Architecture of Enterprise Applications 4
Security I

Haopeng Chen

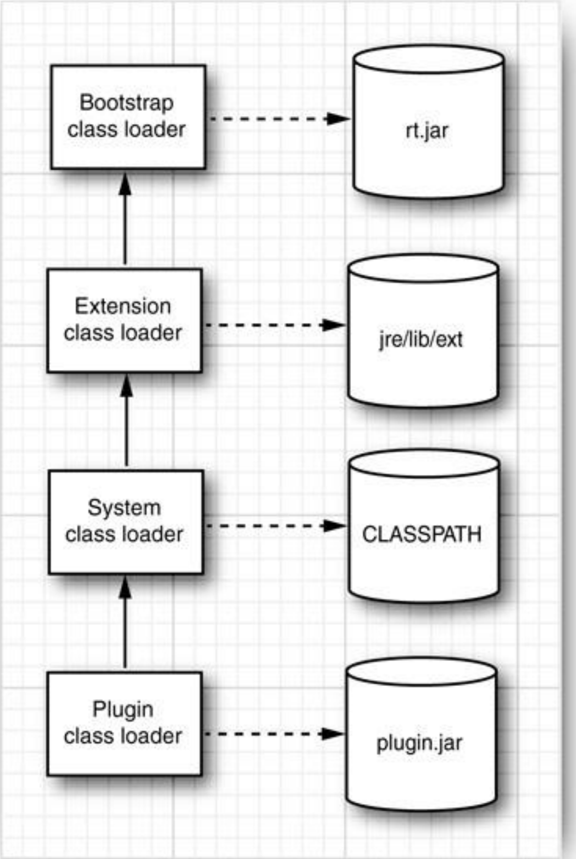
***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents

- SECURITY
  - CLASS LOADERS
  - BYTECODE VERIFICATION
  - SECURITY MANAGERS AND PERMISSIONS

Class Loader

```
URL url = new URL("file:///path/to/plugin.jar");
URLClassLoader pluginLoader = new
   URLClassLoader(new URL[] { url });
Class<?> cl =
   pluginLoader.loadClass("mypackage.MyClass");
```



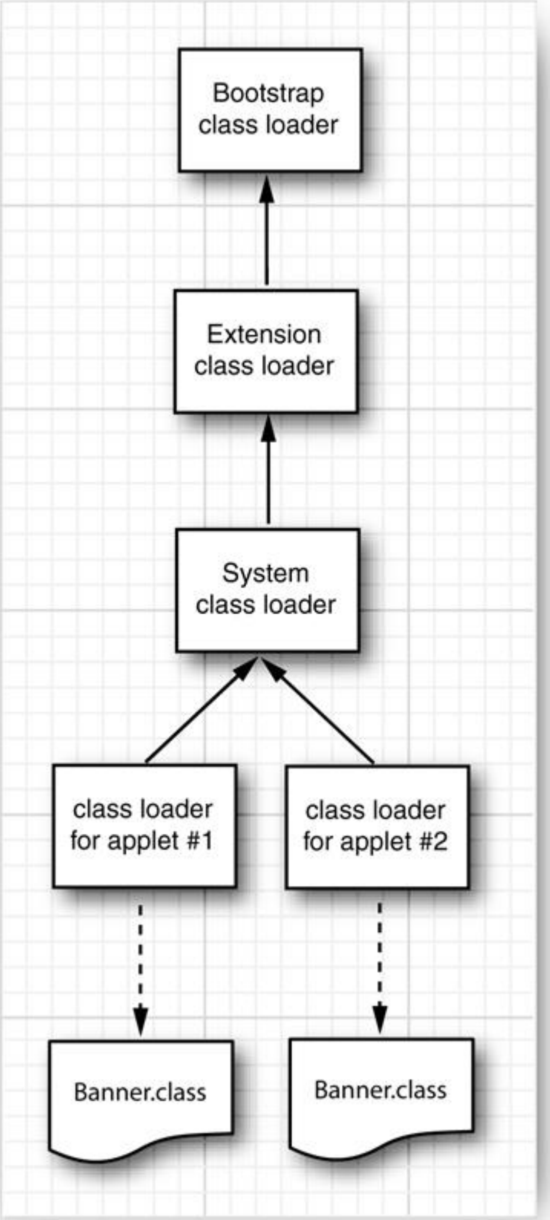
Class Loader

- However, you can set any class loader by calling

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

- Then retrieve the context class loader:

```
Thread t = Thread.currentThread();
ClassLoader loader = t.getContextClassLoader();
Class cl = loader.loadClass(className);
```



