Building Secure OSGi Applications

Karl Pauls
Marcel Offermans

luminis

eclipseCON™ 2008

Who are we?

Who are we?

- Karl Pauls

Who are we?

- Karl Pauls
- Marcel Offermans

Who are we?

- Luminis

- Karl Pauls
- Marcel Offermans

Who are we?

- Luminis

- Karl Pauls
- Marcel Offermans

Who are we?

- Luminis
- Karl Pauls
- Marcel Offermans

Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

Preparation...

- Copy from the memory stick:
  - the ZIP file if you want to use VMware;
  - the folder with the project files if not.
- Alternatively, you can download the folder from: https://opensource.luminis.net/confluence/x/AYAq (13 MB)

Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

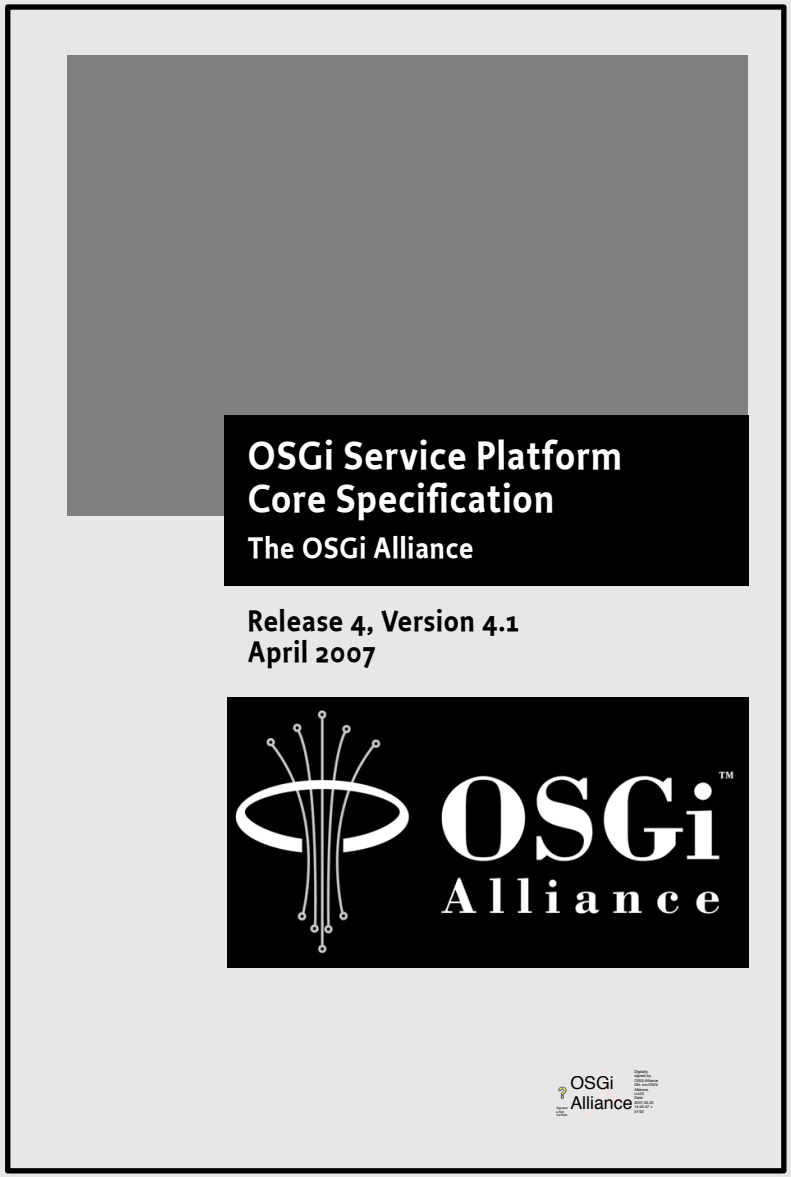
OSGi today

OSGi technology is the dynamic module system for Java™

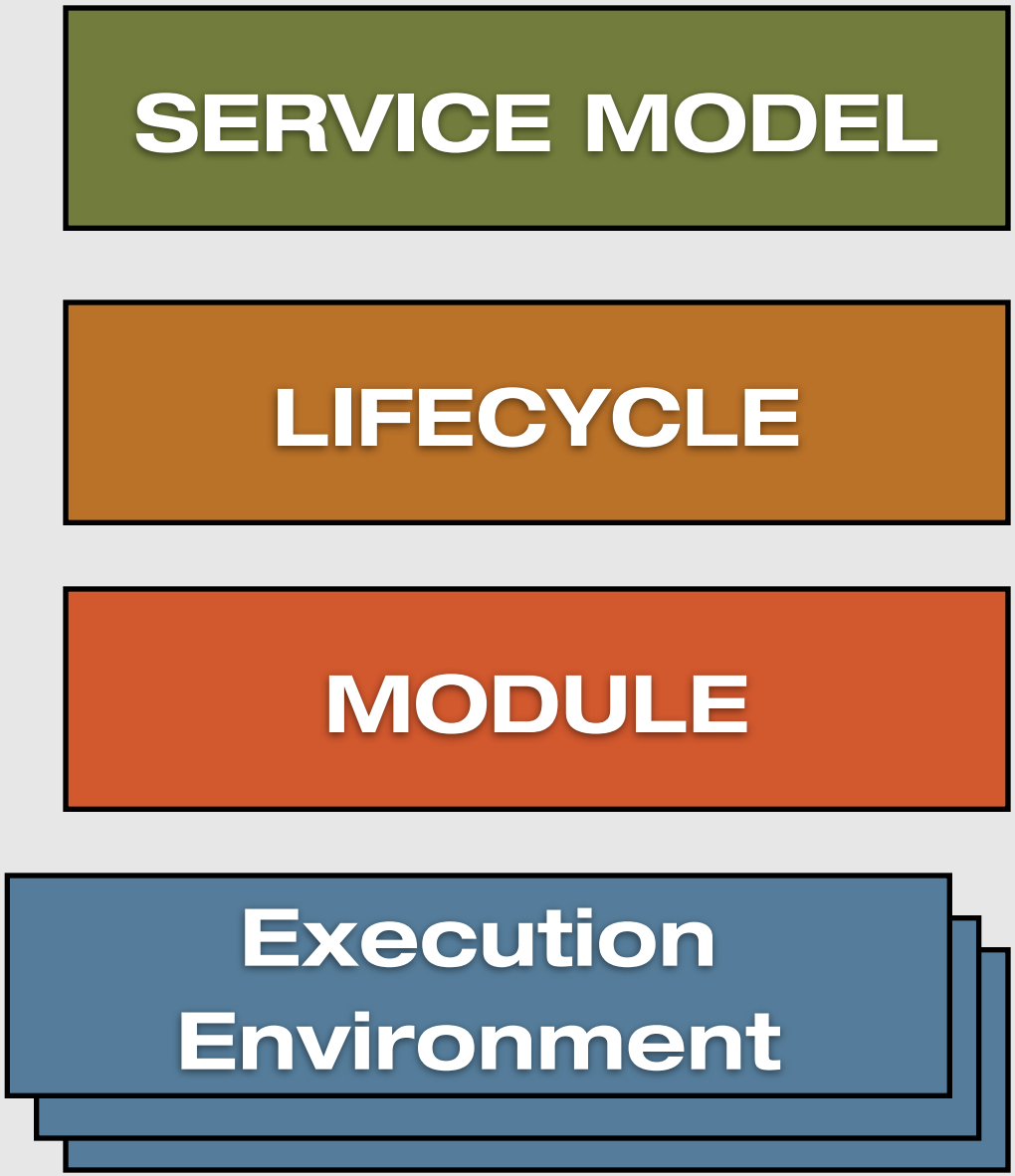
OSGi technology is Universal Middleware.

OSGi technology provides a service-oriented, component-based environment for developers and offers standardized ways to manage the software lifecycle. These capabilities greatly increase the value of a wide range of computers and devices that use the Java™ platform.

OSGi Specification



OSGi Framework Layering



L3 - Provides a publish/find/bind service model to decouple bundles

L2 - Manages the life cycle of a bundle in a framework without requiring the vm to be restarted

L1 - Creates the concept of a module (aka. bundles) that use classes from each other in a controlled way according to system and bundle constraints

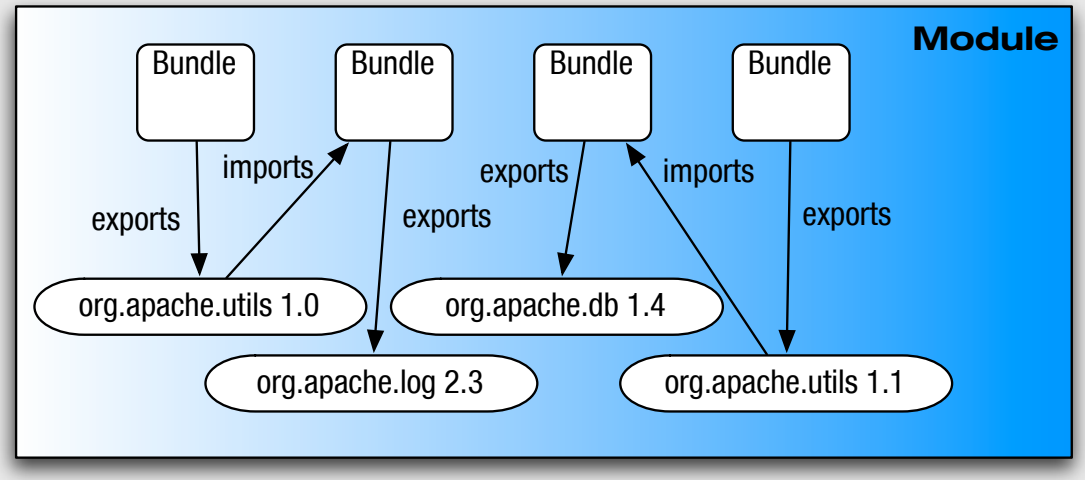
L0 -

OSGi Minimum Execution Environment
CDC/Foundation
JavaSE

Module Layer (1/3)

- Unit of deployment
  is the bundle i.e., a JAR
- Separate class loader
  per bundle
  - Class loader graph
  - Independent namespaces
  - Class sharing at the Java package level

Module Layer (1/3)

