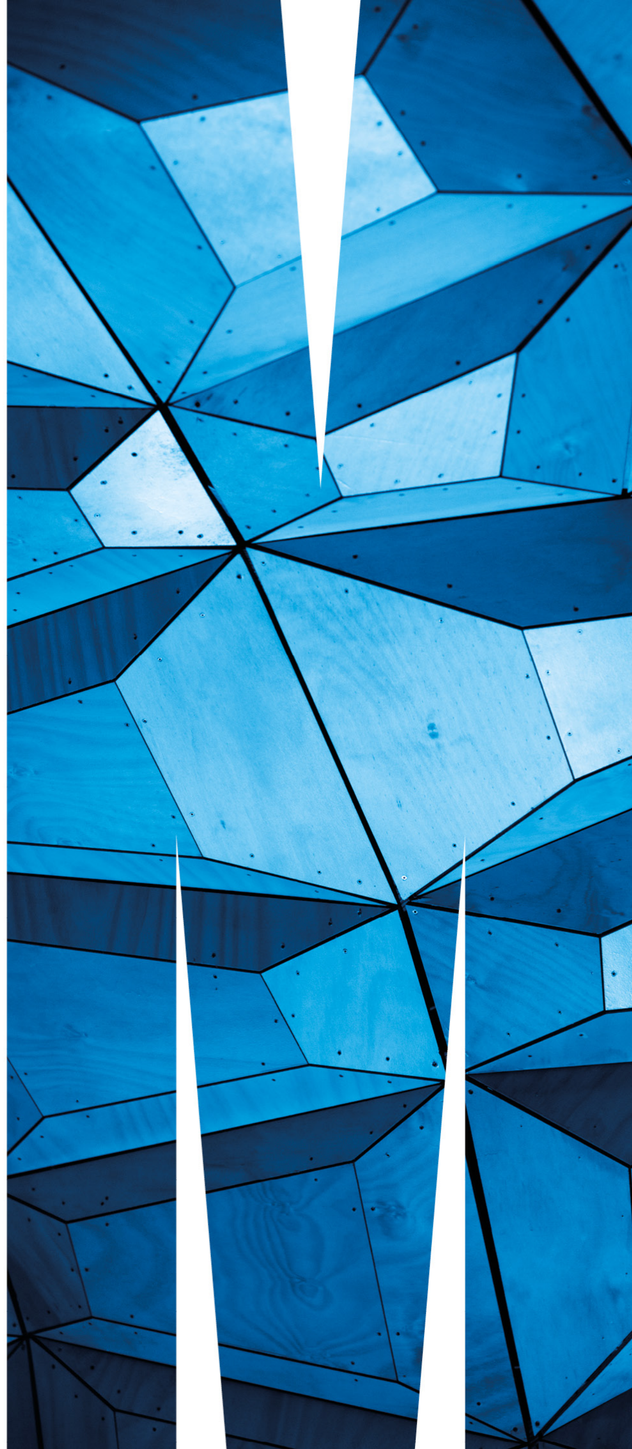
MONASH University

FIT5003: Introduction to Java Security

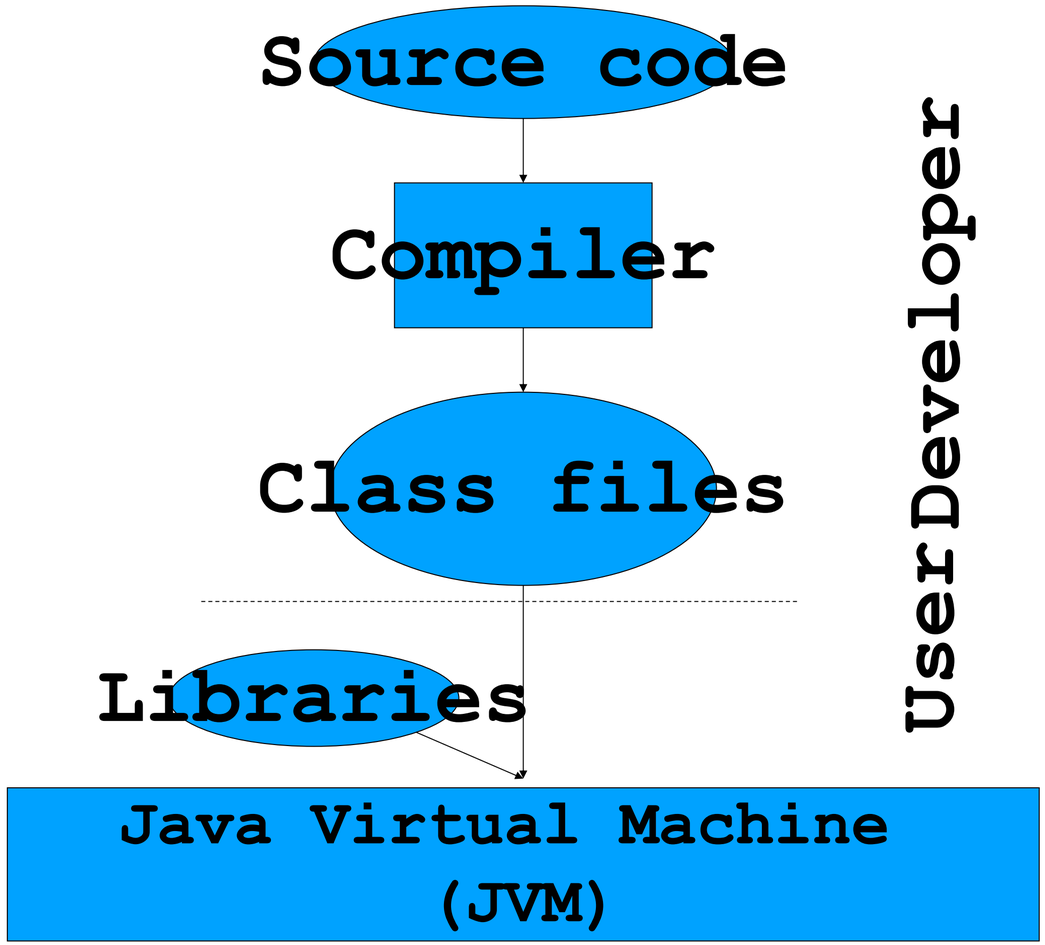
Dr Xiaoning Du
Department of Software Systems and Cybersecurity
Faculty of Information Technology



Learning Outcomes of This Lecture