Class Loader

- Our own ClassLoader

```
class CryptoClassLoader extends ClassLoader
 {
     /**
        Constructs a crypto class loader.
        @param k the decryption key
     */
     public CryptoClassLoader(int k)
     {
        key = k;
     }

     protected Class findClass(String name)
        throws ClassNotFoundException
     {
        byte[] classBytes = null;
        try
        {
           classBytes = loadClassBytes(name);
        }
```

Class Loader

```
  catch (IOException e)
   {
      throw new ClassNotFoundException(name);
   }

   Class cl = defineClass(name, classBytes, 0,
                         classBytes.length);
   if (cl == null)
      throw new ClassNotFoundException(name);
   return cl;
}

/**
   Loads and decrypt the class file bytes.
   @param name the class name
   @return an array with the class file bytes
*/
private byte[] loadClassBytes(String name)
   throws IOException
{
```

Class Loader

```
      String cname = name.replace('.', '/') + ".caesar";
      FileInputStream in = null;
      in = new FileInputStream(cname);
      try {
         ByteArrayOutputStream buffer = new ByteArrayOutputStream();
         int ch;
         while ((ch = in.read()) != -1) {
            byte b = (byte) (ch - key);
            buffer.write(b);
         }
         in.close();
         return buffer.toByteArray();
      }
      finally
      {
         in.close();
      }
   }
 private int key;
}
```

Class Loader

- Use our own ClassLoader

```
public class Caesar
 {
    public static void main(String[] args)
    {
       if (args.length != 3)
       {
          System.out.println("USAGE: java Caesar in out key");
          return;
       }

       try
       {
          FileInputStream in = new FileInputStream(args[0]);
          FileOutputStream out = new FileOutputStream(args[1]);
          int key = Integer.parseInt(args[2]);
```

Class Loader

```
        int ch;
         while ((ch = in.read()) != -1)
         {
            byte c = (byte)(ch + key);
            out.write(c);
         }
         in.close();
         out.close();
      }
      catch (IOException exception)
      {
         exception.printStackTrace();
      }
   }
}
```

Class Loader

```
public void runClass(String name, String key)
{
   try
   {
      ClassLoader loader = new
                 CryptoClassLoader(Integer.parseInt(key));
      Class c = loader.loadClass(name);
      String[] args = new String[] {};

      Method m = c.getMethod("main", args.getClass());
      m.invoke(null, (Object) args);
   }
   catch (Throwable e)
   {
      JOptionPane.showMessageDialog(this, e);
   }
}
```

Byte Verification

- Here are some of the checks that the verifier carries out:
  - Variables are initialized before they are used.
  - Method calls match the types of object references.
  - Rules for accessing private data and methods are not violated.
  - Local variable accesses fall within the runtime stack.
  - The runtime stack does not overflow.

- `java -noverify Hello`

Byte Verification

```
static int fun()
{
   int m;
   int n;
   m = 1;
   n = 2;
   int r = m + n;
   return r;
}

0 iconst_1 04
1 istore_0 3B
2 iconst_2 05
3 istore_1 3C
4 iload_0  1A
5 iload_1  1B
6 iadd       60
7 istore_2 3D
8 iload_2  1C
9 ireturn  AC
```

```
static int fun()
{
   int m;
   int n;
   m = 1;
   m = 2;
   int r = m + n;
   return r;
}

0 iconst_1 04
1 istore_0 3B
2 iconst_2 05
3 istore_0 3B
4 iload_0  1A
5 iload_1  1B
6 iadd       60
7 istore_2 3D
8 iload_2  1C
9 ireturn  AC
```

Security Manager and Permission

- Operations checked by the security manager include the following:
  - Creating a new class loader
  - Exiting the virtual machine
  - Accessing a field of another class by using reflection
  - Accessing a file
  - Opening a socket connection
  - Starting a print job
  - Accessing the system clipboard
  - Accessing the AWT event queue
  - Bringing up a top-level window

