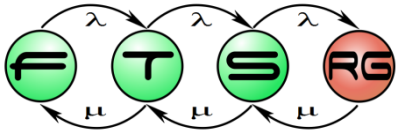
# Az Eclipse, mint integrált fejlesztőkörnyezet



# Mottó

- Any problem in computer science can be solved with another level of indirection.

David Wheeler

# Tartalomjegyzék

- Grafikus felület összekapcsolása
  - Workbench Selection Service
  - Adapterek használata (Properties nézet)
- Erőforráskezelés
  - Fájlrendszer
  - Projektkezelés
  - Háttérfolyamatok
  - Esettanulmány: Java fejlesztőeszköz

- Egyszerre több Workbenchpart látszik
  - Nézetek és szerkesztők
  - Több fejlesztőtől!
  - Együttműködnek
- Közös megoldások

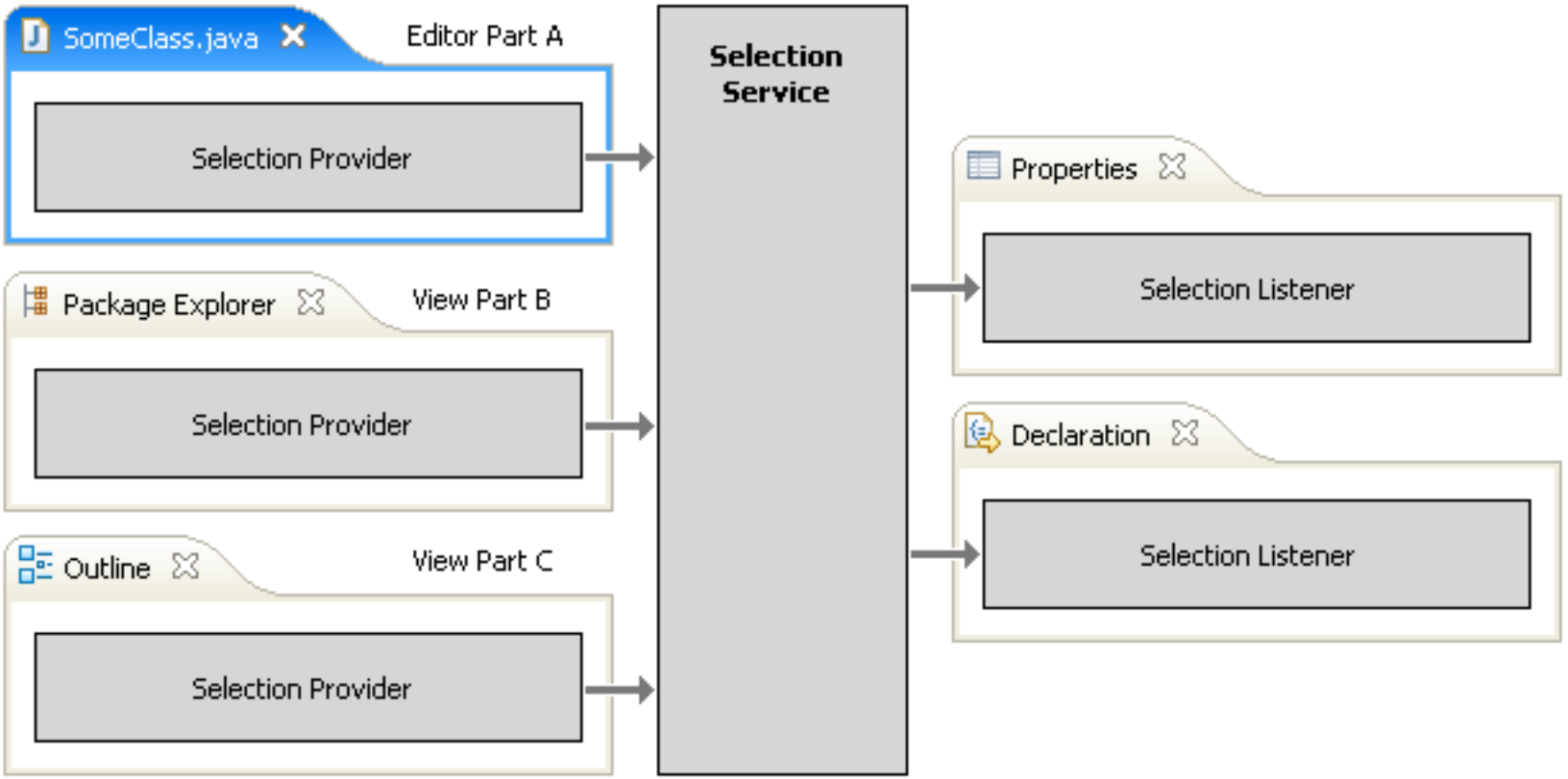
# Workbench Selection Service

# Kijelölés követése

- Gyakori UI feladat: master-detail elemek
- Workbenchparton belül:
  - SelectionChangedListener
  - Master-detail adatkötés
- WorkbenchPartok között
  - Nem ismerjük a másik part osztályát
    - Ritkán jó – erős csatolás
  - Általában ismert az ID (de már ez is közvetlen függés!)

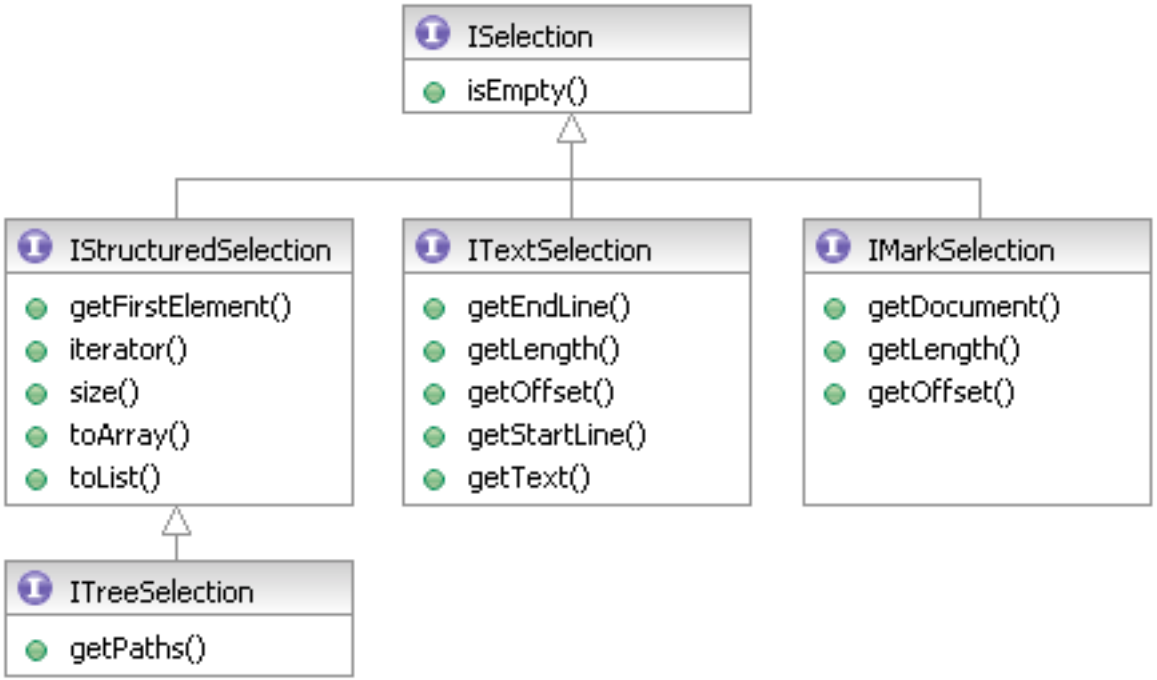
# Kijelölés követése

- SelectionService - indirekció



# Kijelölés

- ISelection hierarchia



# ISelectionProvider interfész

- Kijelölés változását jelzi
- Implementáció
  - JFace viewerek megvalósítják
  - Mi is írhatunk
- Regisztráció

```
getSite().setSelectionProvider(viewer);
```

# JFace implementáció

| Viewer | Selection type |
|---|---|
| ComboViewer | IStructuredSelection |
| ListViewer | IStructuredSelection |
| TreeViewer | IStructuredSelection, ITreeSelection |
| +-- CheckboxTreeViewer | IStructuredSelection, ITreeSelection |
| TableViewer | IStructuredSelection |
| +-- CheckboxTableViewer | IStructuredSelection |
| TextViewer | ITextSelection, IMarkSelection |
| +-- SourceViewer | ITextSelection, IMarkSelection |

# Kijelölés figyelése

- ISelectionService megkeresés
  - `getSite().getWorkbenchWindow().getSelectionService()`
- Aktuális kijelölés lekérése
  - `ISelection getSelection()`
    - Aktuális kijelölés lekérése
  - `ISelection getSelection(String partId)`
    - Megadott Part kijelölésének lekérése

# Kijelölés változásai

- ISelectionListener interfész
- Egyetlen metódus:
  - `public void selectionChanged(IWorkbenchPart sourcepart, ISelection selection)`
  - Paraméterek:
    - Forrás
    - Kijelölés

# Változásfigyelés

- Minden változásra figyelés
  - `void addSelectionListener(ISelectionListener listener)`
  - `void removeSelectionListener(ISelectionListener listener)`
- Adott part változásaira figyelés
  - `void addSelectionListener(String partId, ISelectionListener listener)`
  - `void removeSelectionListener(String partId, ISelectionListener listener)`

# Takarítás

- Regisztrált listenereket távolítsuk el!
  - Garbage collection nem elég

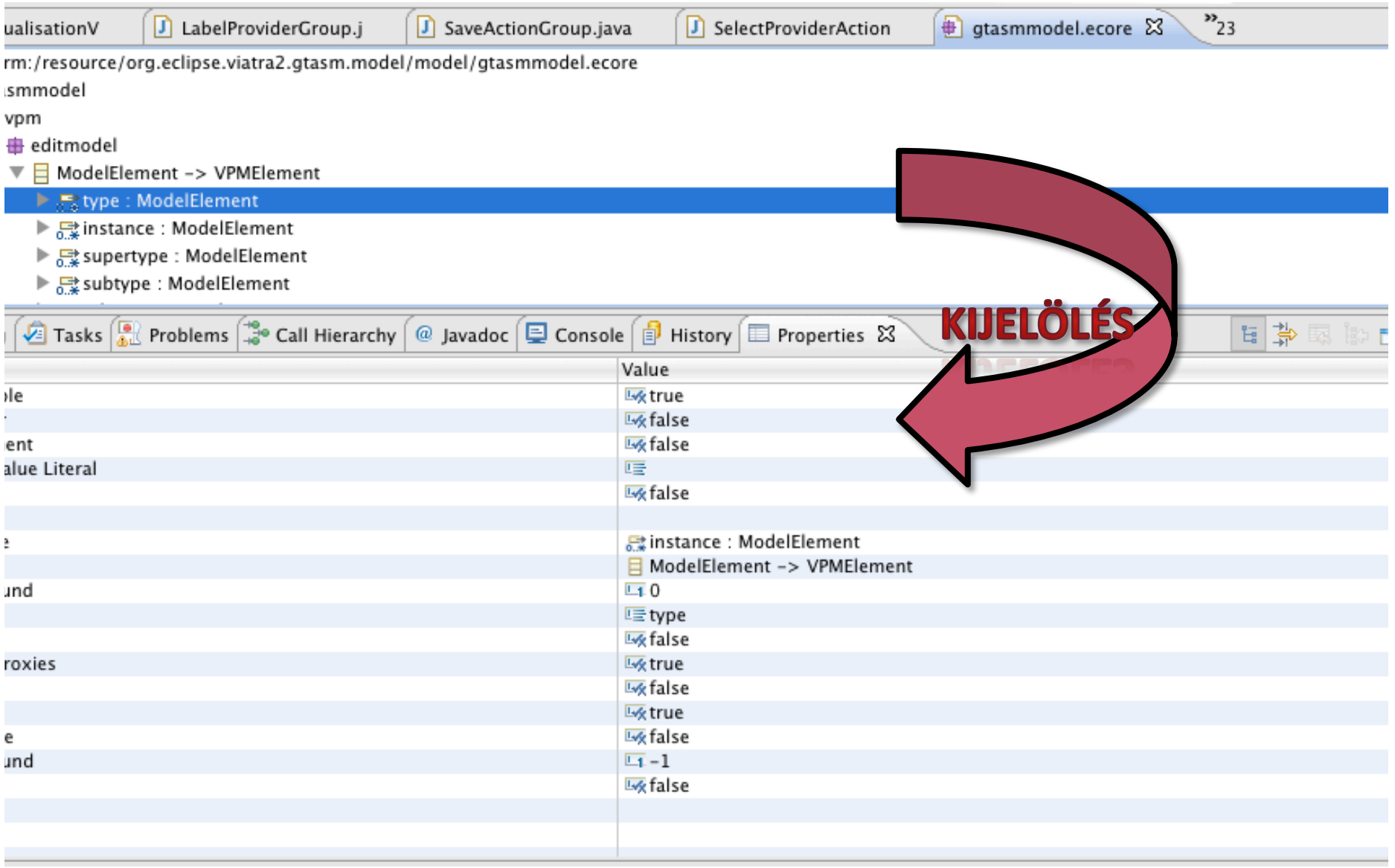
```
public void dispose() {
  ISelectionService s = getSite().
    getWorkbenchWindow().
    getSelectionService();
  s.removeSelectionListener(mylistener);
  super.dispose();
}
```

