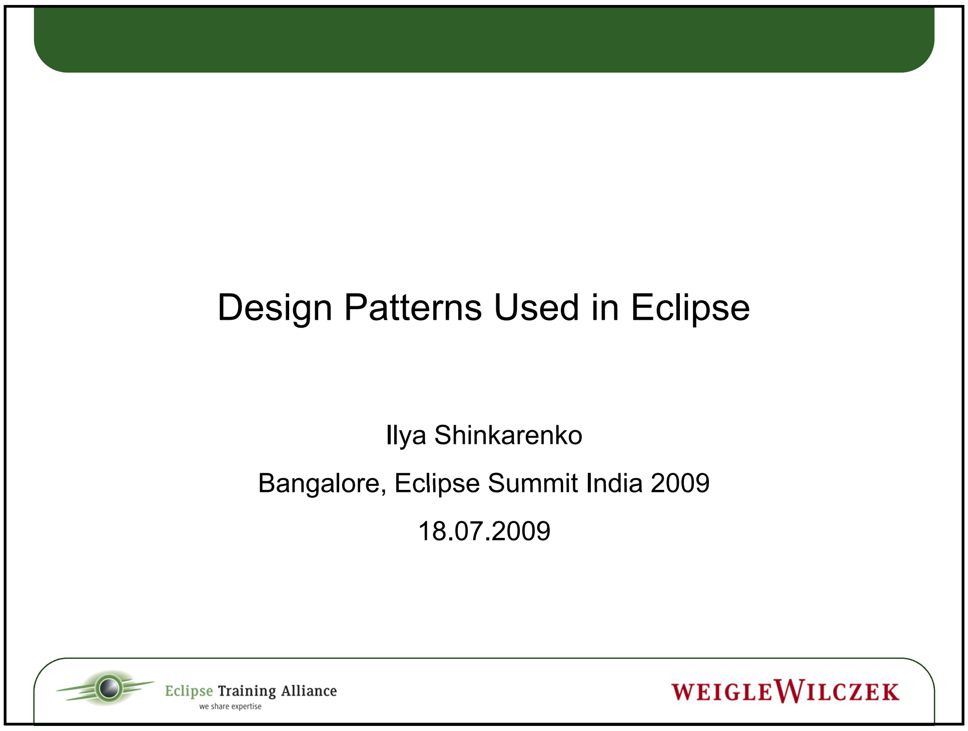


Design Patterns Used in Eclipse

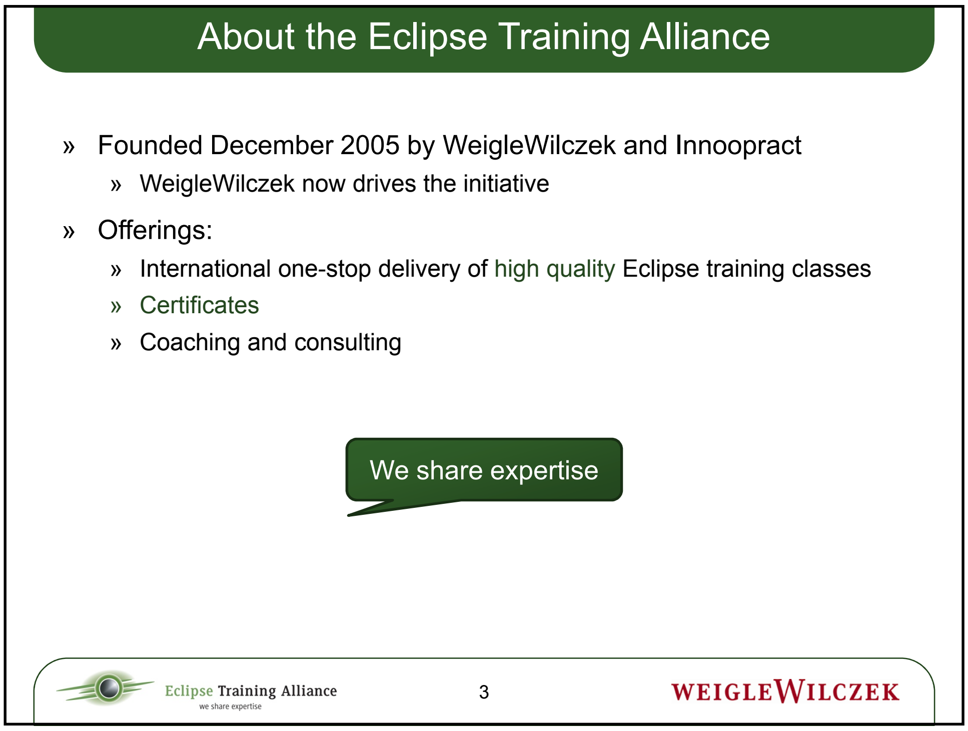
Ilya Shinkarenko

Bangalore, Eclipse Summit India 2009

18.07.2009

About me: Ilya Shinkarenko

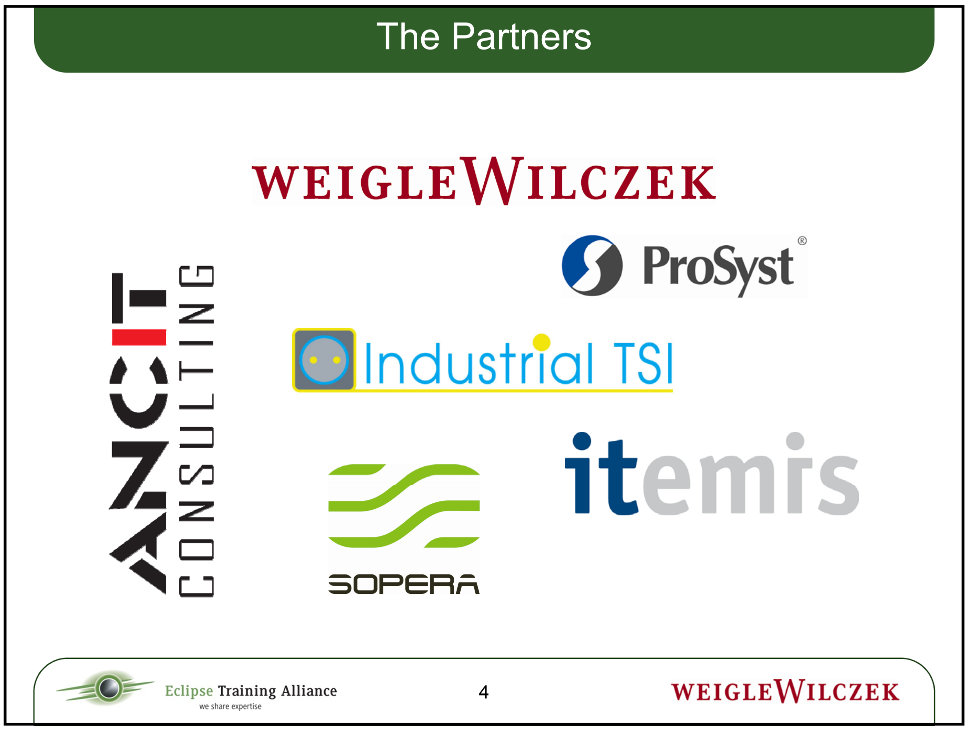
- » Software Engineer / Architect
  - » Eclipse RCP, OSGi
  - » Spring, JBoss AS
  - » Other stuff: Android OS, Flash/Flex, Red5 etc.
- » Working in IT since 1997
  - » Java since 1999
  - » Eclipse expertise since 2004
- » Consulting, workshops, seminars
  - » Eclipse RCP and basically everything around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



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



The Partners

WEIGLEWILCZEK

ProSyst®

Industrial TSI

ANCIT CONSULTING

SOPERA

itemis

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

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

Training courses
Avail yourself of our specialists' long-time Eclipse expertise.

On-site
Need flexibility regarding contents and location?

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!

eclipseCON We're sponsoring EclipseCon 2009

See the talks of our partners.

Contact India

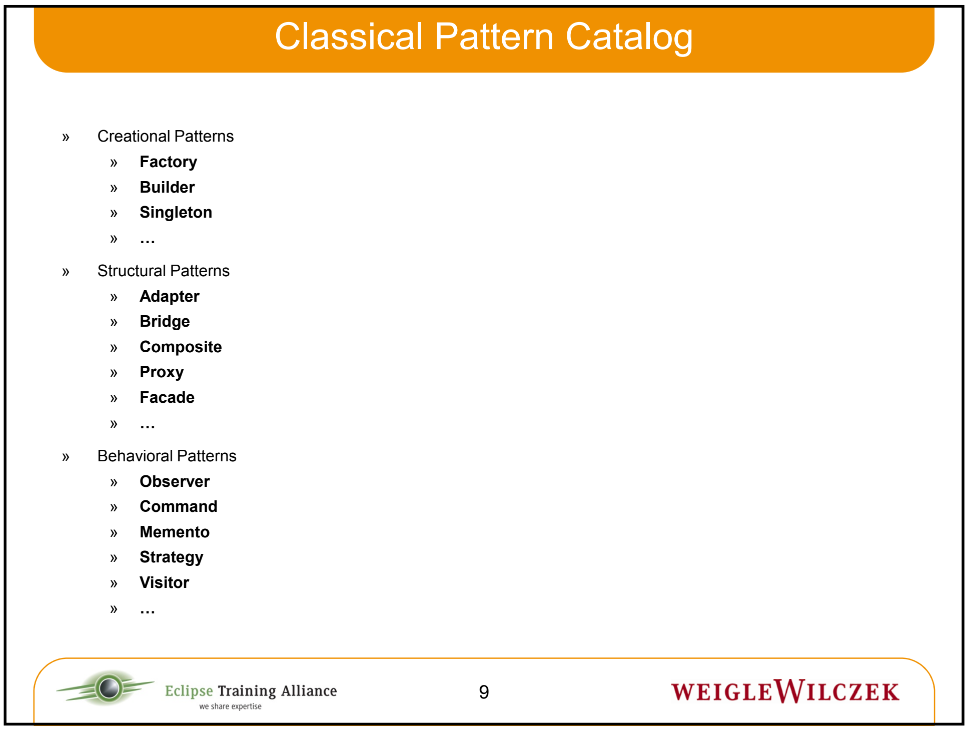
Please contact us for details:

