

## Design Patterns Used in Eclipse

Ilya Shinkarenko

Bangalore, Eclipse Summit India 2009

18.07.2009



Eclipse Training Alliance  
we share expertise

WEIGLEWILCZEK

## About me: Ilya Shinkarenko

- » Software Engineer / Architect
  - » Eclipse RCP, OSGi
  - » Spring, JBoss AS
  - » Other stuff: Android OS, Flash/Flex, Red5 etc.
- » Working in IT since 1997
  - » Java since 1999
  - » Eclipse expertise since 2004
- » Consulting, workshops, seminars
  - » Eclipse RCP and basically everything around Eclipse
  - » HCI, Usability, Screen Design
  
- » E-Mail: [ilya@shinkarenko.org](mailto:ilya@shinkarenko.org)



Eclipse Training Alliance  
we share expertise

2

WEIGLEWILCZEK

## About the Eclipse Training Alliance

- » Founded December 2005 by WeigleWilczek and Innoopract
  - » WeigleWilczek now drives the initiative
- » Offerings:
  - » International one-stop delivery of high quality Eclipse training classes
  - » Certificates
  - » Coaching and consulting

We share expertise



Eclipse Training Alliance  
we share expertise

3

**WEIGLEWILCZEK**

## The Partners

**WEIGLEWILCZEK**



**ANCI**  
CONSULTING



**itemis**



Eclipse Training Alliance  
we share expertise

4

**WEIGLEWILCZEK**

## Our Training Classes

- » RCP
  - » Developing RCP
  - » Advanced RCP
- » OSGi
  - » Developing OSGi
  - » Advanced OSGi
- » Other
  - » Administration Training
  - » Architecture Training
  - » Developer Training
  - » Eclipse SOA Workshop
  - » HYPERIC System Administration Training
  - » Methodology Training
- » Modelling
  - » Eclipse Modeling (EMF & GMF)
  - » EMF Professional
  - » Graphical Modeling with GMF
  - » Model-Driven Development with Eclipse Modeling
  - » MDD in the context of Software Engineering
  - » Advanced MDD with Eclipse Modeling and openArchitectureWare



Eclipse Training Alliance  
we share expertise

5

**WEIGLEWILCZEK**

## Contact Europe

Please contact us for details:

Eclipse Training Alliance  
c/o Weigle Wilczek GmbH  
Heiko Seeberger  
Martinstrasse 42-44  
D-73728 Esslingen

Phone +49 711 45 99 98 0  
Fax +49 711 45 99 98 29  
[www.eclipse-training.net](http://www.eclipse-training.net)

The screenshot shows the Eclipse Training Alliance website with the following content:

- Navigation:** About us | Training courses | On-site | Certification
- Header:** Eclipse Training Alliance logo and "we share expertise" tagline. Large "ECLIPSE" text with "RCP OSGi Modeling SOA" listed to the right.
- Training courses:** Avail yourself of our specialist long-time Eclipse expertise. [more »](#)
- On-site:** Need flexibility regarding contents and location? [more »](#)
- Certification:** Become a "Certified Eclipse Developer"! 25% off\* for orders until 01.04.09.
- The Eclipse Training Alliance - we share expertise:**
  - The Eclipse Training Alliance is a unique pool of experts throughout the world committed to applied Eclipse and OSGi related learning.
  - Founded in 2005 by WeigleWilczek, a leader in application development with OSGi and Eclipse RCP - the Alliance pools the extensive Eclipse know-how of its associate companies. This concept allows us to cover a broad scope of topics - worldwide and in real-time.
  - Capitalize on many years of Eclipse expertise either in one of our scheduled training courses or in tailored on-site training and coaching sessions.
- News:** Visit us on EclipseCon! See the talks of our partners.
- Footer:** Legal notice | FAQ



Eclipse Training Alliance  
we share expertise

6

**WEIGLEWILCZEK**

## Contact India

Please contact us for details:

ANCiT Consulting  
Mr.Imran S  
84, Rajiv Gandhi Nagar,  
Sowripalayam,  
Coimbatore, TN, India  
PIN 641028

T +91-422-6461778  
[certification@ancitconsulting.com](mailto:certification@ancitconsulting.com)

**ANCiT**  
CONSULTING



Eclipse Training Alliance  
we share expertise

7

**WEIGLEWILCZEK**

## About You

- » Are you at the moment / have you been actively involved in:
  - » ... development with Eclipse RCP?
  - » ... development of plug-ins for Eclipse IDE?
  - » ... development for OSGi / Equinox platform?



Eclipse Training Alliance  
we share expertise

8

**WEIGLEWILCZEK**

## Classical Pattern Catalog

- » Creational Patterns
  - » **Factory**
  - » **Builder**
  - » **Singleton**
  - » ...
- » Structural Patterns
  - » **Adapter**
  - » **Bridge**
  - » **Composite**
  - » **Proxy**
  - » **Facade**
  - » ...
- » Behavioral Patterns
  - » **Observer**
  - » **Command**
  - » **Memento**
  - » **Strategy**
  - » **Visitor**
  - » ...



## Patterns in Eclipse

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>» Classical Pattern Catalog           <ul style="list-style-type: none"> <li>» Creational Patterns               <ul style="list-style-type: none"> <li>» <b>Factory</b></li> <li>» <b>Builder</b></li> <li>» <b>Singleton</b></li> <li>» ...</li> </ul> </li> <li>» Structural Patterns               <ul style="list-style-type: none"> <li>» <b>Adapter</b></li> <li>» <b>Bridge</b></li> <li>» <b>Composite</b></li> <li>» <b>Proxy</b></li> <li>» <b>Facade</b></li> <li>» ...</li> </ul> </li> <li>» Behavioral Patterns               <ul style="list-style-type: none"> <li>» <b>Observer</b></li> <li>» <b>Command</b></li> <li>» <b>Memento</b></li> <li>» <b>Strategy</b></li> <li>» <b>Visitor</b></li> <li>» ...</li> </ul> </li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>» Eclipse Pattern Catalog           <ul style="list-style-type: none"> <li>» Platform               <ul style="list-style-type: none"> <li>» <b>Singleton</b>: getting Workbench / Plug-in / Service</li> </ul> </li> <li>» OSGi               <ul style="list-style-type: none"> <li>» <b>Bridge</b>: OSGi Services</li> <li>» <b>Whiteboard</b>: pluggable listeners</li> </ul> </li> <li>» Workspace Resources               <ul style="list-style-type: none"> <li>» <b>Proxy</b> and <b>Bridge</b>: Accessing File System</li> <li>» <b>Composite</b>: the workspace</li> <li>» <b>Observer</b>: tracking resource changes</li> <li>» <b>Visitor</b>: traversing the resource tree</li> </ul> </li> <li>» Core Runtime               <ul style="list-style-type: none"> <li>» <b>IAdaptable</b> and <b>Adapter Factories</b>: Property View</li> </ul> </li> <li>» SWT               <ul style="list-style-type: none"> <li>» <b>Composite</b>: composing widgets</li> <li>» <b>Strategy</b>: defining the layout</li> <li>» <b>Observer</b>: responding to events</li> </ul> </li> <li>» JFace               <ul style="list-style-type: none"> <li>» <b>Pluggable adapter</b>: Connecting widget to model</li> <li>» <b>Strategy</b>: customize a viewer without subclassing</li> <li>» <b>Command</b>: Actions</li> </ul> </li> <li>» UI Workbench               <ul style="list-style-type: none"> <li>» <b>Memento</b>: persisting UI state</li> <li>» <b>Virtual Proxy</b>: lazy loading with E.P.</li> </ul> </li> </ul> </li> </ul> |
|---|---|