# Segédanyag

- Marc R. Hoffmann: Eclipse Workbench: Using the Selection Service
  - http://eclipse.org/articles/Article-WorkbenchSelections/article.html

# Adapterek készítése

# Példa: Properties nézet



# Példa: Properties nézet

- Kijelölés feldolgozása
  - IPropertySource interfész (+ IPropertySource2)

```
public interface IPropertySource {
        public Object getEditableValue();
        public IPropertyDescriptor[] getPropertyDescriptors();
        public Object getPropertyValue(Object id);
        public boolean isPropertySet(Object id);
        public void resetPropertyValue(Object id);
        public void setPropertyValue(Object id, Object value);
}

public interface IPropertySource2 extends IPropertySource {
     boolean isPropertyResettable(Object id);
     public boolean isPropertySet(Object id);
}
```

# Interfész implementáció

- Hogyan implementáljuk az interfészt?
  - Közvetlen implementáció
    - `class FileProperties implements IPropertySource2`

# Interfész implementáció

- Hogyan implementáljuk az interfészt?
  - Közvetlen implementáció
    - `class FileProperties implements IPropertySource2`
  - Többféle interfész kellhet -> áttekinthetetlen
  - Interfész átvétele -> API változás!
  - Kellemetlen függőségek
    - IFile (később) tudna Properties nézetről (GUI-Core link)

# Interfész implementáció

- Hogyan implementáljuk az interfészt?
  - Adapter osztály készítése

```
public class FigureProperties implements IPropertySource2{

        IFigure figure;

        public FigureProperties(IFigure figure){this.figure = figure;}
        public IFigure toFigure() { return this.figure; }

        @Override
        public Object getEditableValue() {…}
        @Override
        public Object getPropertyValue(Object id) {…}
        @Override
        public boolean isPropertySet(Object id) {…}
        @Override
        public void resetPropertyValue(Object id) {…}
        @Override
        public void setPropertyValue(Object id, Object value) {…}


}
```

# Interfész implementáció

- Hogyan implementáljuk az interfészt?
  - Adapter osztály készítése

Csatolás a modellhez

```
public class FigureProperties implements IPropertySource2{

        IFigure figure;

        public FigureProperties(IFigure figure){this.figure = figure;}
        public IFigure toFigure() { return this.figure; }

        @Override
        public Object getEditableValue() {…}
        @Override
        public Object getPropertyValue(Object id) {…}
        @Override
        public boolean isPropertySet(Object id) {…}
        @Override
        public void resetPropertyValue(Object id) {…}
        @Override
        public void setPropertyValue(Object id, Object value) {…}


}
```

# Interfész implementáció

- Hogyan implementáljuk az interfészt?
  - Adapter osztály készítése

```
public class FigureProperties implements IPropertySource2{

        IFigure figure;

        public FigureProperties(IFigure figure){this.figure = figure;}
        public IFigure toFigure() { return this.figure; }

        @Override
        public Object getEditableValue() {…}
        @Override
        public Object getPropertyValue(Object id) {…}
        @Override
        public boolean isPropertySet(Object id) {…}
        @Override
        public void resetPropertyValue(Object id) {…}
        @Override
        public void setPropertyValue(Object id, Object value) {…}


}
```

Csatolás a modellhez

Interfész megvalósítás

# Adapterek csatolása

- Adapterek bedrótozhatóak

```
public class AdaptableShape implements IFigure,
IAdaptable {

  @Override
  public Object getAdapter(Class adapter) {
    if (IPropertySource2.class.equals(adapter)) {
     return new FigureProperties(this);
    }
    return null;
  }
…
}
```

# Adapterek csatolása

- Adapterek bedrótozhatóak

```
public class AdaptableShape implements IFigure,
IAdaptable {

  @Override
  public Object getAdapter(Class adapter) {
    if (IPropertySource2.class.equals(adapter)) {
     return new FigureProperties(this);
    }
    return null;
  }
…
}
```

Típus szerint kérünk

# Adapterek csatolása

- Adapterek bedrótozhatóak

```
public class AdaptableShape implements IFigure,
IAdaptable {

  @Override
  public Object getAdapter(Class adapter) {
    if (IPropertySource2.class.equals(adapter)) {
     return new FigureProperties(this);
    }
    return null;
  }
…
}
```

Típus szerint kérünk

Nem nyertünk sokat!

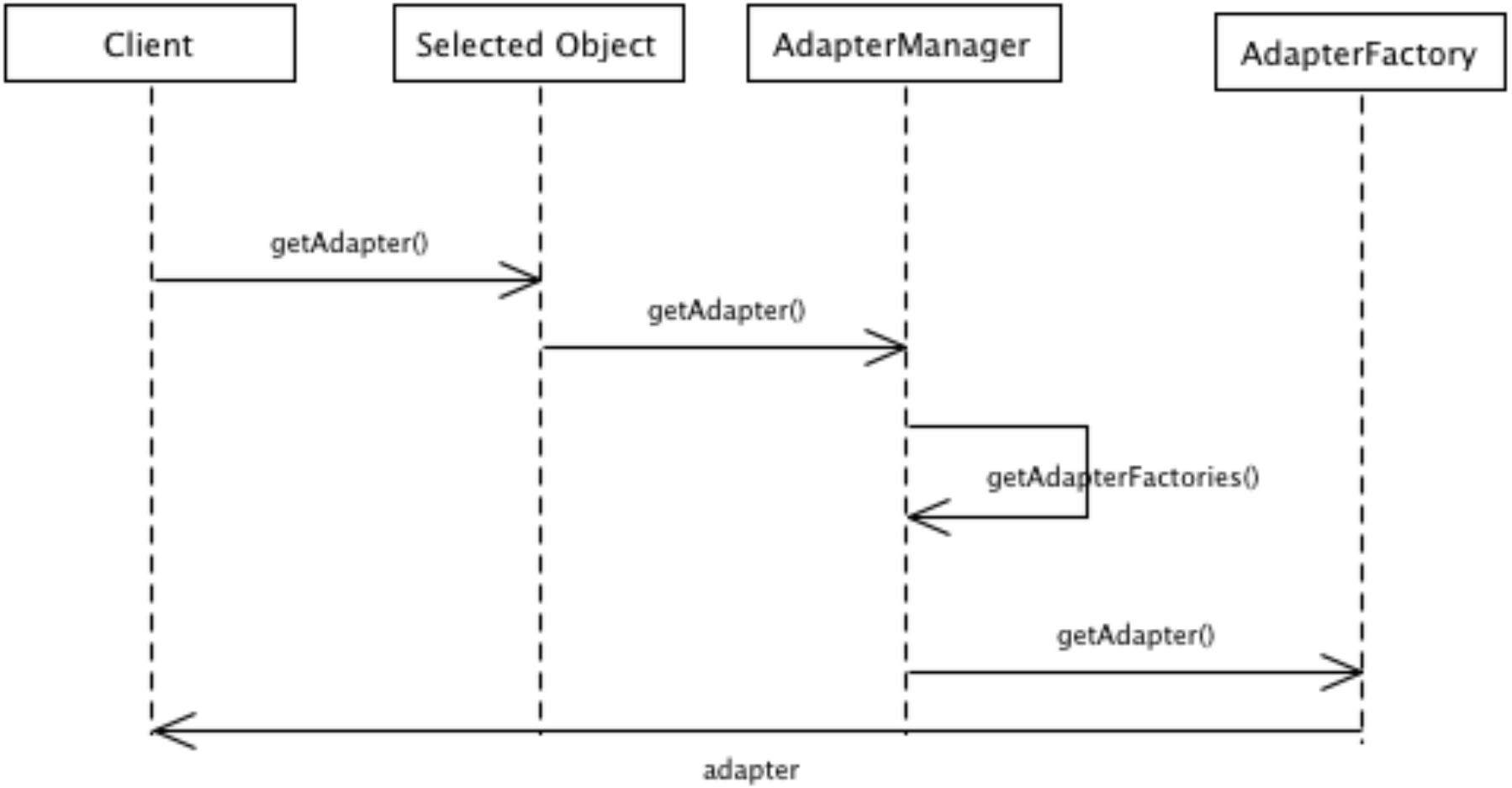
# Adapterek csatolása

- Platform bővítési lehetőséget ad

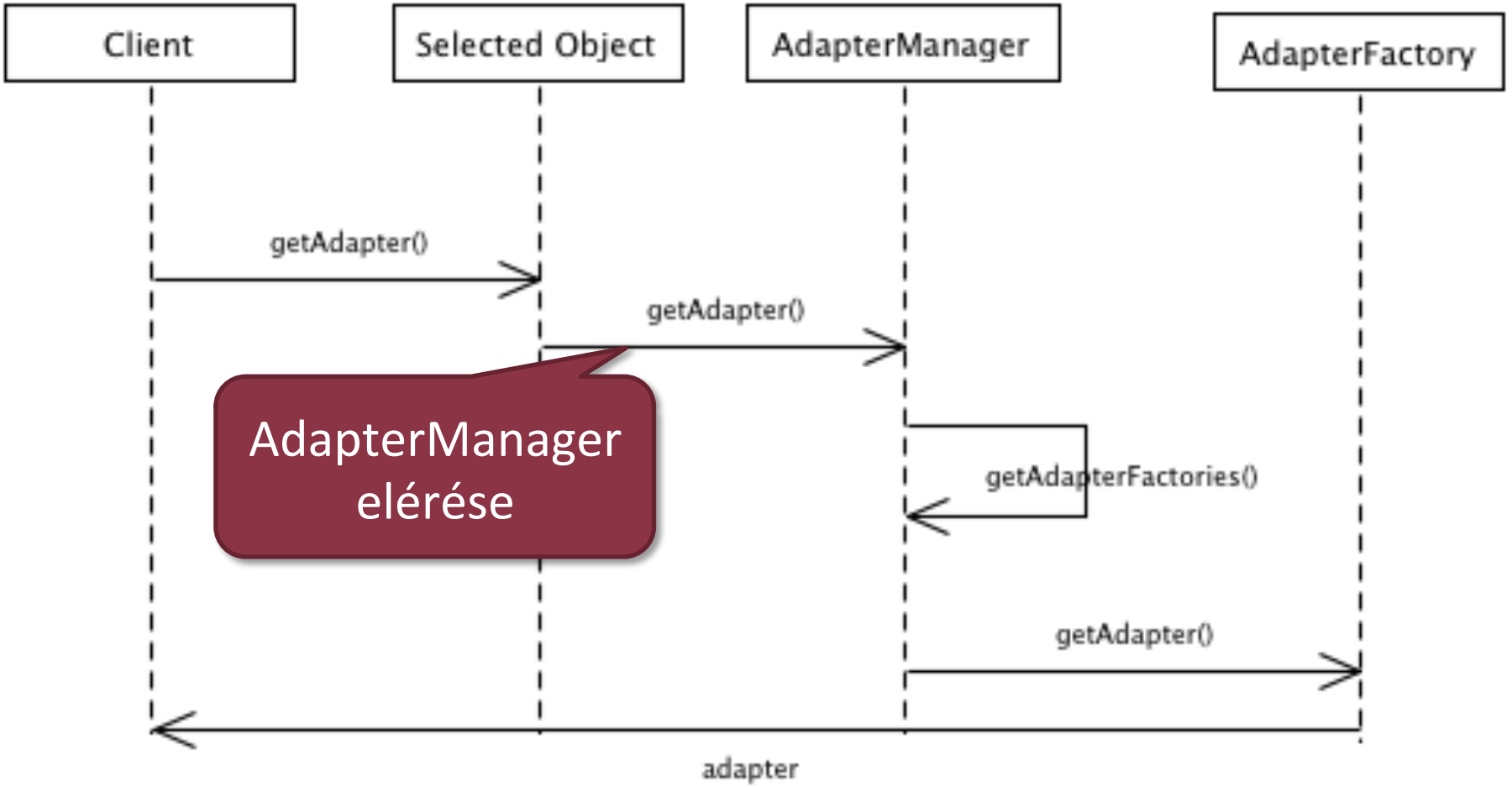
```
public class AdaptableShape implements IFigure,
IAdaptable {

  @Override
  public Object getAdapter(Class adapter) {
    if (IPropertySource2.class.equals(adapter)) {
     return new FigureProperties(this);
    }
     return Platform.getAdapterManager().
      getAdapter(this, adapter);
  }
…
}
```