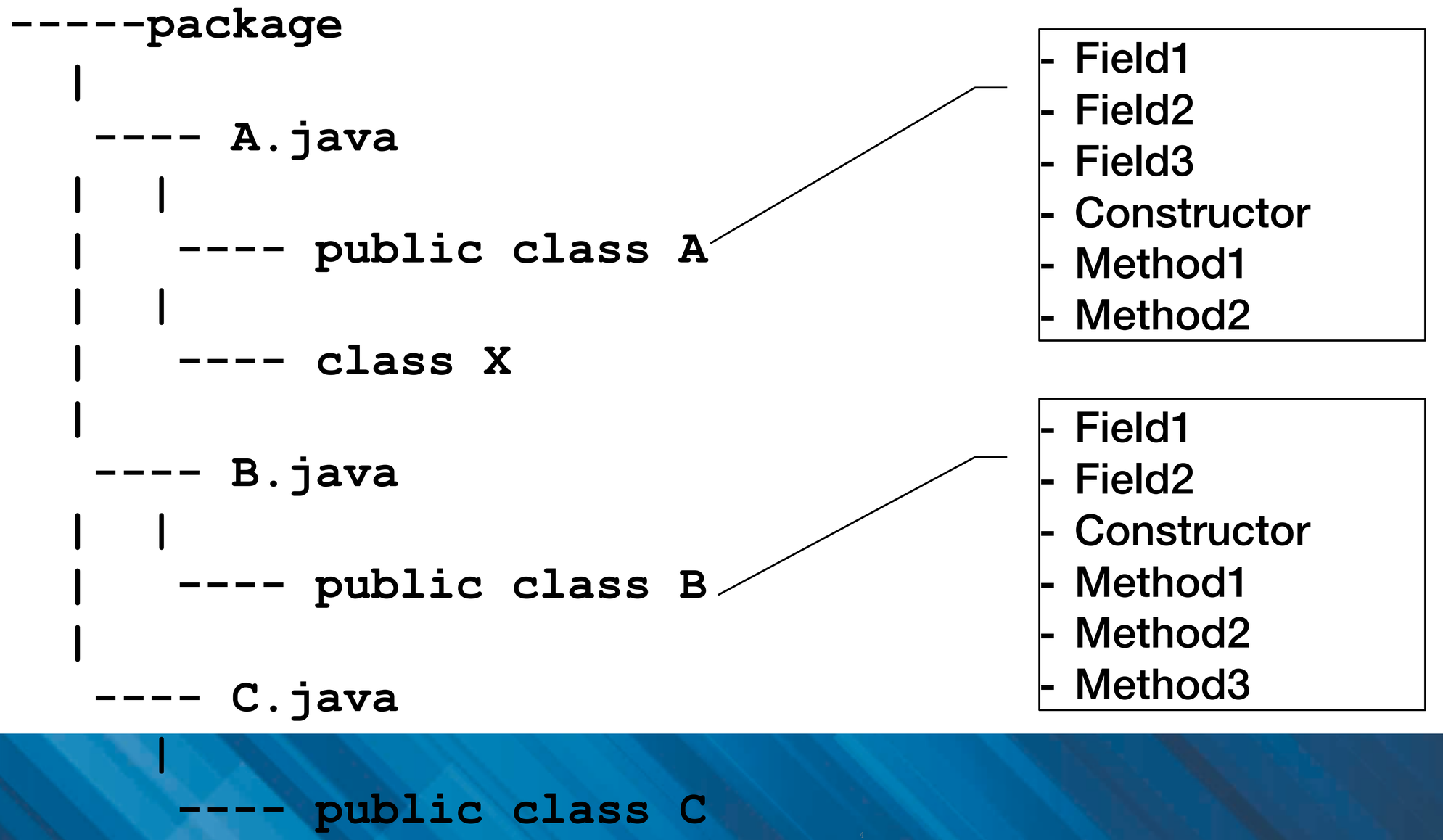
- Understand basic Java structures
- Learn about the Java Security Architecture and the Security Manager
- Cryptography Java library (with examples)
- Keystore and Keytool functionality in Java
- Java Vulnerabilities/exploits using Java reflection