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

T +91-422-6461778
certification@ancitconsulting.com

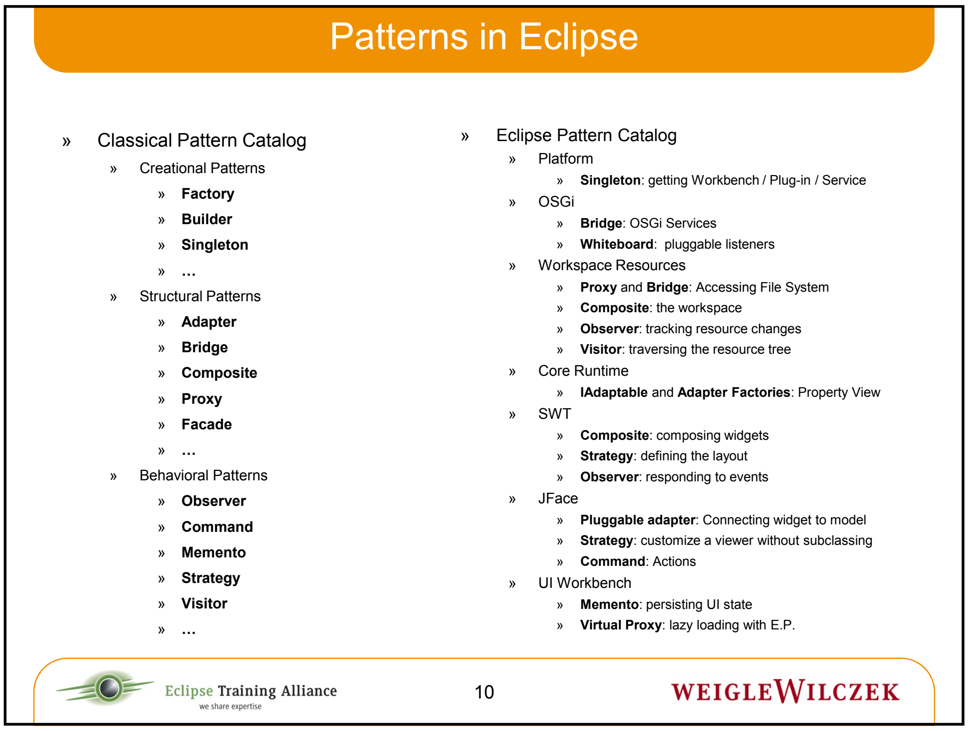
About You

» 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?



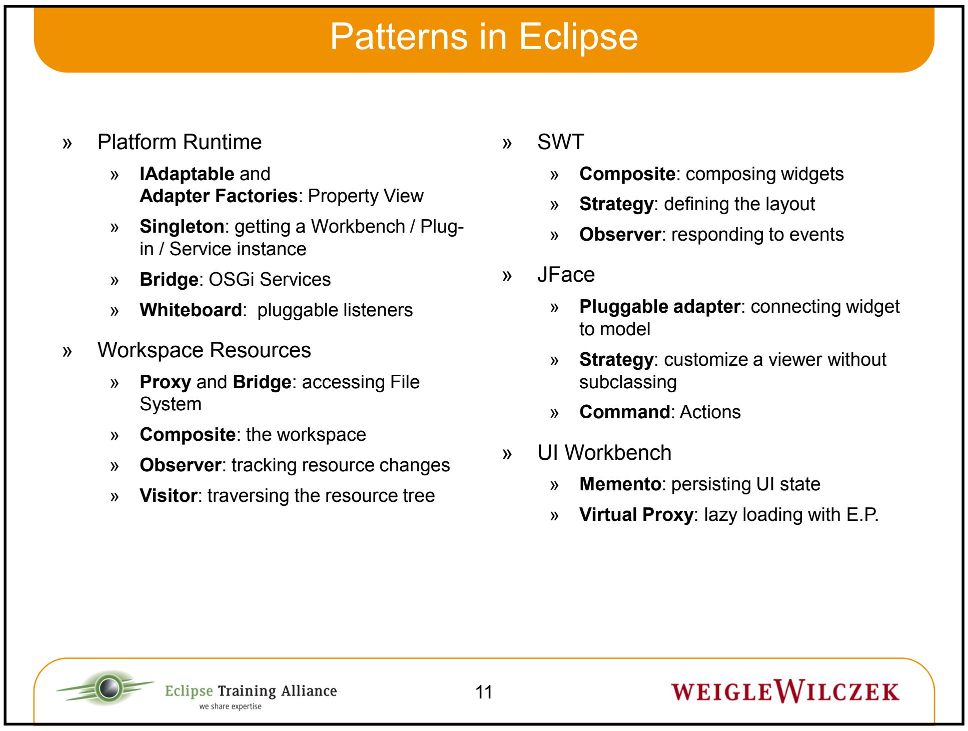
Classical Pattern Catalog

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



Patterns in Eclipse

- Classical Pattern Catalog
  - Creational Patterns
    - **Factory**
    - **Builder**
    - **Singleton**
    - **…**
  - Structural Patterns
    - **Adapter**
    - **Bridge**
    - **Composite**
    - **Proxy**
    - **Facade**
    - **…**
  - Behavioral Patterns
    - **Observer**
    - **Command**
    - **Memento**
    - **Strategy**
    - **Visitor**
    - **…**
- Eclipse Pattern Catalog
  - Platform
    - **Singleton**: getting Workbench / Plug-in / Service
  - OSGi
    - **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
  - Core Runtime
    - **IAdaptable** and **Adapter Factories**: Property View
  - 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.

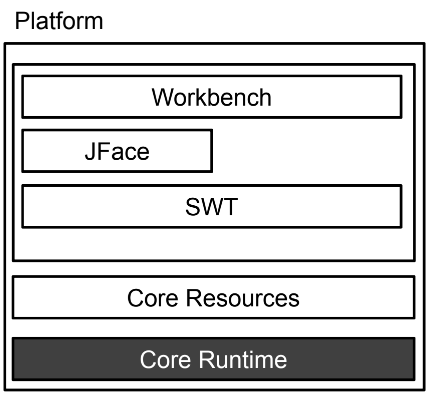


Patterns in Eclipse

- Platform Runtime
  - **IAdaptable** and **Adapter Factories**: Property View
  - **Singleton**: getting a Workbench / Plug-in / 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.

Eclipse Training Alliance we share expertise 11 WEIGLEWILCZEK

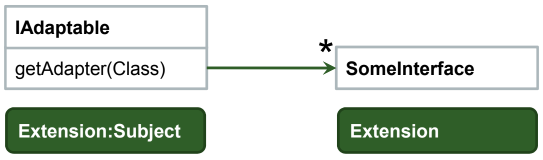
Runtime



Eclipse Training Alliance we share expertise 12 WEIGLEWILCZEK

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."*

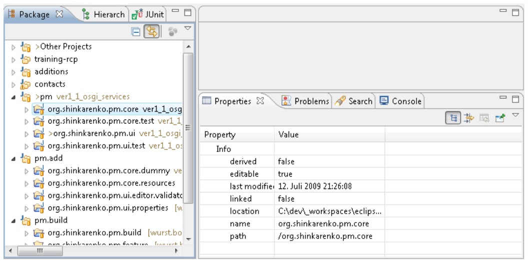


Need for Adapters

» Add a service interface to a type without exposing it in that type

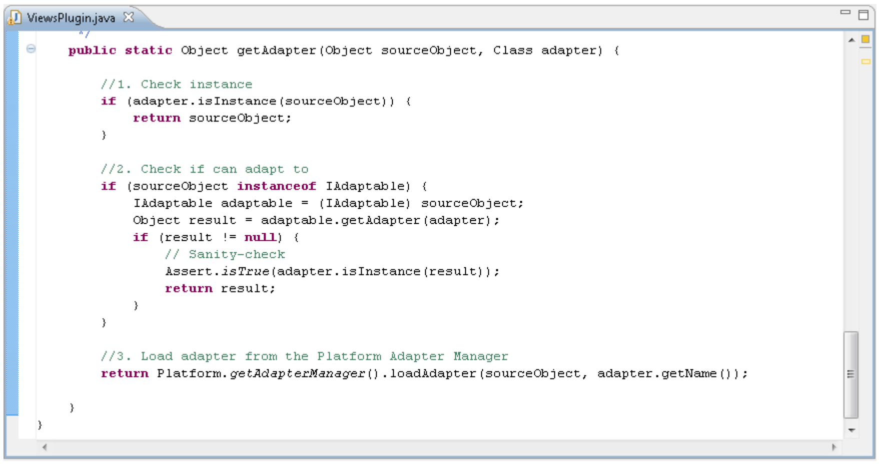
» Add behavior to preexisting types such as `IFile`, `Person`, etc.