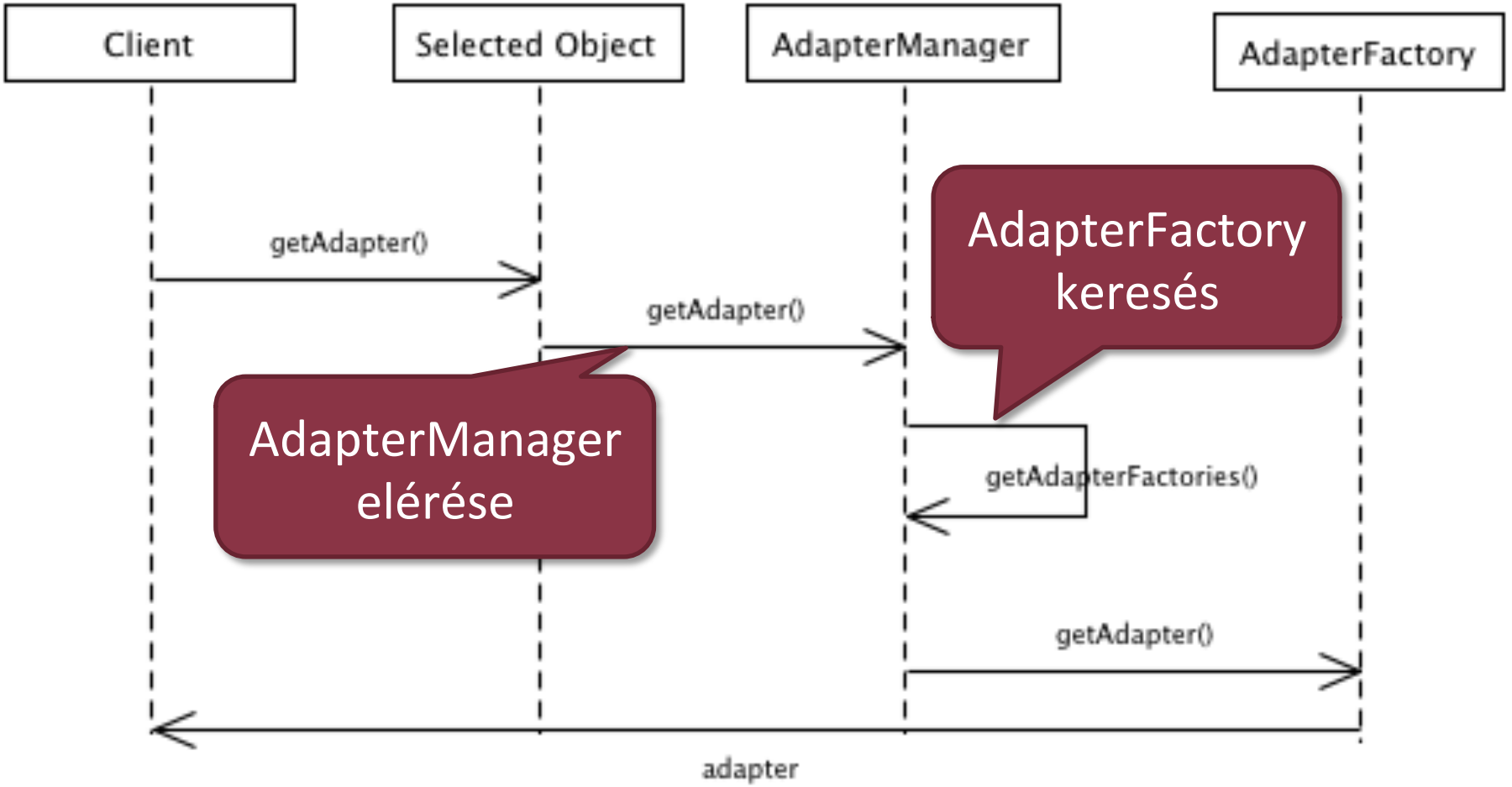
# Adapterek elkérése



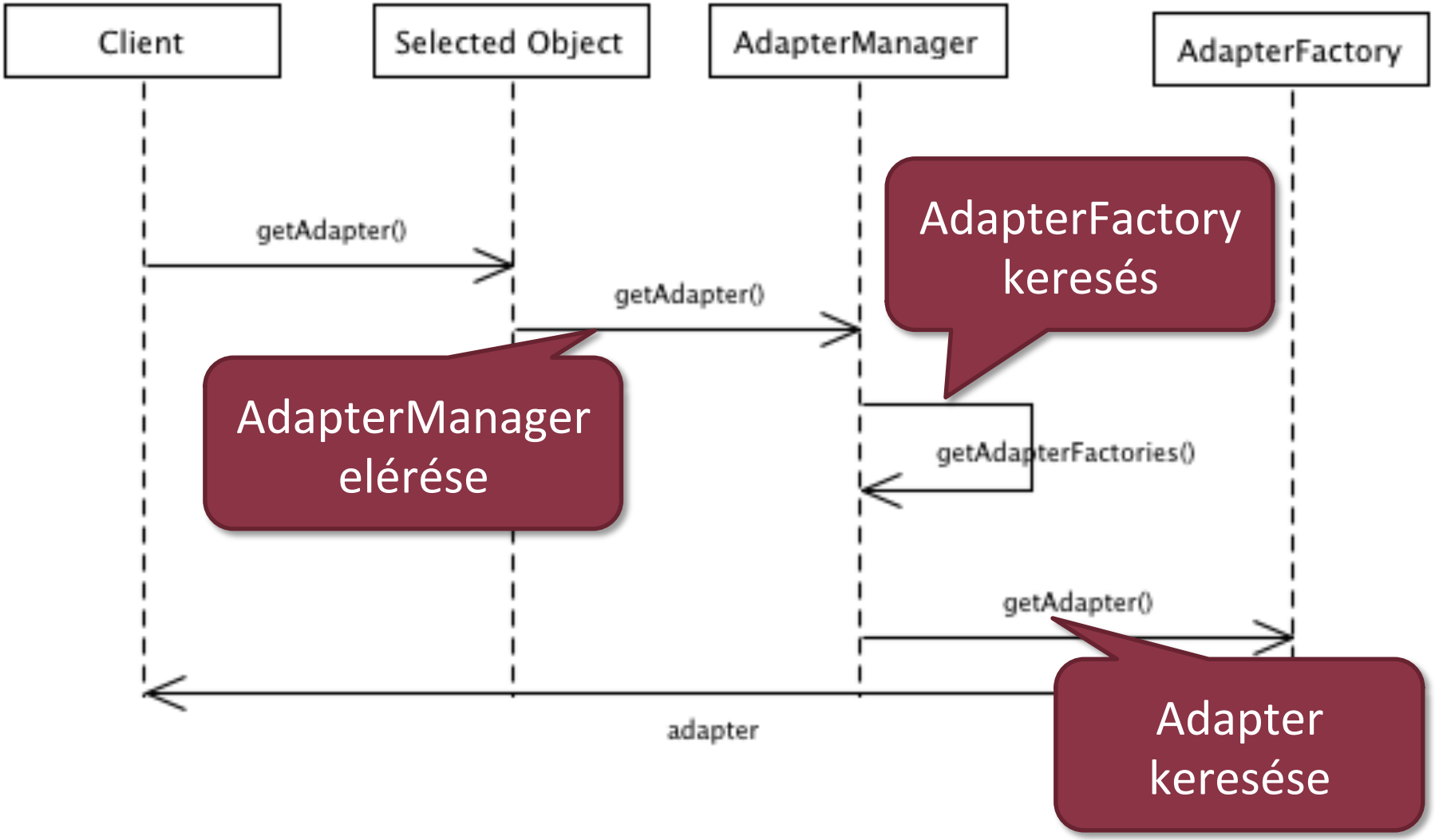
# Adapterek elkérése



# Adapterek elkérése



# Adapterek elkérése



# Adapterek regisztrálása

- AdapterFactory
  - Osztály, amely képes a modellelemből Adaptert előállítani
  - Követelmény
    - Ne legyen egy modellelemhez két különböző adapter
      - Többnyire elég, ha equals() és hashCode() megegyezik ☺

# Segédanyagok

- Dicky Johan: Take control of your properties
  - http://eclipse.org/articles/Article-Properties-View/properties-view.htm
- Jeffrey Ricker: Eclipse Adapters: A Hands-on Hand Holding Explanation
  - http://eclipse.org/resources/resource.php?id=407
- Wayne Beaton: Adapters
  - http://www.eclipse.org/articles/article.php?file=Article-Adapters/index.html

# Erőforráskezelés

# Erőforrások

- Erőforrások
  - Az Eclipse fájlrendszer alapú
    - Nincs köztes repository
    - A fájlok módosítása történhet Eclipse-ből vagy kívülről (közvetlenül)
  - Erőforrások
    - Létrejönnek/módosulnak/törlődnek
    - Nem akarjuk az állapotot több helyen tárolni
      - Állapotmentes referencia kell
      - Eclipse: csak Handle-t kap a kliens

# Erőforrások

- Handle
  - Proxy és Bridge tervezési minta összekötése
  - Proxy
    - helyettes létrehozása egy objektum számára
    - hozzáférés kontrollálása
    - érvénytelen állapot kialakulásának elkerülése
  - Bridge
    - leválasztja az interfészt és az implementációt
    - ezek így függetlenül módosulhatnak
    - erős elválasztás

# Erőforrások

- Alkalmazás a fájlrendszerre
  - Egy handle: kulcs a fájlhoz
  - Egy info objektum, mely a fájl állapotát tárolja
    - A handle interfészekhez csak egy info implementáció van.
  - Handle: IFile, IFolder, IProject, IWorkspaceRoot
    - Nem implementálandóak!