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

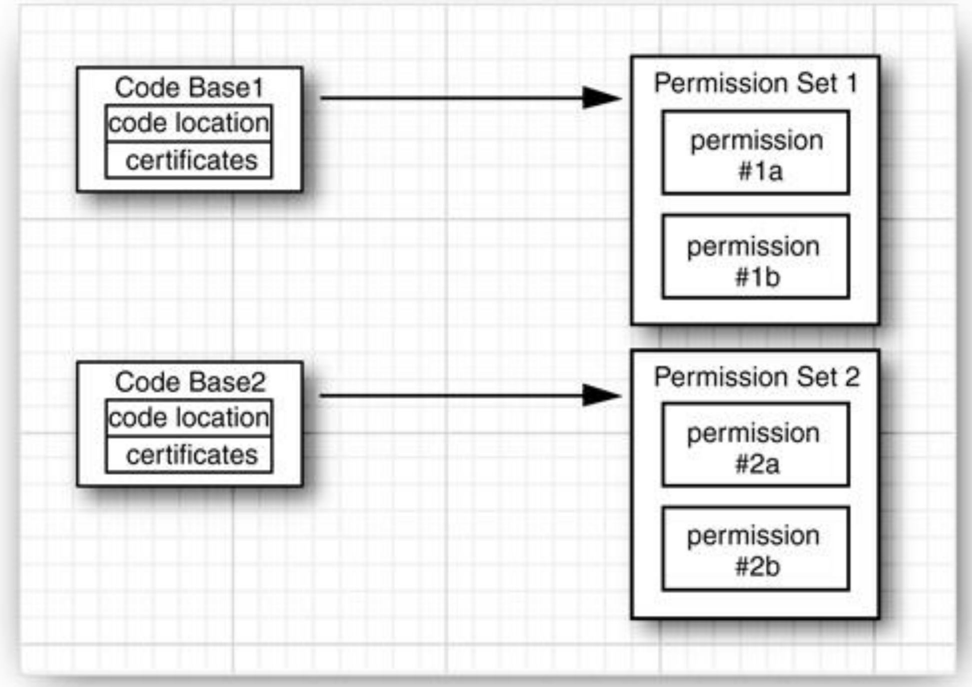
Java Platform Security

- Code:

```
FilePermission p = new FilePermission("/tmp/*", "read,write");
```

- Permission file:

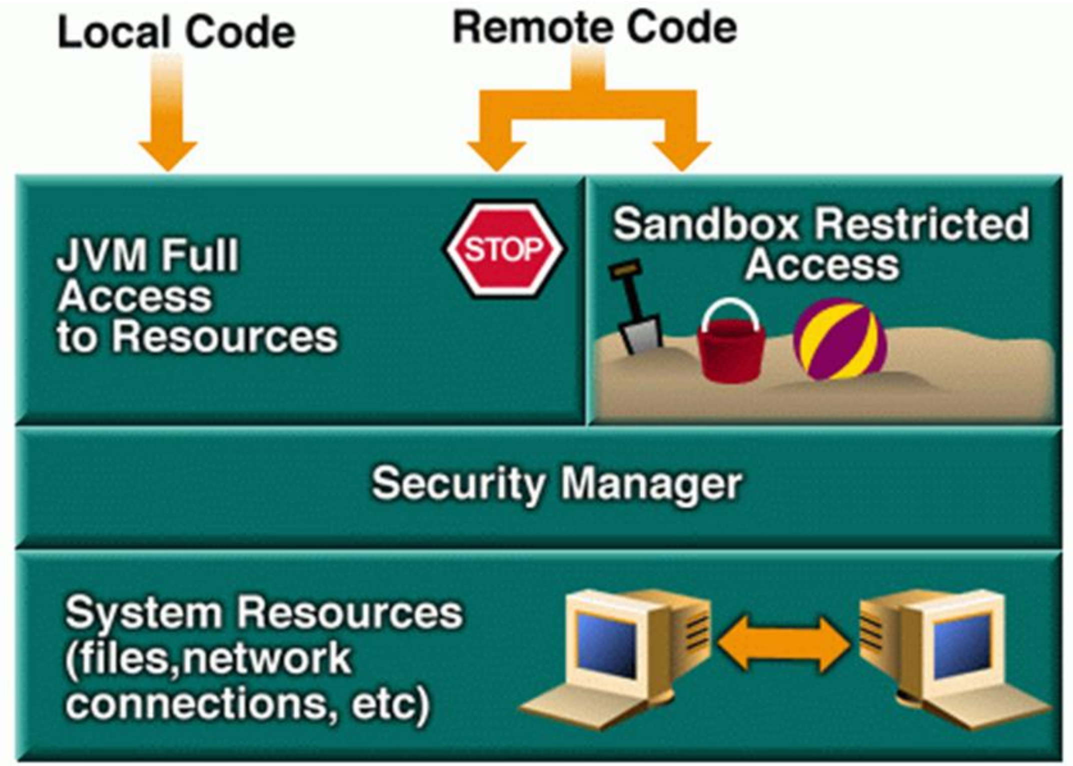
```
java.io.FilePermission "/tmp/*", "read,write";
```



