Security

Security

Security Mechanisms in Java

Virtual machine

- erroneous array accesses
- forbidden casts
- …

Class loader

- loading of code
- Byte-code verification

Security manager

- allowed and forbidden operations

Encryption technologies

- code signing
- authentication

Security

Introduction

Class Loader

Security Manager and Permissions

Summary

Class Loading

In Java code is loaded incrementally on demand

Process of loading class code
(illustrated by execution of program `MyProgram`):

- Loading of class file **`MyProgram.class`**
- Loading of all super classes and classes of variables of class **`MyProgram`**
- Execution on **`main`** in class **`MyProgram`**
- Loading of all classes used when executing **`main`**
  - classes of variables and instances
  - classes of methods

Class Loader

Loading of classes done by `ClassLoader` objects

- Accessing **`ClassLoader`** objects

```
Class clazz = Class.forName("MyProgram");
ClassLoader loader = clazz.getClassLoader();
```

```
ClassLoader loader = ClassLoader.getSystemClassLoader();
```

Standard ClassLoader!

- Class loaders support
  - Loading classes explicitly

```
Class myClass = loader.loadClass("mypack.MyClass");
```

  - and defining new classes

```
byte[] classCode = ...;
loader.defineClass("MyDefinedClass", classCode, 0, classCode.length);
```

- Class loaders can be set
  - for thread one can define a **`ContextClassLoader`**

```
Thread thread = Thread.currentThread();
thread.setContextClassLoader(loader);
```

Then all classes loaded in this thread are loaded by this class loader!

Kinds of Class Loaders

Java uses a hierarchy of class loaders

- Bootstrap class loader:
  - loads all system classes from **rt.jar**

which contains standard library!

- Extension class loader:
  - loads all extensions from directory **jre/lib/ext**

Better name would be "application class loader"

- `SystemClassLoader`:
  - loads classes in class path (CLASSPATH)

- Special class loader:
  - Application-specific class loader as extension of **`SystemClassLoader`**

Hierarchy of Class Loaders

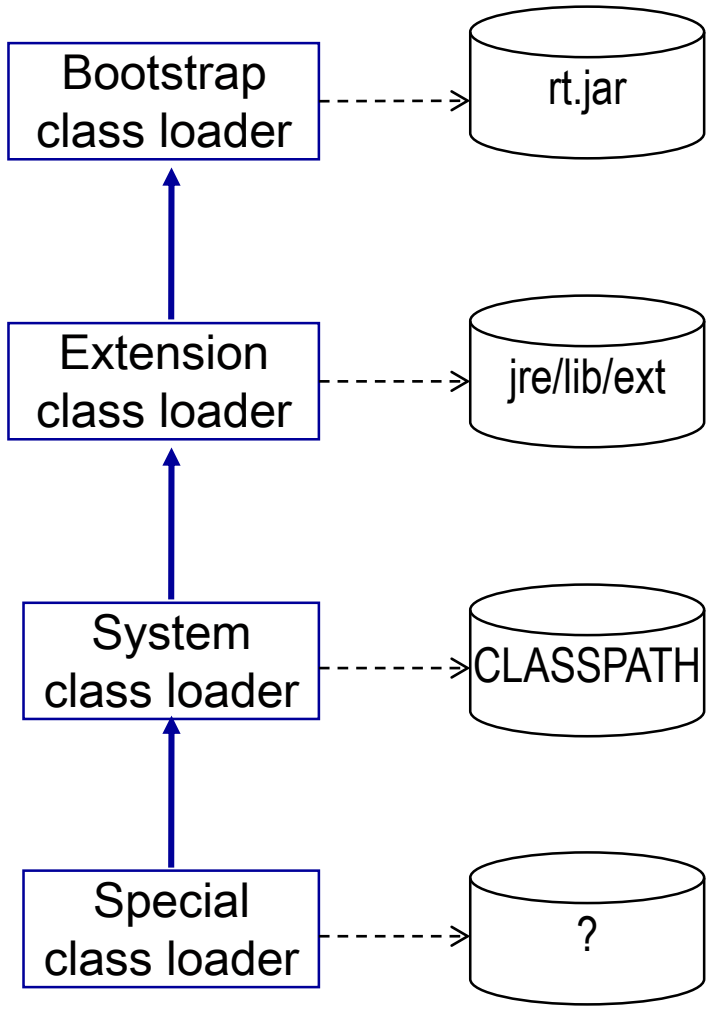
Hierarchy class loader with parent and child relation