» Our goal: we want to adapt `Person` to `IPropertySource`


Package
Hierarch
JUnit
>Other Projects
training-rcp
additions
contacts
>pm ver1_1_osgi_services
org.shinkarenko.pm.core ver1_1_osg
org.shinkarenko.pm.core.test ver1_1
>org.shinkarenko.pm.ui ver1_1_osgi
org.shinkarenko.pm.ui.test ver1_1_o
pm.add
org.shinkarenko.pm.core.dummy ve
org.shinkarenko.pm.core.resources
org.shinkarenko.pm.ui.editor.validatc
org.shinkarenko.pm.ui.properties [w
pm.build
org.shinkarenko.pm.build [wurst.bo
Properties
Problems
Search
Console

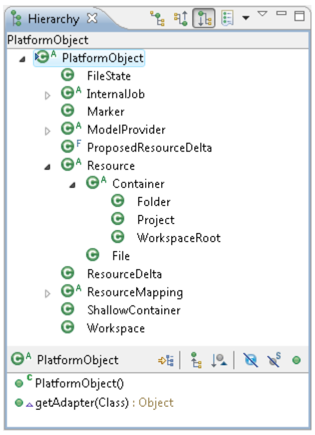
Property	Value
Info	
derived	false
editable	true
last modifie	12. Juli 2009 21:26:08
linked	false
location	C:\dev_workspaces\eclips...
name	org.shinkarenko.pm.core
path	/org.shinkarenko.pm.core

Getting an Adapter:



```
    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

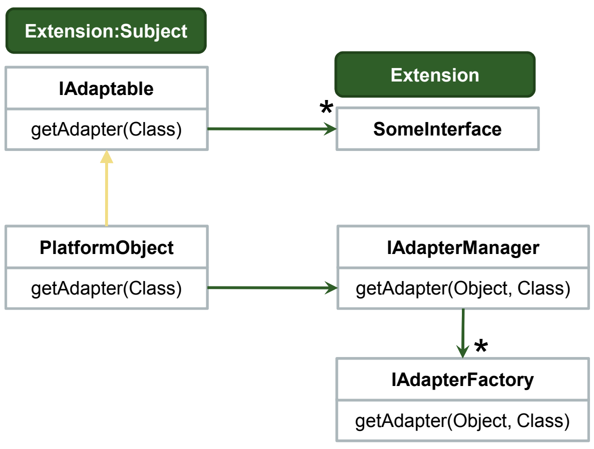
» Eclipse provides the interface `IAdaptable` to address the adaption of an object:

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

» Since model objects should not depend on Eclipse, `AdapterFactories` can adapt all objects. Even if the objects do not implement `IAdaptable`…

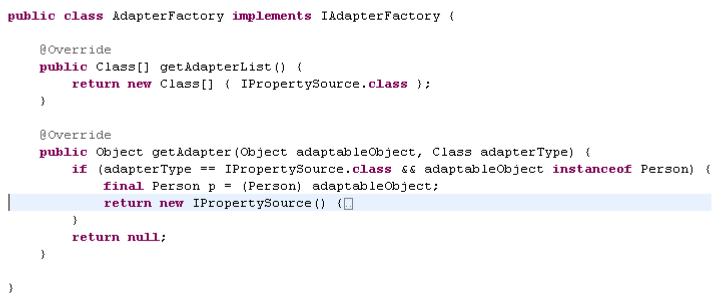
  » How does this work?

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;
    }

}
```

Eclipse Training Alliance
we share expertise

WEIGLEWILCZEK

Registering the Adapter Factory

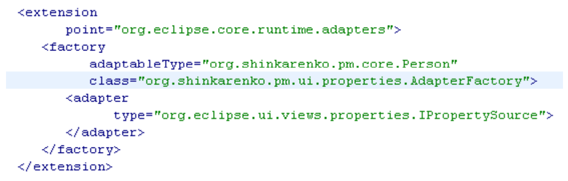
» Implementations of `IAdapterFactory` can be registered with the platform:

» programmatically:

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

» or declaratively:

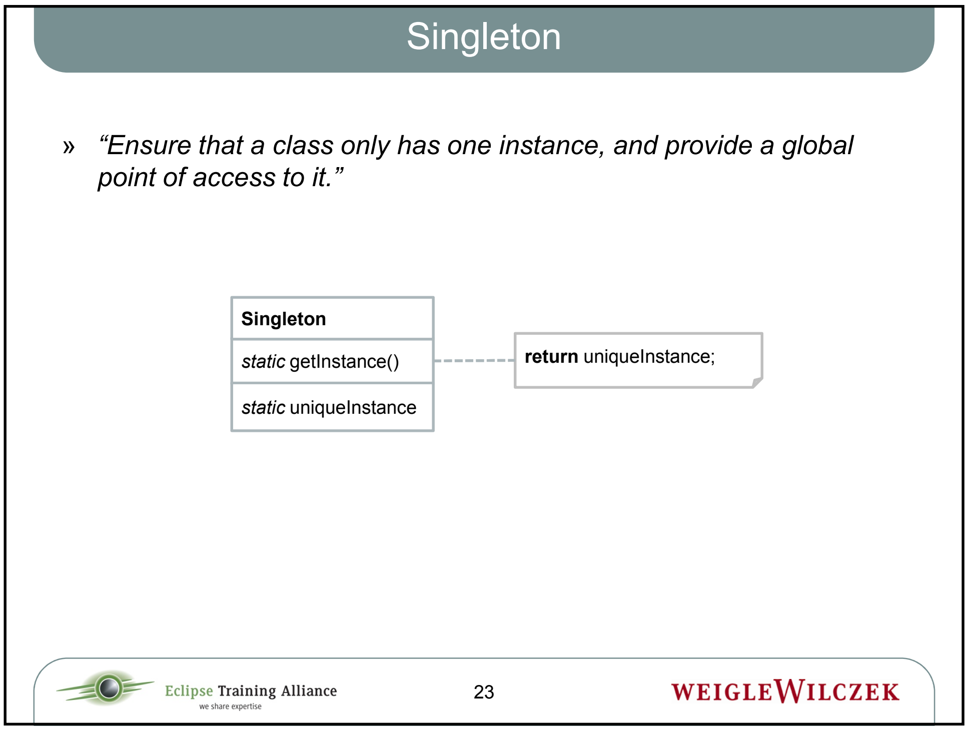
» see E.P. `org.eclipse.core.runtime.adapters`



```
<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>
```

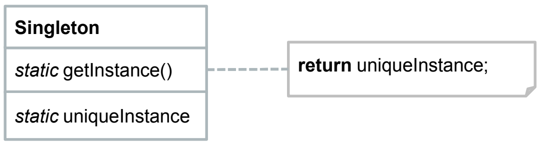
Eclipse Training Alliance
we share expertise

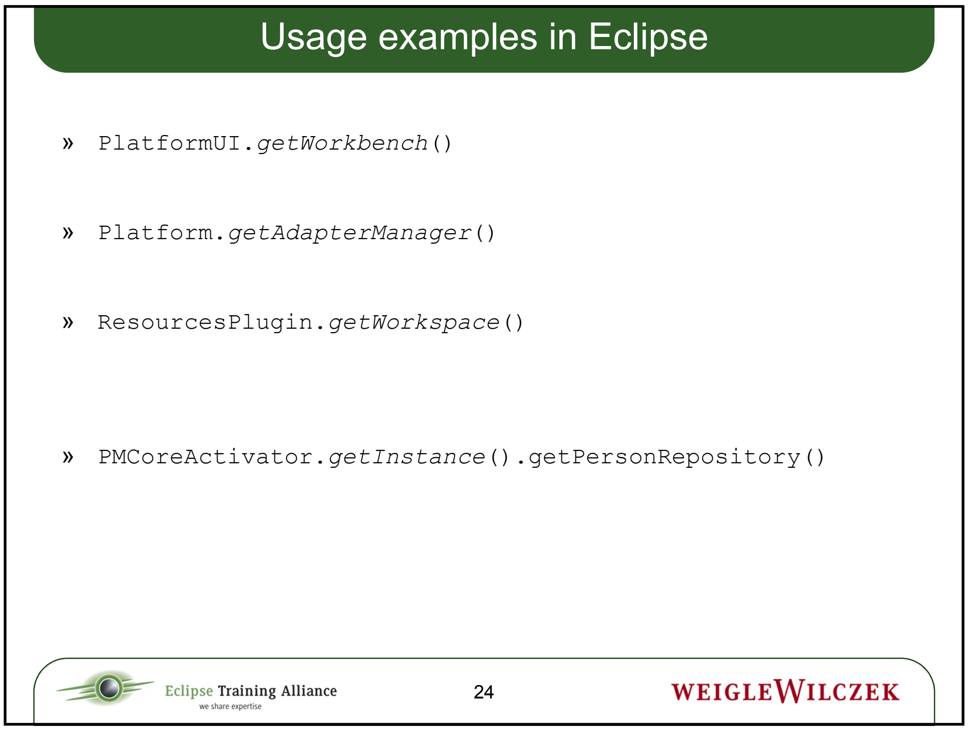
WEIGLEWILCZEK



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()`

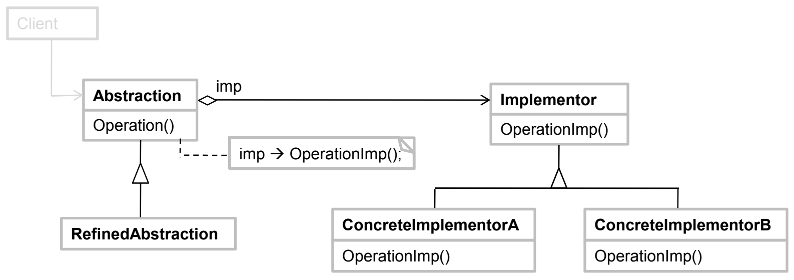
» `PMCoreActivator.getInstance().getPersonRepository()`