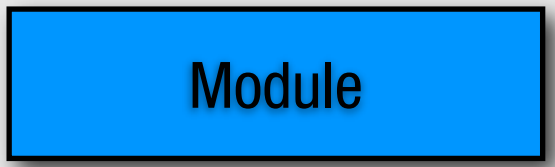


- Unit of deployment is the bundle i.e., a JAR
- Separate class loader per bundle
  - Class loader graph
  - Independent namespaces
  - Class sharing at the Java package level

Module

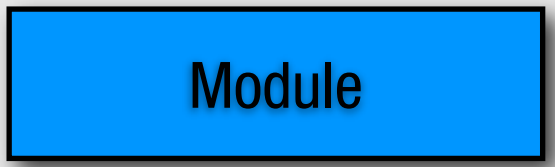
Module Layer (2/3)

- Multi-version support
  - i.e., side-by-side versions
- Explicit code boundaries and dependencies
  - i.e., package imports and exports
- Support for various sharing policies
  - i.e., arbitrary version range support
- Arbitrary export/import attributes
  - Influence package selection



Module Layer (3/3)

- Sophisticated class space consistency model
  - Ensures code constraints are not violated
- Package filtering for fine-grained class visibility
  - Exporters may include/exclude specific classes from exported package
- Bundle fragments
  - A single logical module in multiple physical bundles
- Bundle dependencies
  - Allows for tight coupling when required

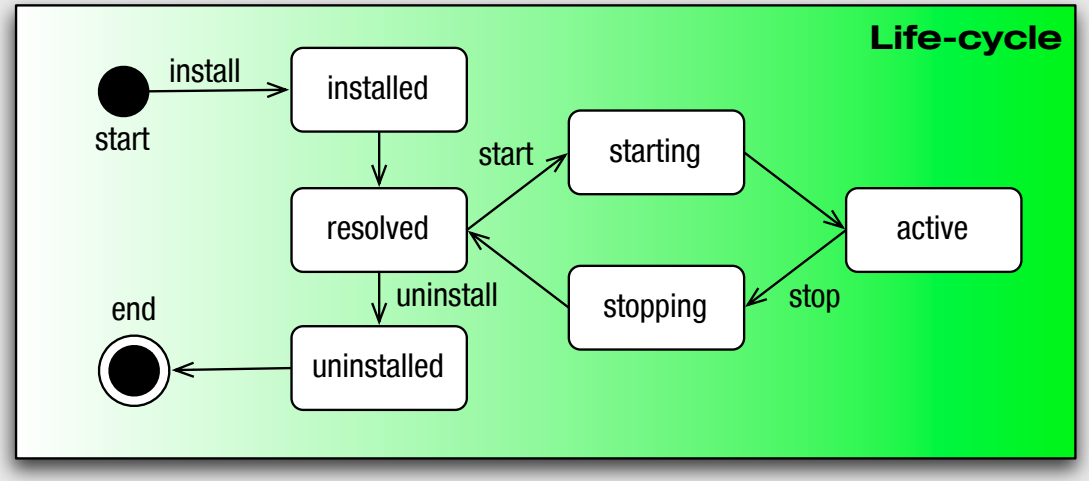


Life-cycle Layer

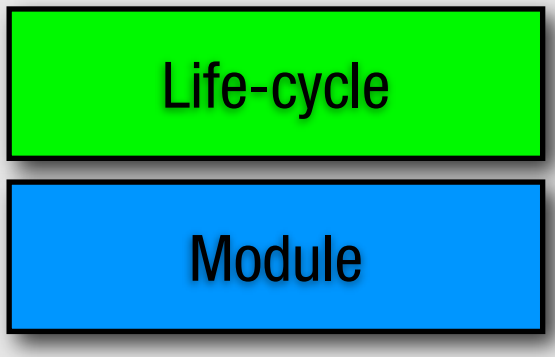
- Managed life cycle
  - States for each bundle;
- Allows updates of existing bundles.
  - Dynamically install, start, update, and uninstall

Module

Life-cycle Layer

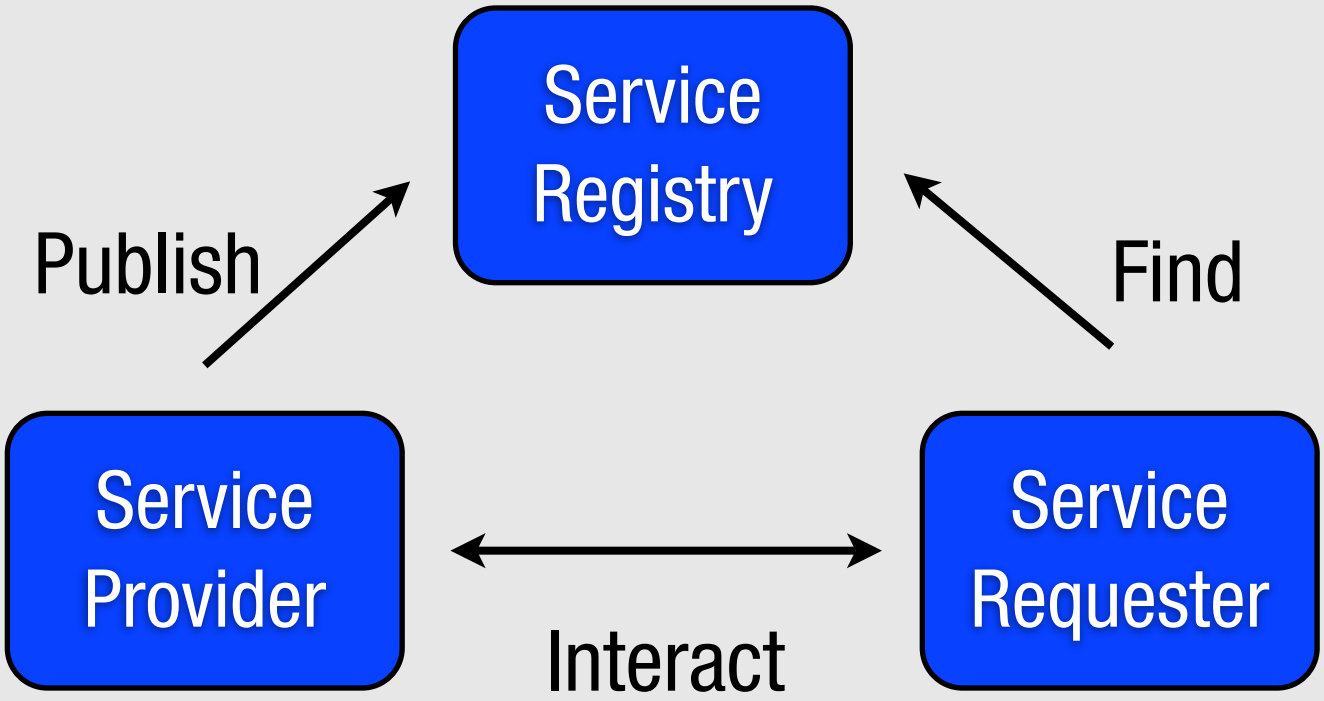


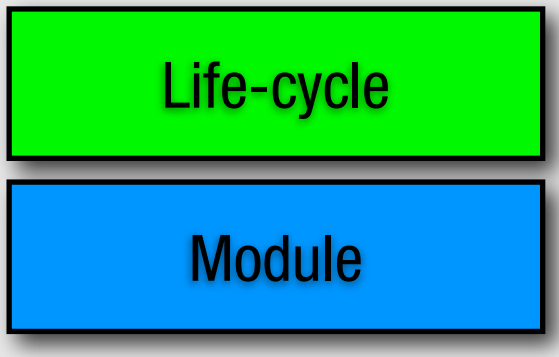
- Managed life cycle
  - States for each bundle;
- Allows updates of existing bundles.
  - Dynamically install, start, update, and uninstall



Service Layer

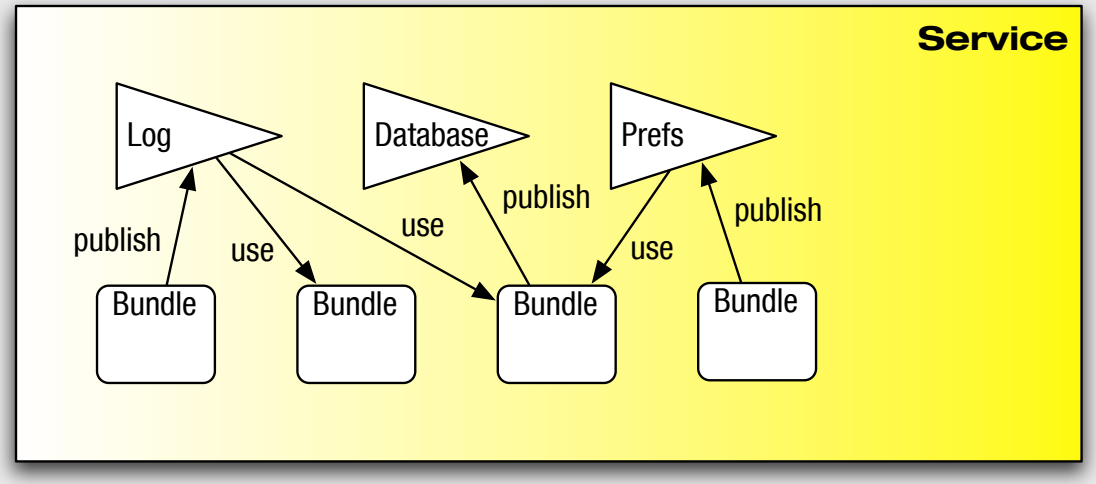
- OSGi framework promotes service oriented interaction pattern among bundles