Determine priority of class loading

- first Bootstrap class loader
- then Extension class loader
- then System class loader
- finally one of the specializations

Class loaders first delegate loading of a class to its parent class loader

and only loads class if parent fails



Example: Specific Class Loader

URLClassLoader

- Loads classes from URLs

```
URL pluginUrl = new URL("file:c:/plugins.jar");
URLClassLoader pluginLoader = new URLClassLoader(
                                  new URL[] { pluginUrl },
                                  ClassLoader.getSystemClassLoader()
                              );

Class<?> cl = pluginLoader.loadClass("plugin1.PluginClass");
cl.getMethod("test").invoke(null);
```

Plugin-JAR not in CLASSPATH!

superior class loader

Example: Implementation of Class Loader

Class loader as specialization of **`ClassLoader`**

- decryption of encrypted Byte code files
- overriding method **`findClass`**

```
public class CryptoClassLoader extends ClassLoader {
    private final String path;
    private final int key;

    public CryptoClassLoader(String path, int key) {...}

    @Override
    protected Class<?> findClass(String name) throws ClassNotFoundException {
        byte[] classBytes = null;
        try {
            classBytes = loadAndDecryptClassBytes(name);
        } catch (IOException e) { throw new ClassNotFoundException(name); }
        Class<?> clazz = defineClass(name, classBytes, 0, classBytes.length);
        if (clazz == null ) throw new ClassNotFoundException(name);
        return clazz;
    }

    private byte[] loadAndDecryptClassBytes(String name) throws IOException {
        ...
    }
```

Method **findClass** called by **loadClass**!

Uniqueness of Classes by ClassLoader

Uniqueness of classes by:

- fully qualified name (e.g., `java.sql.Date`)
- class loader which has loaded this class

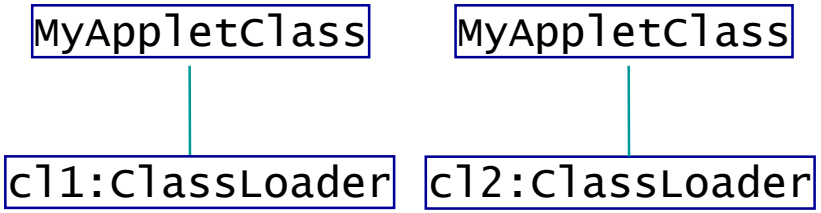
Class loader object defines namespace for classes!

Then VM can

- load and run classes with same name from different sources
- load and use different versions of a class

Example:

- Applets loaded from different servers are loaded by different class loaders
- Applets can use classes with equal name



Security

Introduction

Class Loader

Security Manager and Permissions

Summary

Security Manager

Security manager allows checking if certain (possibly dangerous) operations are allowed

For example:

- accessing a file
- opening socket connection
- accessing system properties
- terminating application
- creating and setting a class loader
- accessing the AWT event queue
- opening a top-level window (Frame)
- installation of a security manager
- ...

Security Check

Example: Security check in `System.exit`

```
public void exit(int status) {
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        checkExit(status);
    }
    Shutdown.exit(status);
}
```

can throw SecurityException!!

Installation of a Security Manager

Per default to security manager installed

- ➔ no checks are performed

Installation of a security manager in application by

```
System.setSecurityManager(SecurityManager sm)
```

or at program start

```
java -Djava.security.manager ...
```

Available security manager :

- `SecurityManager`
- ~~`RMISecurityManager`~~ deprecated
- Application-specific security managers a specialization of `SecurityManager`