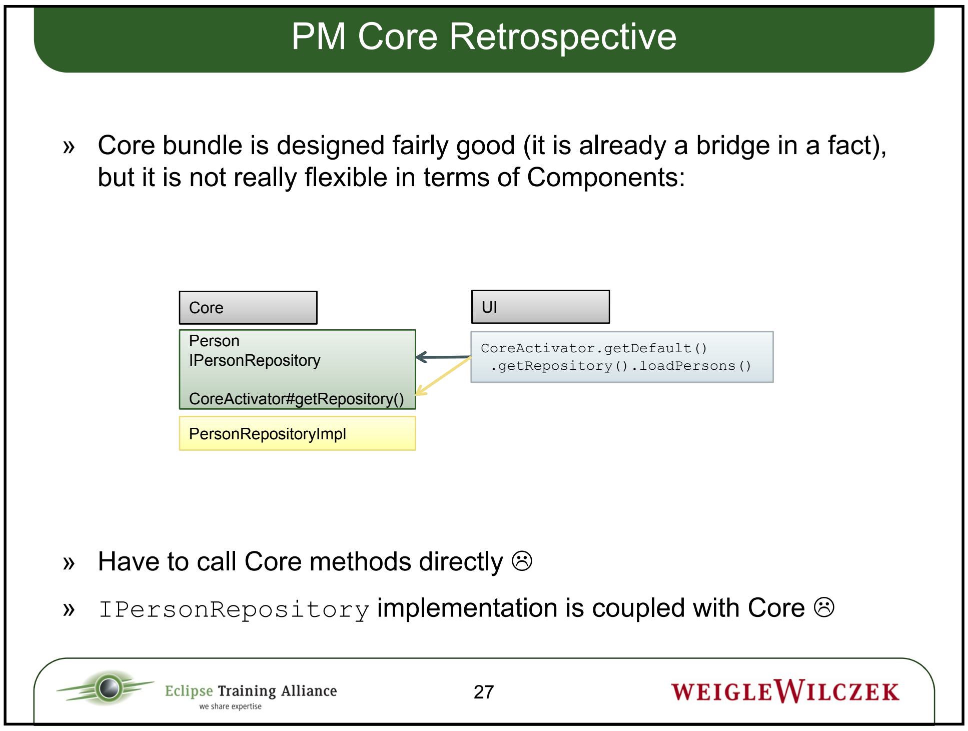
Singleton Drawbacks

- 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

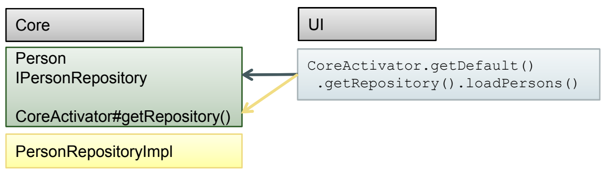
- *"Decouple an abstraction from its implementation so that the two can vary independently"*





PM Core Retrospective

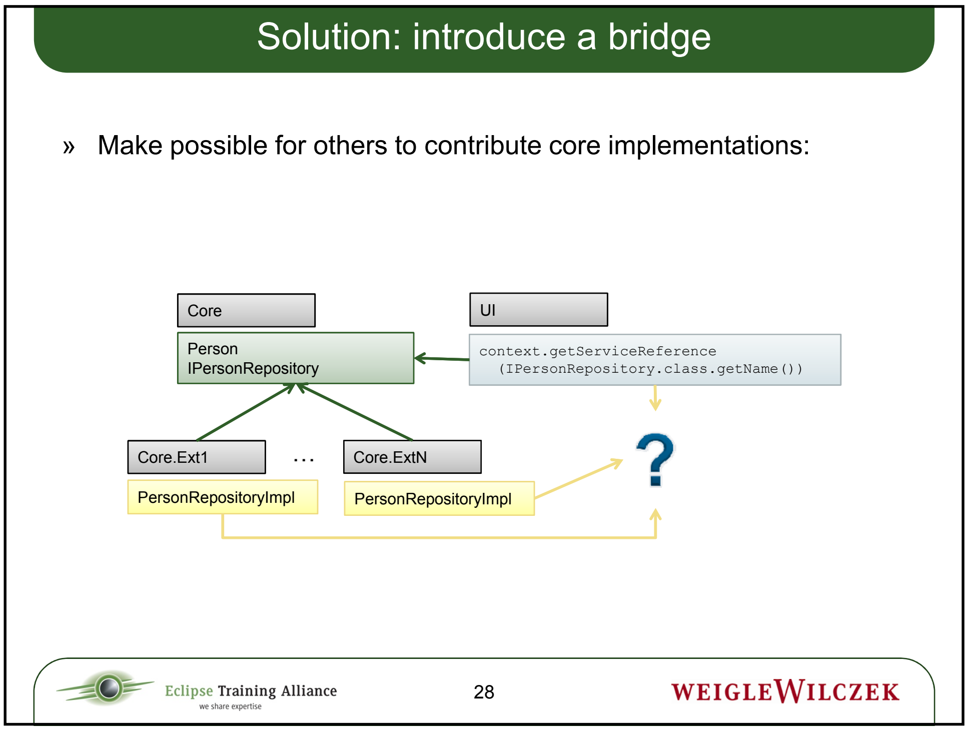
» Core bundle is designed fairly good (it is already a bridge in a fact), but it is not really flexible in terms of Components:



```
CoreActivator.getDefault()
 .getRepository().loadPersons()
```

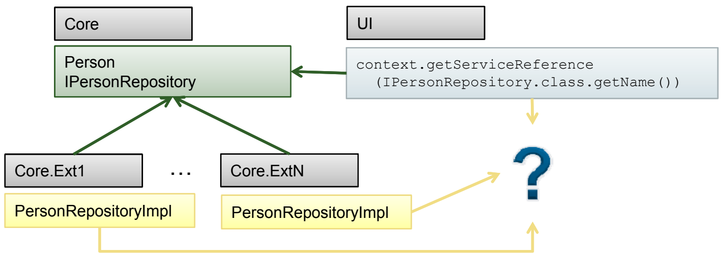
» Have to call Core methods directly ☹

» `IPersonRepository` implementation is coupled with Core ☹



Solution: introduce a bridge

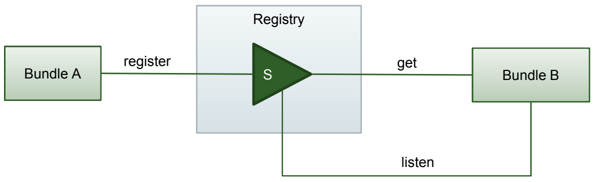
» Make possible for others to contribute core implementations:



```
context.getServiceReference
  (IPersonRepository.class.getName())
```

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);
}
```

Eclipse Training Alliance
we share expertise

29

WEIGLEWILCZEK

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();
```

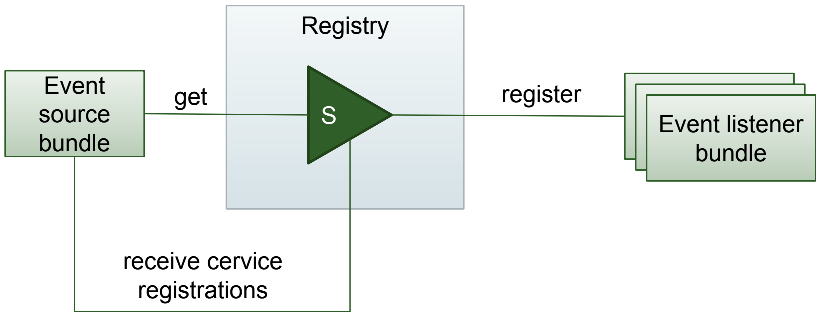
Eclipse Training Alliance
we share expertise