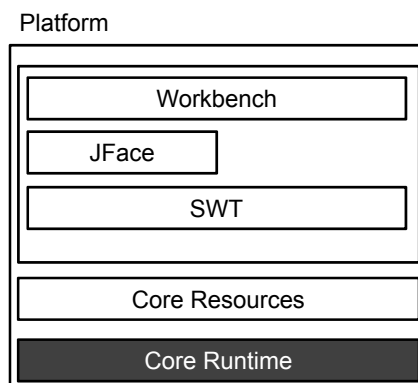


## Patterns in Eclipse

- » Platform Runtime
  - » **Adaptable** and **Adapter Factories**: Property View
  - » **Singleton**: getting a Workbench / Plugin / Service instance
  - » **Bridge**: OSGi Services
  - » **Whiteboard**: pluggable listeners
- » Workspace Resources
  - » **Proxy** and **Bridge**: accessing File System
  - » **Composite**: the workspace
  - » **Observer**: tracking resource changes
  - » **Visitor**: traversing the resource tree
- » SWT
  - » **Composite**: composing widgets
  - » **Strategy**: defining the layout
  - » **Observer**: responding to events
- » JFace
  - » **Pluggable adapter**: connecting widget to model
  - » **Strategy**: customize a viewer without subclassing
  - » **Command**: Actions
- » UI Workbench
  - » **Memento**: persisting UI state
  - » **Virtual Proxy**: lazy loading with E.P.

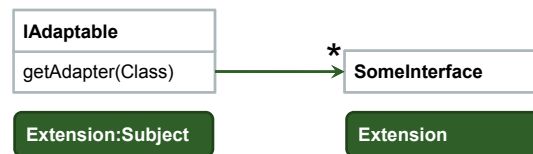


## Runtime



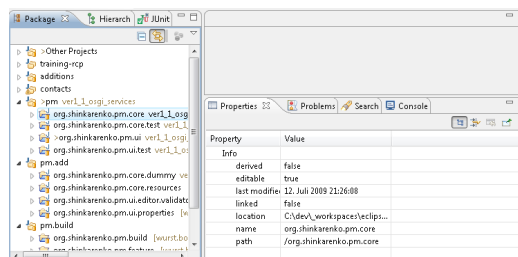
## Extension Interface: IAdaptable

- » "Anticipate that an object's interface needs to be extended in the future. An Extension Object lets you add interfaces to a class and lets clients query whether an object has a particular extension."



## Need for Adapters

- » Add a service interface to a type without exposing it in that type
- » Add behavior to preexisting types such as `IFile`, `Person`, etc.
- » Our goal: we want to adapt `Person` to `IPropertySource`



## Getting an Adapter:

```

ViewsPlugin.java
public static Object getAdapter(Object sourceObject, Class adapter) {
    //1. Check instance
    if (adapter.isInstance(sourceObject)) {
        return sourceObject;
    }

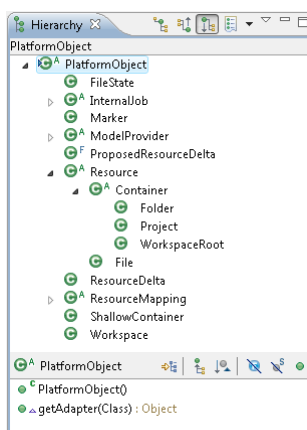
    //2. Check if can adapt to
    if (sourceObject instanceof IAdaptable) {
        IAdaptable adaptable = (IAdaptable) sourceObject;
        Object result = adaptable.getAdapter(adapter);
        if (result != null) {
            // Sanity-check
            Assert.isTrue(adapter.isInstance(result));
            return result;
        }
    }

    //3. Load adapter from the Platform Adapter Manager
    return Platform.getAdapterManager().loadAdapter(sourceObject, adapter.getName());
}

```



## Use of AdapterManager in Platform



```

public abstract class PlatformObject implements IAdaptable {
    public Object getAdapter(Class adapter) {
        return AdapterManager.getDefault().getAdapter(this, adapter);
    }
}

```



## IPropertySource

- » How does the Properties View work?
  - » The object in the Current Selection must be an IPropertySource
- » What does it mean: „the object must be an IPropertySource“?
  - » be a direct implementor of IPropertySource:
 

```
public class Person implements IPropertySource { /**/ }
```
  - » or **can adapt** to IPropertySource



## IAdaptable

- » What does it mean „the object **can adapt** to IPropertySource“?
  - » to implement IAdaptable and be able to return an IPropertySource on request
 

```
public class Person implements IAdaptable { /**/ }
```
  - » or to have a registered IAdapterFactory which would be able to adapt Person to IPropertySource



## What is an Adapter?

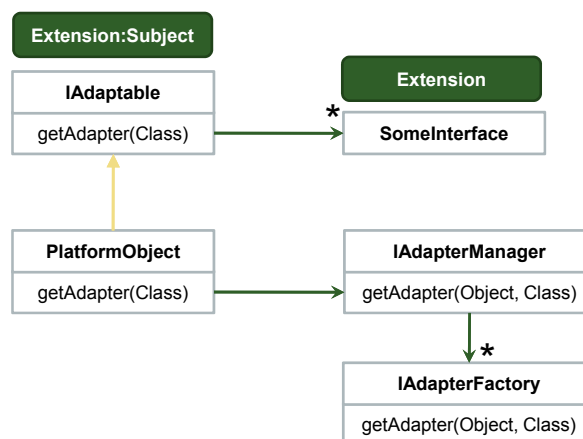
- » In object-oriented software systems, an adapter simply adapts (converts) an object of type A to another object of relevant type B
- » Eclipse provides the interface `IAdaptable` to address the adaption of an object:

```
public interface IAdaptable {
    public Object getAdapter(Class adapter);
}
```

- » Since model objects should not depend on Eclipse, `AdapterFactory`s can adapt all objects. Even if the objects do not implement `IAdaptable`...
  - » How does this work?