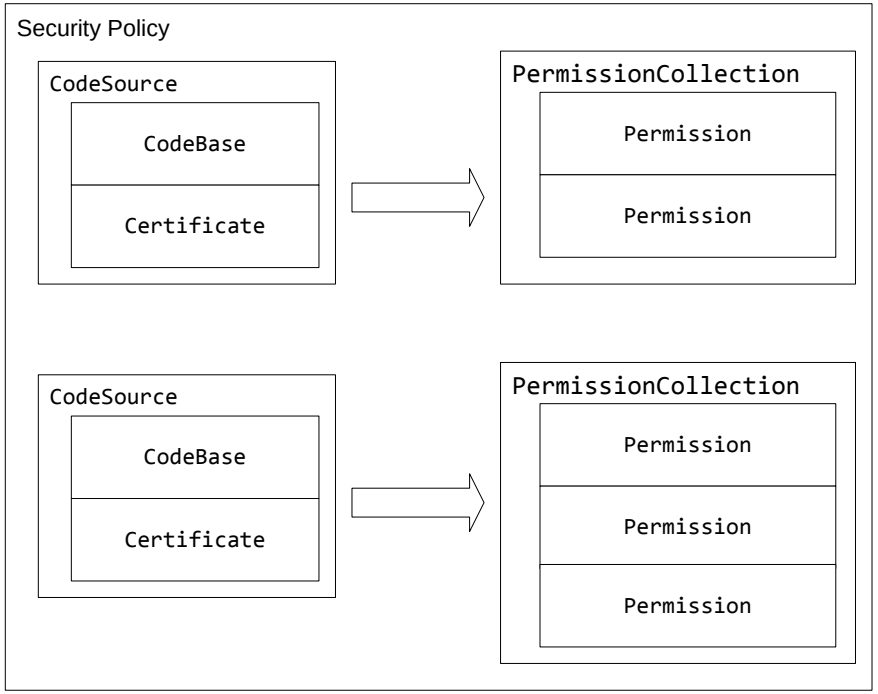
Security Model in Java

Security policies which define mapping
- from code source
- to permissions

i.e., code from specific source is given certain permissions

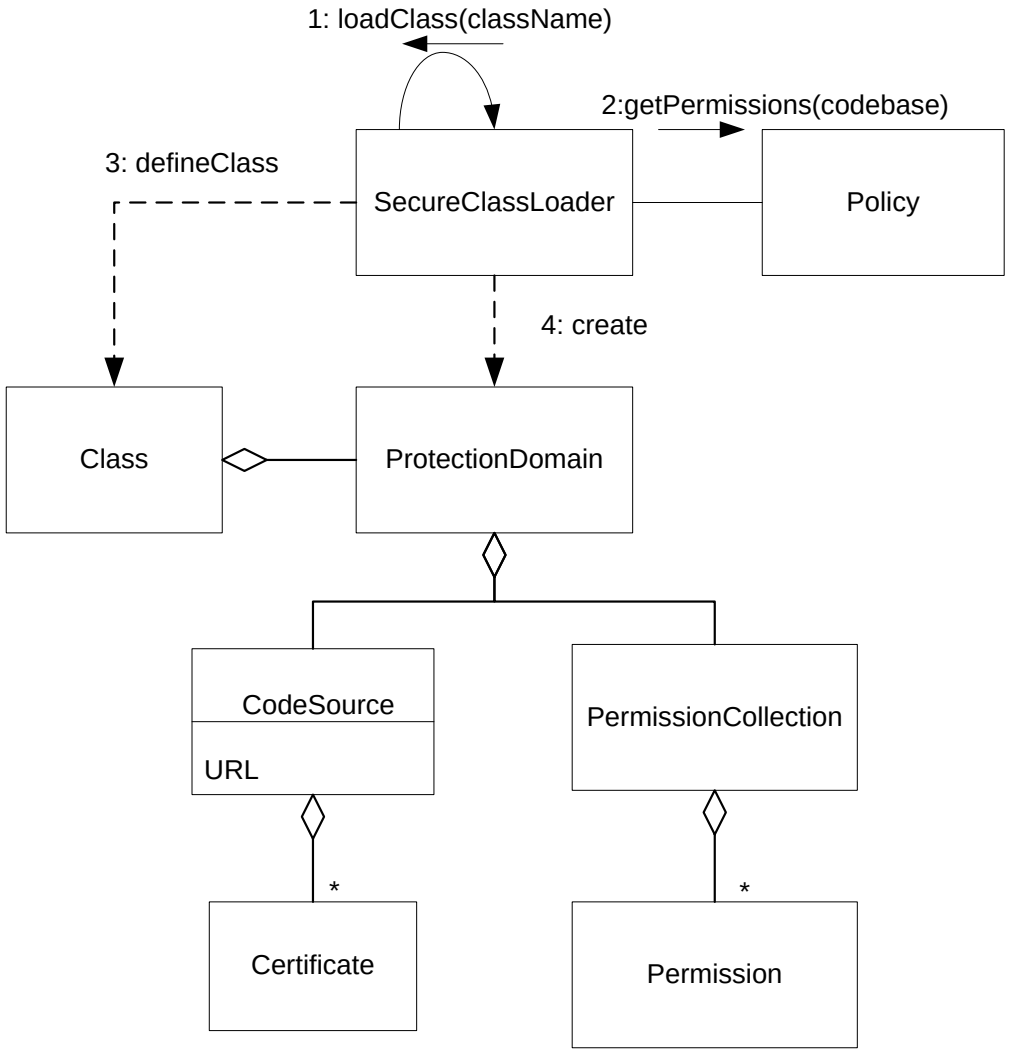
Code sources and permissions represented by classes and objects
- Class **`CodeSource`** with **`CodeBase`** and **`Certificate`**
- Class system of **`Permission`**s
- **`PersmissionCollection`** are collections of permissions