Java Basics



- object-oriented
- strongly typed
- garbage collection
  - objects no longer in use are removed automatically from memory
- exception handling
- very similar to C/C++, but *cleaner* and *simpler*
  - no more pointers
  - no more struct and union
  - no more (stand alone) functions
  - no more multiple inheritance
  - no more operator overloading

Java Basics



Java Basics

Access Control Modifier	Class Accessibility	Member (Field or Method) Accessibility
Public	All	All if class is accessible
Protected	N/A	Same package OR subclass
“default” (Package private)	Same package	Same package
Private	N/A	Only same class (not subclass)

Java Basics

CreditCard.class
Test.class

```
public class CreditCard {
    public String acctNo = "1234";
}

public class Test {
    public static void main(String[] args) {
        CreditCard cc = new CreditCard();
        System.out.println("account is " +
cc.acctNo);
    }
}
```

```
javac CreditCard.java
javac Test.java

java -cp . Test
```

```
MU00106169X:JavaExamples clii0040$ java -cp . Test
account is 1234
```

Java Basics

```
public class CreditCard {
    public String acctNo = "1234";
}
```

private

```
javac CreditCard.java
```

CreditCard.class

```
java -cp . Test
```

```
MU00106169X:JavaExamples clii0040$ java -cp . Test
Exception in thread "main" java.lang.IllegalAccessError:
tried to access field CreditCard.acctNo from class Test
    at Test.main(Test.java:4)
```

Java Basics

```
public class CreditCard {
    public void transfer500AUD () {
        transfer(500);
    }
}
```

5

We should never assume that third-party code is secure.

Java Security

Why?

Security

When running unknown Java programs, e.g., may contain malicious payloads, we need to control the execution of the programs.

Java Security

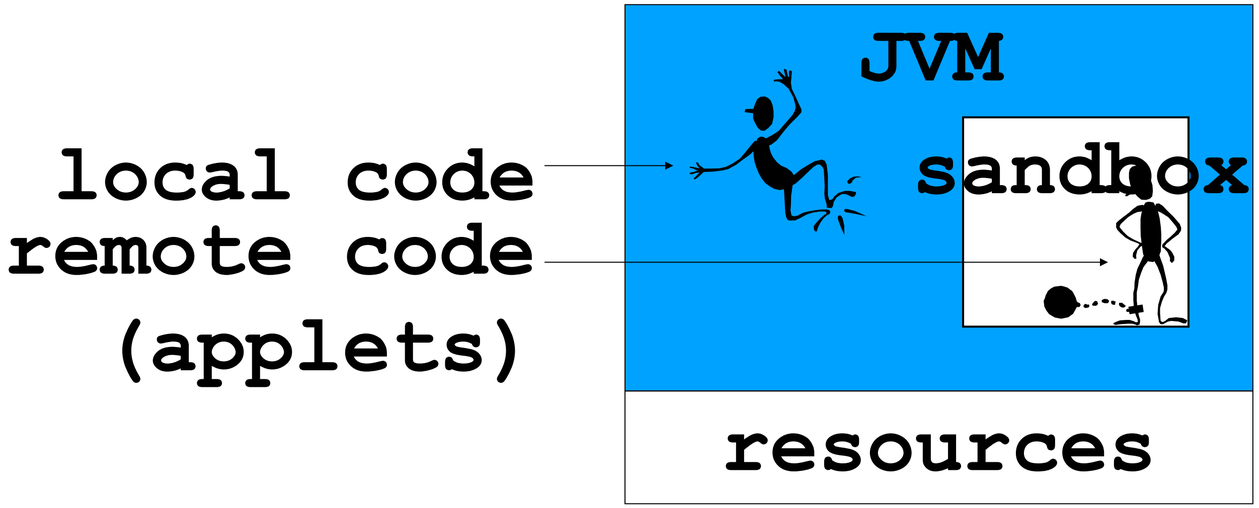
Java Technology uses three mechanisms to ensure safety.