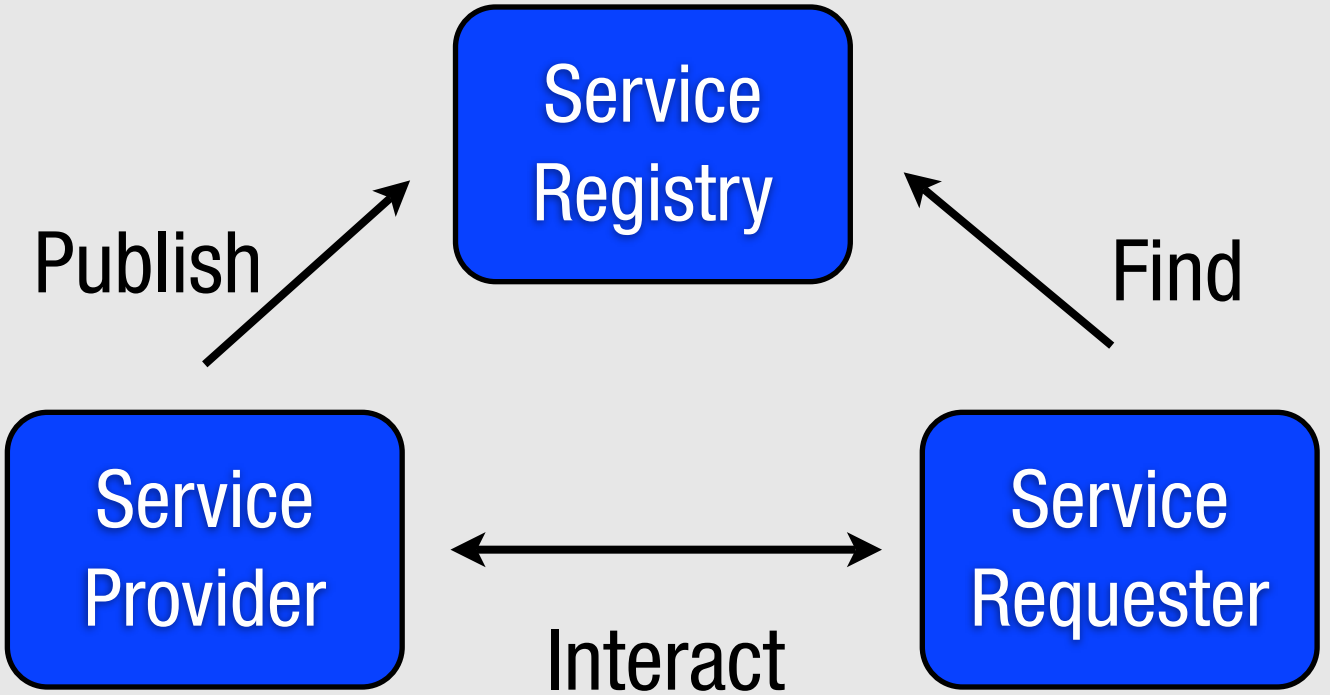


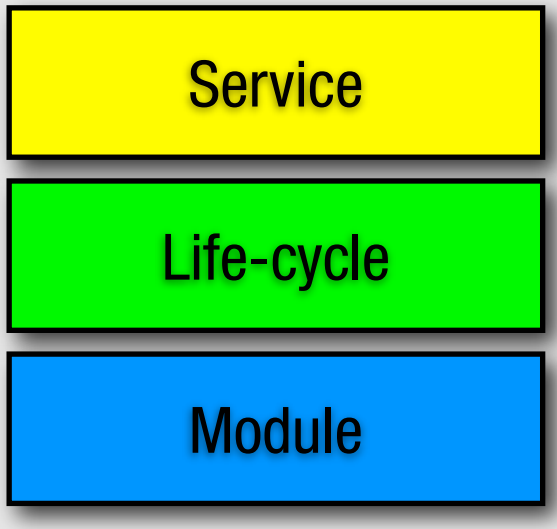


Service Layer

- OSGi framework promotes service oriented interaction pattern among bundles

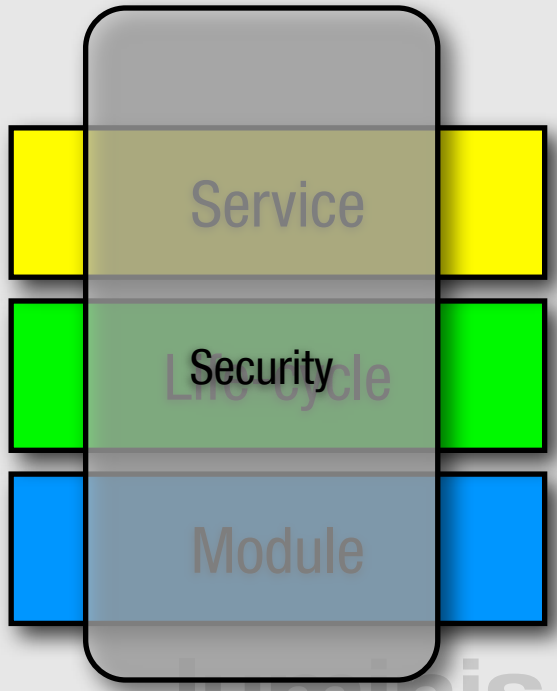






Security

- Optional Security Layer based on Java permissions
- Infrastructure to define, deploy, and manage fine-grained application permissions
- Well defined API to manage permissions
- Code authenticated by location or signer



Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

Security Concepts Overview

- OSGi uses codebased security following the Java Security Model
  - Makes use of Protection Domain
  - The stack walk based Permission Check
  - Signed bundles
- User based security is supported by the UserAdmin service but not integrated in the standard permission check as with JAAS
- Additionally, PermissionAdmin and ConditionalPermissionAdmin provide sophisticated management infrastructure

Protection Domain

- Encapsulates characteristics of a domain
  - One protection domain per bundle
- Encloses a set of classes whose instances are granted a set of permissions
  - Set of permissions associated with each bundle
- Permission check consults all protection domains on the stack

Permission Check

- Invoked either by call to SecurityManager.check* or AccessController.checkPermission
  - SecurityManager is old way to do it
  - OSGi requires usage of the SecurityManager for full functionality
- Privileged calls used to cut off stack walk
  - Disregard code on the stack earlier then the latest privileged call.
- Merges context of parent thread as well

Algorithm

AccessController.checkPermission(Permission p)

Algorithm

AccessController.checkPermission(Permission p)

E.class

Algorithm

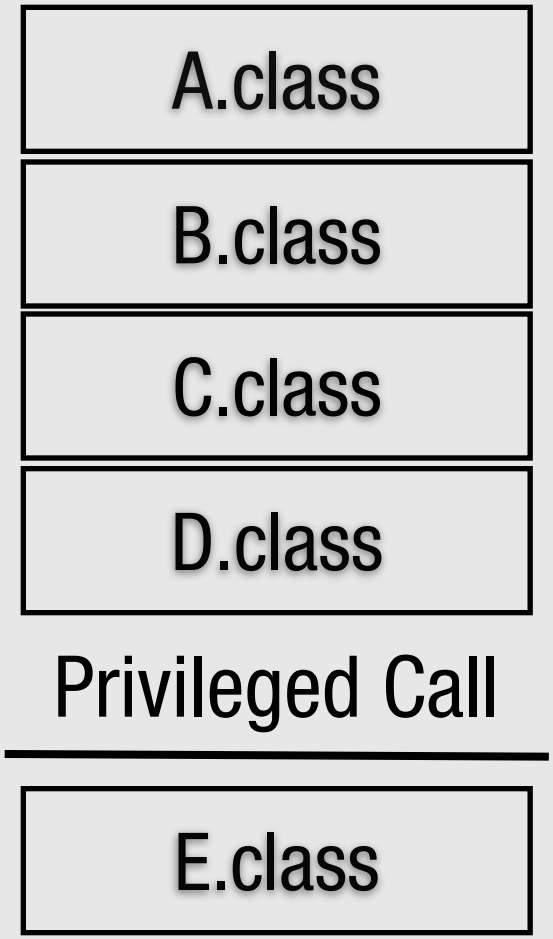
AccessController.checkPermission(Permission p)

Privileged Call

E.class

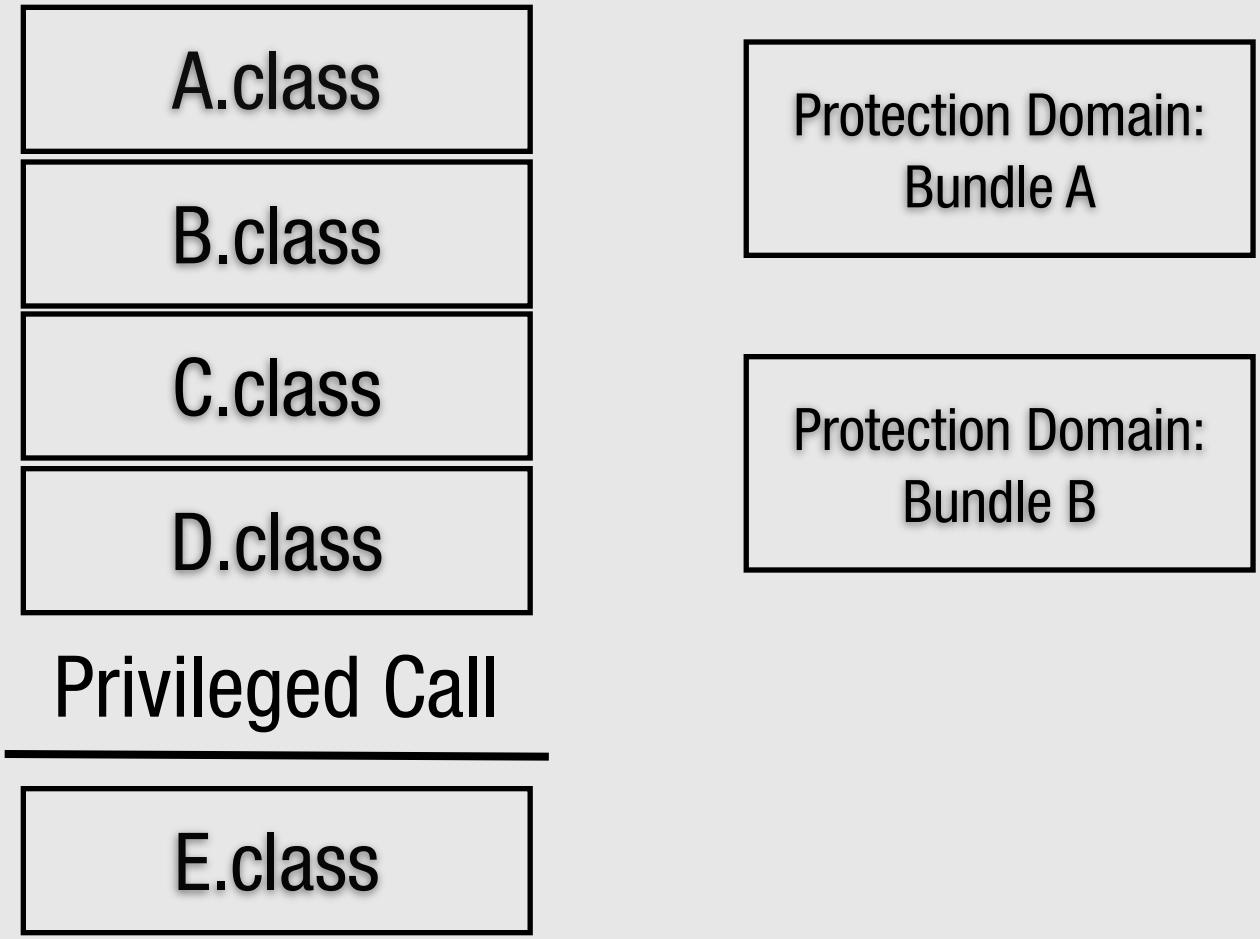
Algorithm

AccessController.checkPermission(Permission p)