## IAdapterFactory



## IAdapterFactory

- » Such a factory provides adapters for given adaptable types:

```
public class AdapterFactory implements IAdapterFactory {
    @Override
    public Class[] getAdapterList() {
        return new Class[] { IPropertySource.class };
    }

    @Override
    public Object getAdapter(Object adaptableObject, Class adapterType) {
        if (adapterType == IPropertySource.class && adaptableObject instanceof Person) {
            final Person p = (Person) adaptableObject;
            return new IPropertySource() { }
        }
        return null;
    }
}
```



## Registering the Adapter Factory

- » Implementations of IAdapterFactory can be registered with the platform:

- » programmatically:

```
Platform.getAdapterManager()
    .registerAdapters(adapterFactory, Person.class);
```

- » or declaratively:

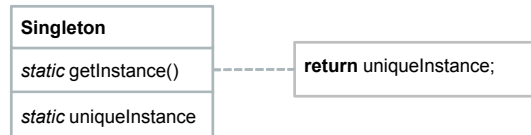
- » see E.P. org.eclipse.core.runtime.adapters

```
<extension
    point="org.eclipse.core.runtime.adapters">
    <factory
        adaptableType="org.shinkarenko.pm.core.Person"
        class="org.shinkarenko.pm.ui.properties.AdapterFactory">
        <adapter
            type="org.eclipse.ui.views.properties.IPropertySource">
        </adapter>
    </factory>
</extension>
```



## Singleton

- » *“Ensure that a class only has one instance, and provide a global point of access to it.”*



## Usage examples in Eclipse

- » `PlatformUI.getWorkbench()`
- » `Platform.getAdapterManager()`
- » `ResourcesPlugin.getWorkspace()`
- » `PMCoreActivator.getInstance().getPersonRepository()`



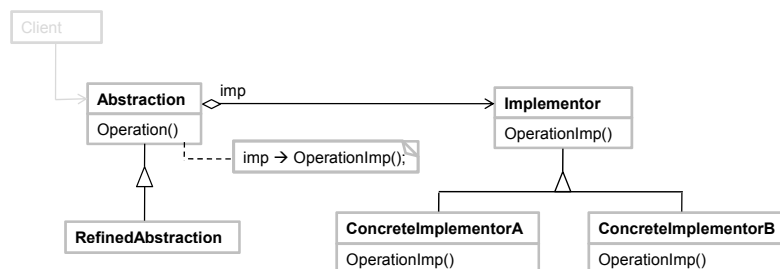
## Singleton Drawbacks

- » Use of *static*:
  - » Classloading issues
  - » May behave in an unpredictable fashion in dynamic OSGi environments
- » Coupling between the Singleton and the Client(s)
  - » Bridge pattern will help ;-)



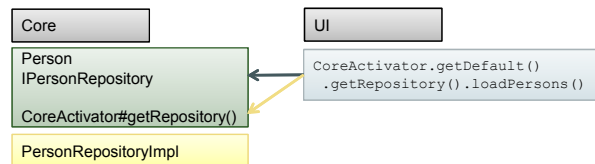
## Bridge

- » “Decouple an abstraction from its implementation so that the two can vary independently”



## PM Core Retrospective

- » Core bundle is designed fairly good (it is already a bridge in a fact), but it is not really flexible in terms of Components:

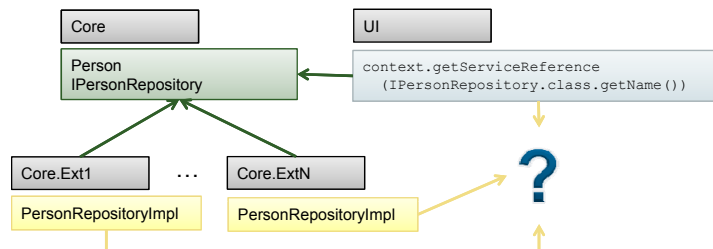


- » Have to call Core methods directly ☹
- » IPersonRepository implementation is coupled with Core ☹



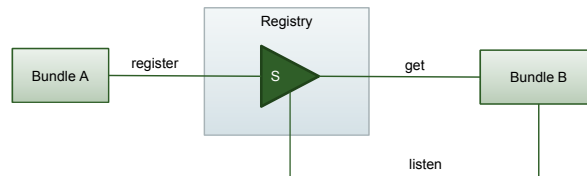
## Solution: introduce a bridge

- » Make possible for others to contribute core implementations:



## OSGi Services

- » Bundles may register services in an OSGi Service Registry



```

@Override
public void start(BundleContext context) throws Exception {
    super.start(context);
    plugin = this;

    final IPersonRepository personRepository = new DummyPersonRepository();
    context.registerService(IPersonRepository.class.getName(), personRepository, null);
}

@Override
public void stop(BundleContext context) throws Exception {
    plugin = null;
    super.stop(context);
}
  
```



## OSGi Services

- » Other bundles may get the registered services:

```

final String serviceName = IPersonRepository.class.getName();
final ServiceReference serviceReference = ctx.getServiceReference(serviceName);
if (serviceReference == null) {
    throw new PMException("Person Repository Service cannot be found");
}
personRepository = (IPersonRepository) ctx.getService(serviceReference);
  
```

- » Or listen to their lifecycle:

```

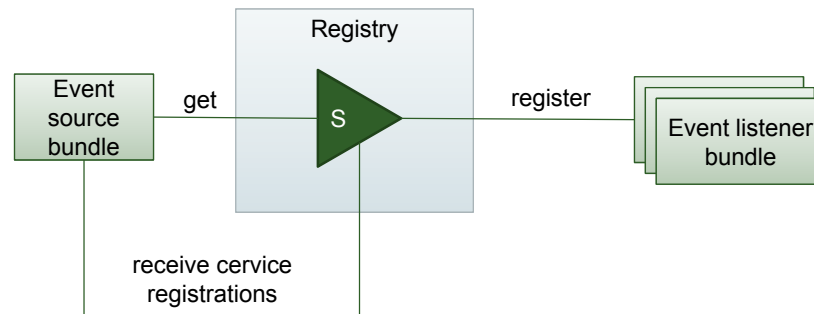
final String serviceName = IPersonRepository.class.getName();
final ServiceTracker serviceTracker = new ServiceTracker(ctx, serviceName,
    new ServiceTrackerCustomizer() {
        @Override
        public void removedService(ServiceReference reference, Object service) {
            log("Removed service: " + service);
        }
        @Override
        public void modifiedService(ServiceReference reference, Object service) {
            log("Modified service: " + service);
        }
        @Override
        public Object addingService(ServiceReference reference) {
            personRepository = (IPersonRepository) ctx.getService(reference);
            log("Adding service ref: " + reference + "-->" + personRepository);
            return personRepository;
        }
    });
serviceTracker.open();
  
```



## Whiteboard: Pluggable Listeners

### » Listeners Considered Harmful: The “Whiteboard” Pattern

» <http://www.osgi.org/wiki/uploads/Links/whiteboard.pdf>



## Listener support in IPersonRepository

```

public interface IPersonRepository {
    * Creates the given person.[]
    String create(Person person) throws PMException;

    * Loads a person by the given ID.[]
    Person load(String id) throws PMException;

    * Loads all persons from the repository sorted by ID.[]
    List<Person> loadAll() throws PMException;

    * Updates the given (@link Person).[]
    void update(Person person) throws PMException;

    * Deletes the given (@link Person).[]
    void delete(Person person) throws PMException;

    * Adds the given (@link IPersonListener) to the repository's listener list.[]
    void addListener(IPersonListener listener);

    * Removes the given (@link IPersonListener) to the repository's listener.[]
    void removeListener(IPersonListener listener);
}
  