30

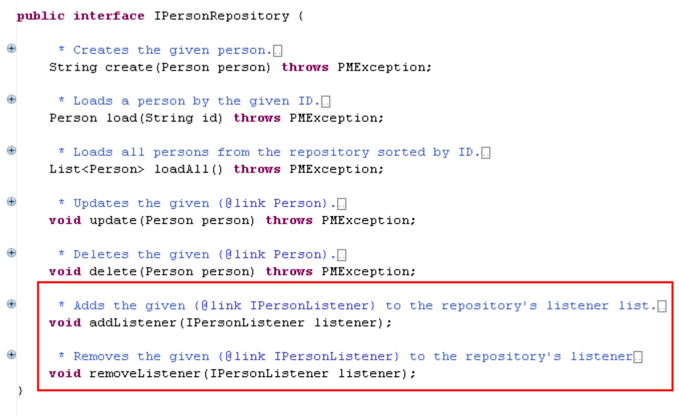
WEIGLEWILCZEK

Whiteboard: Pluggable Listeners

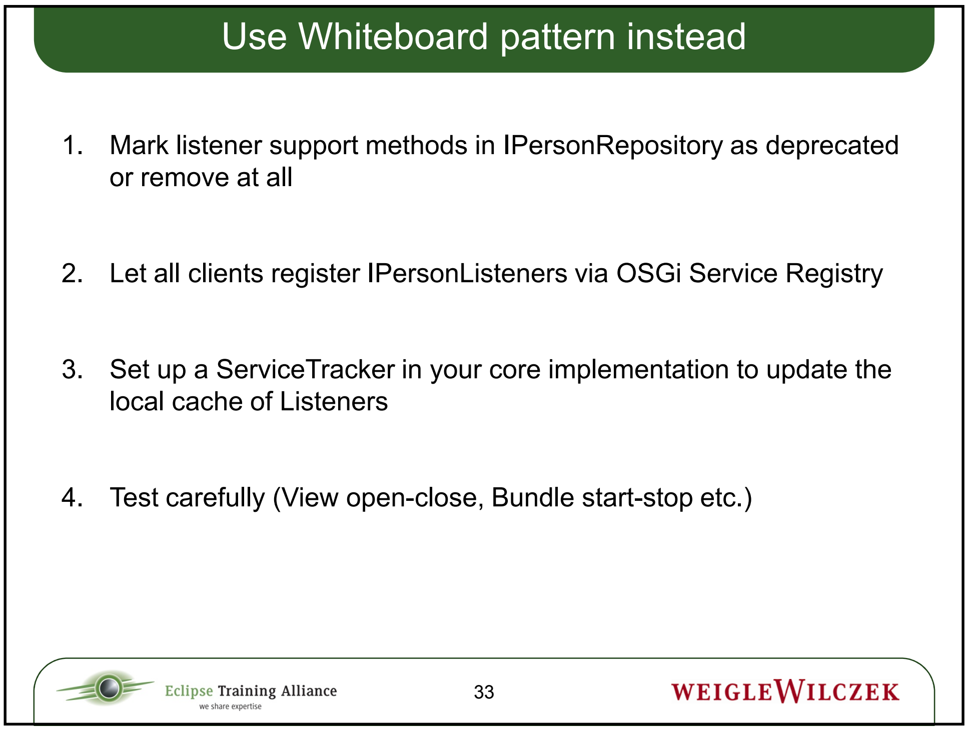
» **Listeners Considered Harmful: The “Whiteboard” Pattern**
  » http://www.osgi.org/wiki/uploads/Links/whiteboard.pdf



Listener support in IPersonRepository



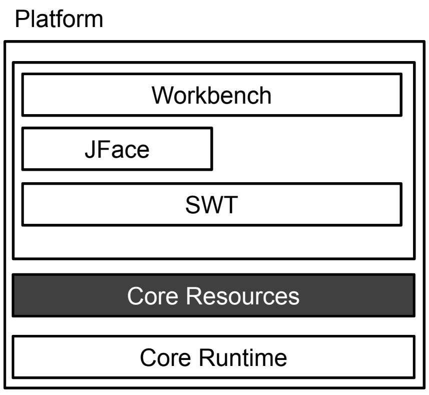
```
public interface IPersonRepository {

     * 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

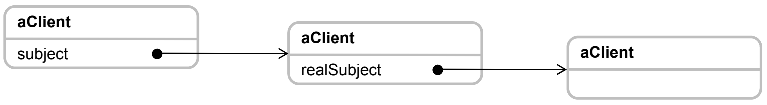
1. Mark listener support methods in IPersonRepository as deprecated or remove at all
2. Let all clients register IPersonListeners via OSGi Service Registry
3. Set up a ServiceTracker in your core implementation to update the local cache of Listeners
4. Test carefully (View open-close, Bundle start-stop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

Core Workspace: Resources



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:

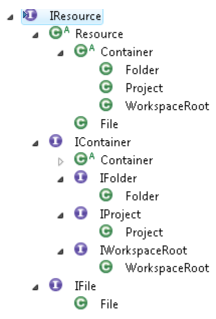


IFile as a Proxy

- 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`



```
IResource
  Resource
    Container
      Folder
      Project
      WorkspaceRoot
    File
  IContainer
    Container
    IFolder
      Folder
    IProject
      Project
    IWorkspaceRoot
      WorkspaceRoot
  IFile
    File
```

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.

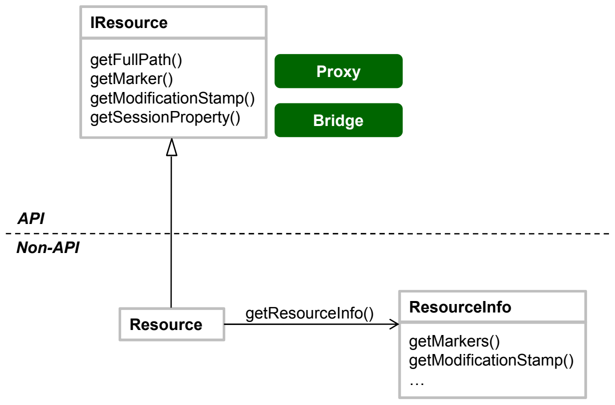
» Handles are created from a parent handle:

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

» Handles are used to create the underlying resource:

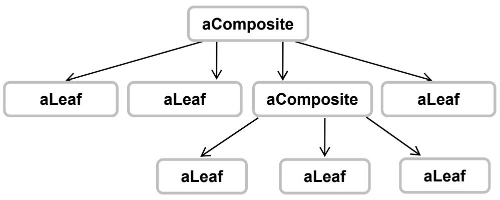
```
folder.create(...);
```

IResource Is a Proxy and a Bridge

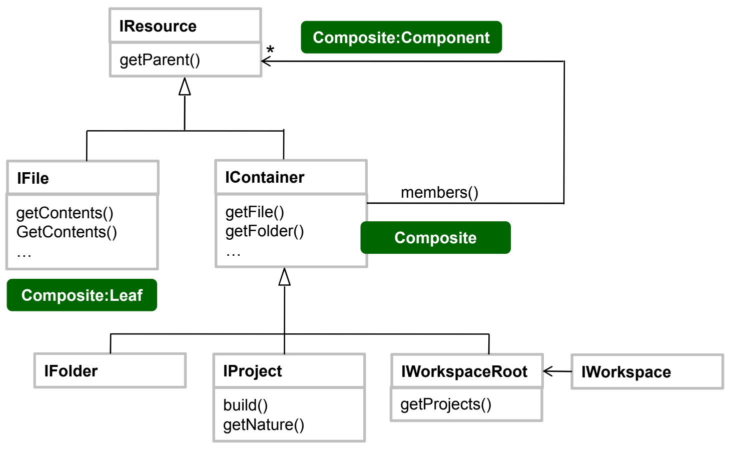


Composite: IWorkspace

- » 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:

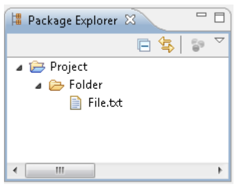


IWorkspace as a Composite



IWorkspace

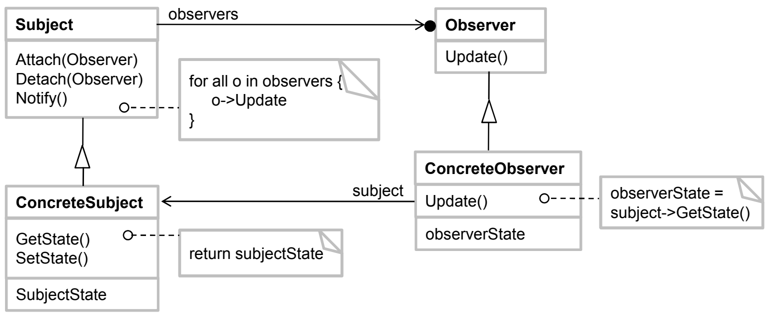
» `IWorkspace` is a Composite of `IContainer`s and `IFile`s



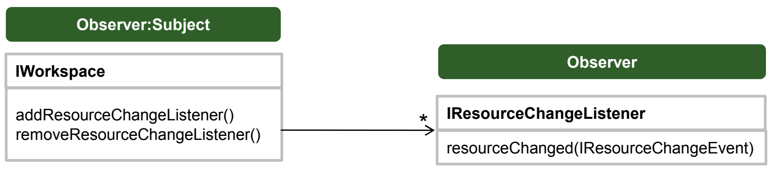
» Access the Singleton workspace instance from the static accessor `ResourcesPlugin.getWorkspace()`

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.”*

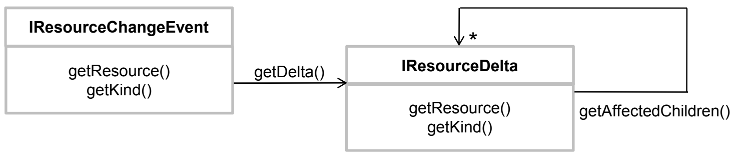


IResourceChangeListener


Observer:Subject
IWorkspace
addResourceChangeListener()
removeResourceChangeListener()
*
Observer
IResourceChangeListener
resourceChanged(IResourceChangeEvent)

```
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:

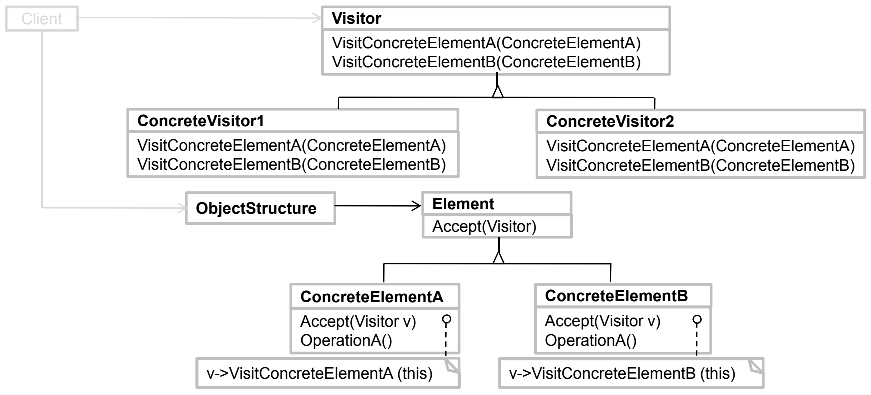
```
public interface IResourceDeltaVisitor {
     public boolean visit(IResourceDelta delta) throws CoreException;
}
```