- Language design features(bounds checking on arrays, legal type conversions etc):
  - *The Java language is designed to enforce type safety. Anything in Java happens inside an object and each object is an instance of a class.*
  - *To implement the type safety enforcement, each object, before usage, needs to be allocated. Java allows usage of primitive types but only inside properly allocated objects.*

- An access control mechanism that controls what the code can do(file access, network access etc).
- Code signing: code authors can use standard cryptographic algorithms to authenticate java programming language code. Users of the code can determine who created the code and whether the code is altered or not after it was signed.

Java Security (History)

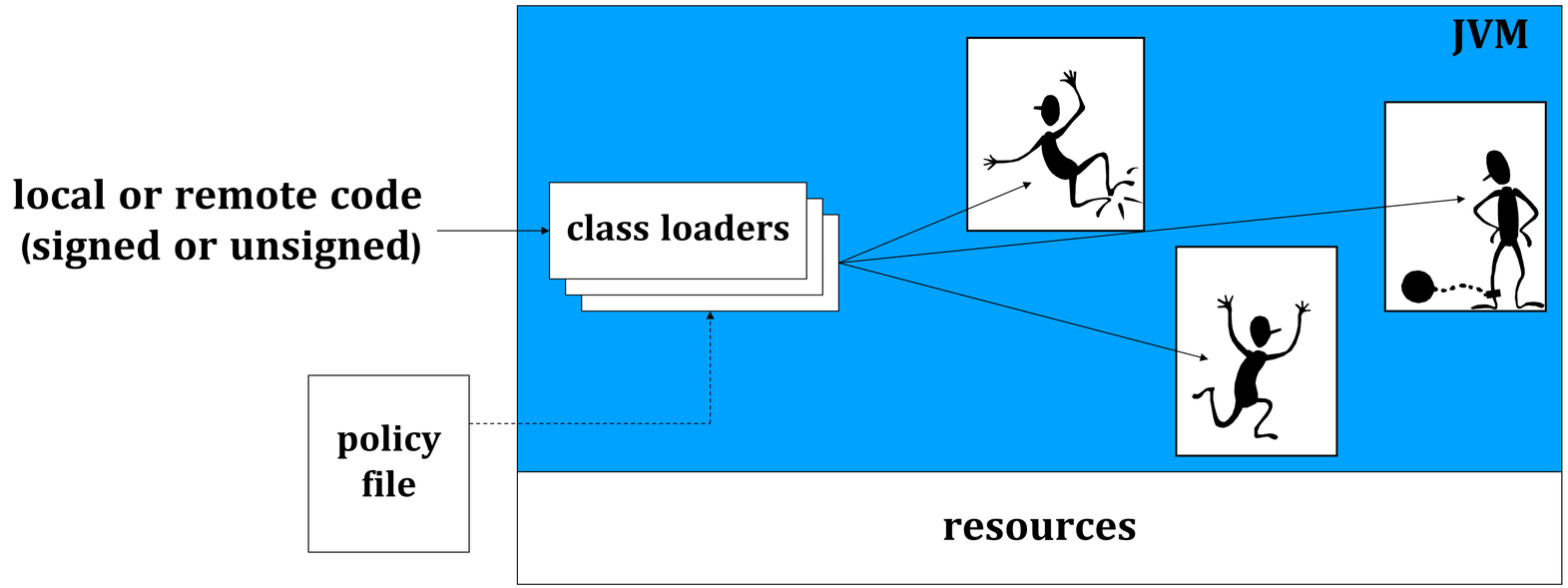
- A Java applet is a small application in Java bytecode. It is a self-contained application and can be executed on any JVM residing on any device
- limit the resources that can be accessed by applets



https://docs.oracle.com/javase/6/docs/technotes/guides/security/spec/security-spec.doc2.html