# Erőforrások

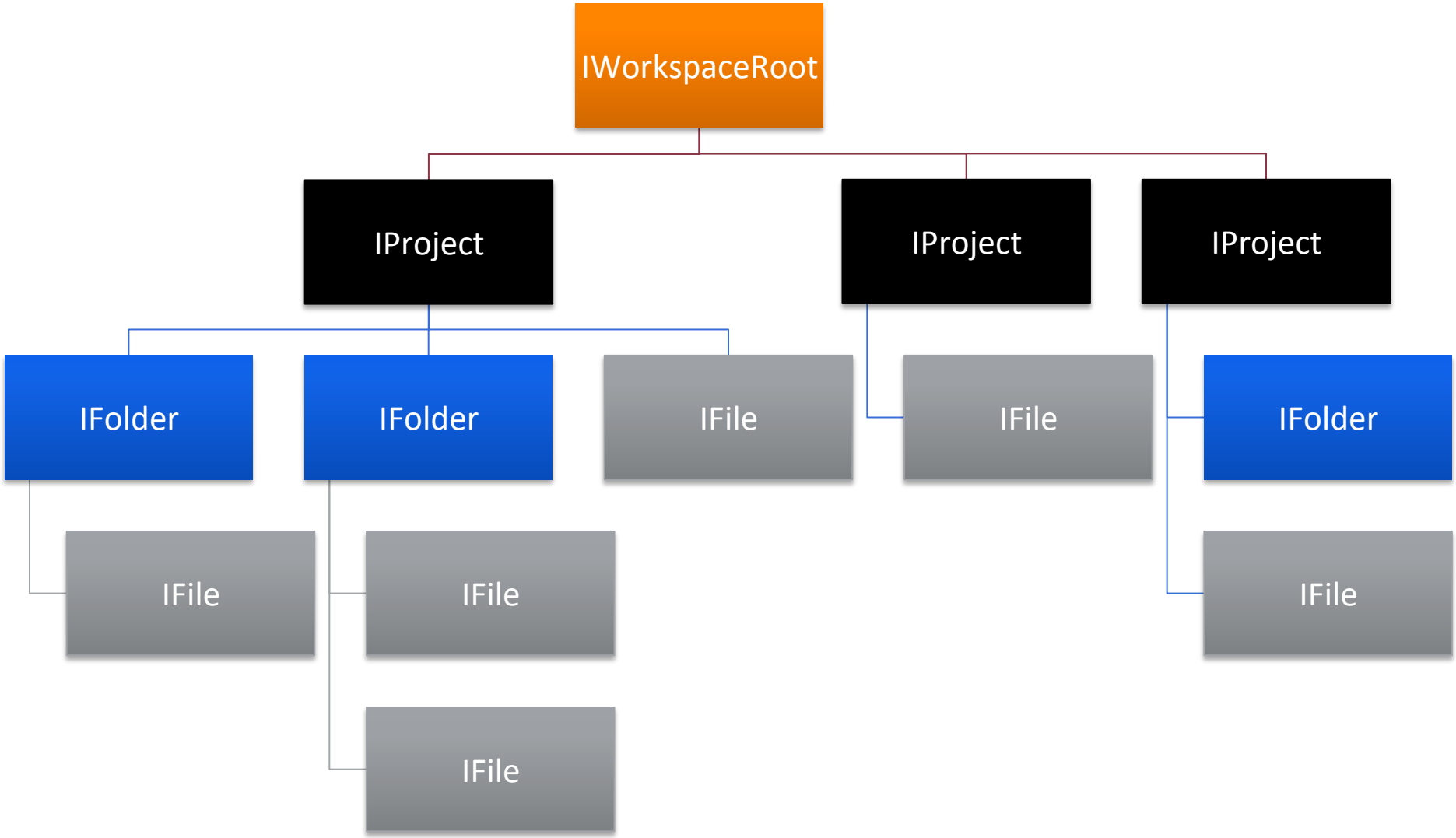
- Handle
  - Érték típusok, az egyenlőséget az equals() metódussal vizsgáljuk
    - Hash-elhetőek
    - Több handle mutathat ugyanarra az erőforrásra
  - Definiálhatják az erőforrás viselkedését, de az állapotát nem tárolhatják
  - Egy handle mutathat nem létező erőforrásra
  - Néhány művelet csak a handle-ben levő információkra alapoz, ezek nem létező erőforrás esetén is sikeresek

# Erőforrások

- Handle
  - Ha egy műveletnek szüksége van az erőforrásra, CoreException-t dob, ha az nem létezik
  - Létezés tesztelése: exists()
  - A handle egy szülő handle-ből keletkezik
  - A handle használható az erőforrás létrehozására is

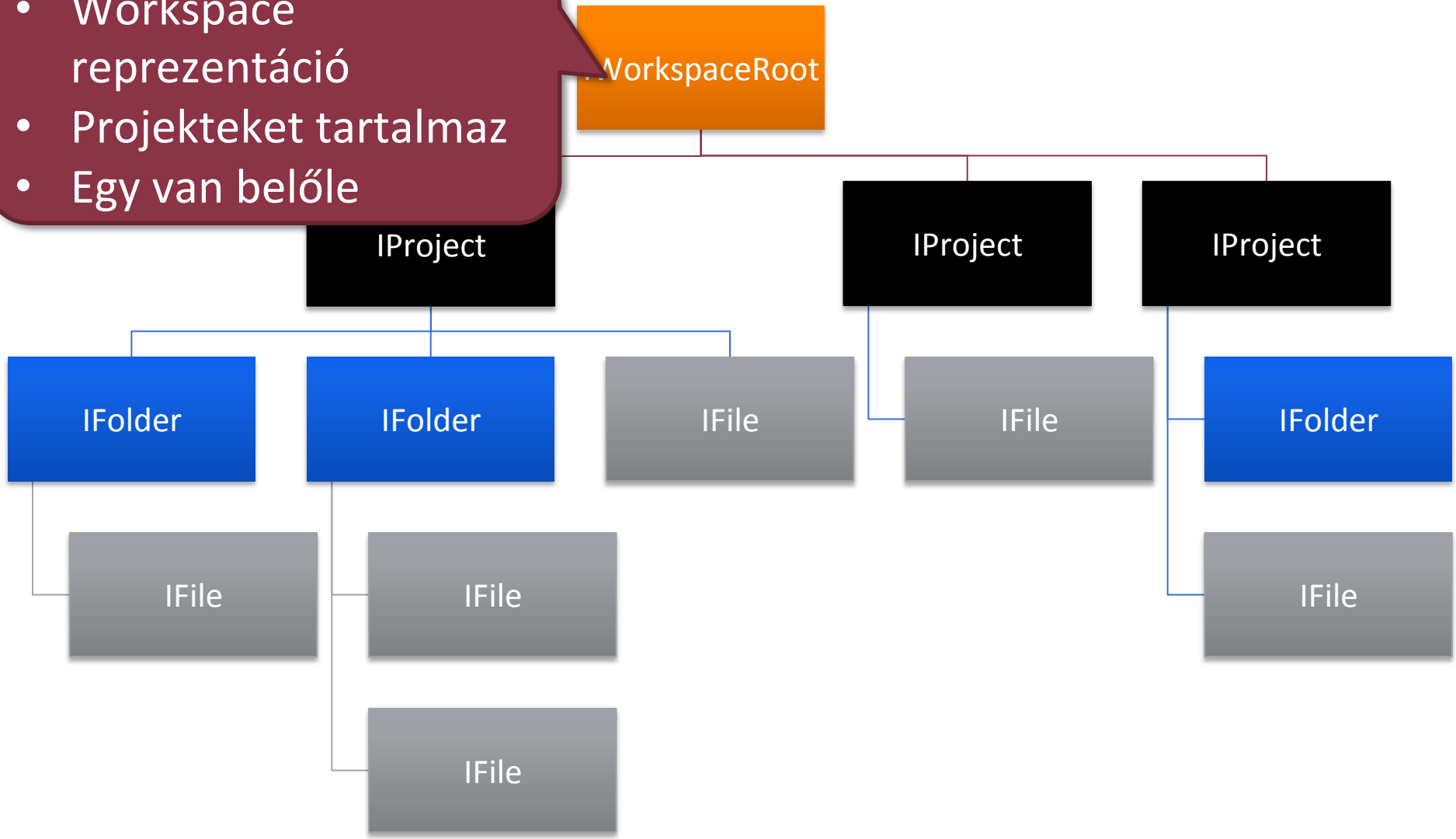
```
IProject project;
IFolder folder =
  project.getFolder(“someFolder”);
folder.create(...);
```