```



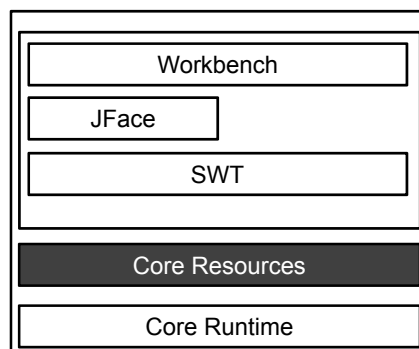
## Use Whiteboard pattern instead

1. Mark listener support methods in IPersonRepository as deprecated or remove at all
2. Let all clients register IPersonListeners via OSGi Service Registry
3. Set up a ServiceTracker in your core implementation to update the local cache of Listeners
4. Test carefully (View open-close, Bundle start-stop etc.)



## Core Workspace: Resources

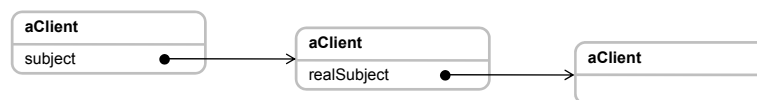
Platform



## Proxy and Bridge: Accessing File System

» Proxy: *"Provide a surrogate or place holder for another object to control access to it"*

» Proxy structure at a runtime:



## IFile as a Proxy

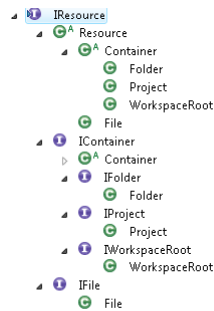
» How to address a resource in a Workspace?

» give a handle for a resource, not the full resource

» the handle acts like a key for a resource

## IResource hierarchy

- » The handles are defined as interfaces IFile, IFolder, IProject, and IWorkspaceRoot



## Some characteristics of the resource handles

- » Small objects. Once created, none of their fields will ever change. Use them in maps as keys.
- » Handles define the behavior of a resource, but they do not keep any resource state information.
- » A handle can refer to non-existing resources.

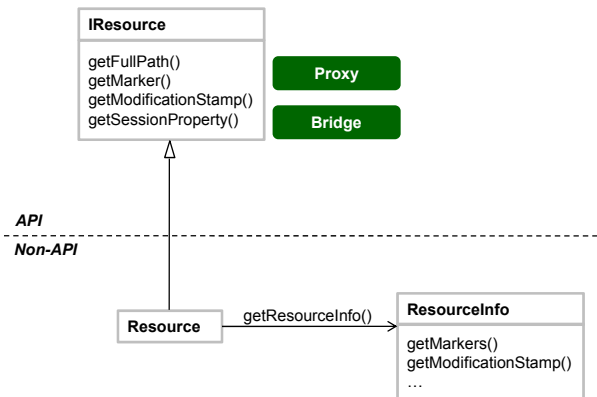
- » Handles are created from a parent handle:

```
IProject project;
IFolder folder = project.getFolder("someFolder");
```

- » Handles are used to create the underlying resource:

```
folder.create(...);
```

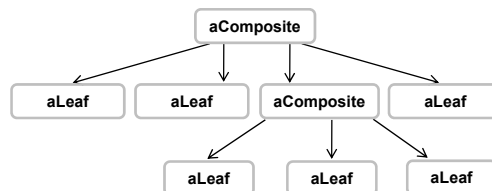
## IResource Is a Proxy and a Bridge



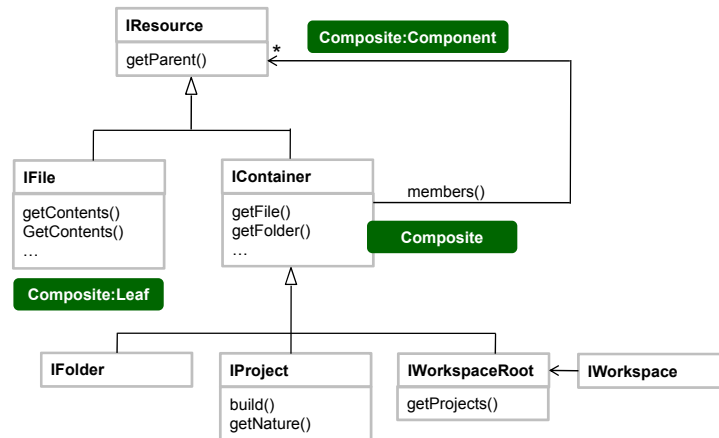
## Composite: IWorkspace

» Composite: “Compose object into tree structures to represent part/whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.”

» A typical Composite object structure:

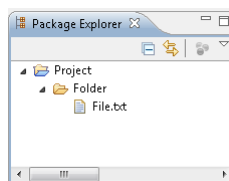


## IWorkspace as a Composite



## IWorkspace

- » IWorkspace is a Composite of IContainers and IFiles

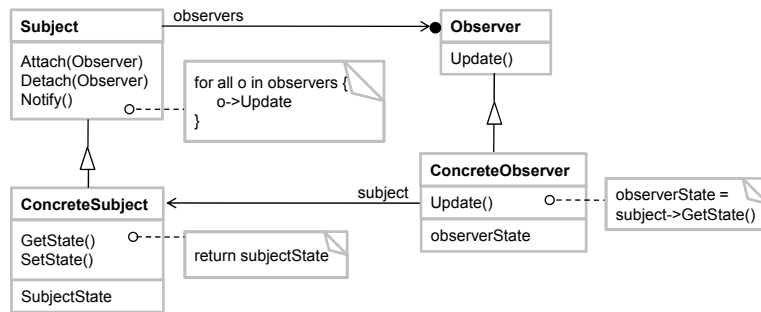


- » Access the Singleton workspace instance from the static accessor `ResourcesPlugin.getWorkspace()`

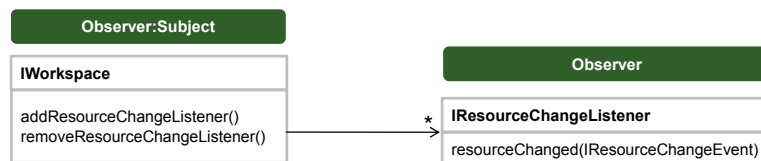


## Observer: tracking resource changes

» Observer: „Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.”