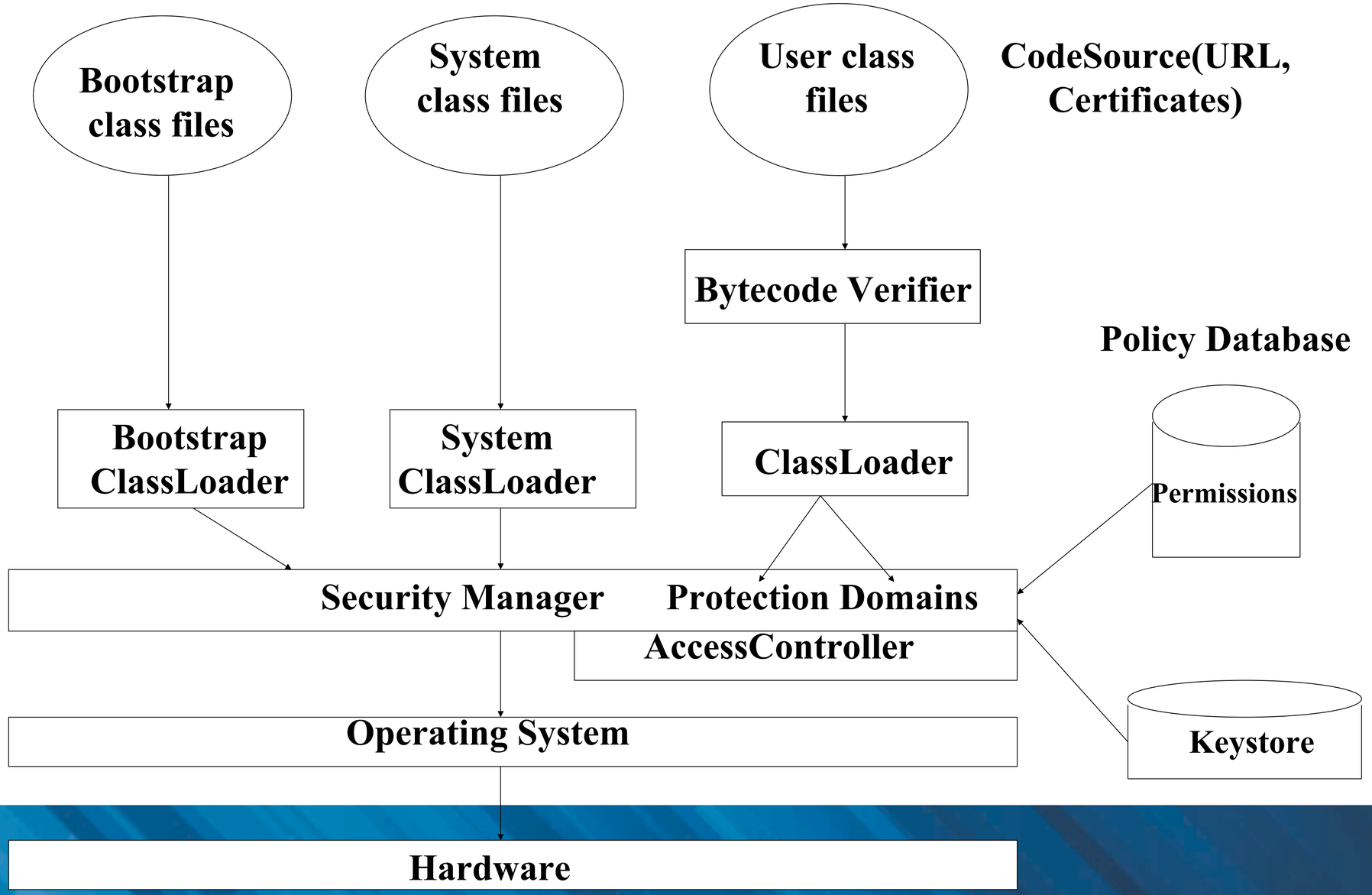
Java Security (History)

every code (remote or local) has access to the system resources based on what is defined in a *policy file*



The class loader is where you can place security features, restrictions, or access control.