Security policies are defined in policy files (see below)



Process of Creation of a ProtectionDomain

1. Load class by `SecureClassLoader`
2. Getting Permissions from Policy based on CodeBase
3. Definition of class (`defineClass`)
4. Creation of ProtectionDomain for class with CodeSource and PermissionCollections



Operations with Security Checks

Process of a security check

- Accessing installed SecurityManager by **`System.getSecurityManager`**
- When installed (**`!= null`**), calling **`checkPermission`** method of **`SecurityManager`**
- execution of operation after passing check
- or throwing a **`SecurityException`** when check not passed

```
public void <checkedOperation>(...) {
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        security.checkPermission(new …Permission(…));
    }
    <uncheckedOperation>(...);
}
```

can throw SecurityException!!

```
public void connect(SocketAddress endpoint, int timeout) throws IOException {
    …
    SecurityManager security = System.getSecurityManager();
    if (security != null) {
        security.checkPermission(new SocketPermission(host+":"+port,
                SecurityConstants.SOCKET_CONNECT_ACTION));
    …
}
```

simplified

checkPermission

Check of permissions with method **checkPermission** of **SecurityManager**

```
public void checkPermission(Permission p)
```

Algorithms for checking Permission p

Pseudocode

```
checkPermission(Permission p) throws SecurityException {
   for all classes clazz of methods on call stack {
        permColl =  clazz.getProtectionDomain().getPermissions();
        if (! exists q in permColl with q.implies(p) )
            throw SecurityException(...);
   }
   permission p granted
}
```

Example: read from SocketInputStream

```
public class DemoCallStack {
    public static String readData() throws IOException {
        String fileName = "data.txt";
        try (BufferedReader r = new BufferedReader(new FileReader(fileName))) {
            String line = r.readLine();
            while (line != null) {
                …
                line = r.readLine();
            }
        }
    }
    public static void main(String[] args) throws IOException {
          readData();
    }
```

Method requiring permission

Stacktrace:

- SocketInputStream.read()
- ...
- BufferedReader.readLine()
- DemoCallStack.readLine(BufferedReader) line: 27
- DemoCallStack.readData() line: 17
- DemoCallStack.main(String[]) line: 11

Classes of all methods on call stack must have permission to read from socket!