Java Platform Security

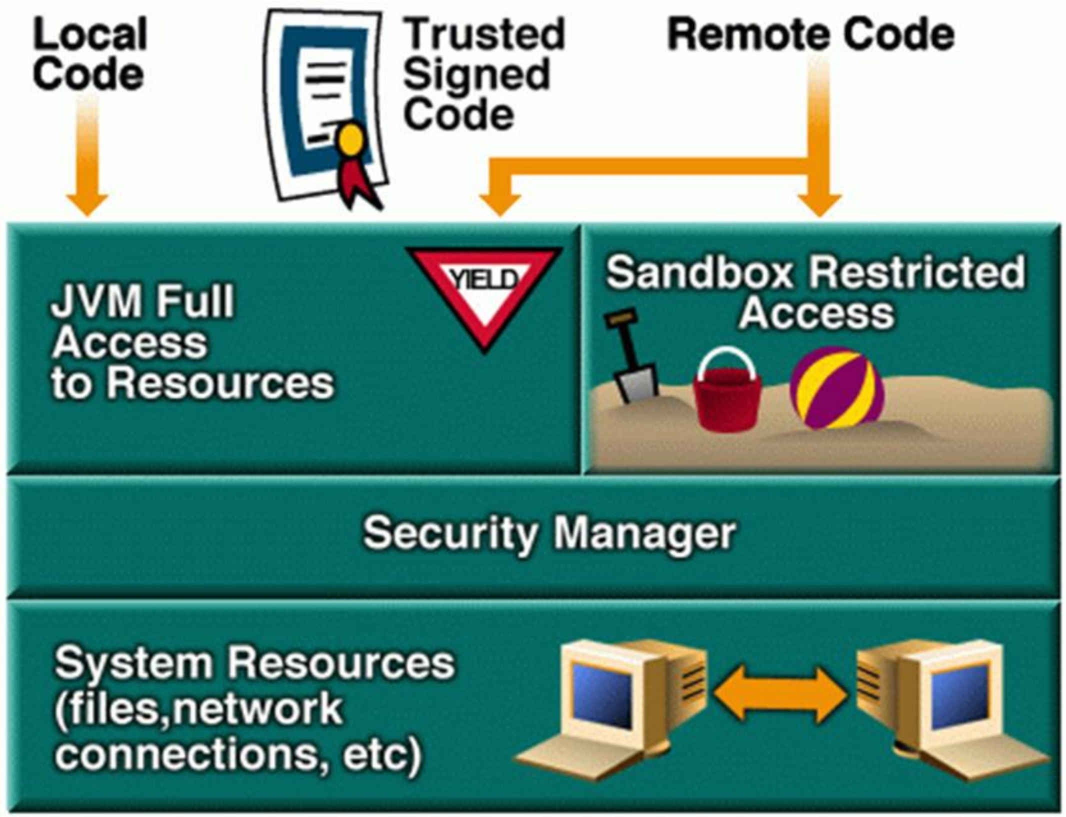
REliable, INtelligent & Scalable Systems

- JDK1.0


Local Code
Remote Code
JVM Full Access to Resources
STOP
Sandbox Restricted Access
Security Manager
System Resources (files,network connections, etc)