Processing the Resource Delta

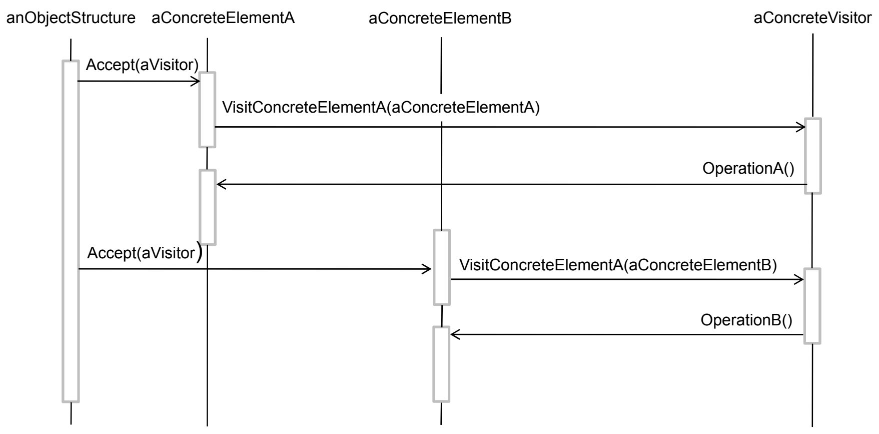
```
IWorkspace ws = ResourcesPlugin.getWorkspace();
ws.addResourceChangeListener(new IResourceChangeListener() {
    @Override
    public void resourceChanged(IResourceChangeEvent event) {
        IResourceDelta delta = event.getDelta();
        IResourceDeltaVisitor resourceDeltaVisitor = new IResourceDeltaVisitor()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

        }
    }
});
```

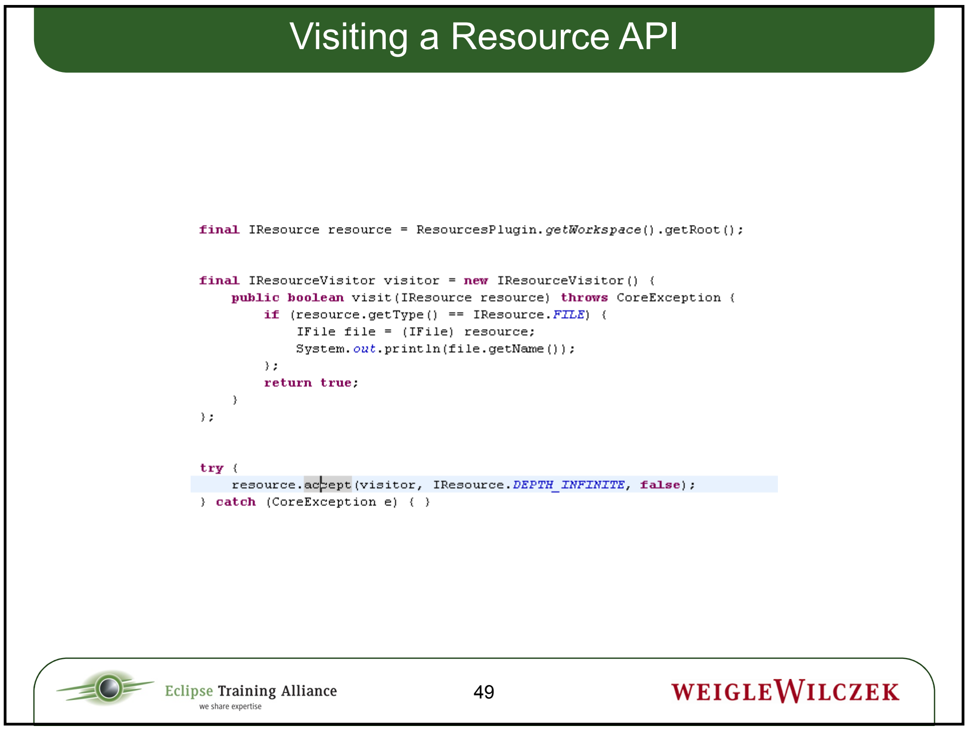
Visitor: traversing the resource tree

» 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."*


Client
Visitor
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor1
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor2
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ObjectStructure
Element
Accept(Visitor)
ConcreteElementA
Accept(Visitor v)
OperationA()
ConcreteElementB
Accept(Visitor v)
OperationA()
v->VisitConcreteElementA (this)
v->VisitConcreteElementB (this)

Visitor Interaction Diagram

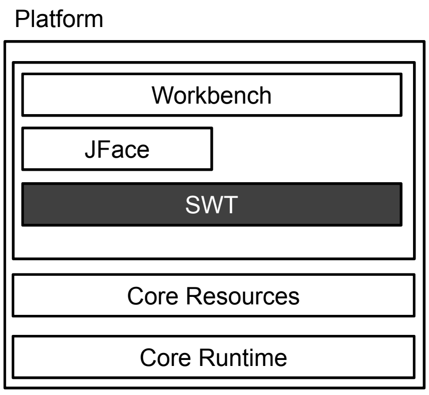




Visiting a Resource API

```java
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) { }
```

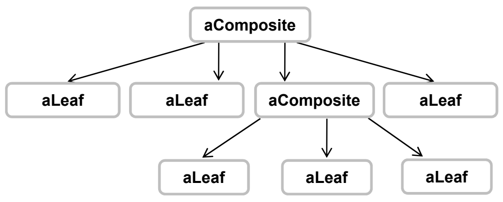
SWT



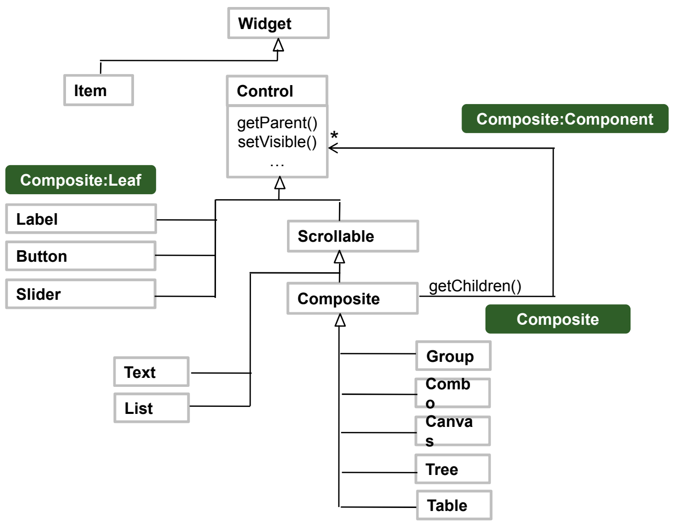
Composite: composing widgets

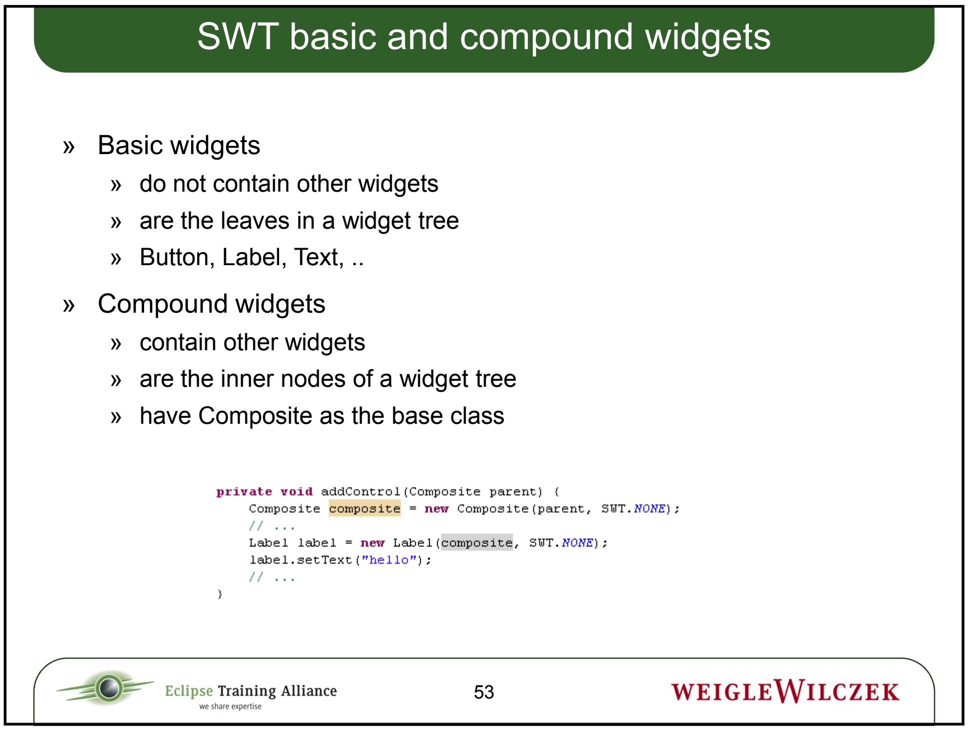
» Composite: "*Compose object into tree structures to represent part/whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.*"