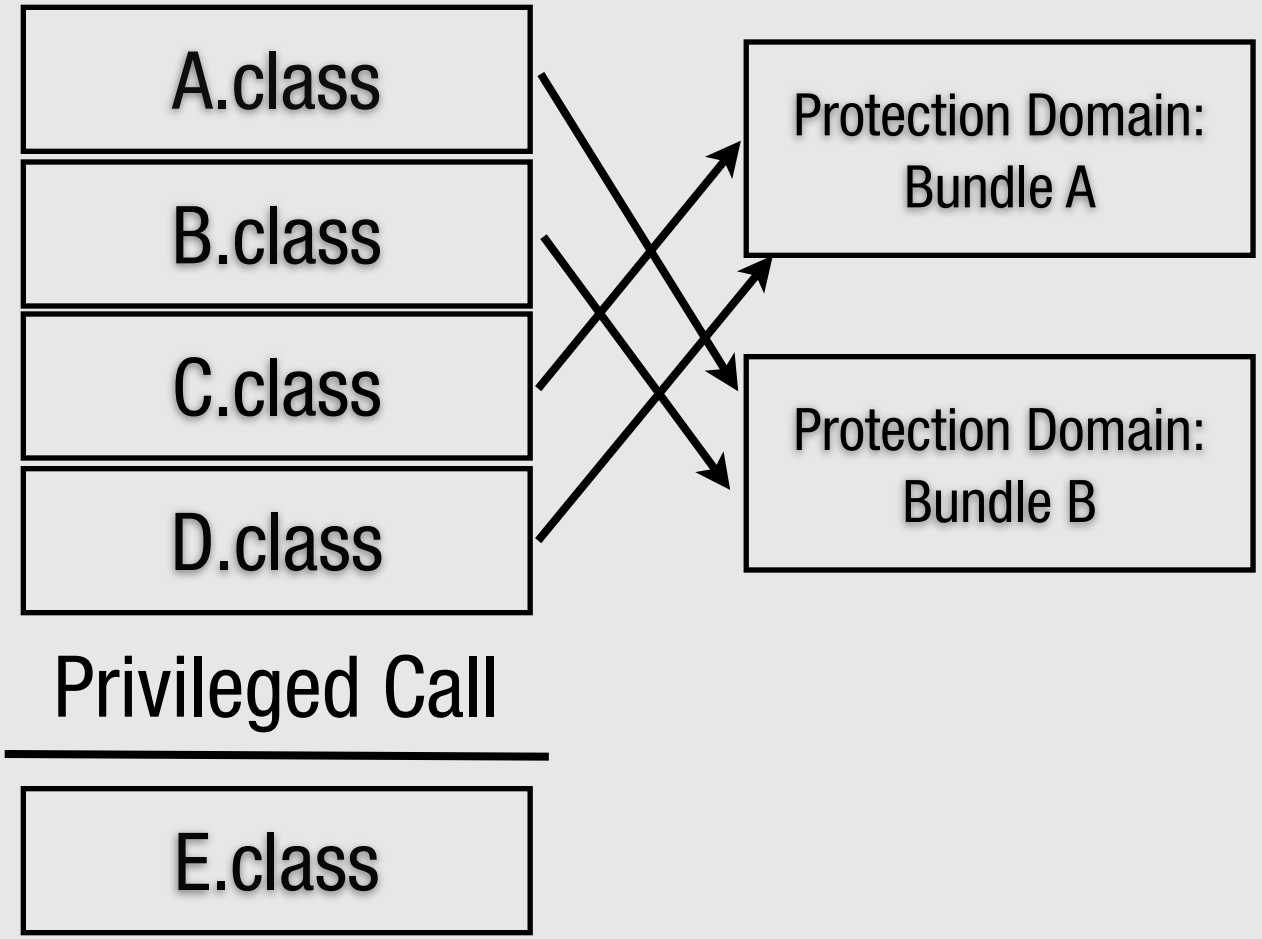
Algorithm

AccessController.checkPermission(Permission p)



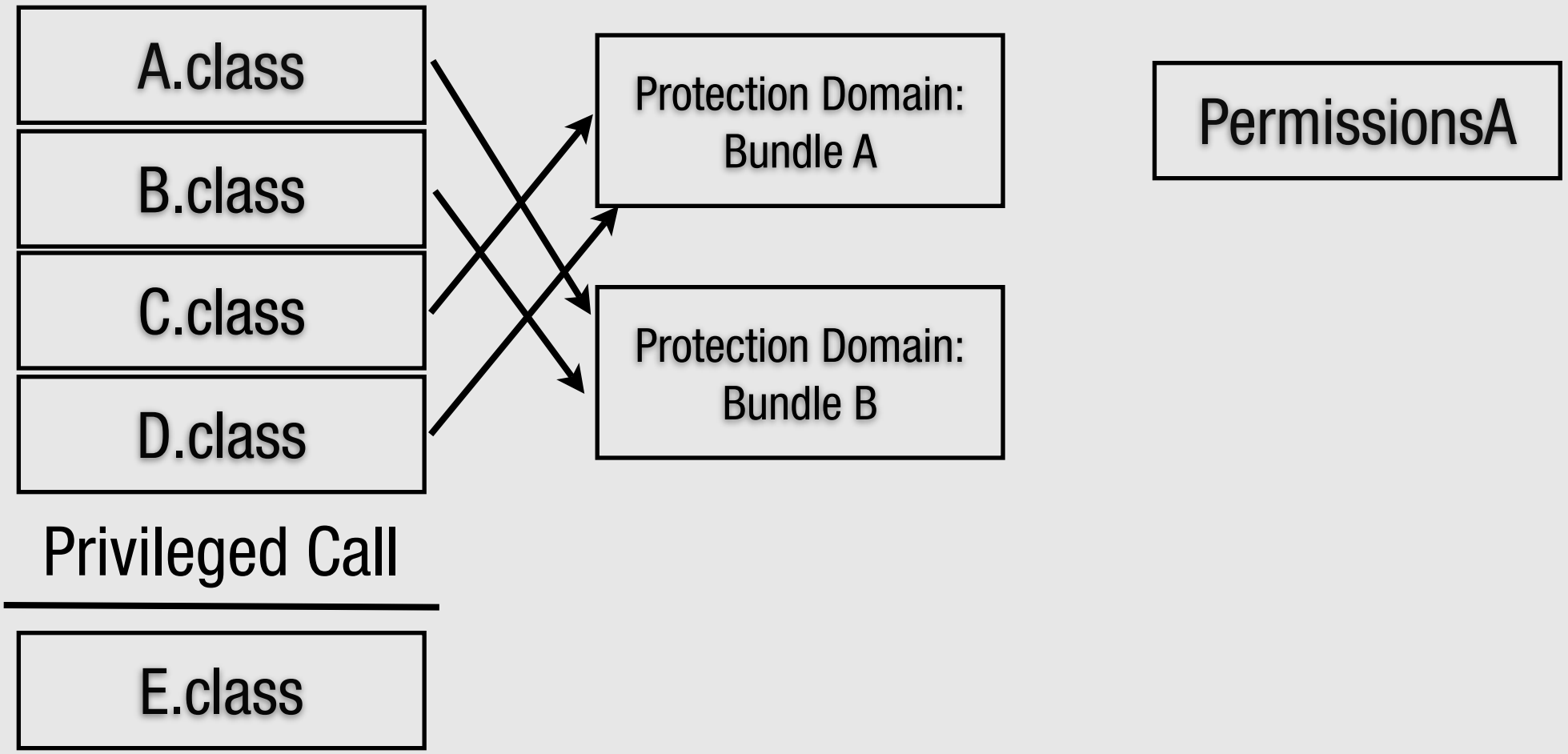
Algorithm

AccessController.checkPermission(Permission p)