Java Security Architecture



Java Security: Security Manager

- The class java.lang.SecurityManager is the focal point of authorization.
- Security Manager is concrete class, with a public constructor and appropriate checks in place to ensure that it can be invoked in an authorized manner.
- It consists of a number of check methods, e.g,: CheckPermission method is used to check to see if the requested access has the permission based on policy.

Java Security: Security Manager

- the JVM allows only one SM to be active at a time
- there is a default SM provided by the JDK
- Java programs (applications, applets, beans, …) can replace the default SM by their own SM only if they have permission to do so
- invoking the SecurityManager constructor or the setSecurityManager() method will call the checkPermissions() method of the current SM and verify if the caller has the needed permissions

Java Security: Bootstrap Class Loader

The bootstrap classloader is platform specific machine instructions that kick off the whole classloading process.

Bootstrap classes - Classes that comprise the Java platform, including the classes in rt.jar and several other important jar files in $JAVA_HOME/jre/lib

The bootstrap classloader also takes care of loading all of the code needed to support the basic Java Runtime Environment (JRE), including classes in the java.util and the java.lang packages.

Java Security: System Class Loader

classes from the system class path, which are set by the CLASSPATH environment variable

CLASSPATH is a parameter in the JVM that specifies the location of user-defined classes and packages

By default, the CLASSPATH is the current working directory (.)

```
java -Djava.system.class.loader=com.test.MyClassLoader MyApplication
```

Java Security: Class Loader

- Is an important part in **security chain** and loads java byte codes into the JVM.
- It works in conjunction with the **security manager and access controller** to enforce security rules.
- It is involved in enforcing some security decisions earlier in an object's lifetime than the security manager.
- Information about the URL from which the code is originated, and the code's signers is initially available to the ClassLoader.

Java Security: Class Loader

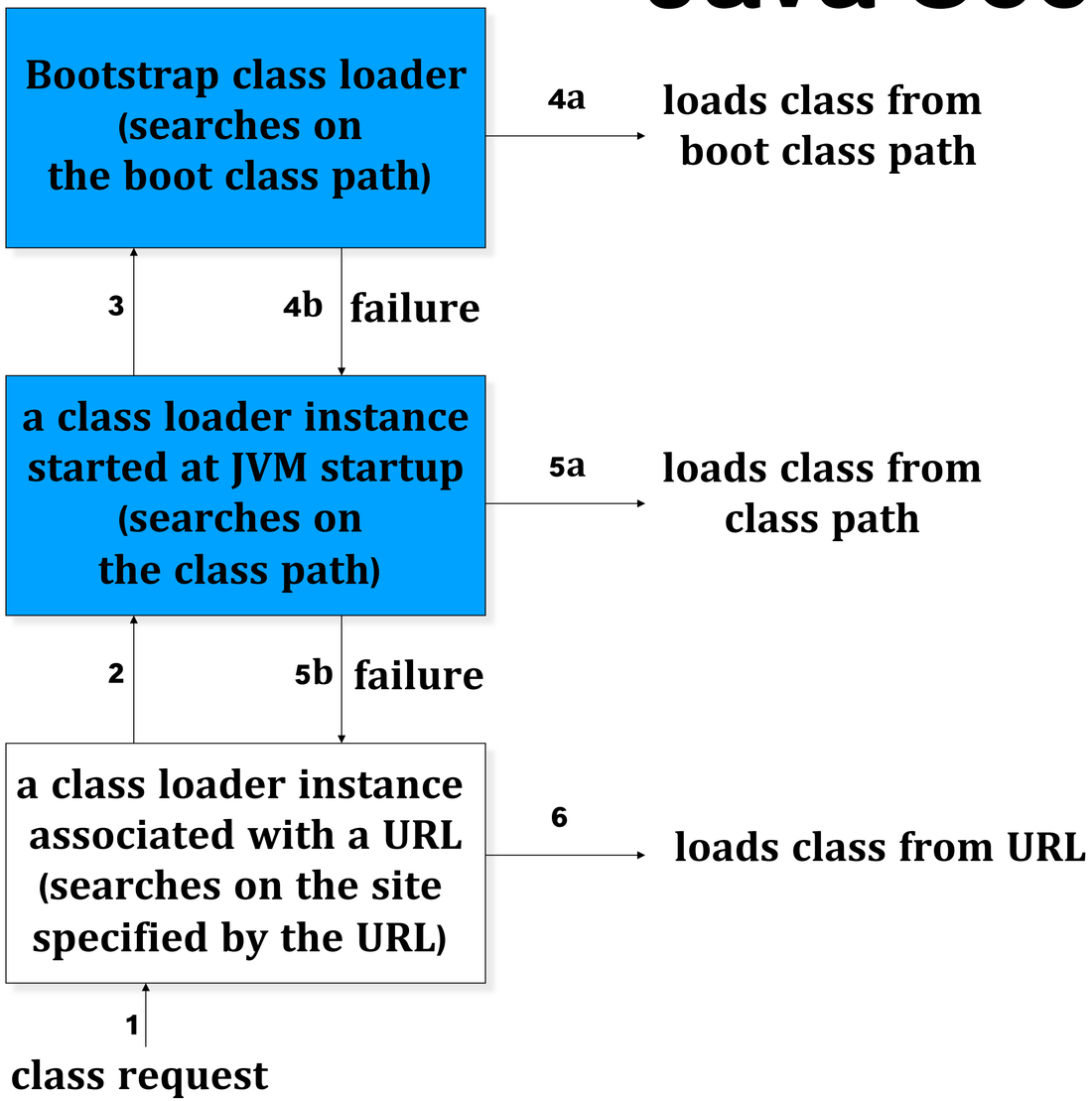
- java.security.SecureClassLoader provides security features beyond the standard Java2 security model, and you can customize a ClassLoader from it.