» A typical Composite object structure:



SWT basic and compound widgets

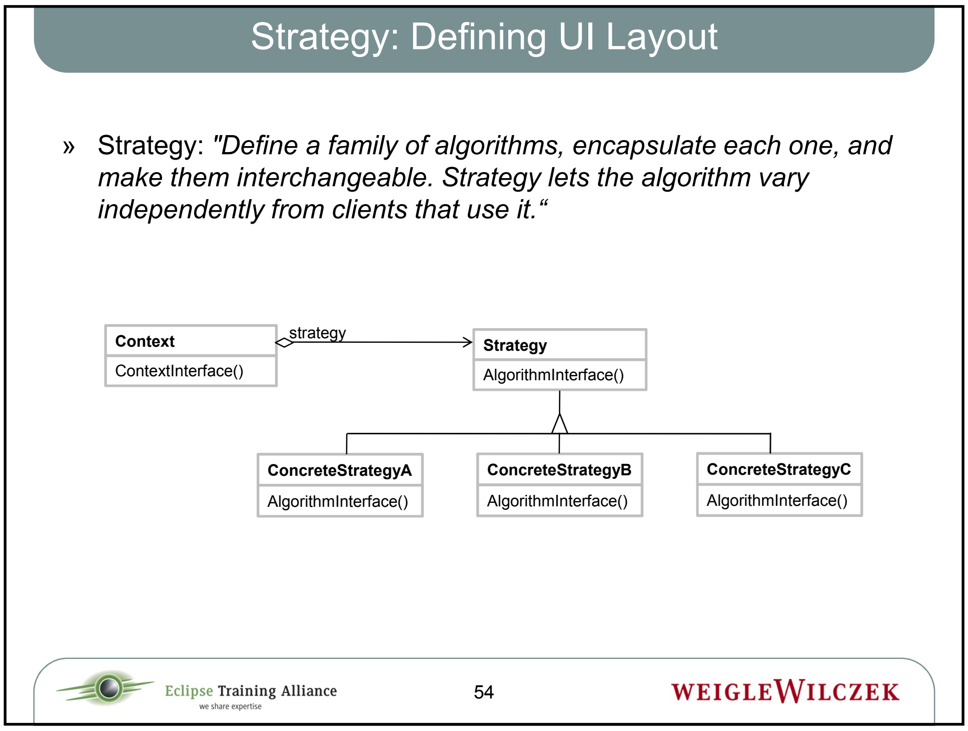




SWT basic and compound widgets

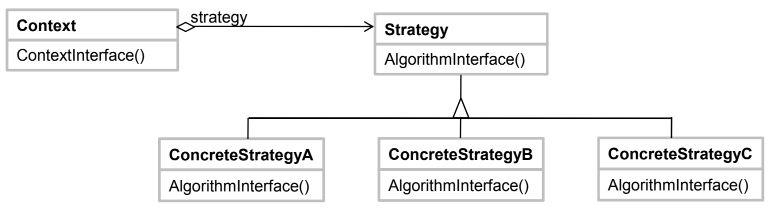
- Basic widgets
  - do not contain other widgets
  - are the leaves in a widget tree
  - Button, Label, Text, ..
- Compound widgets
  - contain other widgets
  - are the inner nodes of a widget tree
  - have Composite as the base class

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

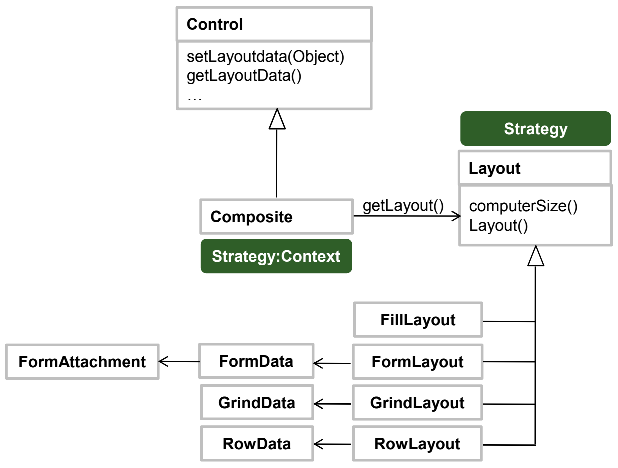


Strategy: Defining UI Layout

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



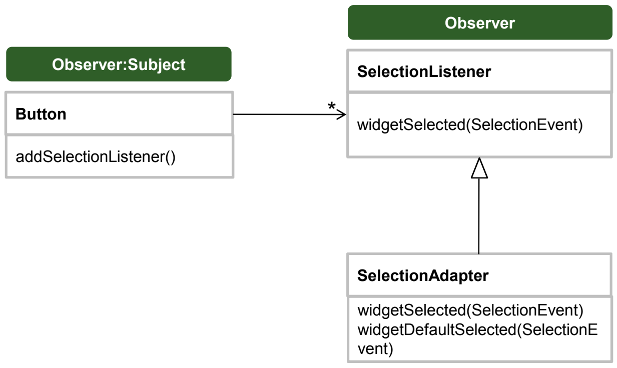
SWT Layout Managers



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");
    }

}
```

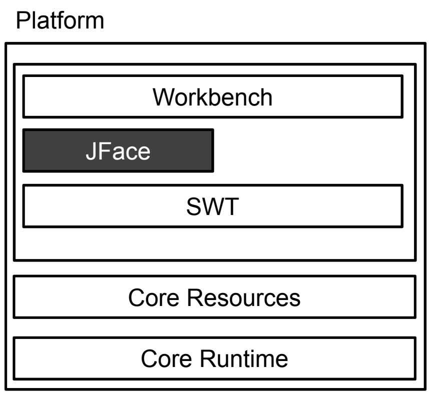
Observer: responding to Events


Observer:Subject
Button
addSelectionListener()
*
Observer
SelectionListener
widgetSelected(SelectionEvent)
SelectionAdapter
widgetSelected(SelectionEvent)
widgetDefaultSelected(SelectionEvent)

Adding Listeners to SWT widgets

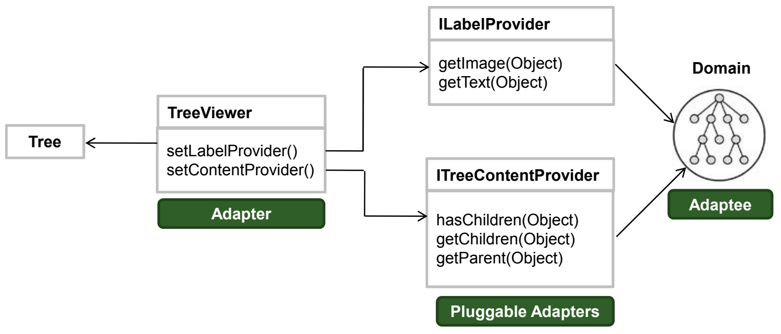
```
Text txtName = new Text(cmp, SWT.PUSH);

//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

    }
});
```

JFace



Pluggable Adapters: Label- and Content Providers

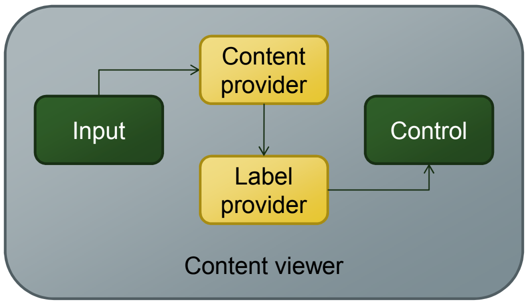
» Adapt the domain knowledge so that the Viewer can understand it and render properly:
  » Content: `IContentProvider`
  » Rendering: `ILabelProvider`



A Content Viewer ...

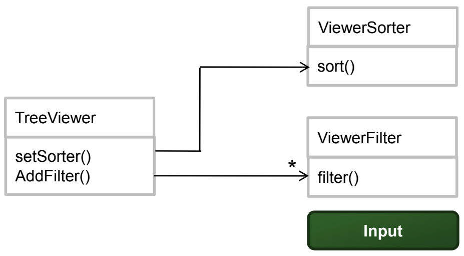
» ... delegates handling of input changes to a content provider:
   » The viewer queries the content provider for (an) element(s) to be shown

» ... delegates mapping the elements to be displayed to labels and images to a label provider



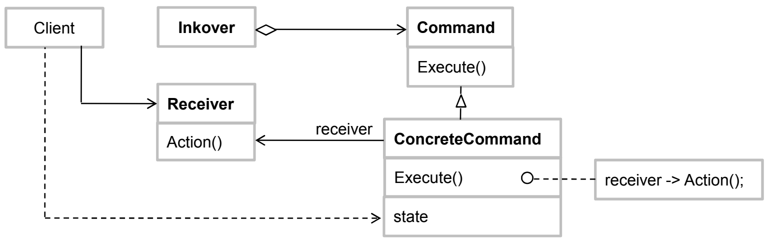
Strategy: Customizing without Subclassing

» `ViewerSorter` and `ViewerFilter` are strategies

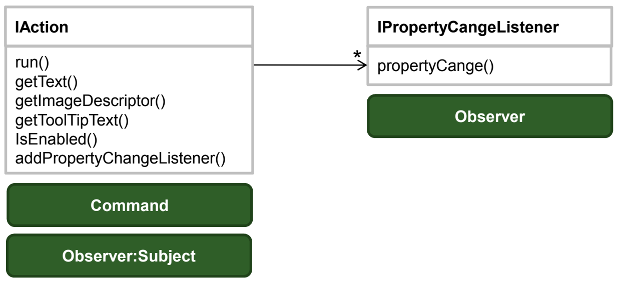


Command: Actions

» 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

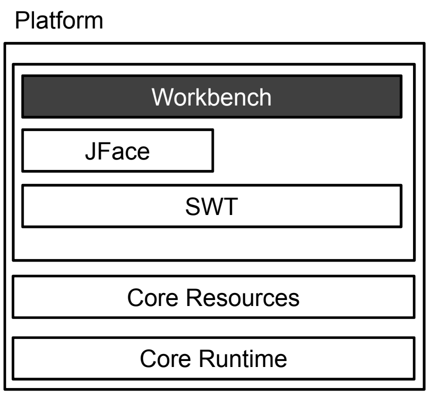
- IAction defines a run() method to be called to execute the request.
- Stores all the “decorative” information:
  - to present the action in a menu / toolbar / button
  - action's label, icon, tooltip, and enablement state
- Can be used by multiple menu items / toolbar items / buttons at the same time. 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 / …

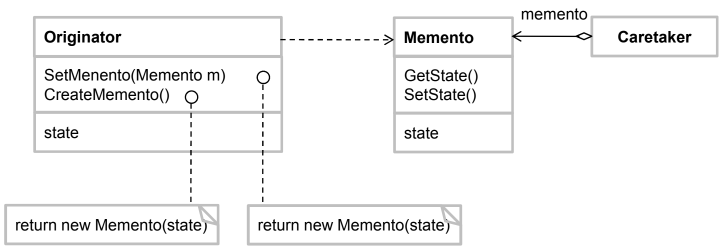
```java
final IAction action = new Action("Exit JVM") {
    @Override
    public void run() {
        System.exit(0);
    }
};
// do not override getters, use setters instead:
action.setEnabled(true);
action.setDescription("Description");
action.setToolTipText("Tooltip");