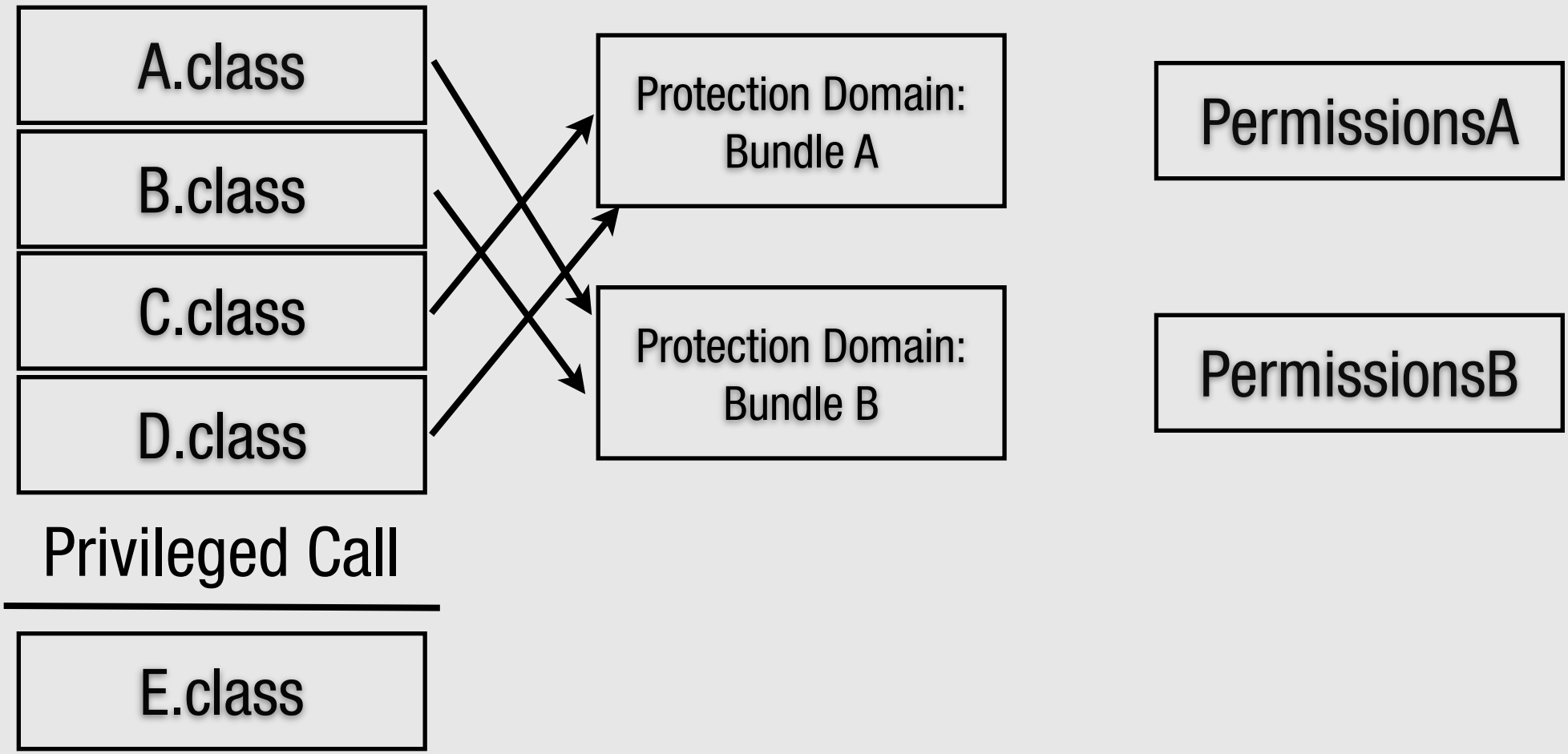
Algorithm

AccessController.checkPermission(Permission p)



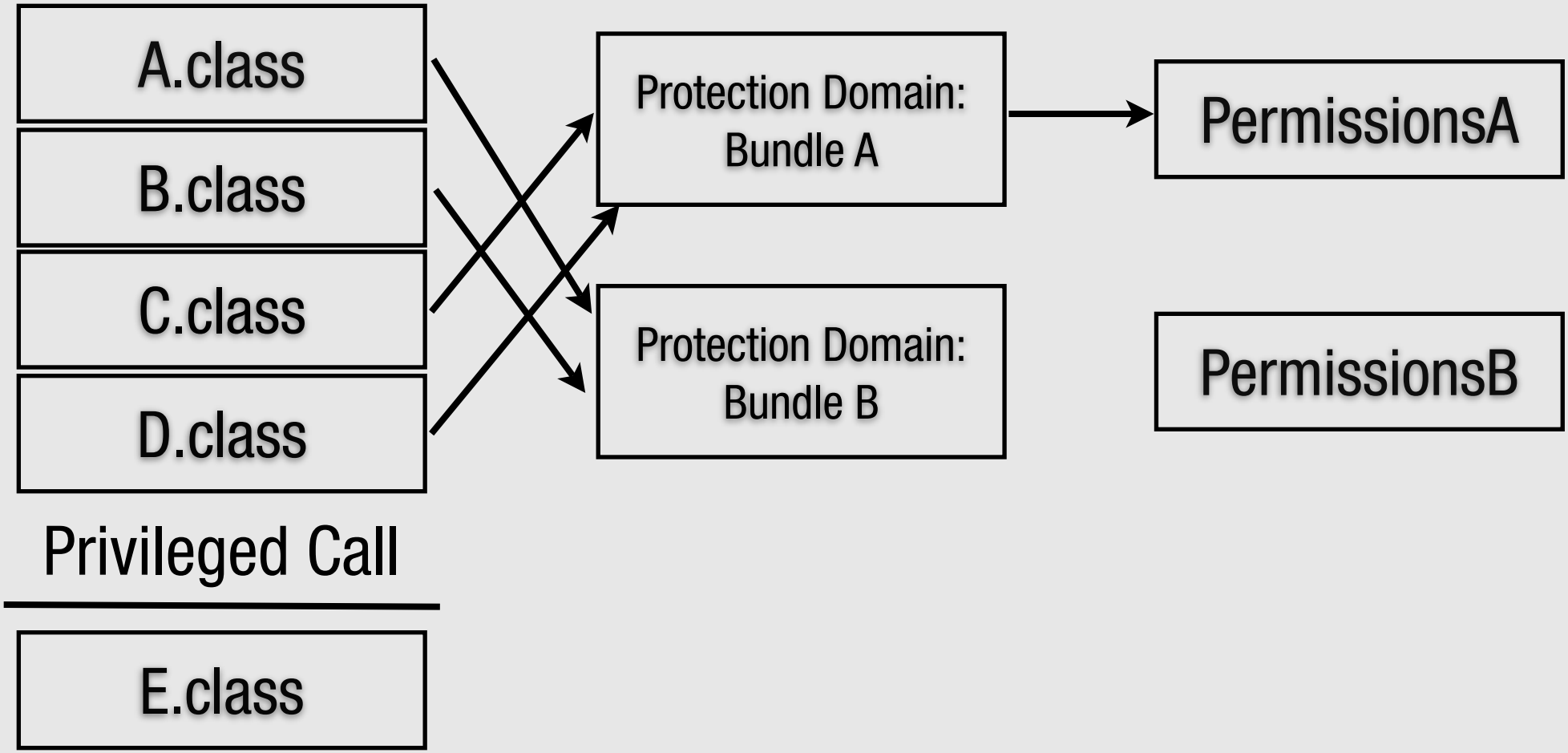
Algorithm

AccessController.checkPermission(Permission p)



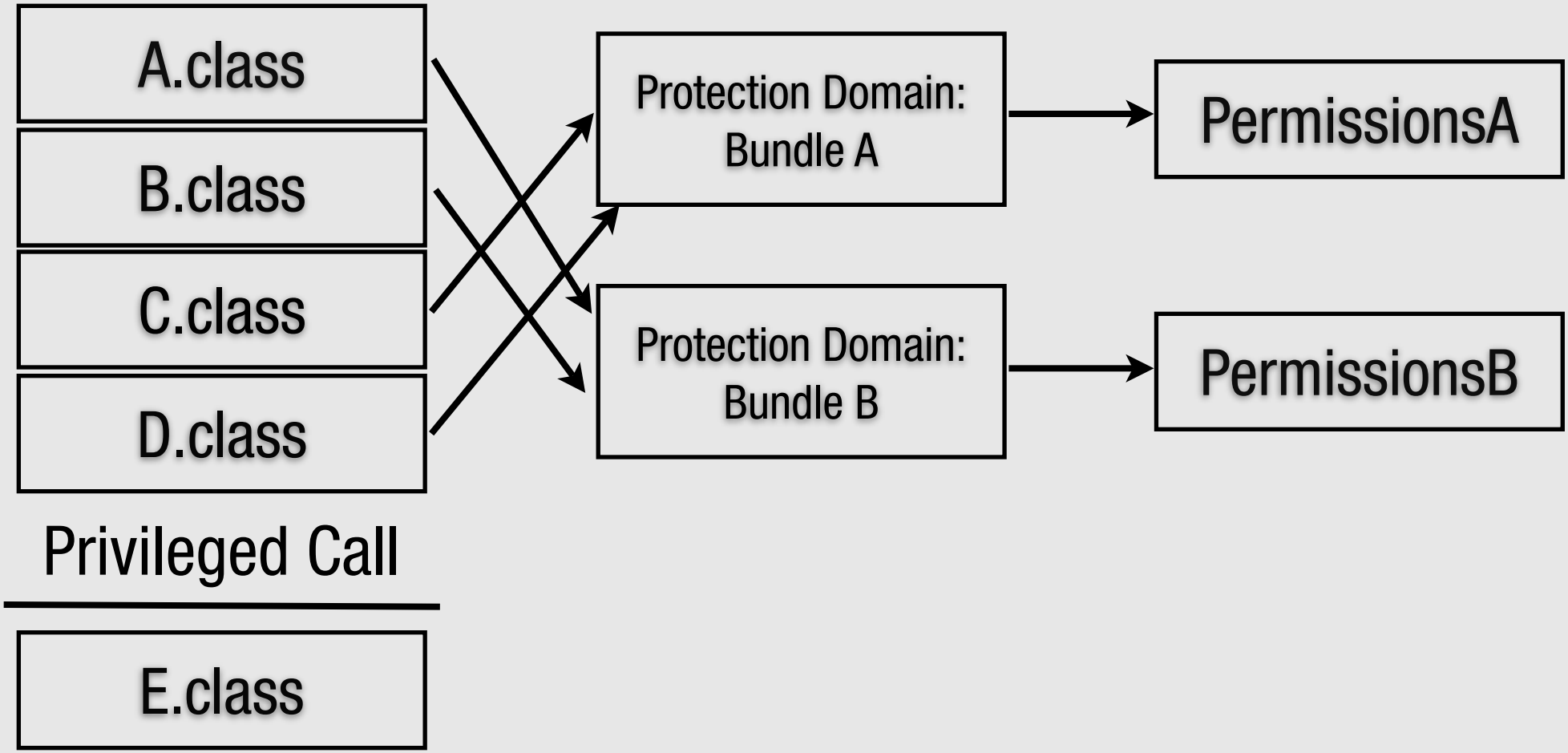
Algorithm

AccessController.checkPermission(Permission p)