Method implies of Permission

Permission implements method

```
boolean implies(Permission permission)
```

which checks of this permission implies permission given as parameter

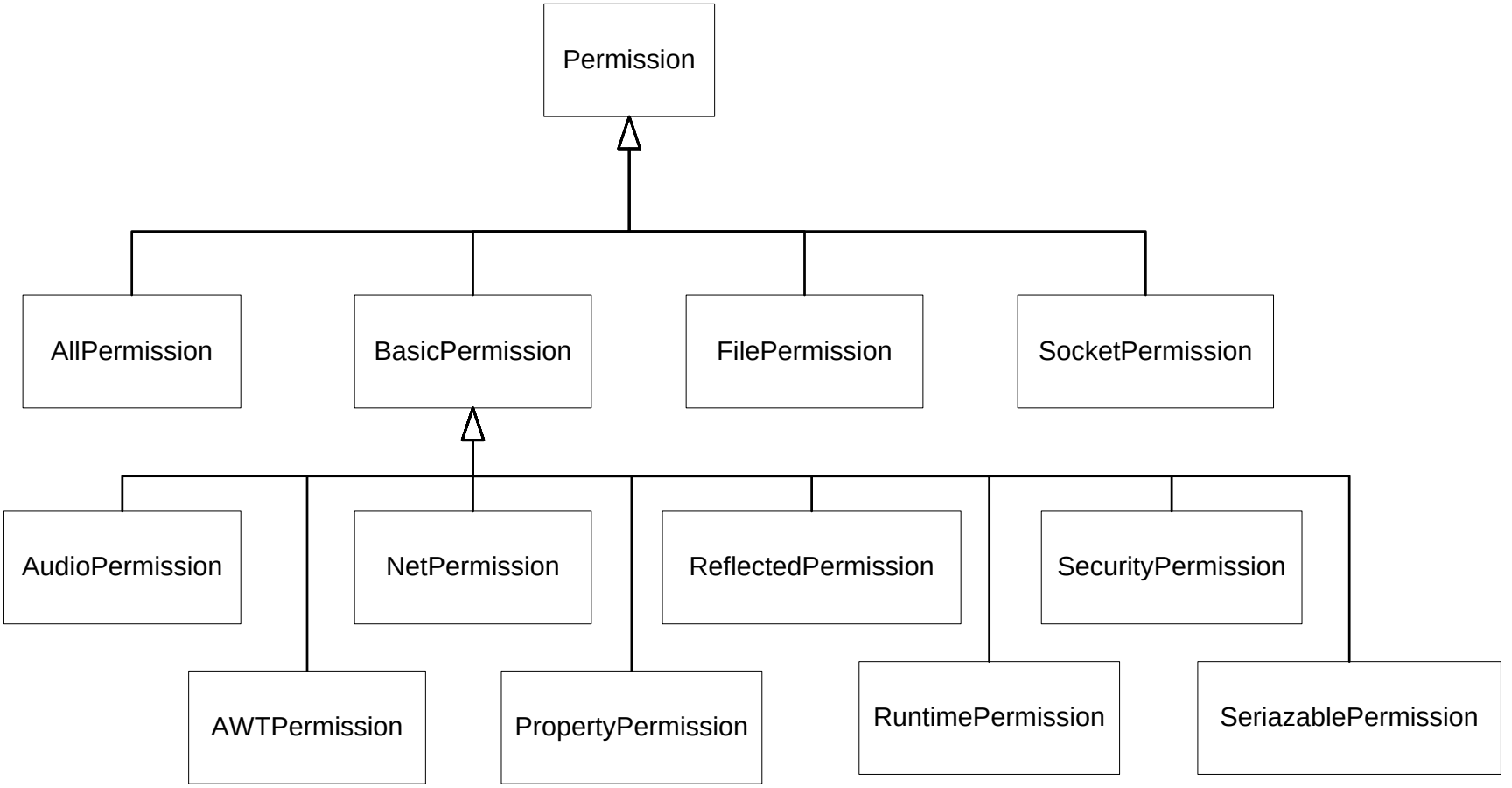
Examples:

```
RuntimePermission("*")
   implies
RuntimePermission("ExitVM")

FilePermission(“C:\temp\*“, “read“)
   implies
FilePermission(“C:\temp\MyFile“, “read“)

SocketPermission( “*:1024-65535", “connect”)
   implies
SocketPermission “yourserver.com:1099", “connect“)
```

Hierarchy of Permission Classes



Are configured with different parameters!

Policy Files

Permissions are usually defined in **policy files**

which contain a sequence of **grant entries**

- mapping of code source
- to set of permissions

```
grant Codesource {
  Permission_1;
  Permission_2;
};
```

Code source consists of

- a URL for the Codebase and
- names of trusted certifiers

```
grant
  codebase codebase-URL
  signedby certificate-name ... {
```

Permission in the form

- keyword **`permission`**
- class name of **`Permission`** class
- a permission-specific target for the permission (e.g. a directory, address, or operation)
- optionally a list of permission-specific actions

```
grant
  codebase codebase-URL
  signedby certificate-name ...
{
permission  permission-className
                  target
                  action1, ...;
...
};
```

Example Policy File

all Codesources!

```
grant {
   permission java.lang.RuntimePermission "stopThread";
   permission java.net.SocketPermission "localhost:1024-", "listen";
   permission java.util.PropertyPermission "java.version", "read";
   permission java.util.PropertyPermission "java.vendor", "read";
        ...
};

grant codeBase "file:${java.home}/lib/ext/*" {
   permission java.security.AllPermission;
};

grant codeBase “www.ssw.uni-linz.ac.at/classes/" {
   permission java.net.SocketPermission “*:1024-65535", "connect";
   permission java.io.FilePermission “${user.home}${/}-“,
                                        “read “, “ write “, “execute“;
   ...
};

...
```

incl. all subdirectories!

