*

Controlled through *top* and *bottom* settings

- applied to current view
- set by user with double slider
- members defined outside that range are hidden

```
public class Animal {  
  
    public String name;  
    public int age;  
    public Animal( String name, int age ) {...}  
    public void incAge() {...}  
}  
  
public class Person extends Animal {  
  
    public Person spouse;  
    public Person[] children;  
    public Address address;  
    public static int numberOfPeople = 0;  
    public Person( String name, int age, Address address ) {...}  
    public Person getSpouse() {...}  
    public void marry( Person spouse ) {...}  
    public void addChild( Person child ) {...}  
}  
  
public class Student extends Person implements Cloneable {  
  
    public Student spouse;  
    public int number;  
    public Student( String name, int age, Address address, int number ) {...}  
    public void marry( Student spouse ) {...}  
}
```

Java Object/Class Browser

Browser Instance Class Preferences

Instance: Student **Class Student**

number 4712
spouse [Student](#)

fields
number int
spouse [Student](#)

static fields

methods
marry void ([Student](#))

static methods

constructors
Student ([java.lang.String](#), int, [Address](#), int)

enum values

Pin <> Object Animal Person **Student**

The screenshot shows the Java Object/Class Browser window. The title bar reads "Java Object/Class Browser". Below the title bar is a menu bar with "Browser", "Instance", "Class", and "Preferences". The main area is split into two panes. The left pane, titled "Instance: Student", shows the instance's state with "number" set to 4712 and "spouse" set to a "Student" object. The right pane, titled "Class Student", shows the class's structure, including fields (number: int, spouse: Student), static fields, methods (marry: void (Student)), static methods, and a constructor (Student (java.lang.String, int, Address, int)). At the bottom, a class hierarchy diagram shows the path from Object to Animal to Person to Student, with Student highlighted in red.

Java Object/Class Browser

Browser Instance Class Preferences

Instance: Student

address 34 High Street, Kirkcaldy
age 19
children Person[]
 ■ *id* inaccessible
name Josephine Bloggs
number 4712
spouse Student

Class Student

static fields

■ *lastId* inaccessible
numberOfPeople 2

methods

■ *addChild* void (Person)
 ■ *clone* java.lang.Object () throws java.lang.CloneNotSupportedException
 ■ *equals* boolean (java.lang.Object)
 ■ *finalize* void () throws java.lang.Throwable
 ■ *getClass* java.lang.Class ()
 ■ *getSpouse* Person ()
 ■ *hashCode* int ()

Pin ◀ ▶

Object Animal Person Student

Java Object/Class Browser

Browser Instance Class Preferences

Instance: Student

address 34 High Street, Kirkcaldy
age 19
children Person[]
■ *id* inaccessible
name Josephine Bloggs
number 4712
spouse Student

Class Student

static fields

■ *lastId* inaccessible
numberOfPeople 2

methods

addChild void (Person)
getSpouse Person ()
incAge void ()
marry void (Student)
marry void (Person)

static methods

Pin

Object Animal Person Student

Java Object/Class Browser

Browser Instance Class Preferences

Instance: Student

address 34 High Street, Kirkcaldy

age 19

children Person[]

■ *id* inaccessible

name Josephine Bloggs

spouse Person

Class Student

spouse Person

static fields

■ *lastId* inaccessible

numberOfPeople 2

methods

addChild void (Person)

getSpouse Person ()

incAge void ()

marry void (Person)