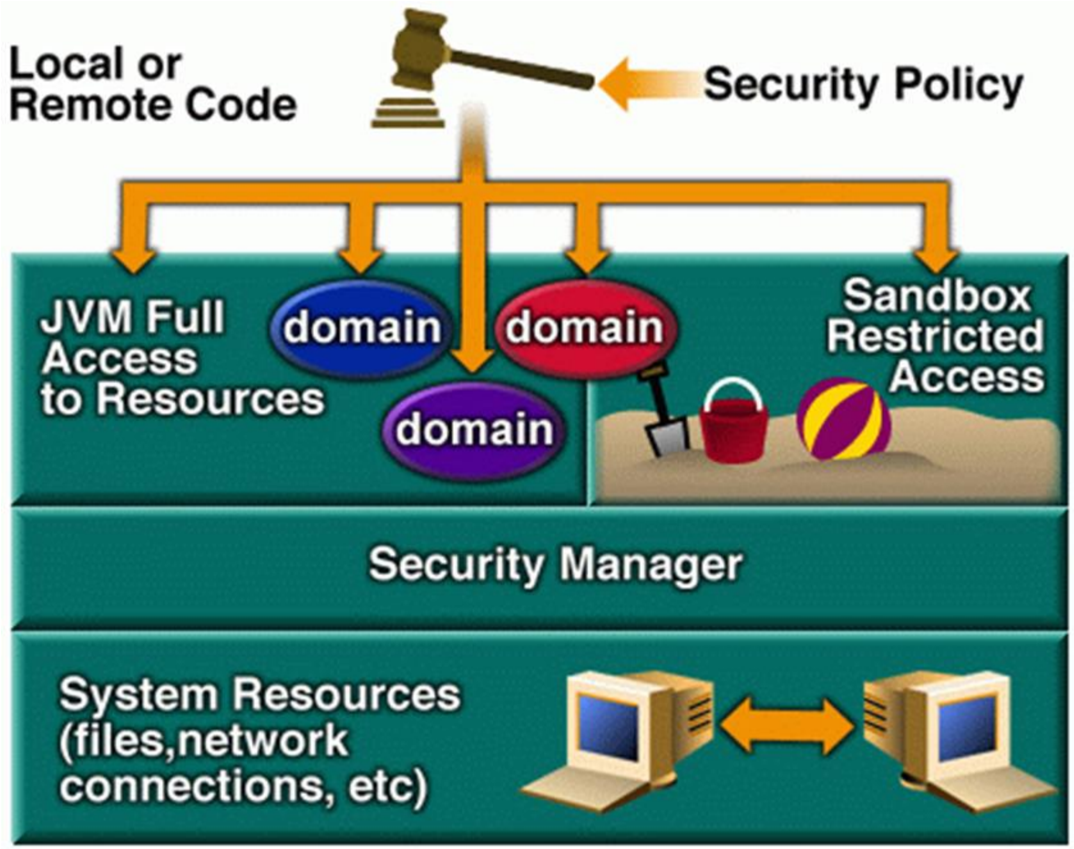
Java Platform Security

- JDK1.1


Local Code
Trusted Signed Code
Remote Code
JVM Full Access to Resources
YIELD
Sandbox Restricted Access
Security Manager
System Resources (files,network connections, etc)