- ClassLoader loads classes into VM and is responsible for the namespaces at runtime. Namespaces as identically named identifiers can reference different objects.

    1. Classes loaded by a class loader instance belong to the same name space.
    2. Since classes with the same name may exist on different Web sites, different Web sites are handled by different instances of the applet class loader.
    3. A class in one namespace cannot access a class in another namespace, e.g., classes from different Web sites cannot access each other.

- Enforce a search order that prevents trusted system classes from being replaced by classes from less trusted sources.

Java Security: Class Loader



JVM: invokes the class loader associated with the requesting program

class loader: has the class already been loaded?

yes:

does the program have permission to access the class?

yes: return object reference

no: security exception

no:

does the program have permission to create the requested class?

yes:

first delegate loading task to parent

if parent returns success, then return (class is loaded)

if parent returned failure, then load class and return

no: security exception

Java Security: Byte Code Verifier

- Checks a classfile for validity:
- Code should have only valid instructions and register use.
- Code does not overflow/underflow stack.
- Does not convert data types illegally.
- Accesses objects correct types.
- Method calls use correct number and types of parameters.
- References to other classes use legal names.

Java Security: CodeSource

- Java Code is downloaded over a network, so the verification of author is critical to maintain a secure environment.
- The object java.security.CodeSource describes a piece of code from three aspects:
  - -The code's origin, which is specified as an URL.
  - -The code's signature, which is hash of the code, encrypted with the private key
  - -The digital certificates containing public keys corresponding to the set of private keys are used to sign the code

Java Security: Keystore

- Keystore is a password-protected database that holds private keys and certificates.
- The password is selected at the time of creation.
- Each database entry can be guarded by its own password for extra security.
- Certificates accepted into the keystore are considered to be trusted.

Java Security: Keystore

Keystore

```
keytool -genkey -alias mykey -keyalg RSA -
keypass changeit -keystore keystore.jks -
storepass changeit
```

Java Security: Security Policy

- SecureClassLoader assigns permissions when loading classes, by asking policy object to look up the permissions for the code source of each class.

- Own Policy class can be installed to carry out mapping from code sources to permissions.

Java Security: Permission

- Permission classes represent access to various system resources such as files, sockets and so on.
- A collection of permissions constructs a custom security policy for an installation.
- Permission classes represent approvals, but not denials.
- Granted permissions is also called "privileges"

Access Policy

The policy file(s) specify what permissions are allowed for code from a specified code source, and executed by a specified principal.

```
$ /usr/libexec/java_home
/Library/Java/JavaVirtualMachines/j
dk1.8.0_152.jdk/Contents/Home
```

$JAVA_HOME/jre/lib/security/java.security
$JAVA_HOME/jre/lib/security/java.policy

Access Policy

$JAVA_HOME/jre/lib/security/java.security

```
policy.url.1=file:${java.home}/lib/security/
                    java.policy
policy.url.2=file:${user.home}/.java.policy
```

$JAVA_HOME/jre/lib/security/java.policy

```
grant codeBase "file:${{java.ext.dirs}}/*" {
  permission java.security.AllPermission;
};
grant {
  permission java.util.PropertyPermission
"java.version", "read";
};
```