static methods

Pin

Object

Animal

Person

Student

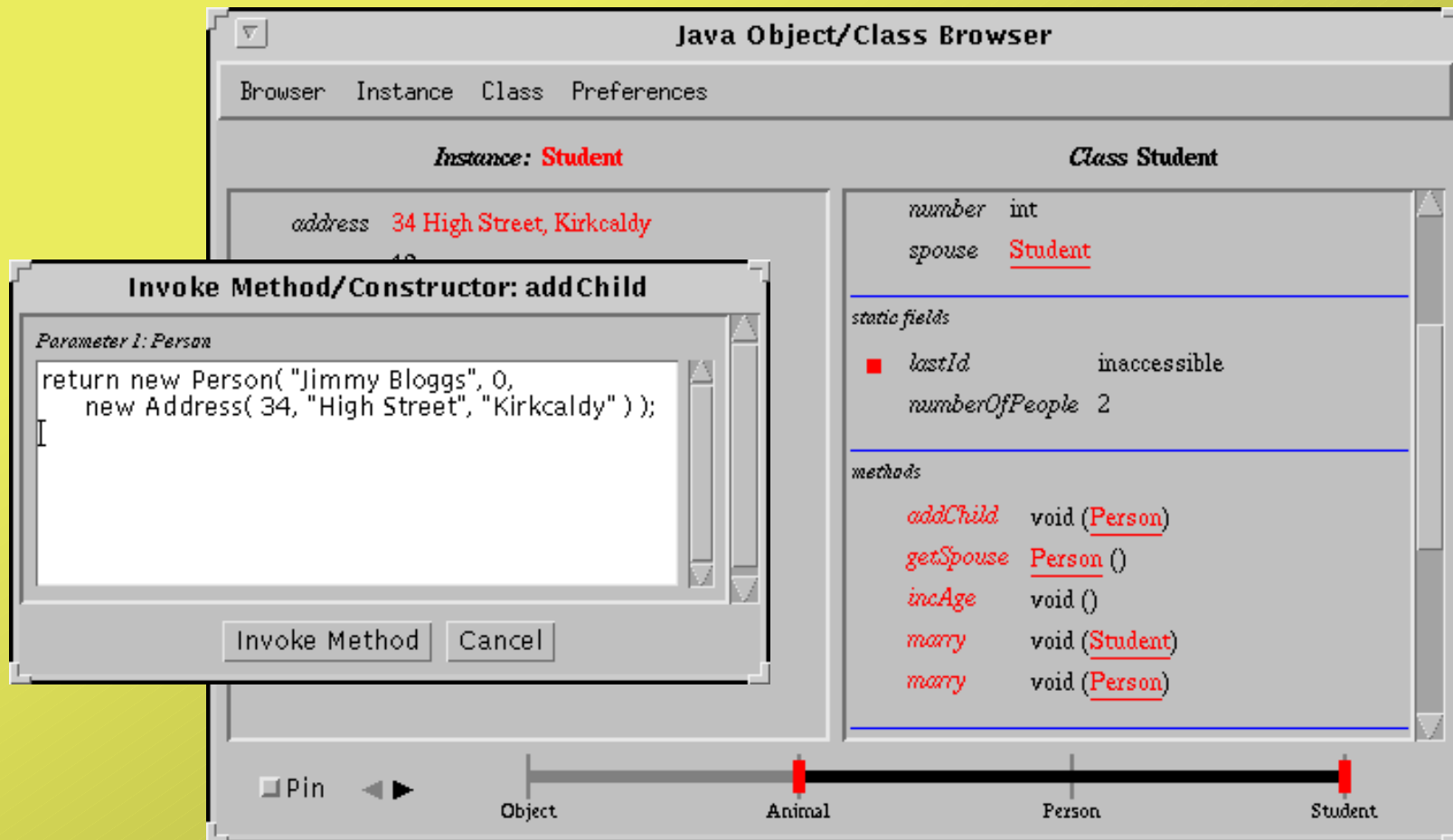
Invoking Methods

Methods can be called interactively

- may require parameter values
- user enters fragment of code for each parameter
 - each returns an *Object*
- compiled and executed dynamically

Possible failures

- invalid code fragments
- types of parameters don't match



API

```
package ocb;
import java.awt.*;

/**
 * Defines the interface for classes that implement OCBs.
 */
public interface OCBInterface {

    public void displayObject( Object anObject );
    public void displayClass( Class aClass );
    public Object getDisplayedObject();
    public Class getDisplayedClass();
    public void clear();
    public void clear( String paneName );
    public void close();
    public Container getOCBContainer();
    public void addCallback( Callback cb );
    public void removeCallback( Callback cb );
    public Callback[] getCallbacks();
    public void addCustomDisplay( Class theClass, DisplayAsString customDisplay );
    public void removeCustomDisplay( Class theClass );
}

public class OCB implements OCBInterface {

    public OCB() {...}
    public OCB( Point position, Dimension size ) {...}
    public OCB( Container container ) {...}

    ...
}
```

Implementation

Object details discovered using core reflection package

- comprehensive, fairly easy to use

Dynamic compilation

- with current reflection support this requires platform-specific implementation
 - fork process to invoke *javac*
 - call compiler implementation classes
- could be cleanly supported by reflection package

Implementation Status

Fully implemented

- except:
 - customisation of colour coding
 - loading and saving settings
- only tested on SPARC/Solaris displaying on OpenWindows
- access to PJama persistent roots not tested

Requirements

- any JVM supporting JDK 1.1 or later

Availability

- freely available, details at

<http://www-ppg.dcs.st-and.ac.uk/Java/OCB/>

Further Development

Static visualisation

- object graph structures
- define new classes during method invocation
- graphical customisation
- method source code
- security issues e.g. viewing private members

Dynamic visualisation

- active monitoring
- thread states
 - reflection package support?

Conclusions

Implemented:

- object and class display tool in pure Java
 - with advantages and disadvantages that gives
- flexible class views

To do:

- use with persistent stores
- evaluation of practical usefulness