# Workspace hierarchia



# Workspace hierarchia

- Workspace reprezentáció
- Projekteket tartalmaz
- Egy van belőle

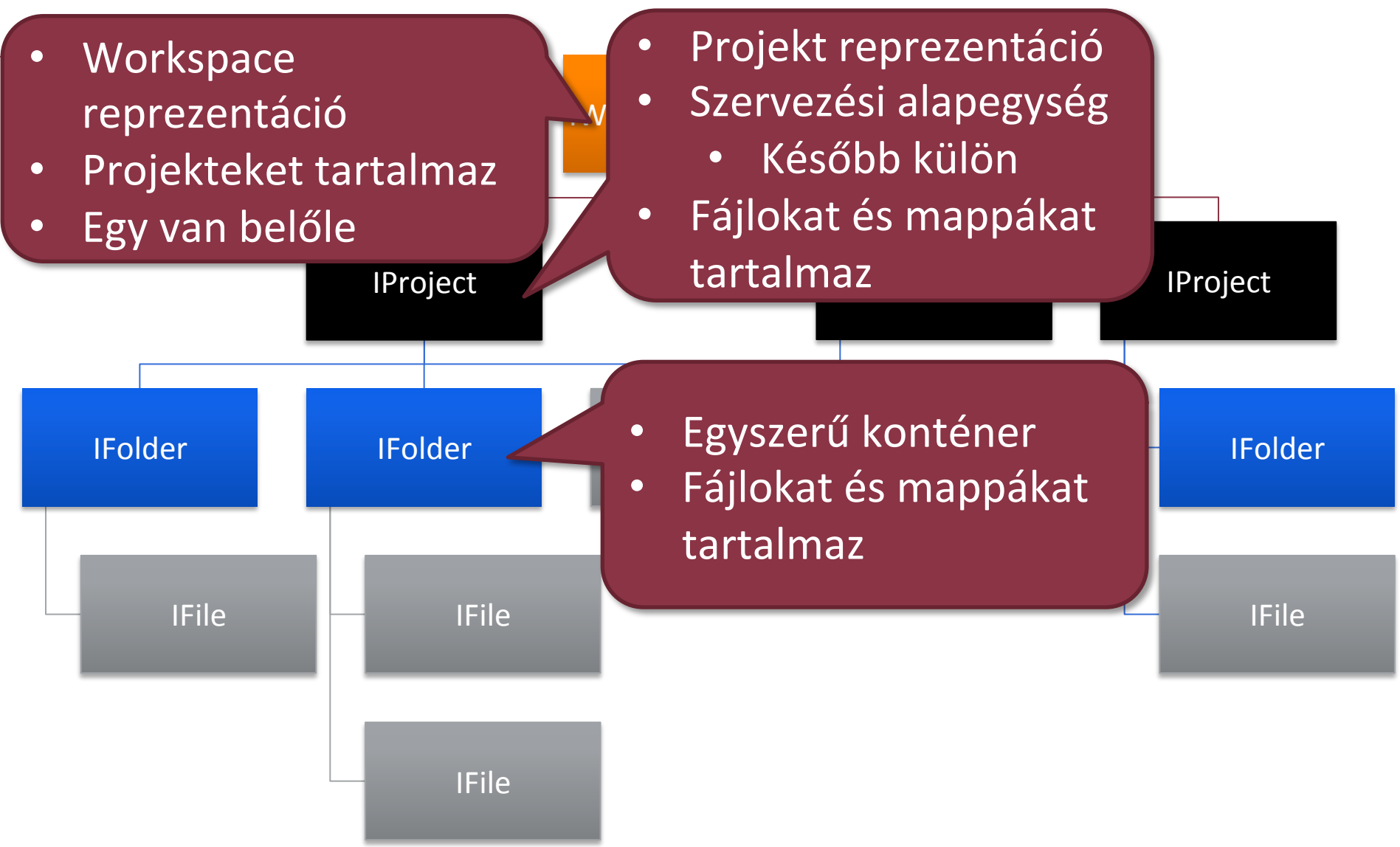


# Workspace hierarchia

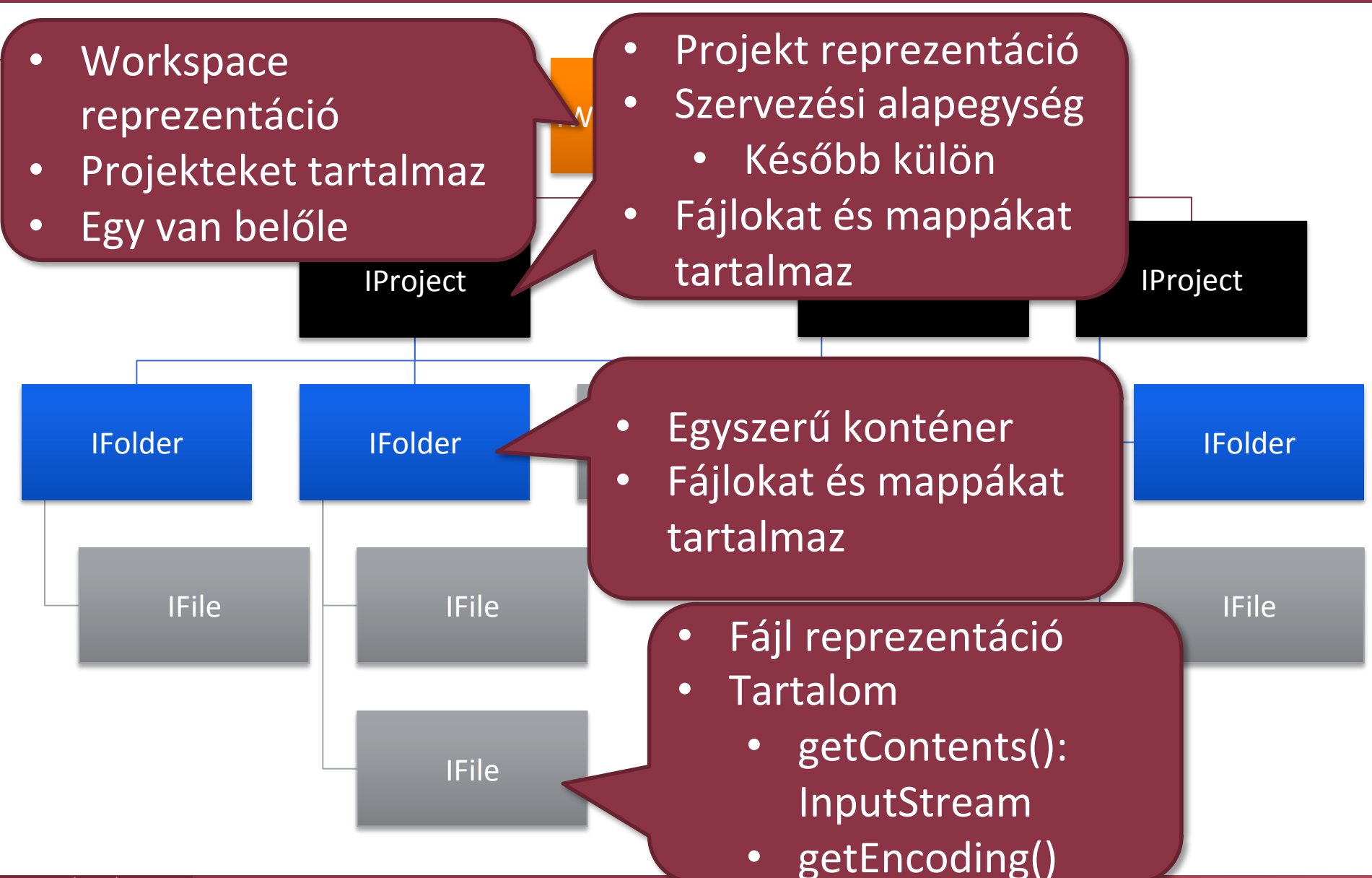
- Workspace reprezentáció
- Projekteket tartalmaz
- Egy van belőle

- Projekt reprezentáció
- Szervezési alapegység
  - Később külön
- Fájlokat és mappákat tartalmaz

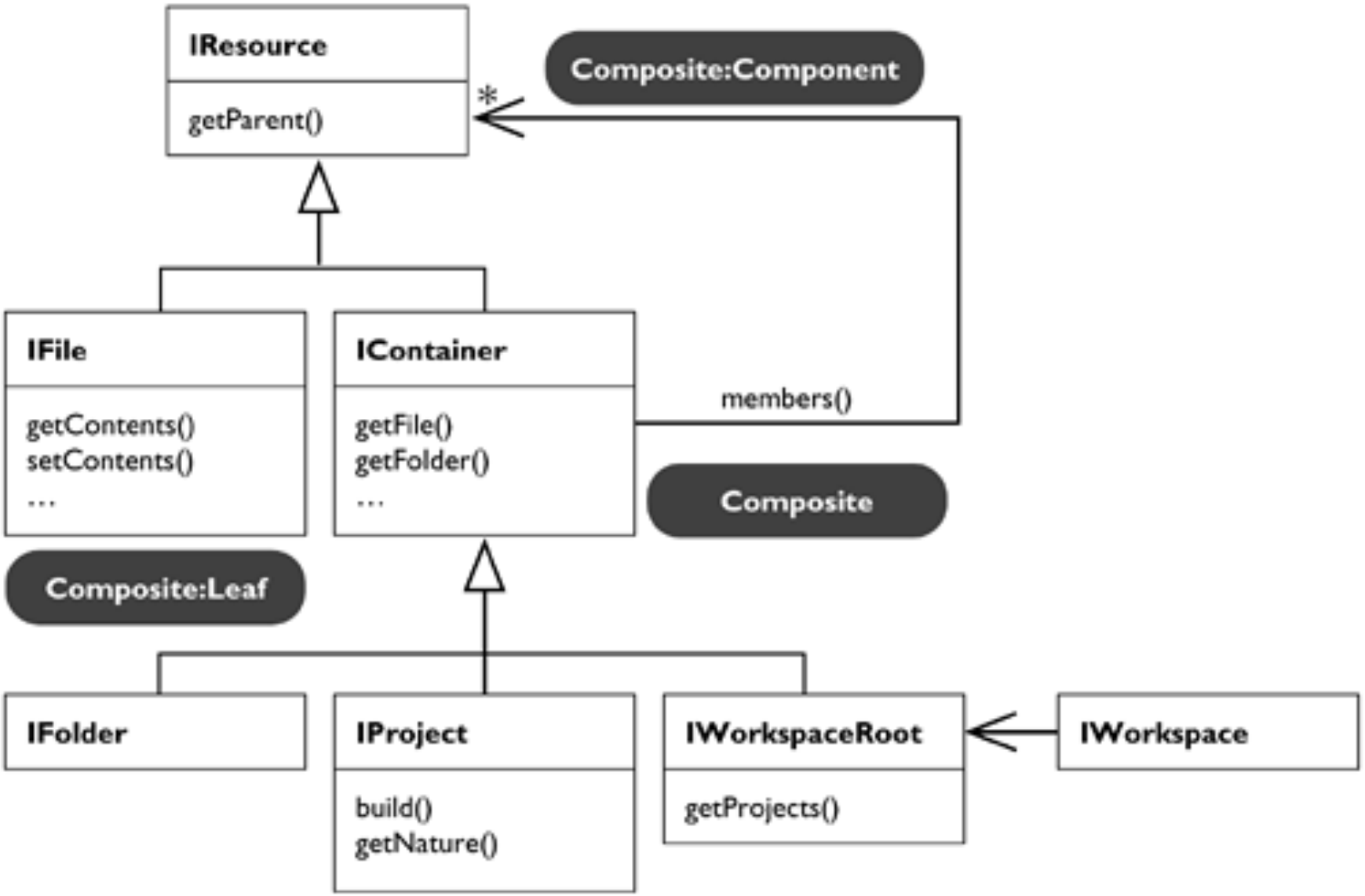
# Workspace hierarchia



# Workspace hierarchia



# Workspace hierarchia – UML diagram

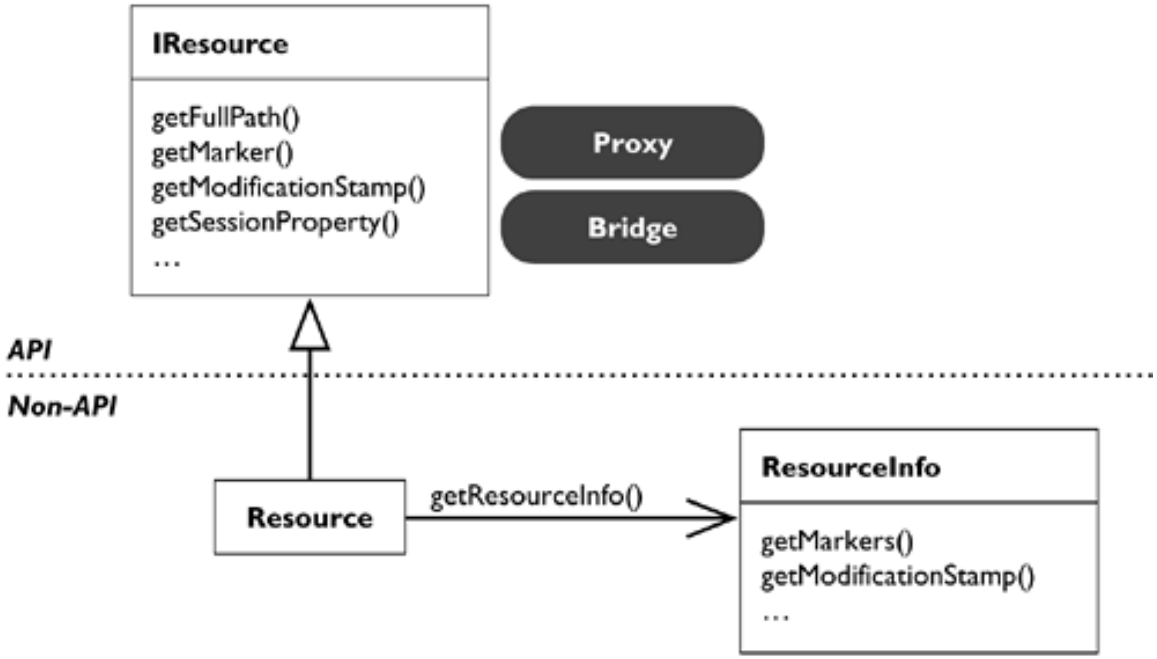


# Erőforrás kijelölés

- IPath implementáció megadásával
  - **Kivétel**: Path osztály példányosítható
  - Cél:
    - Útvonal kijelölése
    - Útvonalműveletek (pl. összefűzés) támogatása
- Értelmezhető
  - Fizikai fájlrendszeren
    - Platformfüggetlen ekkor is!
  - Eclipse fájlrendszeren
    - ~workspace relatív

# Erőforrások

- Handle
  - Mivel a handle állapot nélküli, biztosan nem tárol érvénytelen állapotot a kliensben



# Erőforrások

- ResourceInfo
  - Elemek állapotát tárolja
  - Speciális állapotok: leszármazott osztályok (pl. ProjectInfo)
  - Fa struktúrában a teljes workspace adatai a memóriában
    - “element tree”
  - Az info visszaadása a fa bejárásával történik a handle-ben levő path segítségével
    - Lemezművelet nélkül megtalálhatóak a keresett elemek

# Workspace - composite pattern

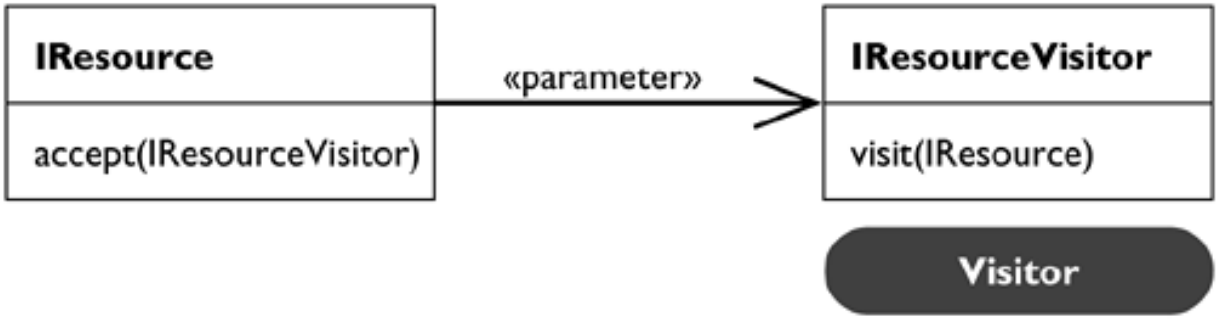
- Az erőforrás fa a composite mintát követi
  - A composite minta lényege:
    - elemek fa struktúrába
    - rész/egész viszony alapján
  - A fa gyökere: IWorkspaceRoot
    - ResourcePlugin.getWorkspace()