## IResourceChangeListener



```

IWorkspace ws = ResourcesPlugin.getWorkspace();
ws.addResourceChangeListener(new IResourceChangeListener() {
    @Override
    public void resourceChanged(IResourceChangeEvent event) {
        if (IResourceChangeEvent.POST_CHANGE == event.getType()) {
            //
        }
    }
});
    
```



## IResourceDelta Records a Tree of Changes



- » A resource delta describes a single change and multiple changes using the same structure.
- » It is easy to process a resource delta recursively top-down when updating an observer.
- » You can reuse the traversal logic of a resource delta with an IResourceDeltaVisitor:

```

public interface IResourceDeltaVisitor {
    public boolean visit(IResourceDelta delta) throws CoreException;
}
  
```



## Processing the Resource Delta

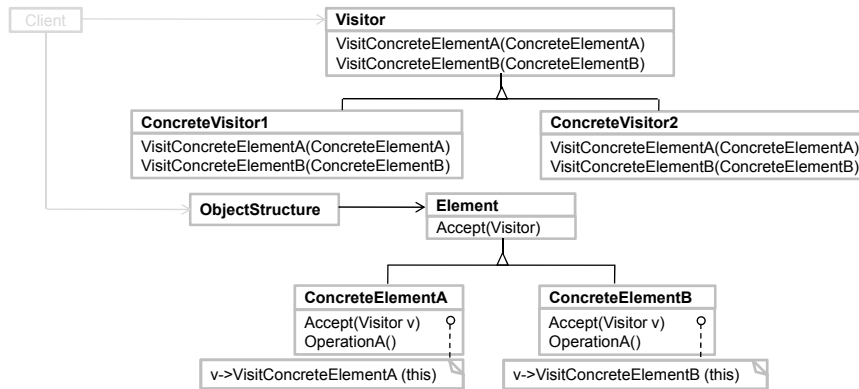
```

IWorkspace ws = ResourcesPlugin.getWorkspace();
ws.addResourceChangeListener(new IResourceChangeListener() {
    @Override
    public void resourceChanged(IResourceChangeEvent event) {
        IResourceDelta delta = event.getDelta();
        IResourceDeltaVisitor resourceDeltaVisitor = new IResourceDeltaVisitor() {
            @Override
            public boolean visit(IResourceDelta delta) throws CoreException {
                System.out.println(delta);
                return true;
            }
        };
        try {
            delta.accept(resourceDeltaVisitor);
        } catch (CoreException e) {
        }
    }
});
  
```

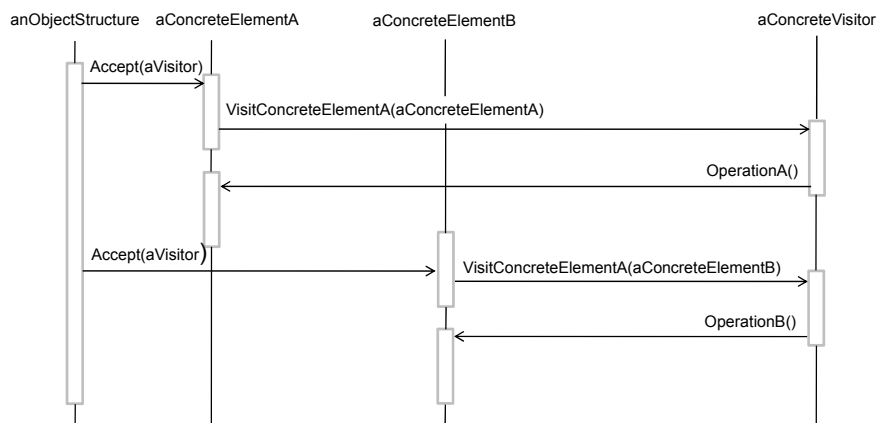


## Visitor: traversing the resource tree

» Visitor: "Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates."



## Visitor Interaction Diagram



## Visiting a Resource API

```

final IResource resource = ResourcesPlugin.getWorkspace().getRoot();

final IResourceVisitor visitor = new IResourceVisitor() {
    public boolean visit(IResource resource) throws CoreException {
        if (resource.getType() == IResource.FILE) {
            IFile file = (IFile) resource;
            System.out.println(file.getName());
        }
        return true;
    }
};

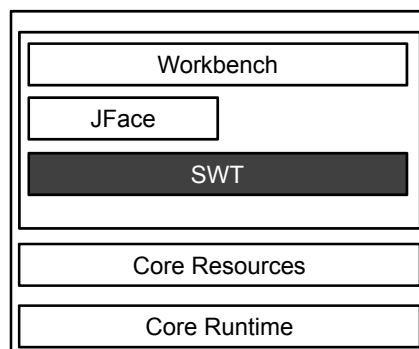
try {
    resource.accept(visitor, IResource.DEPTH_INFINITE, false);
} catch (CoreException e) { }

```



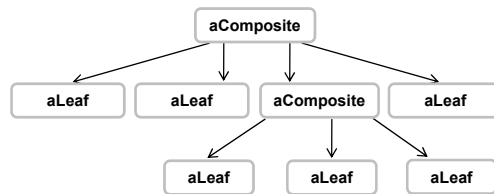
## SWT

Platform

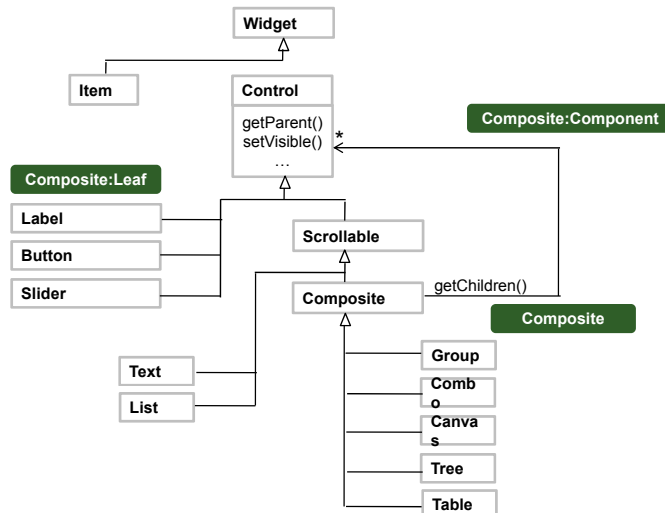


## Composite: composing widgets

- » Composite: *“Compose object into tree structures to represent part/whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.”*
- » A typical Composite object structure:



## SWT basic and compound widgets



## SWT basic and compound widgets

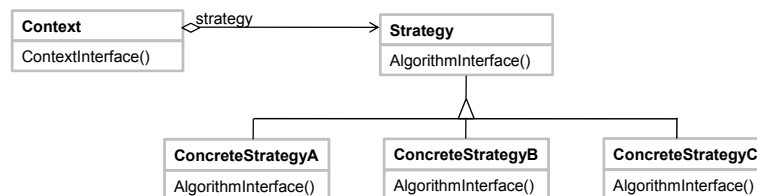
- » Basic widgets
  - » do not contain other widgets
  - » are the leaves in a widget tree
  - » Button, Label, Text, ..
- » Compound widgets
  - » contain other widgets
  - » are the inner nodes of a widget tree
  - » have Composite as the base class

```
private void addControl(Composite parent) {
    Composite composite = new Composite(parent, SWT.NONE);
    // ...
    Label label = new Label(composite, SWT.NONE);
    label.setText("hello");
    // ...
}
```