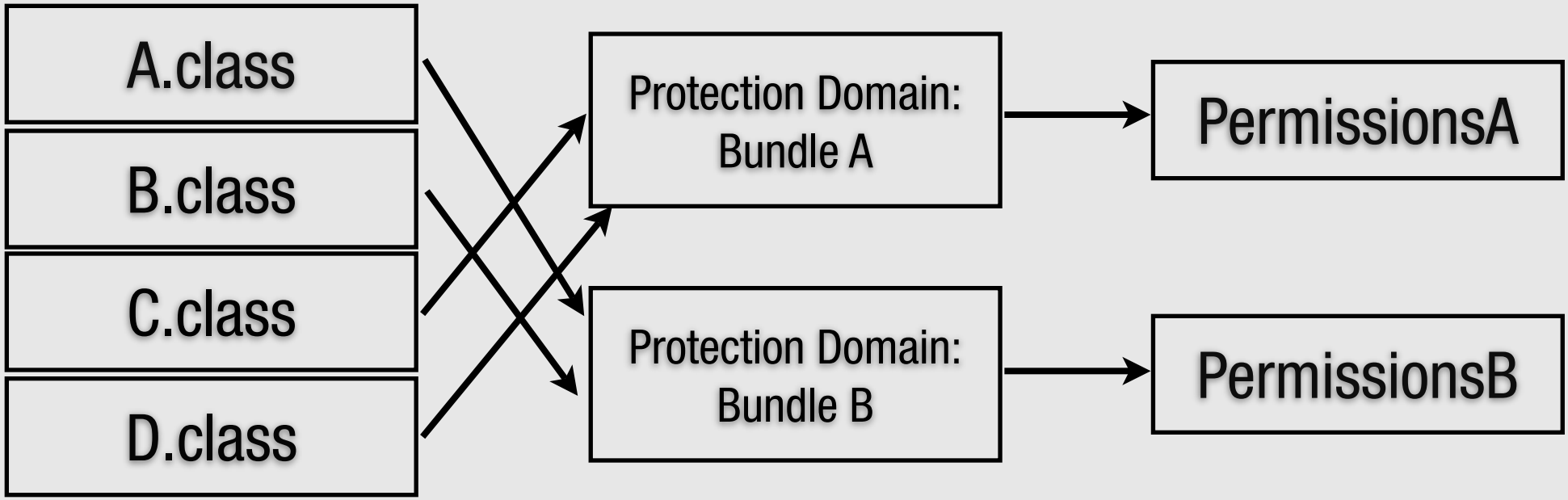
Algorithm

AccessController.checkPermission(Permission p)



Algorithm

AccessController.checkPermission(Permission p)

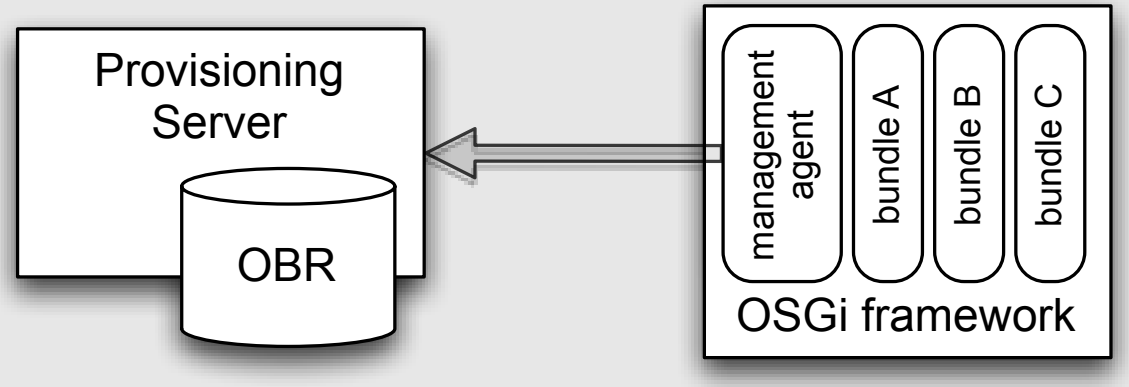


Privileged Call

E.class

```
if (!(PermissionsA.implies(p) &&
      PermissionsB.implies(p))
{
      throw new SecurityException();
}
```

Deployment Topology



- Management Agent, responsible for:
  - life cycle management of the framework
  - security
  - Can use SynchronousBundleListener for on the fly configuration

Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- **Enabling Security** and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

Enable Security: Equinox

- Properties for security manager, keystore, signed bundles support
  - -Djava.security.manager=""
  - -Dosgi.framework.keystore=file:lib/keystore.ks
  - -Dosgi.signedcontent.support=true
- Java Security Policy must give AllPermission
  - -Djava.security.policy=all.policy
    - grant { permission java.lang.AllPermission };

Enable Security: Equinox

- Properties for security manager, keystore, signed bundles support
  - -Djava.security.manager=""
  - -Dosgi.framework.keystore=file:lib/keystore.ks
  - -Dosgi.signedcontent.support=true
- Java Security Policy must give AllPermission
  - -Djava.security.policy=all.policy
    - grant { permission java.lang.AllPermission };

```
java -Djava.security.manager="" -Djava.security.policy=all.policy \
   -Dosgi.framework.keystore=file:keystore.ks -Dosgi.signedcontent.support=true
   -jar org.eclipse.equinox.launcher.jar -noExit
```

Enable Security: Felix

- Felix security is still experimental
  - Not all permission checks implemented
  - Configuration and documentation needs improvements
- Properties for security manager, keystore, keystore password, keystore type
- Java Security Policy must give AllPermission
  - -Djava.security.policy=all.policy
    - grant { permission java.lang.AllPermission };

Enable Security: Felix

- Felix security is still experimental
  - Not all permission checks implemented
  - Configuration and documentation needs improvements
- Properties for security manager, keystore, keystore password, keystore type
- Java Security Policy must give AllPermission
  - -Djava.security.policy=all.policy
    - grant { permission java.lang.AllPermission };

```
java -Djava.security.manager -Djava.security.policy=all.policy
  -Dfelix.keystore=keystore.ks -Dfelix.keystore.pass=luminis -jar felix.jar
```

Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

Setting up your environment

- Memory stick contains VMware image and player
  - Linux account: jars/jars
- Alternatively memory stick contains separate project folder set-up for:
  - Java 5
  - Eclipse Classic 3.3.1.1
  - Ant 1.7

Environment

- Folder structure
  - offermans_pauls_security
    - Building_Secure_Applications.pdf
    - r4_core_book.pdf
    - build.xml - ant clean deploy -> bundles in deploy
    - bin - scripts to run OSGi frameworks with security enabled
      - clean_{equinox,felix}.sh
      - run_{equinox,felix}.sh
    - lib - contains equinox and felix specific resources
    - deploy - task and example bundles
    - workspace/project - contains the task stubs and examples

OSGi Environment

- The felix shell and obr is used
  - use help and obr help command to see commands
  - start bundles with obr start
- Examples and task bundles are created by package e.g.,
  - task1.Activator -> task1.jar = task1 in obr
  - example1.Activator -> example1.jar = example1 in obr
- invoke ant to rebuild, package, and make available via obr

Dry run

```
>ant

> sh bin/clean_equinox.sh
> sh bin/run_equinox.sh
-> obr start task1

> sh bin/clean_felix.sh
> sh bin/run_felix.sh
-> obr start task1
```

Task 1 - Running Secure

- Launch Felix & Equinox with security enabled
- Create a bundle (task1.Activator) that
  - Checks for a security manager;
  - Checks for AllPermissions.

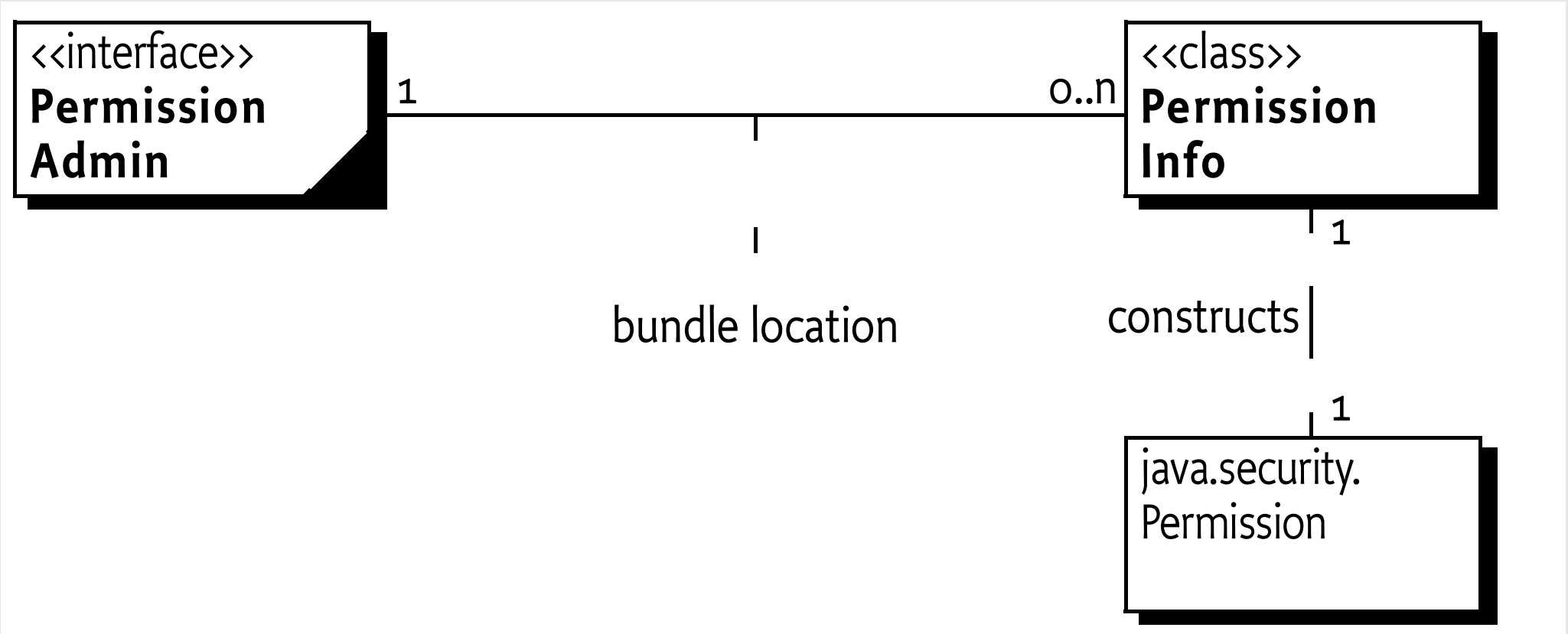
Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

Permission Admin (1/3)

- Old (pre 4.0) way of managing permissions
- Provides information about current permissions
- Allows a management agent to set permissions per bundle
- Permissions are based on bundle locations with a fallback to a set of default permissions

PermissionAdmin (2/3)



PermissionAdmin (3/3)