# Erőforrás bejárás - Visitor

- Kevesebb kódolással is be lehet járni a fát
- Visitor
  - egy művelet megvalósítása
  - végrehajtás: egy objektumstruktúrán
- Fabejárás elkülönítve a végrehajtástól
  - elég egyszer megvalósítani

# Erőforrás bejárás - Visitor

- Struktúra




- Az accept() metódus valósítja meg a bejárást, és visszahívja a visitor-t minden elemre
- Ha a visit() true-val tér vissza, akkor az elem gyerekeit is be kell járni

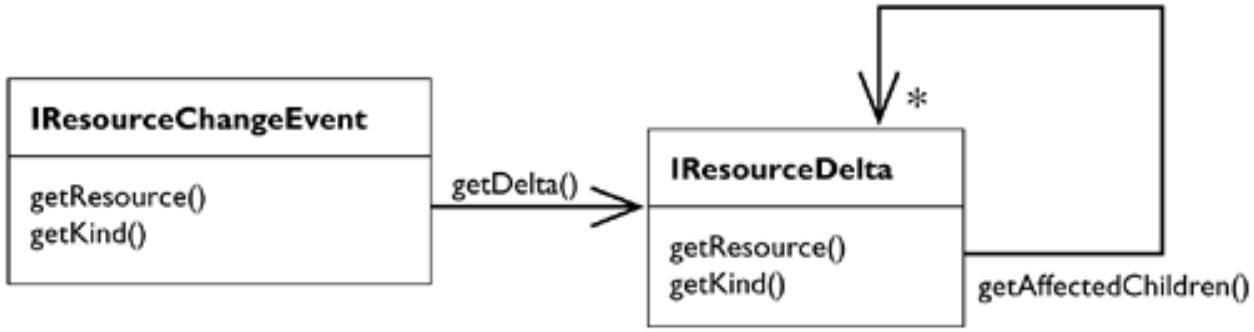
# Erőforrásváltozások követése - Observer

- Az erőforrások változhatnak
  - Eclipse-en belüli változtatások miatt
  - Fájlrendszer szinkronizáció miatt
- Állapotváltozásokat követni kell
  - Hatékony állapotfrissítés
  - Workspace-ben kell regisztrálni

# Erőforrásváltozások követése - Observer

- Kétféle módosítási minta
  - Push – a listener részletes információkat kap a változásokról
  - Pull – a listenernek kell lekérdezni az új állapotot
- Fastruktúra változásainak tárolása
  - Eclipse: ResourceDelta
    - Az erőforrás-fa állapot-változását írja le
    - Szintén fa struktúra
  - ResourceDelta átvihető push és pull módon is

# Erőforrásváltozások követése - Observer




- A resourceDelta az egyszeres és többszörös változásokat ugyanazzal a struktúrával írja le
- Egyszerűen be lehet járni a delta tree-t (ld. visitor minta)
- Mivel az erőforrások csak handle-k, törölt elemekre is hivatkozhatnak

# Kötegelt változtatások

- Minden eseményvezérelt rendszerben veszélyes a túl sok esemény érkezése
- Lehetőség van atomi kompozit akciók létrehozására
  - IWorkspaceRunnable
  - Atomi futtatás
  - Csak egy értesítés keletkezik, a futás végén

# Kötegelt változtatások - példa

```
public static void createMarker(final iResource
  resource, final Map attributes, final String
  markerType) throws CoreException {
  IWorkspaceRunnable r = new IWorkspaceRunnable() {
    public void run(IProgressMonitor monitor) throws
  CoreException {
      IMarker marker =
  resource.createMarker(markerType);
      marker.setAttributes(attributes);
    }
  };
  resource.getWorkspace().run(r, null);
}
```

Az “execute around method” minta alapján

A run() metódus az execute around metódus

# Markerek

- Resource-okat megjelölhetünk
  - Pl. hibás
  - Állapot elmentődik
    - Ha kérjük!
    - Automatikusan
- IMarker interfész csatolása
  - Marker típus regisztrálva

# Források

- John Arthorne: How You've Changed!
  - http://eclipse.org/articles/Article-Resource-deltas/resource-deltas.html
- Dejan Glozic: Mark My Words!
  - http://eclipse.org/articles/Article-Mark%20My%20Words/mark-my-words.html

# Projektek kezelése

# Projekttípusok

- Projektek
  - Példa: Java projekt, Plug-in projekt (Java is!), …
  - Feladatok
    - Azonosító -> plug-in kód vonatkozik-e
    - Konfiguráció -> projekttípus függő
    - Fordítás -> fordítók regisztrációja

# Projekt definíció

- Projekt megadása:
  - Nature kiterjesztési pont
    - org.eclipse.core.resources.natures
    - Absztrakt leíró
    - Hozzácsatolható több szolgáltatás

# Projekt létrehozása – 1.

- IProject létrehozás
  - NE származtassunk új implementációt
  - Helyette

```
try {
    IProject p = ResourcesPlugin.getWorkspace().
            getRoot().getProject();
    p.create(new NullProgressMonitor());
} catch (CoreException e1) {
    //Exception handling
}
```

# Projekt létrehozása – 1.

- IProject létrehozás
  - Létrehozás után állítsuk be a nature-t
    - Ne létrehozás előtt, ld. https://bugs.eclipse.org/bugs/show_bug.cgi?id=323371
- Nature tesztelés
  - IProject.hasNature(natureID)

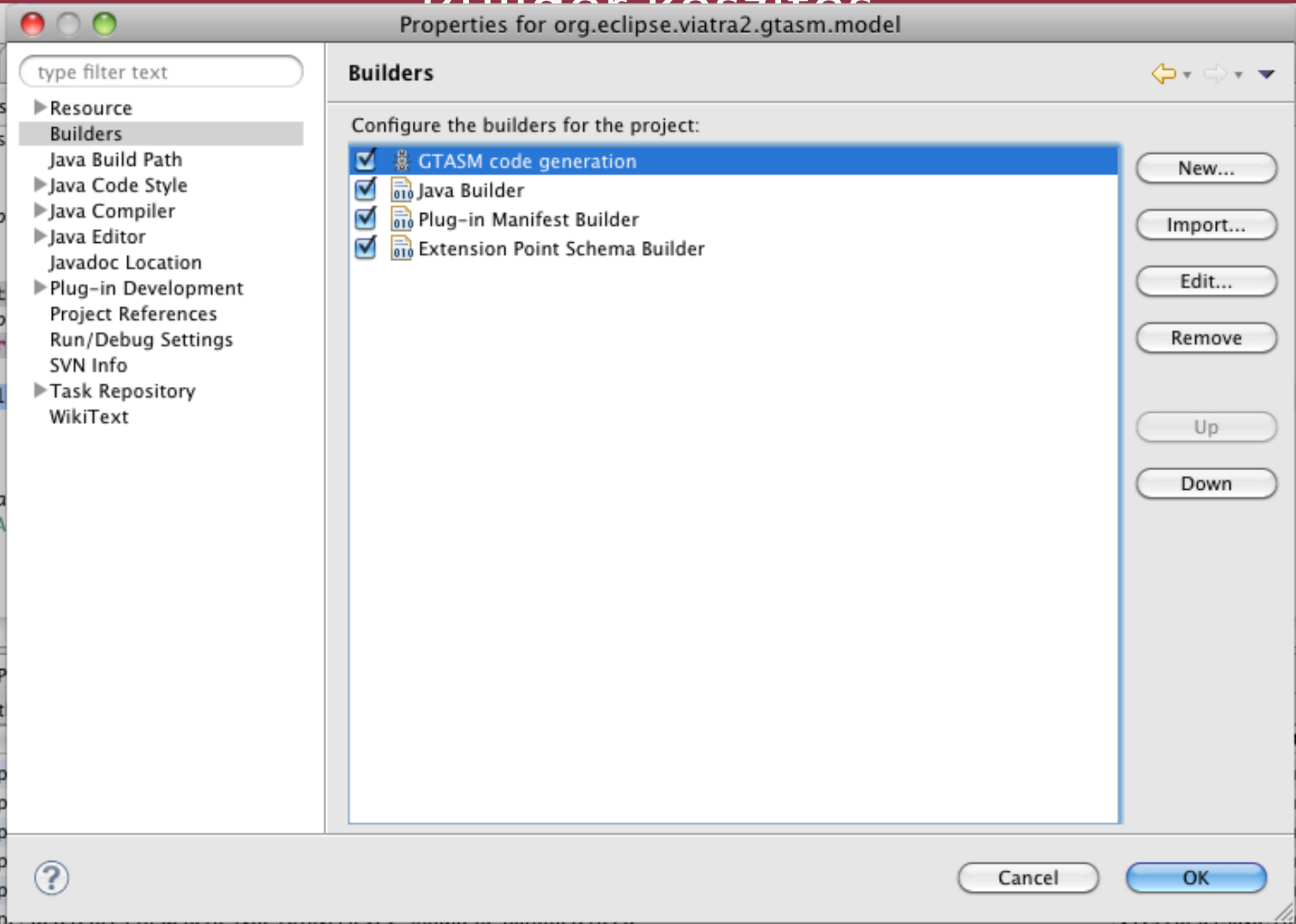
# Fordítás

- Automatikus fordítás
  - Inkrementális fordító
  - Feladattípusok
    - INCREMENTAL
      - Build Project/Build All
    - AUTO
      - Automatikusan triggerelt build
    - FULL
      - Korábbi állapotok eldobása és újrafordítás
    - CLEAN
      - Korábbi állapotok eldobása

# Builder készítés

- Külső eszköz becsatolása
  - Ant build script
  - Batch file/Shell script
  - Külső fordító
  - …

# Builder készítés



# Builder készítés

- Integrált
  - Kiterjesztés: org.eclipse.core.resources.builder
  - IncrementalProjectBuilder absztrakt osztály
- Regisztráció
  - nature-höz kötötten
  - Kódból

# project.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
	<name>org.eclipse.viatra2.gtasm.interpreter</name>
	<comment></comment>
	<projects>
	<project>viatra_gtasm_emf_model</project>
	<project>viatra_pattern_matcher</project>
	</projects>
	<buildSpec>
	<buildCommand>
	<name>org.eclipse.jdt.core.javabuilder</name>
	<arguments>
	</arguments>
	</buildCommand>
	<buildCommand>
```

# project.xml

```
        <name>org.eclipse.pde.ManifestBuilder</name>
        <arguments>
        </arguments>
        </buildCommand>
        <buildCommand>
        <name>org.eclipse.pde.SchemaBuilder</name>
        <arguments>
        </arguments>
        </buildCommand>
        </buildSpec>
        <natures>
        <nature>org.eclipse.pde.PluginNature</nature>
        <nature>org.eclipse.jdt.core.javanature</nature>
        </natures>
</projectDescription>
```

# Források

- John Artorne: Project Builders and Natures
  - http://eclipse.org/articles/Article-Builders/builders.html

# Háttérfolyamatok

Job API

# Háttérfolyamatok

- Hosszú műveleteket háttérszálon célszerű futtatni
  - Ekkor a felhasználói felület nem szaggat
- Eclipse alatt támogatás
  - Job: Egy háttérben elvégzendő feladat
  - Ütemezett végrehajtás
    - Időzítés
    - Ismétlés
  - Állapotjelzés a felhasználói felületen
    - Állapotjelzés
    - Szöveges állapot

# Job

- Plug-in függőség: org.eclipse.core.runtime
- Saját leszármazottat kell készíteni a Job osztályból
- Legfontosabb metódusok:
  - run(IProgressMonitor monitor)
    - A jobhoz tartozó végrehajtható kód – mindig felüldefiniálandó
  - setPriority(int priority)
    - Prioritás beállítása (Job.LONG, Job.SHORT konstansok)
  - setName(String name)
    - Név beállítása (megjelenik a felhasználói felületen)
  - schedule(long delay)
    - Késleltetett időzítés
  - schedule()
    - Azonnali időzítés (amilyen hamar csak lehet)

# Hello, world! Job