Table of Permission (Excerpt)

Permission	Target	Action
java.io.FilePermission	File	read, write, execute, delete
iava.net.SocketPermission	Socket (host, port)	accept,connect, listen, resolve
java.util.PropertyPermission	Property l	read, write
java.lang.RuntimePermission	createClassLoader createSecurityManager exitVM stopThread queuePrintJob ...	
java.net.NetPermission	setDefaultAuthenticator specifyStreamHandler requestPassword-Authentication	
java.awt.AWTPermission	showWindowWithoutWarningBanner accessClipboard acccessEventQueue listenToAllAWTEvents readDisplayPixels	
java.security.SecurityPermission	getPolicy, setPolicy ...	
...		

Example: SecurityManager

Installation of a SecurityManagers

```
System.setSecurityManager(new SecurityManager());
```

Then files cannot be read or written

```
try (BufferedReader reader = new BufferedReader(new InputStreamReader(
                                                  new FileInputStream(filename)))) {
  for (String line = reader.readLine(); line != null; line = reader.readLine()) {
    System.out.println(line);
  }
} catch (Throwable t) { System.out.println("Unable to read file: " + t); }

try (BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(
                                    new FileOutputStream(filename, true)))) {
  writer.append("Hello World!\n");
} catch (Throwable t) { System.out.println("Unable to write file: " + t); }
```

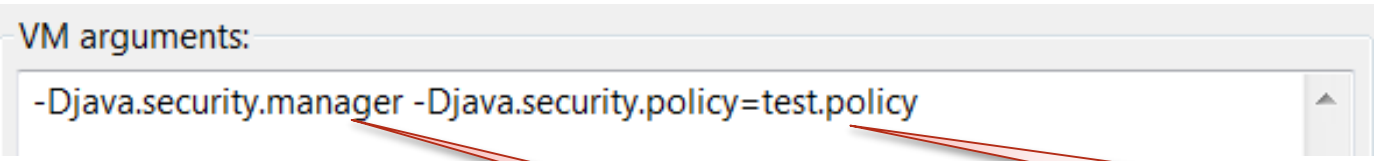
```
Unable to read file: java.security.AccessControlException: access denied
("java.io.FilePermission" "test.txt" "read")
Unable to write file: java.security.AccessControlException: access denied
("java.io.FilePermission" "test.txt" "write")
```

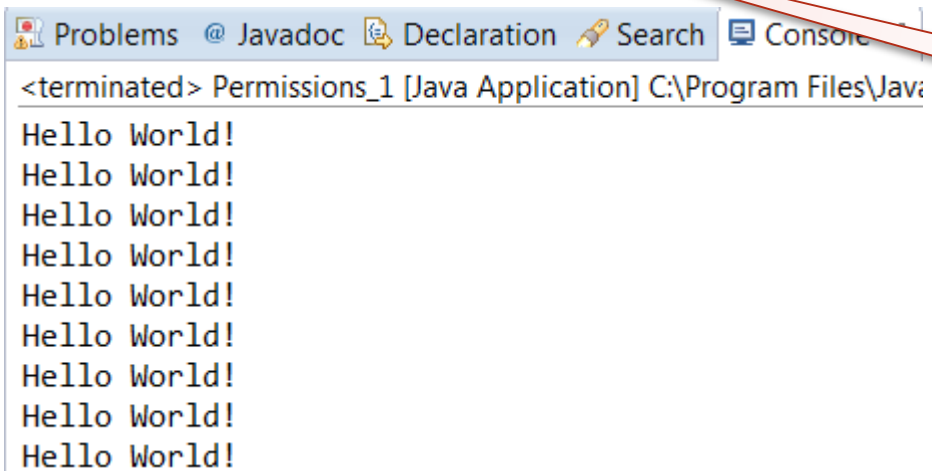
Example: SecurityManager

Policy file with permissions allow access to files

```
grant codeBase "file:C:/.../UE03_WS/Security_Permissions_1/bin/-" {
   permission java.io.FilePermission "test.txt", "read";
   permission java.io.FilePermission "test.txt", "write";
};
```

```
java -Djava.security.manager -Djava.security.policy=test.policy ...
```





Sets Policy-File!

Sets SecurityManager!

Security

Introduction

Class Loader

Security Manager and Permissions

Summary

Literature

Horstmann, Cornell: Core Java 2, Volume II . Advance Features, Sun Microsystems, 2008: Chapter 9

Scott Oaks: Java Security, O'Reilly, 2001

http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html

https://docs.oracle.com/javase/tutorial/security/