- Relative FilePermissions are assumed to be relative to the bundle storage area
- All permission changes need AllPermission
  - the first thing a management agent has to do is give itself AllPermission
- If ConditionalPermissionAdmin is present (as is the case in our environment) then default permissions are ignored unless the ConditionalPermissionAdmin has not been set-up with at least one entry

PermissionInfo

- Permission representation used
- Encapsulates three pieces of information
  - type - class name of the permission
  - name - name argument of the permission
  - actions - actions argument of the permission

PermissionInfo

- Permission representation used
- Encapsulates three pieces of information
  - type - class name of the permission
  - name - name argument of the permission
  - actions - actions argument of the permission

```
new PermissionInfo(
    AdminPermission.class.getName(), "(id=10)",
    AdminPermission.EXECUTE);
```

Example

```
PermissionAdmin admin = getPermissionAdmin();


admin.setPermissions(
   context.getBundle().getLocation(),
   new PermissionInfo[]{
      new PermissionInfo(
           AllPermission.class.getName(), "", "")});

PermissionInfo[] previous = admin.getDefaultPermissions();

admin.setDefaultPermissions(new PermissionInfo[0]);

// unset
admin.setDefaultPermissions(previous);
```

Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

OSGi specific permissions

- OSGi specifications define special permissions for framework and service related tasks
- The core framework specification defines:
  - AdminPermission - for all framework specific actions
  - PackagePermission - for package import and export
  - ServicePermission - for service providing and usage
  - BundlePermission - for extensions/fragments
- Custom permissions can be used if they have been exported by a bundle or the classpath

PackagePermission

- A bundle‘s authority to import/export a package
- Name is the package as dot-separated string
  - Wildcards are supported
- Two actions: EXPORT and IMPORT.
  - EXPORT implies IMPORT

PackagePermission

- A bundle‘s authority to import/export a package
- Name is the package as dot-separated string
  - Wildcards are supported
- Two actions: EXPORT and IMPORT.
  - EXPORT implies IMPORT

```
Import-Package: net.luminis.pub.foo, net.luminis.bar
Export-Package: net.luminis.bar
```

PackagePermission

- A bundle's authority to import/export a package
- Name is the package as dot-separated string
  - Wildcards are supported
- Two actions: EXPORT and IMPORT.
  - EXPORT implies IMPORT

Import-Package: net.luminis.pub.foo, net.luminis.bar
Export-Package: net.luminis.bar

```
System.getSecurityManager().checkPermission(
    new PackagePermission("net.luminis.pub.foo", PackagePermission.IMPORT));
System.getSecurityManager().checkPermission(
    new PackagePermission("net.luminis.bar", PackagePermission.EXPORT));
```

PackagePermission

- A bundle's authority to import/export a package
- Name is the package as dot-separated string
  - Wildcards are supported
- Two actions: EXPORT and IMPORT.
  - EXPORT implies IMPORT

Import-Package: net.luminis.pub.foo, net.luminis.bar
Export-Package: net.luminis.bar

```
System.getSecurityManager().checkPermission(
    new PackagePermission("net.luminis.pub.foo", PackagePermission.IMPORT));
System.getSecurityManager().checkPermission(
    new PackagePermission("net.luminis.bar", PackagePermission.EXPORT));

  new PackagePermission("net.luminis.pub.*", PackagePermission.IMPORT);
  new PackagePermission("net.luminis.bar", PackagePermission.EXPORT);
```

ServicePermission

- A bundle‘s authority to register/get a service
- Name is the name of the service interface as a dot separated string
  - Wildcards may be used for the classname
- Two Actions: GET and REGISTER

ServicePermission

- A bundle's authority to register/get a service
- Name is the name of the service interface as a dot separated string
  - Wildcards may be used for the classname
- Two Actions: GET and REGISTER

```
context.getServiceReference("net.luminis.pub.Foo");
context.registerService("net.luminis.pub.Bar", new Bar(), null);
```

ServicePermission

- A bundle's authority to register/get a service
- Name is the name of the service interface as a dot separated string
  - Wildcards may be used for the classname
- Two Actions: GET and REGISTER

```
context.getServiceReference("net.luminis.pub.Foo");
context.registerService("net.luminis.pub.Bar", new Bar(), null);

System.getSecurityManager().checkPermission(
   new ServicePermission("net.luminis.pub.Foo", ServicePermission.GET));
System.getSecurityManager().checkPermission(
   new ServicePermission("net.luminis.pub.Bar", ServicePermission.REGISTER));
```

ServicePermission

- A bundle's authority to register/get a service
- Name is the name of the service interface as a dot separated string
  - Wildcards may be used for the classname
- Two Actions: GET and REGISTER

```
context.getServiceReference("net.luminis.pub.Foo");
context.registerService("net.luminis.pub.Bar", new Bar(), null);

System.getSecurityManager().checkPermission(
   new ServicePermission("net.luminis.pub.Foo", ServicePermission.GET));
System.getSecurityManager().checkPermission(
   new ServicePermission("net.luminis.pub.Bar", ServicePermission.REGISTER));

new ServicePermission("net.luminis.pub.*", ServicePermission.GET);
new ServicePermission("net.luminis.pub.Bar", ServicePermission.REGISTER);
```

BundlePermission

- A bundle‘s authority to require/provide/attach a bundle/fragment
- Name is the bundle symbolic name
  - Wildcards may be used
- Four Actions: PROVIDE, REQUIRE, HOST, and FRAGMENT
  - PROVIDE implies REQUIRE

AdminPermission (1/3)

- A bundle‘s authority to perform specific privileged administrative operations or get sensitive informations about a bundle.
- Name is a filter expression. The filter gives access to the following parameters:
  - signer - A DN chain of bundle signers
  - location - The location of a bundle
  - id - The bundle ID of the bundle
  - name - The symbolic name of a bundle

AdminPermission (2/3)

- There are eleven Actions:
  - class - load a class from a bundle
  - execute - start/stop bundle and set bundle startlevel
  - extensionLifecycle - manage extension bundle
  - lifecycle - manage bundle (update/uninstall/etc.)
  - listener - add/remove synchronous bundle listeners
  - metadata - get manifest and location
  - resolve - refresh and resolve a bundle
  - resource - get/find resources from a bundle
  - startlevel - set startlevel and initial bundle startlevel
  - context - get bundle context

AdminPermission (3/3)

```
context.installBundle("file:bundle.jar").start();
```

AdminPermission (3/3)

```
context.installBundle("file:bundle.jar").start();
```

```
System.getSecurityManager().checkPermission(
        new AdminPermission(bundle));
```

AdminPermission (3/3)

```
context.installBundle("file:bundle.jar").start();
```

```
System.getSecurityManager().checkPermission(
        new AdminPermission(bundle));
```

```
new AdminPermission(
    "(&(signer=o=luminis)(name=net.luminis.*)(location=file://*)(id>=10))",
    AdminPermission.LIFECYCLE + "," + AdminPermission.EXECUTE);
```

Task 2 - Configure Security

- Create a bundle (task2.Activator) that using PermissionAdmin gives:
  - itself AllPermissions;
  - shell, shell.tui, and obr the permissions they need.
  - PackagePermission.IMPORT to all bundles for all packages in the default permissions
- Create a bundle (task2.test.Activator) that:
  - successfully creates a file in its storage area
  - tries to create a file outside its storage area
  - tries to access a service (PermissionAdmin)

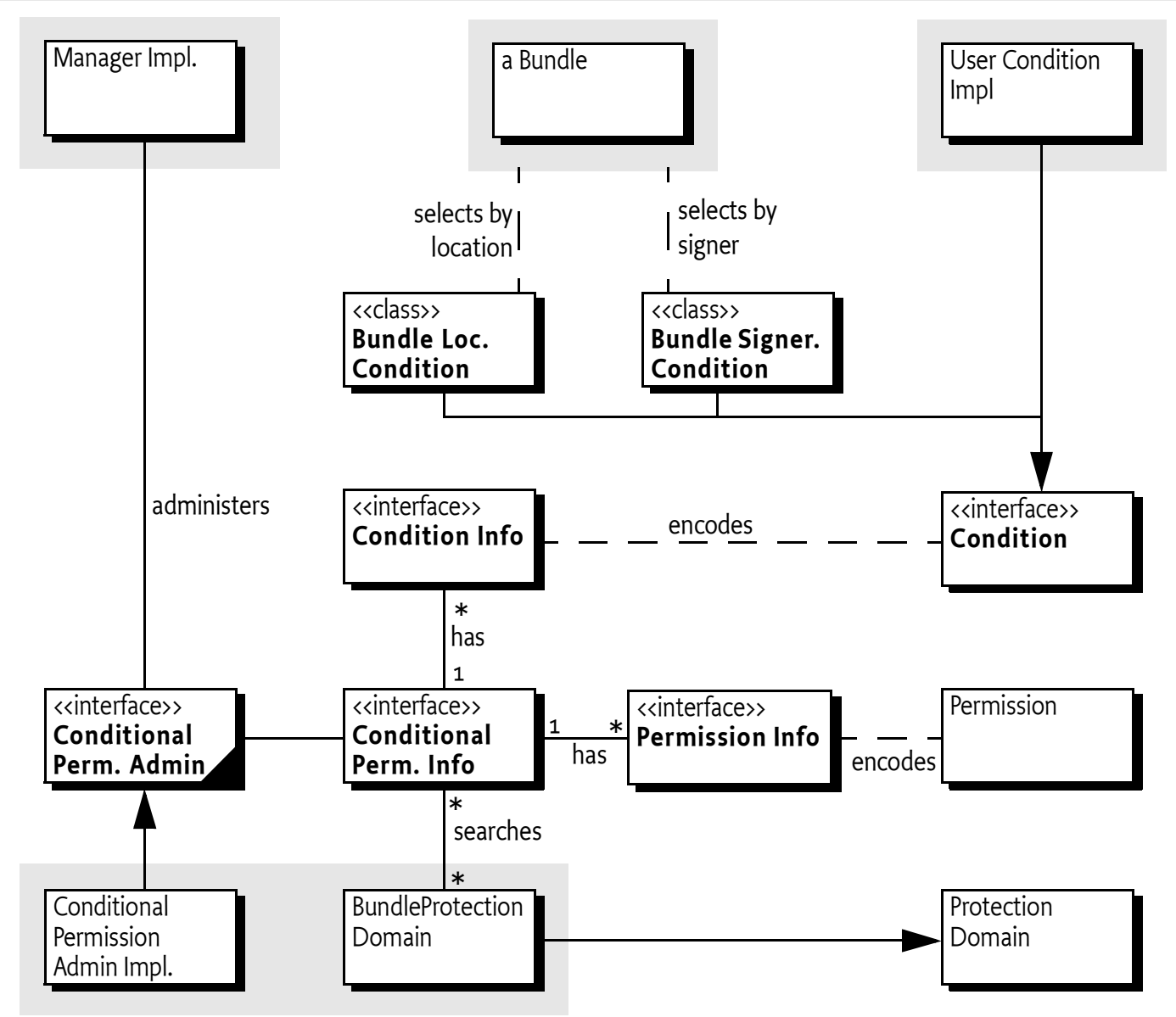
Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