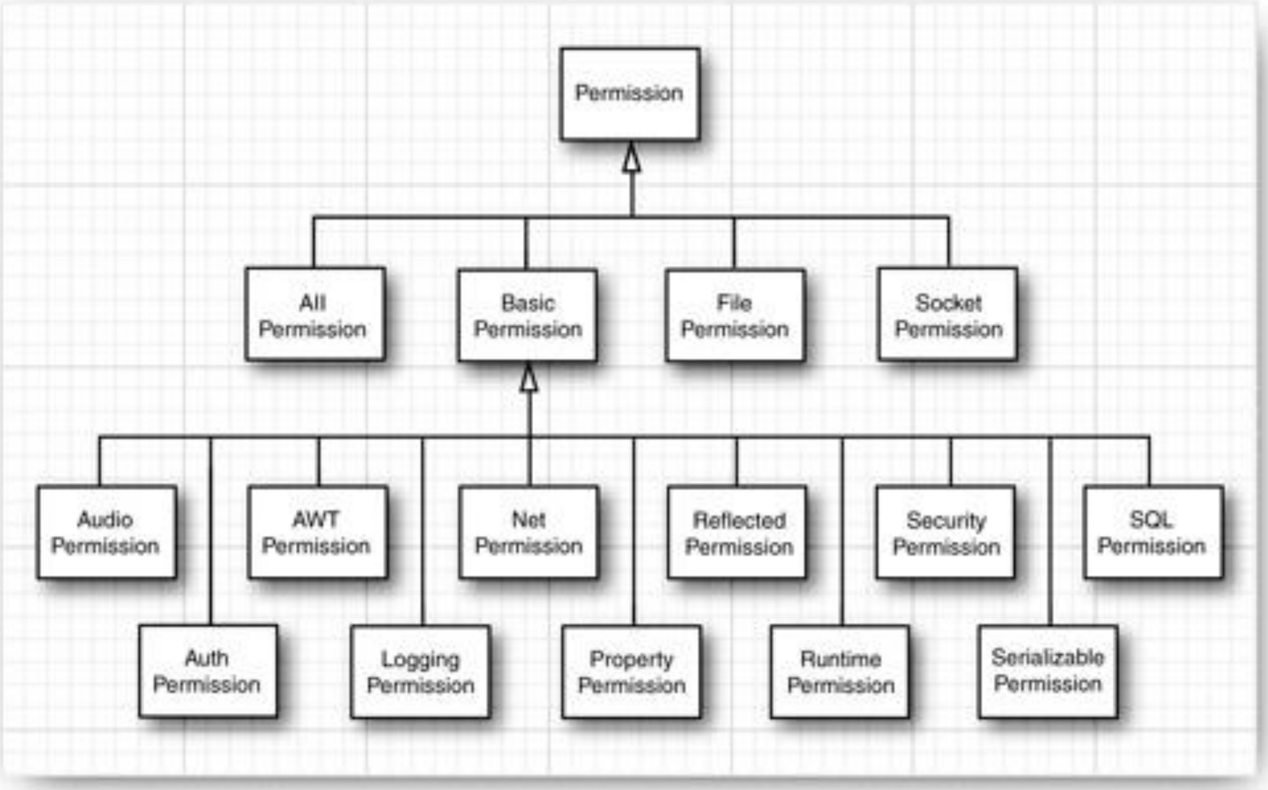
Java Platform Security

- JDK1.2+



Java Platform Security

- A part of the hierarchy of permission classes



```
public void checkExit() {
   checkPermission(new RuntimePermission("exitVM"));
}
```

Security Policy Files

```
grant codeBase "http://www.horstmann.com/classes"
{
   permission java.io.FilePermission "/tmp/*", "read,write";
};
```

- You can install policy files in standard locations. By default, there are two locations:
  - The file java.policy in the Java platform home directory
  - The file .java.policy (notice the period at the beginning of the file name) in the user home directory

- You can change the locations of these files in the java.security configuration file in the jre/lib/security. The defaults are specified as
  - policy.url.1=file:${java.home}/lib/security/java.policy
  - policy.url.2=file:${user.home}/.java.policy

Security Policy Files

```
grant codesource {
   permission1;
   permission2;
    .  .  .
};
```

- The code source contains
  - a code base (which can be omitted if the entry applies to code from all sources)
  - and the names of trusted principals
  - and certificate signers (which can be omitted if signatures are not required for this entry).

Security Policy Files

- code base

```
grant codeBase "www.horstmann.com/classes/" { . . . };
grant codeBase "www.horstmann.com/classes/MyApp.jar"
                   { . . . };
grant codeBase "file:C:/myapps/classes/" { . . . };
```

- Permission

```
 permission className targetName, actionList;
```

Security Policy Files

- An example
  - GetProps.java

```
class GetProps {
  public static void main(String[] args) {
       String s;
       try {
            s = System.getProperty("java.version", "not specified");
                       System.out.println("  The version of your Java is: " + s);
            s = System.getProperty("os.name", "not specified");
                       System.out.println("  The name of your os is: " + s);
            s = System.getProperty("java.home", "not specified");
                       System.out.println("  Your JRE directory is: " + s);
        } catch (Exception e) {
            System.err.println("Caught exception " + e.toString());
        }
 }
```

Security Policy Files

- An example
  - policy file

```
grant {
  permission java.util.PropertyPermission "java.version", "read";
  permission java.util.PropertyPermission "os.name", "read";
  permission java.util.PropertyPermission "java.home", "read";
};
```