```
Job job = new Job("My First Job") {
  protected IStatus
  run(IProgressMonitor monitor) {
      System.out.println("Hello
  World (from a background job)");
      return Status.OK_STATUS;
  } };
job.setPriority(Job.SHORT);
job.schedule(); // start as soon as
  possible
```

# Állapotjelzés

- run() paramétere: IProgressMonitor
  - Interfész a folyamatok nyomon követésére
  - A megvalósító objektum tárolja a folyamat állapotát
- Állapotváltozások
  - Végrehajtandó feladat frissíti
  - Tárolja a megszakítási kérést

# Állapotjelzés: IProgressMonitor

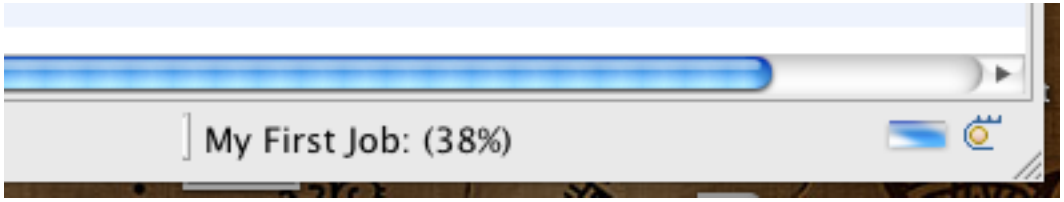
- Feladatok állapotának nyomonkövetése
  - beginTask(String name, int totalWork)
    - Részfolyamat kezdése névvel és elvégzendő lépésszámmal
  - worked(int work)
    - Az elvégzett lépésszám frissítése
  - done()
    - Jelzés, hogy a feladat befejeződött
- Hierarchikus feladatkezelés
  - subTask(String)
    - Jelzés, hogy egy alfolyamat megkezdődött

# Állapotjelzés

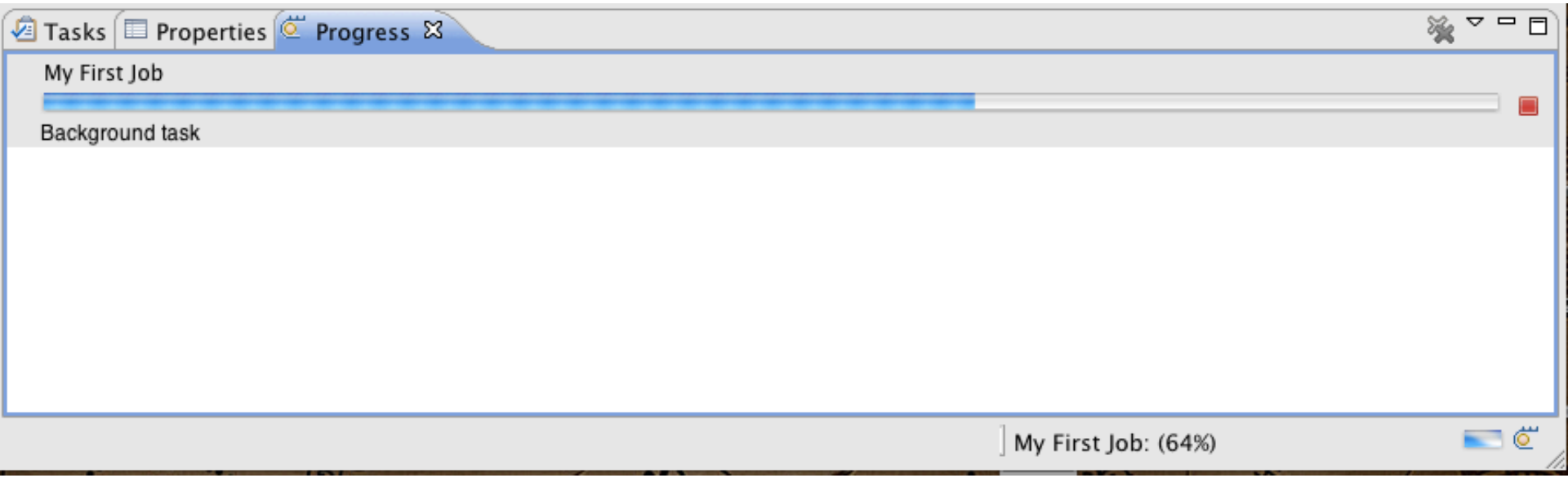
```
Job job = new Job("My First Job") {
  protected IStatus run(IProgressMonitor
  monitor) {
    monitor.beginTask("Background task",
  1000);
    for (int i=0; i<1000; i++){
      calculationStep();
      monitor.worked(1);
    }
    monitor.done();
    return Status.OK_STATUS;
  }
};
```

# Állapotjelzés

- Háromféle megjelenés IDE alatt
  - Státuszsor




  - Állapot nézet (Progress view)



# Állapotjelzés

- Felhasználói folyamat
  - job.setUser(true);
  - Ajánlás: közvetlen felhasználó által kezdeményezett folyamat
    - Modális pop-up
    - Bezárható

# Folyamat megszakítása

- Megszakítás kezdeményezhető
  - Felhasználói felületen
    - Állapot nézetben nyomógombbal
    - Felhasználói folyamat ablakában
  - Kódban
    - A Job `cancel()` metódusát kell meghívni
- Ez csak megszakítási kérés
  - Nem szakítja meg a végrehajtást
  - A kérés figyelése a végrehajtandó kód feladata

# Folyamat megszakítása

- Kód felkészítése megszakítás kezelésére:
  - IProgressMonitor.isCanceled() pollozása
  - Kilépés kétféleképpen
    - `return Status.CANCEL_STATUS;`
    - `OperationCanceledException` dobása
      - Ezt hívási lánc mélyén is lehet

# Folyamat megszakítása - példa

```
Job job = new Job("My First Job") {
  protected IStatus run(IProgressMonitor monitor) {
    monitor.beginTask("Background task", 1000);
    for (int i=0; i<1000; i++){
      calculationStep();
      monitor.worked(1);
      if (monitor.isCancelled())
        return STATUS.CANCEL_STATUS;
    }
    monitor.done();
    return Status.OK_STATUS;
  }
};
```

# Megjelenítő szál visszahívása

- A felhasználói felületet csak a megjelenítő szálon frissítsük
- Megoldás:
  - Display: syncExec(Runnable r)
    - A háttérszál megvárja, amíg a megjelenítő szál végez.
  - Display: asyncExec(Runnable r)
    - A háttérszál nem vár a futtatott parancs elvégzésére

# Feladatütemezés

- A Jobokat a platform ütemezi
  - Prioritás és késleltetés alapján
- Az időzítési kérelemhez késleltetés adható
  - Egy másodperces várakoztatás
    - job.schedule(1000);
  - Ezután a platform ütemezni fogja a végrehajtást
- Ismételt végrehajtás
  - Egy Job csak egyszer várakozhat az ütemezésre
  - Ismétlés: befejezéskor schedule() újrahívása
  - Így lehet periodikusan ismétlődő háttérfolyamatot definiálni

# Egyéb szolgáltatások

- Alfolyamatok létrehozása
- Ütemezi szabályok hozzáadása
- Erőforrásfoglalás
  - Holtpontdetekció

# Job API - Összefoglalás

- Magas szintű támogatás többszálú futáshoz
  - Állapotjelzés a felhasználói felületen
  - Ütemezés
- Kényelmes többszálú fejlesztés

# Források

- Michael Valenta: On the Job: The Eclipse Jobs API
  - http://eclipse.org/articles/Article-Concurrency/jobs-api.html

# Varázslók

# Erőforrás varázslók

- New wizard
  - Varázsló új erőforrás létrehozásához
- Import wizard
  - Külső tartalom importálásáa
- Export wizard
  - Erőforrások exportálása

# Erőforrás varázslók

- JFace varázslók
  - Wizard, WizardPage osztályok
- Sok újrahasznosítható komponens
  - Fájl/mappa választás
  - Projekt tulajdonságok
  - …
- Regisztráció
  - Kiterjesztési pontok segítségével

# New wizard

- Kiterjesztési pont: *org.eclipse.ui.newWizards*
- Fontosabb tulajdonságok
  - Ikon
  - Kategória
  - Típus:
    - Projekt
    - Example
    - Egyéb
  - Perspektívák
    - Mire váltsunk a létrehozás után

# Import/export

- Import
  - Kiterjesztési pont: org.eclipse.ui.importWizards
- Export
  - Kiterjesztési pont: org.eclipse.ui.exportWizards

# Varázslók és kijelölés

# Varázslók és kijelölés

- A varázslók megkapják az aktuális kijelölést
  - Miért?
- Előre kitölthetőek bizonyos rovatok
  - Pl. forrás vagy célmappa
  - Tegyük meg mindig!

# Java erőforrás modell

Java modellek

A Java modellek kezelése

# Erőforrások és Java elemek

- Összefüggések fájlstruktúra és Java modell között

| Java modell | Erőforrásstruktúra |
|---|---|
| Csomagok | Könyvtárak |
| Osztályok | Fájlok |

- Java-centrikus felület
  - Erőforrás rendszer bővítése adapter mintával
    - Java Objektum modell – program struktúra, metódusokig
    - Absztrakt Szintaxisfa – Java osztály utasításszintű felbontása

# Java Objektum Model

- Osztályok
  - IJavaModel - Az összes workspace projekt konténere (gyökér)
  - IJavaProject - Projekt, tartalmazza a csomagokat és classpath elemeket
  - IPackageFragmentRoot - Java könyvtár (jar, zip)
  - IPackageFragment - Egy csomag (vagy egy része)

# Java Objektummodell

- További osztályok
  - ICompilationUnit – Egy .java forrásfájl
  - IPackageDeclaration – A package deklaráció a java fájlban
  - IImportContainer – A java fájl importjai
  - IImportDeclaration – Egyetlen import deklaráció
  - IType – Egy típusdefiníció (forrás v. class fájlban) – osztály, interfész, annotáció, enum
  - IField – Egy osztály egy attribútuma
  - IMethod – Egy osztály egy metódusa
  - IInitializer – Egy osztály vagy példány szintű inicializáló blokk
  - IClassFile – Egy bináris (.class) fájlt reprezentál

# Java Objektum Modell

- WorkingCopy – helyi másolat amin dolgzunk
  - Primary: minden plugin által közösen használt
    - commitWorkingCopy() – változások érvényesítése
    - discardWorkingCopy() – változások eldobása
  - Saját másolat – legyen csak a miénk!
    - getWorkingCopy(owner) – az owner hatáskörébe utalja az új másolatot
    - commitWorkingCopy() – visszamenti az eredetibe a változásokat
    - A beépített refactoring pl. ezt használja, hogy tesztelhesse a változásokat

# Java Objektum Modell

- Események
  - IJavaElementDelta – az elem változott
    - ADDED – hozzáadva
    - REMOVED – törölve
    - CHANGED - változás
      - F_CHILDREN – a gyerek változtak
      - F_MODIFIERS – a típusmódosítók változtak
      - F_CONTENT – a tartalma változott
      - …
- Eseménykezelés lehetséges

# Absztrakt szintaxisfa

- Absztrakt szintaxisfa (Abstract Syntax Tree, AST)
  - Utasítás-szintig lebontja a forráskódot
  - Cél: elemi forráskód analízis és manipuláció
  - Miért nem ilyen a Java Object Model?
    - Erőforrás-takarékosság
    - Sebesség
  - Egy tetszőleges Java elem kibontható (osztály, metódus…)
  - A referenciák (pl. típus, metódus, vagy mező elérésére) nincsenek teljesen kifejtve
    - Sebesség-növekedés
    - Igény szerint kifejthető: binding.getJavaElement()
  - Használat: visitor pattern

# JDT: AST példa