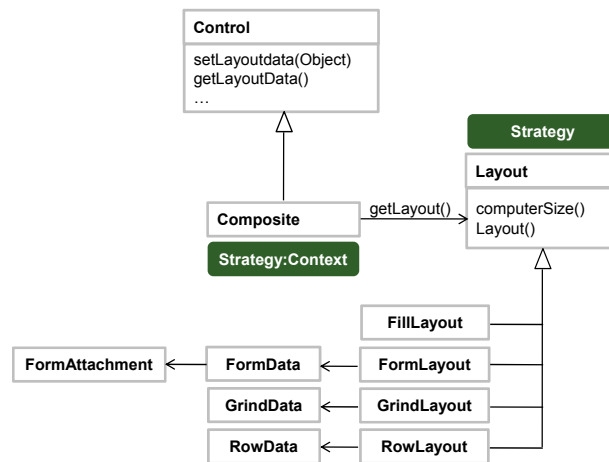


## Strategy: Defining UI Layout

- » Strategy: *"Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it."*



## SWT Layout Managers



## SWT Layout Example

```

public class SampleComposite extends Composite {
    final Text txtFirstName;

    public SampleComposite(final Composite cmpParent, final int style) {
        super(cmpParent, style);

        // 1. Create widgets
        final Label label = new Label(this, SWT.NONE);
        txtFirstName = new Text(this, SWT.BORDER);

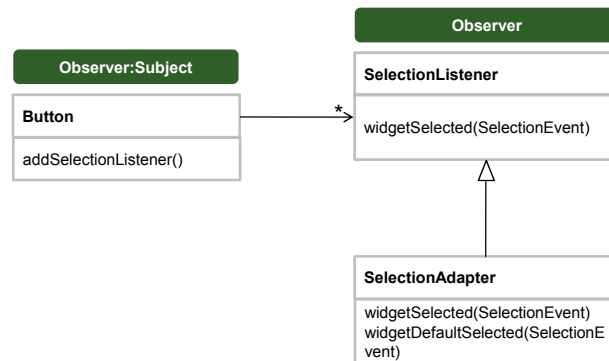
        // 2. Set the Layout Manager
        final GridLayout layout = new GridLayout(2, false);
        this.setLayout(layout);

        // 3. Set the Layout Data (if needed)
        final GridData layoutData = new GridData(GridData.FILL_HORIZONTAL);
        txtFirstName.setLayoutData(layoutData);

        // 4. Set some data (if needed)
        label.setText("First Name");
    }
}
  
```



## Observer: responding to Events



## Adding Listeners to SWT widgets

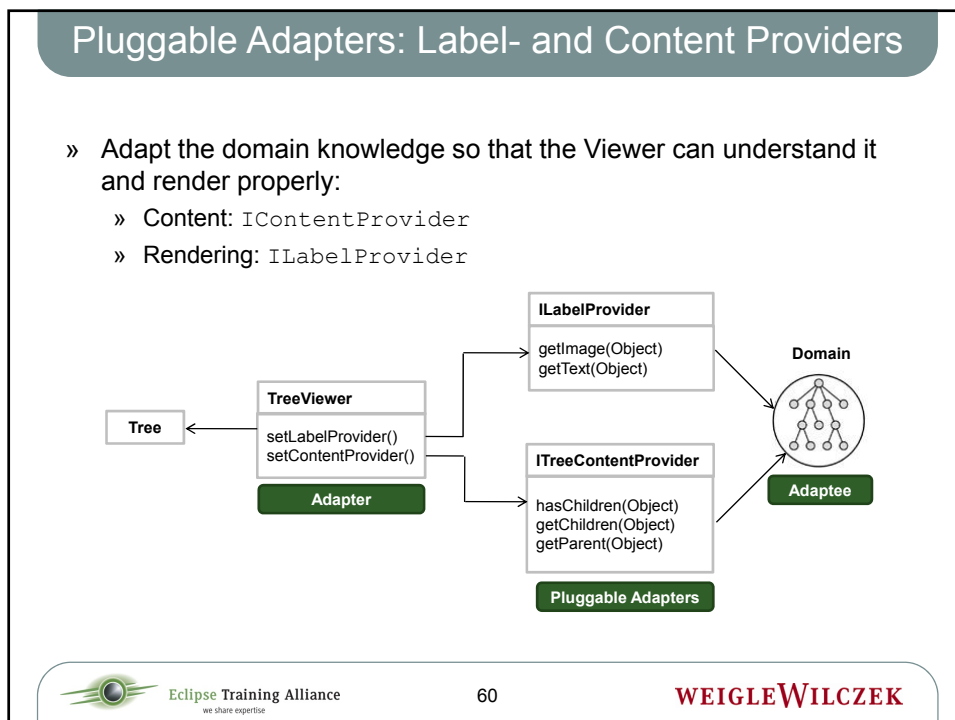
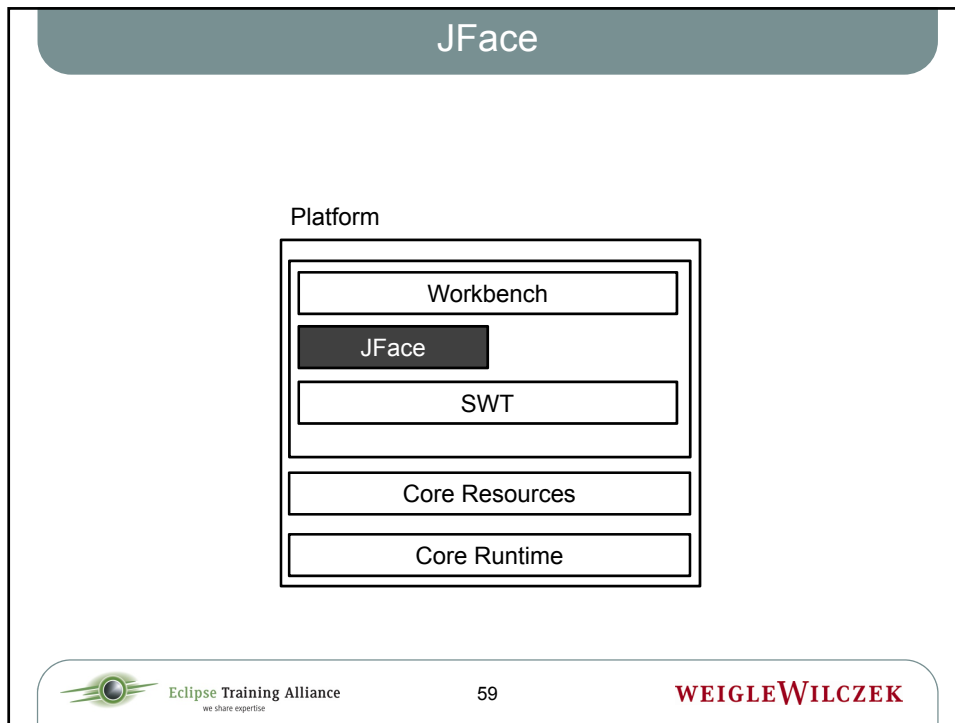
```

Text txtName = new Text(cmp, SWT.PUSH);

//Typed listener:
txtName.addModifyListener(new ModifyListener() {
    @Override
    public void modifyText(ModifyEvent e) {
        // handle modify
    }
});

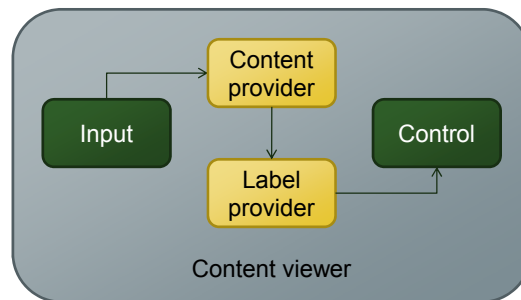
//Generic (untyped) listener:
txtName.addListener(SWT.Modify, new Listener() {
    @Override
    public void handleEvent(Event event) {
        // handle modify
    }
});
  