  - Command line

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

  - If the java.home permission is deleted, an exception will be thrown

Custom Permission

- To implement your permission class, you extend the Permission class and supply the following methods:
  - A constructor with two String parameters, for the target and the action list
  - String getActions()
  - boolean equals()
  - int hashCode()
  - boolean implies(Permission other)

```
p1 = new FilePermission("/tmp/-", "read, write");
p2 = new FilePermission("/tmp/-", "read");
p3 = new FilePermission("/tmp/aFile", "read, write");
p4 = new FilePermission("/tmp/aDirectory/-", "write");
```

- a file permission p1 implies another file permission p2 if
  - The target file set of p1 contains the target file set of p2.
  - The action set of p1 contains the action set of p2.

Custom Permission

- we implement a new permission for monitoring the insertion of text into a text area. The program ensures that you cannot add "bad words" such as sex, drugs, and C++ into a text area.

```
class WordCheckTextArea extends JTextArea {
 public void append(String text) {
   WordCheckPermission p = new WordCheckPermission(text, "insert");
   SecurityManager manager = System.getSecurityManager();
   if (manager != null)
       manager.checkPermission(p);
   super.append(text);
 }
}
```

Custom Permission

```
grant {
    permission WordCheckPermission "sex,drugs,C++", "avoid";
};
```

- If p1 has action avoid and p2 has action insert, then the target of p2 must avoid all words in p1.
  - `WordCheckPermission "sex,drugs,C++", "avoid"`
  - implies the permission
  - `WordCheckPermission "Mary had a little lamb", "insert"`
- If p1 and p2 both have action avoid, then the word set of p2 must contain all words in the word set of p1.
  - `WordCheckPermission "sex,drugs", "avoid"`
  - implies the permission
  - `WordCheckPermission "sex,drugs,C++", "avoid"`
- If p1 and p2 both have action insert, then the text of p1 must contain the text of p2.
  - `WordCheckPermission "Mary had a little lamb", "insert"`
  - implies the permission
  - `WordCheckPermission "a little lamb", "insert"`

Custom Permission

```
import java.security.*;
 import java.util.*;
/** A permission that checks for bad words.
 */
 public class WordCheckPermission extends Permission
 {
    /**
       Constructs a word check permission
       @param target a comma separated word list
       @param anAction "insert" or "avoid"
    */
    public WordCheckPermission(String target, String anAction)
    {
       super(target);
       action = anAction;
    }
```

Custom Permission

```
public String getActions() { return action; }

public boolean equals(Object other)
{
   if (other == null) return false;
   if (!getClass().equals(other.getClass())) return false;
   WordCheckPermission b = (WordCheckPermission) other;
   if (!action.equals(b.action)) return false;
   if (action.equals("insert"))
      return getName().equals(b.getName());
   else if (action.equals("avoid"))
      return badWordSet().equals(b.badWordSet());
   else return false;
}
```

Custom Permission

```
public int hashCode()
{
   return getName().hashCode() + action.hashCode();
}

public boolean implies(Permission other)
{
   if (!(other instanceof WordCheckPermission)) return false;
   WordCheckPermission b = (WordCheckPermission) other;
   if (action.equals("insert"))
   {
      return b.action.equals("insert") &&
         getName().indexOf(b.getName()) >= 0;
   }
   else if (action.equals("avoid"))
   {
```

Custom Permission

```
      if (b.action.equals("avoid"))
         return b.badWordSet().containsAll(badWordSet());
      else if (b.action.equals("insert"))
      {
         for (String badWord : badWordSet())
            if (b.getName().indexOf(badWord) >= 0)
               return false;
         return true;
      }
      else return false;
   }
   else return false;
}
```

Custom Permission

```
    /**
      Gets the bad words that this permission rule describes.
      @return a set of the bad words
   */
   public Set<String> badWordSet()
   {
      Set<String> set = new HashSet<String>();
      set.addAll(Arrays.asList(getName().split(",")));
      return set;
   }

   private String action;
}
```

References

- Core Java (volume II) 9th edition
  - http://horstmann.com/corejava.html
- The Java EE 7 Tutorial
  - http://docs.oracle.com/javaee/7/tutorial/doc/javaeetutorial7.pdf

REin

Thank You!