```
ASTNode root = parser.createAST(null);
root.accept(new ASTVisitor() {
     public boolean visit(CastExpression ce)
     {
          fCastCount++;
     }
});
```

# JDT: AST példa

AST generálás

```
ASTNode root = parser.createAST(null);
root.accept(new ASTVisitor() {
     public boolean visit(CastExpression ce)
     {
          fCastCount++;
     }
});
```

# JDT: AST példa

```
ASTNode root = parser.crea
root.accept(new ASTVisitor() {
     public boolean visit(CastExpression ce)
     {
          fCastCount++;
     }
});
```

Visitor megadása

# JDT: AST példa

```
ASTNode root = parser.createAST(nu
root.accept(new ASTVisitor() {
     public boolean visit(CastExpression ce)
     {
          fCastCount++;
     }
});
```

Class Cast kifejezések figyelése

# JDT: AST példa

```
ASTNode root = parser.createAST(null);
root.accept(new ASTVisitor() {
     public boolean visit(CastExpression ce)
     {
          fCastCount++;
     }
});
```

Számláló növelése

# JDT: Abstract Syntax Tree

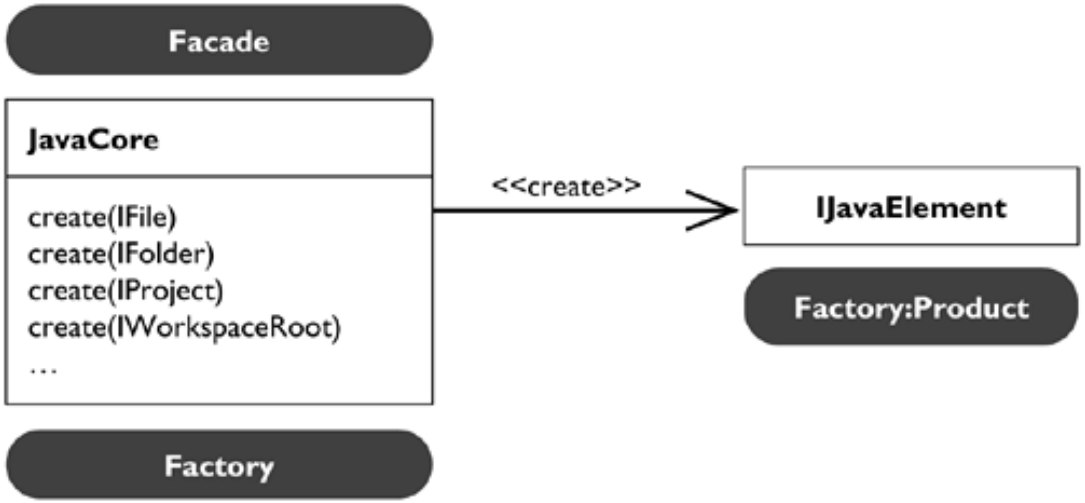
- AST módosítás
  - A forráskód módosítása helyett az AST-n dolgozzunk
  - Leíró megközelítés
    - Változtatások leírása végrehajtás nélkül
    - Több műveletben használható marad az AST
    - Lehetőség van preview készítésére is
  - Módosító megközelítés
    - Közvetlen AST módosítás
    - Az előző megoldásra alapuló implementáció
  - Újraíró megközelítés
    - Őrizzük meg a felhasználói formázást és jelöléseket
    - Generáljunk szerkesztő szkriptet a módosításhoz

# Erőforrások -> Java elemek - Adapter

- IJavaElement.getCorrespondingResource() – a mögöttes erőforrás – nem mindig létezik!
  - Vannak Java elemek, melyek a workspace-en kívül vannak, nincs erőforrás megfelelőjük
  - Java elemek, melyek metódusokat reprezentálnak

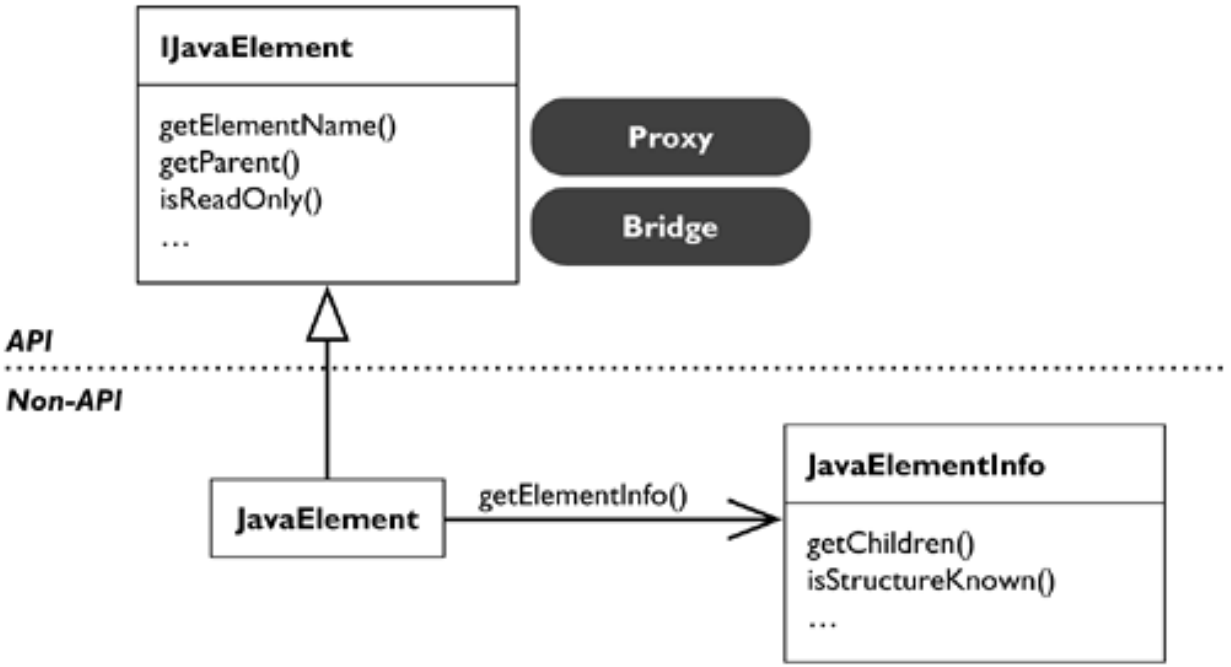
# Erőforrások -> Java elemek - Adapter

- Az IJavaElement lehetővé teszi a mögöttes erőforrások elérését
- Szükség lehet a másik irányra is!
  - Facade: JavaCore – factory metódusok erőforrásokból Java elem létrehozására -> ezek csak handle-k



# Java elemek - (Virtuális) proxy

- Erőforrásokhoz hasonló struktúra

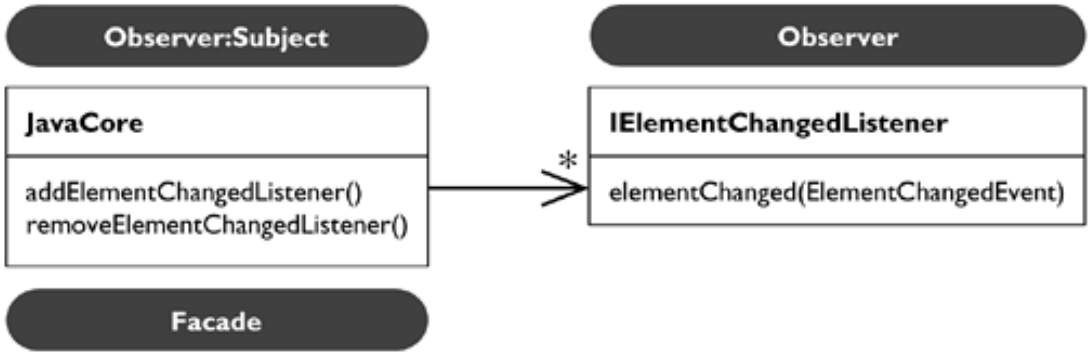


# A Java modell bejárása

- Nincs olyan Visitor megoldás, mint az erőforrásoknál
  - A Java fa lusta módszerrel épül fel
    - Ha már felépült az AST egy része, az már bejárható visitorral
  - Ezért egy teljes bejárás költséges lenne
    - A fordítási egységeket értelmezni kell…
  - Inkább egyedi keresést alkalmazzunk

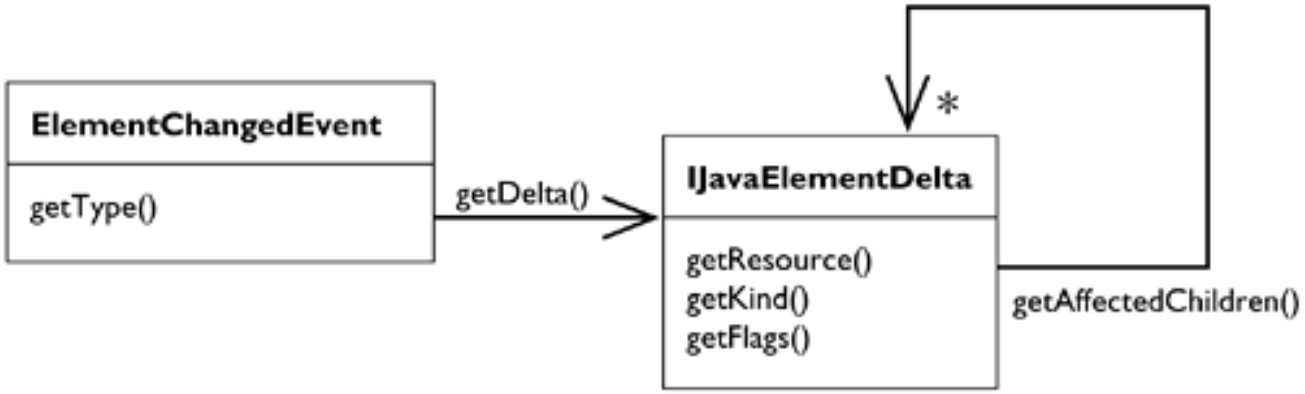
# Java modell - Változásfigyelés

- Hasonló az erőforrásoknál megszokotthoz



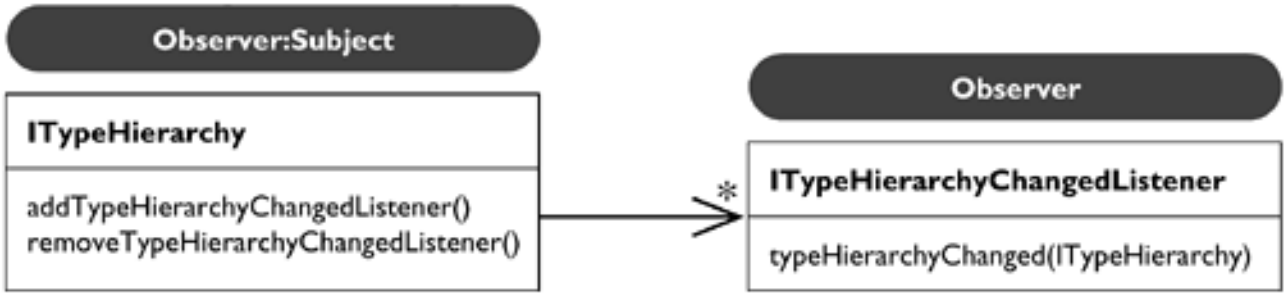
# Java modell - Változásfigyelés

- Az erőforrás deltákkal ellentétben nem mindig a gyökérből indul
- A delta csak az eseménykezelés alatt érvényes (erőforrás-felszabadítás)
- ExecuteAround: JavaCore.run()



# Java modell - Változásfigyelés

- Típushierarchia nem része a fának
- Külön változásfigyelés hozzá



# Összefoglalás

# Tartalomjegyzék

- Grafikus felület összekapcsolása
  - Workbench Selection Service
  - Adapterek használata (Properties nézet)
- Erőforráskezelés
  - Fájlrendszer
  - Projektkezelés
  - Háttérfolyamatok
  - Esettanulmány: Java fejlesztőeszköz

# Mottó

- Any problem in computer science can be solved with another level of indirection.
- … But that usually will create another problem.

David Wheeler