```





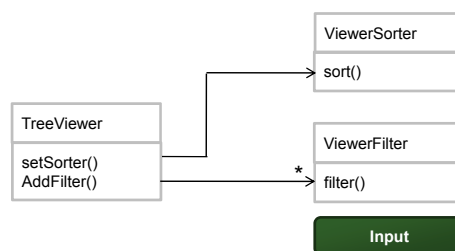
## A Content Viewer ...

- » ... delegates handling of input changes to a content provider:
  - » The viewer queries the content provider for (an) element(s) to be shown
- » ... delegates mapping the elements to be displayed to labels and images to a label provider



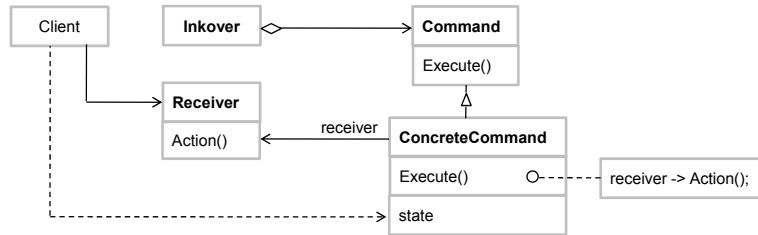
## Strategy: Customizing without Subclassing

- » `ViewerSorter` and `ViewerFilter` are strategies

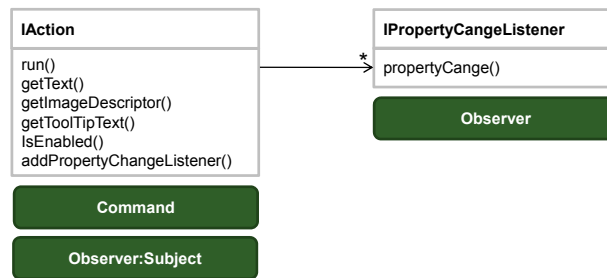


## Command: Actions

» Command: **"Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support for undoable operations."**



## JFace IAction



## JFace Actions

- » IAction defines a run() method to be called to execute the request.
- » Stores all the “decorative” information:
  - » to present the action in a menu / toolbar / button
  - » action's label, icon, tooltip, and enablement state
- » Can be used by multiple menu items / toolbar items / buttons at the same time. Create an Action once and share it.
- » Fires a property change event when an action's state changes.
  - » This allows JFace to keep the state of the widget in sync with the action.



## Action: Default IAction implementation

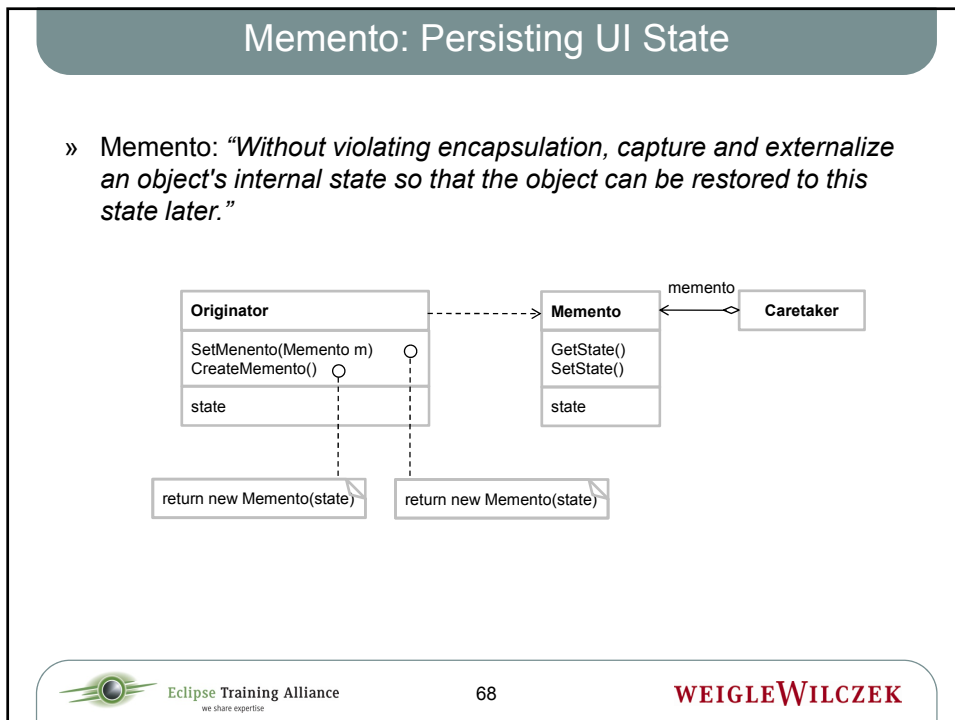
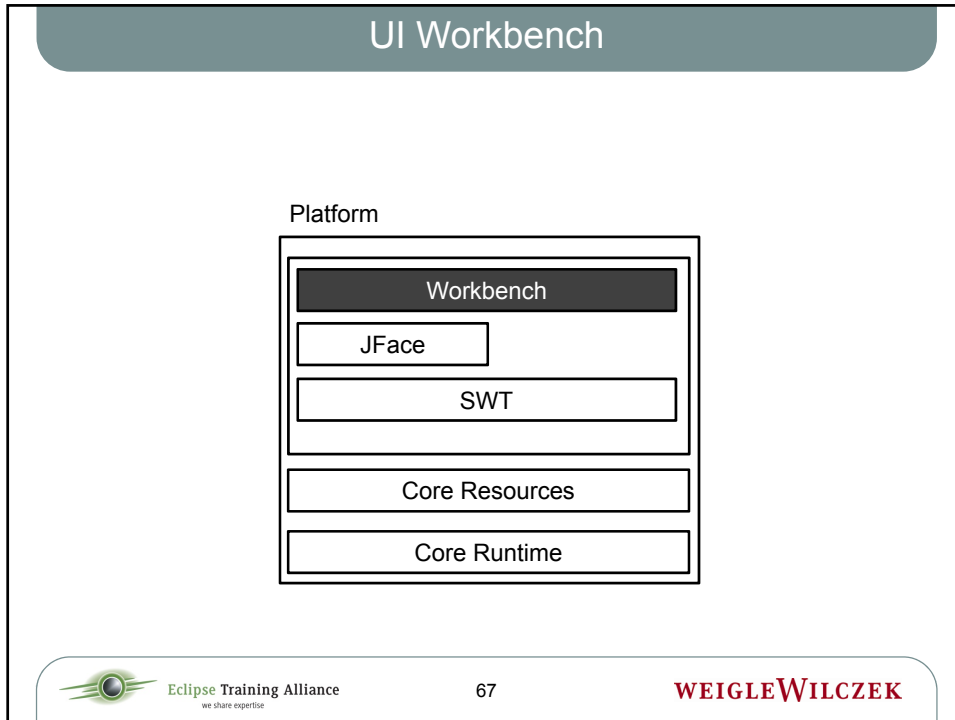
- » Encapsulate the code you want to run in „run“ method
- » Set the decorative attributes
- » Use the action in menu / toolbar / ...

```
final IAction action = new Action("Exit JVM") {
    @Override
    public void run() {
        System.exit(0);
    }
};
// do not override getters, use setters instead:
action.setEnabled(true);
action.setDescription("Description");
action.setToolTipText("Tooltip");

// attach the action
final MenuManager menuManager = new MenuManager("File");
final ToolBarManager toolbarManager = new ToolBarManager(SWT.FLAT);

menuManager.add(action);
toolbarManager.add(action);
```