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

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

UI Workbench



Memento: Persisting UI State

» Memento: *“Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.”*



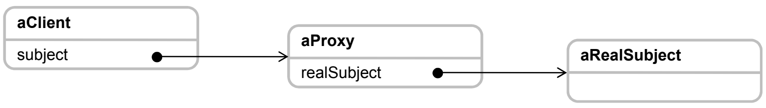
Memento: Persisting UI State

```java
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."*

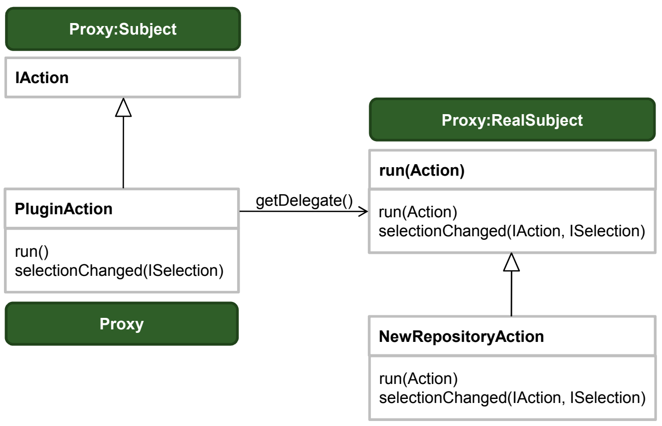
» Proxy structure at runtime:



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

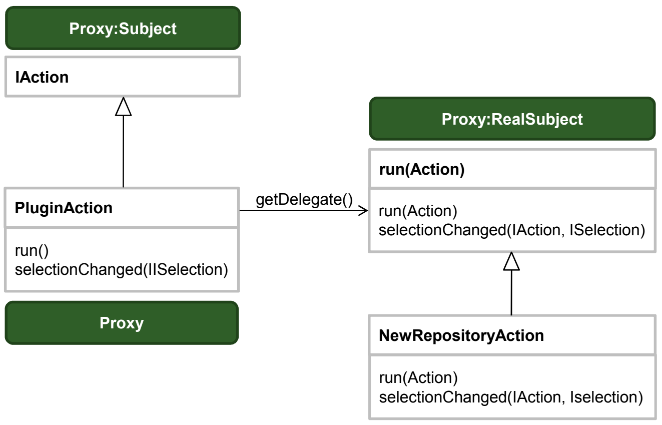
Virtual Proxies: the Lazy Loading Rule

» PluginAction Lazily Loads the Real Action:



Virtual Proxies: the Lazy Loading Rule

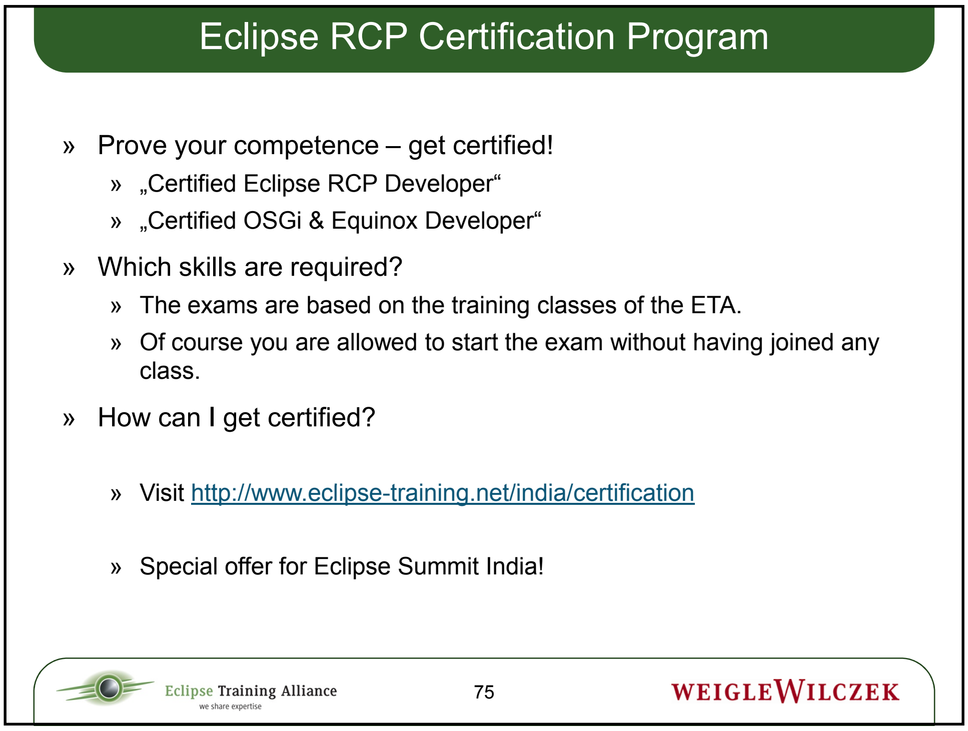
» PluginAction Lazily Loads the Real Action:



IActionDelegate implementation

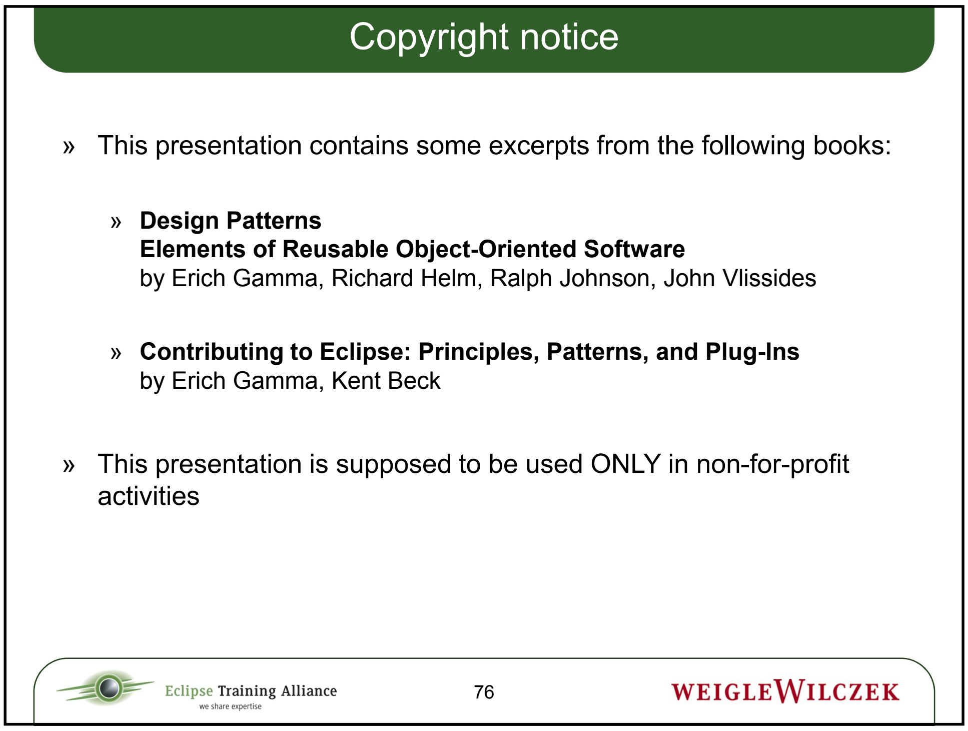
```
/**
 * Our sample action implements workbench action delegate.
 *
 * The action proxy will be created by the workbench and shown in the UI.
 *
 * When the user tries to use the action, this delegate will be created and
 * execution will be delegated to it.
 *
 */
public class SampleAction implements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