Conditional Permission Admin

- New (4.0) way of doing permission management
  - use this exclusively for new implementations
  - interoperability when both PA and CPA are present
- IF all conditions of a set of conditions match THEN apply the supplied permissions
  - More flexible, extensible model
- Conditions evaluation is highly optimized

CondPermAdmin (1/4)



Conditions

- Purpose is to decide if a permission set is applicable or not.
- Can be postponed or immutable
  - allows optimized evaluations
- Custom conditions can be used for more advanced use-cases

BundleLocationCondition

- Condition to test if the location of a bundle matches a pattern.
  - matching is done based on filter string matching rules

BundleLocationCondition

- Condition to test if the location of a bundle matches a pattern.
  - matching is done based on filter string matching rules

```
new ConditionInfo(BundleLocationCondition.class.getName(),
   new String[] {context.getBundle().getLocation()});
new ConditionInfo(BundleLocationCondition.class.getName(),
   new String[] {"*://www.luminis.nl/*"});
```

Example

```
ConditionalPermissionAdmin condPermAdmin =
getConditionalPermissionAdmin();

condPermAdmin.addConditionalPermissionInfo(
    new ConditionInfo[] {
        new ConditionInfo(
            BundleLocationCondition.class.getName(),
            new String[]{"*://www.luminis.nl/*"})
        },
    new PermissionInfo[] {
        new PermissionInfo(
            AdminPermission.class.getName(),
            "(!(id=" + context.getBundle().getBundleId() + "))",
            "*")
        });
```

Task 3 - Use Conditions

- Create a bundle (task3.Activator) that using ConditionalPermissionAdmin and BundleLocationConditions gives:
  - itself AllPermission
  - shell, shell.tui, and obr the permissions they need.
  - PackagePermission.IMPORT to all bundles for all packages
- Reuse the second bundle of task two (task2.test.Activator) for testing

Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

Signed Bundles

- Authenticates the signer
- Ensures that the content has not been modified
- Bundle (jar) can be signed by multiple signers
- Basically, normal java jar signing with a few extras
  - All entries must be signed except META-INF
- certificate chains represented as ; separated lists
- matching done using * and - wildcards

Signed Bundles

- Authenticates the signer
- Ensures that the content has not been modified
- Bundle (jar) can be signed by multiple signers
- Basically, normal java jar signing with a few extras
  - All entries must be signed except META-INF
- certificate chains represented as ; separated lists
- matching done using * and - wildcards

cn=marrs,o=iQ,c=NL;cn=hans,o=luminis,c=NL

cn=marrs,o=IQ

***;cn=*,o=luminis**

cn=marrs;-;cn=*,o=luminis

Signing bundles in Eclipse

Signing bundles manually

```
jarsigner -keystore file:lib/keystore.ks  \
    -storepass luminis bundle.jar luminis
```

```
<macrodef name="sign-bundle">
    <attribute name="name" />
    <attribute name="location" default="deploy/@{name}.jar" />
    <sequential>
        <exec executable="jarsigner">
            <arg line="-keystore file:lib/keystore.ks" />
            <arg line="-storepass luminis" />
            <arg line="@{location}" />
            <arg line="luminis" />
        </exec>
    </sequential>
</macrodef>
```

Certificates and Keystores

```
keytool -genkey -keystore keystore.ks -alias marrs -storepass luminis  \
  -keypass luminis -dname "CN=Marcel, OU=iQ, O=luminis, L=Arnhem, C=NL"

keytool -selfcert -keystore keystore.ks -alias marrs -storepass luminis \
  -keypass luminis -dname "CN=Marcel, OU=iQ,  O=luminis, L=Arnhem, C=NL"

keytool -export -v -keystore keystore.ks -alias marrs -file luminis.cert \
  -storepass luminis -keypass luminis

keytool -import -v -keystore keystore.ks -alias luminis -file luminis.cert \
  -storepass luminis -keypass luminis

 keytool -list -keystore keystore.ks -storepass luminis


 marrs, Mar 13, 2008, keyEntry,
 luminis, Mar 13, 2008, trustedCertEntry
```

BundleSignerCondition

- Condition to test if the signer of a bundle matches a pattern
- Uses the wildcard matching

BundleSignerCondition

- Condition to test if the signer of a bundle matches a pattern
- Uses the wildcard matching

```
new ConditionInfo(BundleSignerCondition.class.getName(),
                  new String[]{"*,o=luminis"})
```

Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

Local Permissions

- Defined in a resource inside the bundle
- Defines a set of permissions that are enforced by the framework
- A bundle can get less than these permissions, but never more
- Defaults to All Permissions
- Good way for operators to “audit” the permissions of a bundle

LocalPermissions

- OSGI-INF/permissions.perm

```
# Friday, Feb 24 2005
# ACME, chess game
( ..ServicePermission "..log.LogService" "GET" )
( ..PackagePermission "..log" "IMPORT" )
( ..ServicePermission "..cm.ManagedService" "REGISTER" )
( ..PackagePermission "..cm" "IMPORT" )
( ..ServicePermission "..useradmin.UserAdmin" "GET" )
( ..PackagePermission "..cm" "SET" )
( ..PackagePermission "com.acme.chess" "IMPORT,EXPORT" )
( ..PackagePermission "com.acme.score" "IMPORT" )
```

Tip: local permissions tracing with Apache Felix

```
import java.security.Permission;

public class SecMan extends SecurityManager {
    public void checkPermission(Permission perm, Object context) {
        System.out.println(perm);
        try {
            super.checkPermission(perm, context);
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public void checkPermission(Permission perm) {
        System.out.println(perm);
        try {
            super.checkPermission(perm);
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Tip: local permissions tracing with Apache Felix

```
import java.security.Permission;

public class SecMan extends SecurityManager {
    public void checkPermission(Permission perm, Object context) {
        System.out.println(perm);
        try {
            super.checkPermission(perm, context);
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public void checkPermission(Permission perm) {
        System.out.println(perm);
        try {
            super.checkPermission(perm);
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

```
java -Djava.security.manager=SecMan -Djava.security.policy=all.policy \
     -cp .:felix.jar org.apache.felix.main.Main
```

Task 4 - Signed bundles

- Create an (automatically) signed bundle (task4.Activator) that uses the ConditionalPermissionAdmin and a BundleSignerCondition to give itself and other bundles signed by o=luminis AllPermission
  - use BundleLocationConditions to give the needed permissions to shell, shell.tui, and obr.
- Create an (automatically) signed bundle (task4.test.Activator) that limits itself to certain local permissions (task4/test/permissions.perm)
  - Use the SecurityManager to test that you have the local permissions and that you don‘t have others

Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

Custom Condition

- Conditions must come from the classpath/ system bundle
- Are constructed from ConditionInfo objects
  - static getCondition(Bundle,ConditionInfo) method
  - constructor with (Bundle, ConditionInfo) signature

Custom Condition

- Conditions must come from the classpath/ system bundle
- Are constructed from ConditionInfo objects
  - static getCondition(Bundle,ConditionInfo) method
  - constructor with (Bundle, ConditionInfo) signature

```
class BeforeDateCondition implements Condition {
    private final long m_date;

    public static Condition getCondition(Bundle bundle,
        ConditionInfo info) {
      return new BeforeDateCondition(bundle, info);
    }

    public BeforeDateCondition(Bundle bundle,
        ConditionInfo info) {
      m_date = Long.parseLong(info.getArgs()[0]);
    }

    public boolean isMutable() {
      return m_date > System.currentTimeMillis();
    }

    public boolean isPostponed() {
      return false;
    }

    public boolean isSatisfied() {
      return System.currentTimeMillis() < m_date;
    }

    public boolean isSatisfied(Condition[] conditions,
        Dictionary context) {
      return false;
    }
}
```

Extension Bundles

- Extension bundles can deliver optional parts of the Framework implementation
- Necessary to add custom conditions because they have to come from the classpath
- No Import-Package, Require-Bundle, Bundle-NativeCode, DynamicImport-Package, or Bundle-Activator allowed

Extension Bundles

- Extension bundles can deliver optional parts of the Framework implementation
- Necessary to add custom conditions because they have to come from the classpath
- No Import-Package, Require-Bundle, Bundle-NativeCode, DynamicImport-Package, or Bundle-Activator allowed

```
Fragment-Host: system.bundle; extension:=framework
```

Agenda

- Introduction to OSGi layers and Security
- Java and OSGi Security
- Enabling Security and tutorial environment
- PermissionAdmin and OSGi specific permissions
- ConditionalPermissionAdmin
- Signed Bundles and Local Permissions
- Custom and postponed conditions

Postponed Conditions

- Optimize condition evaluation on multiple evaluations during the same permission check
  - context map can be used to pass settings during evaluation
- Use if evaluation is expensive

```
public boolean isPostponed() {
    return true;
}

public boolean isSatisfied(Condition[] conditions, Dictionary context) {
    // do evaluation for all conditions involved
}
```

task 5 - Custom Postponed

- Create an extension bundle providing a custom postponed condition (task5.extension.AskUserCondition)
  - Should open a swing dialog to ask the user
- Create a bundle (tast5.Activator) that gives
  - itself AllPermission using a BundleSignerCondition;
  - shell, shell.tui, and obr needed permission as before;
  - and AllPermision if the AskUserCondition is satisfied
- Reuse bundle from task 1 to test the condition
- Use Felix (equinox is not supported for task5 :-)

Discussion

- We've showed:
  - how security is integrated into OSGi
  - the relation between Java 2 Security and OSGi
  - how to use both Permission Admin and Conditional Permission admin
  - how to use signed bundles, local permissions, and add custom permissions and conditions at runtime