## Memento: Persisting UI State

```

public class MementoViewPart extends ViewPart {

    private static final String KEY_NAME = "name";
    private static final String DEFAULT_NAME = "";
    private Text txtName;

    private String savedName = DEFAULT_NAME;

    @Override
    public void init(IViewSite site, IMemento memento) throws PartInitException {
        super.init(site, memento);
        if (memento != null) {
            final String name = memento.getString(KEY_NAME);
            this.savedName = (name != null) ? name : DEFAULT_NAME;
        }
    }

    @Override
    public void createPartControl(Composite parent) {
        txtName = new Text(parent, SWT.NONE);
        txtName.setText(savedName);
    }

    @Override
    public void setFocus() {
        txtName.setFocus();
    }

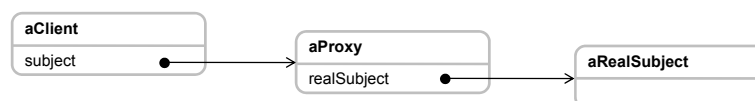
    @Override
    public void saveState(IMemento memento) {
        memento.putString(KEY_NAME, txtName.getText());
    }
}

```



## Virtual Proxy: lazy loading with E.P.

- » Proxy: "Provide a surrogate or place holder for another object to control access to it."
- » Proxy structure at runtime:

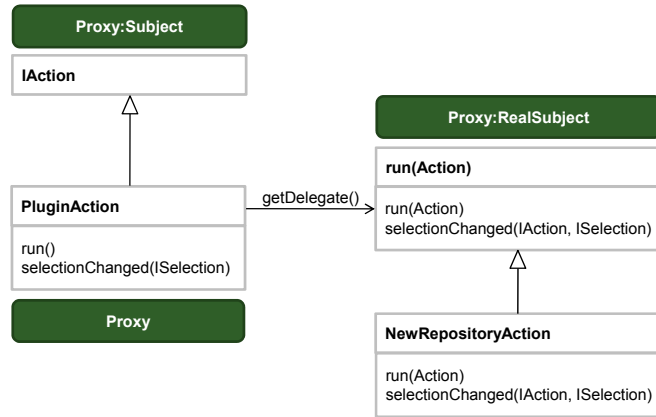


- » Virtual: the proxy has only a descriptor, the instance is created on demand



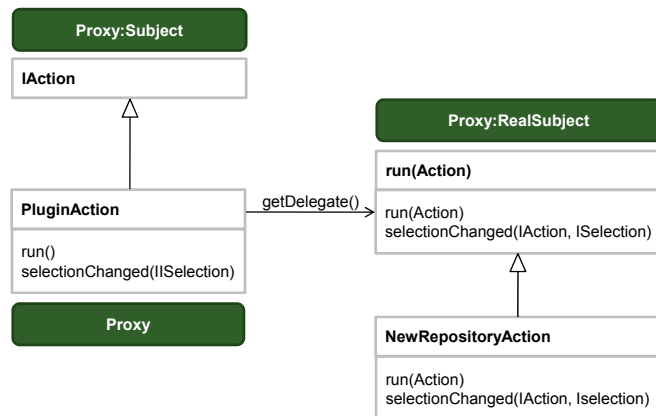
## Virtual Proxies: the Lazy Loading Rule

» PluginAction Lazily Loads the Real Action:



## Virtual Proxies: the Lazy Loading Rule

» PluginAction Lazily Loads the Real Action:



## ActionDelegate implementation

```

/**
 * Our sample action implements workbench action delegate.
 *
 * The action proxy will be created by the workbench and shown in the UI.
 *
 * When the user tries to use the action, this delegate will be created and
 * execution will be delegated to it.
 *
 */
public class SampleAction implements IWorkbenchWindowActionDelegate {

    public void init(IWorkbenchWindow window) {

    }

    public void selectionChanged(IAction action, ISelection selection) {

    }

    /**
     * The action has been activated. The argument of the method represents the
     * 'real' action sitting in the workbench UI.
     *
     * @see IWorkbenchWindowActionDelegate#run
     */
    public void run(IAction action) {
        System.out.println("Hello, my real name is " + action.getText());
    }

    public void dispose() {

    }

}

```



## Eclipse RCP Certification Program



## Eclipse RCP Certification Program

- » Prove your competence – get certified!
  - » „Certified Eclipse RCP Developer“
  - » „Certified OSGi & Equinox Developer“
- » Which skills are required?
  - » The exams are based on the training classes of the ETA.
  - » Of course you are allowed to start the exam without having joined any class.
- » How can I get certified?
  - » Visit <http://www.eclipse-training.net/india/certification>
  - » Special offer for Eclipse Summit India!



## Copyright notice

- » This presentation contains some excerpts from the following books:
  - » **Design Patterns**  
**Elements of Reusable Object-Oriented Software**  
by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  - » **Contributing to Eclipse: Principles, Patterns, and Plug-Ins**  
by Erich Gamma, Kent Beck
- » This presentation is supposed to be used ONLY in non-for-profit activities