Access Policy

dir/ : all .class files, does not include .jar files
dir/* : all .class and .jar files, does not include sub-dir files
dir/- : all .class and .jar files, including sub-dir ones

- **The permission is not granted if not configured**
- **The permission can only be configured so as to be granted, cannot be disabled**
- **The same permission can have multiple configurations. The union results will be recognized.**

Access Policy

```
import java.io.*;
import java.security.*;
public class TestPolicy {
    public static void main(String[] args) {
        String javaVersion=System.getProperty("java.version");
        System.err.println(javaVersion);

        System.setProperty("java.version","1.7.0_45");
        String javaNewVersion=System.getProperty("java.version");
        System.err.println(javaNewVersion);
    }
}
```

```
java TestPolicy
```

```
1.8.0_152
1.7.0_45
```

Security Manager is disabled by default!

Access Policy

```
import java.io.*;
import java.security.*;
public class TestPolicy {
    public static void main(String[] args) {
        String javaVersion=System.getProperty("java.version");
        System.err.println(javaVersion);

        System.setProperty("java.version","1.7.0_45");
        String javaNewVersion=System.getProperty("java.version");
        System.err.println(javaNewVersion);
    }
}
```

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

```
1.8.0_152
Exception in thread "main" java.security.AccessControlException: access denied ("java.util.Pr
opertyPermission" "java.version" "write")
        at java.security.AccessControlContext.checkPermission(AccessControlContext.java:472)
        at java.security.AccessController.checkPermission(AccessController.java:884)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:549)
        at java.lang.System.setProperty(System.java:792)
        at TestPolicy.main(TestPolicy.java:7)
```

Access Policy

$JAVA_HOME/jre/lib/security/java.policy

```
grant codeBase "file:${{java.ext.dirs}}/*" {
    permission java.security.AllPermission;
};
grant {
permission java.util.PropertyPermission "java.version",
"read";
};
```

No "write" permission

java -Djava.security.manager TestPolicy

```
MU00106169X:JavaExamples clii0040$ java -Djava.security.manager TestPolicy
1.8.0_152
Exception in thread "main" java.security.AccessControlException: access denied ("java.util.Pr
opertyPermission" "java.version" "write")
        at java.security.AccessControlContext.checkPermission(AccessControlContext.java:472)
        at java.security.AccessController.checkPermission(AccessController.java:884)
        at java.lang.SecurityManager.checkPermission(SecurityManager.java:549)
        at java.lang.System.setProperty(System.java:792)
        at TestPolicy.main(TestPolicy.java:7)
```

Access Policy

test.policy

```
grant {
 permission java.util.PropertyPermission "java.version", "read";
 permission java.util.PropertyPermission "java.version", "write";
};
```

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

```
MU00106169X:JavaExamples cliii0040$ java -Djava.security.manager
-Djava.security.policy=test.policy TestPolicy
1.8.0_152
1.7.0_45
```

Access Controller

- To decide whether access to a critical system resource should be allowed or denied, based on the security policy currently in effect.
- To mark code as privileged, thus affecting subsequent access determinations.
- To obtain a snapshot of the current calling context, so access-control decisions from a different context can be made with respect to the saved context.

Access Controller

Java Access Control Model

Stack-based access control; SecurityManager checks all frames

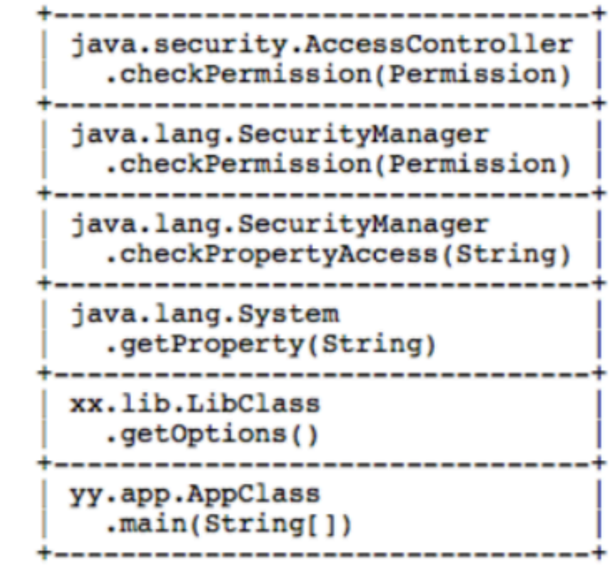
```
package xx.lib;

public class LibClass {
    private static final String OPTIONS = "xx.lib.options";

    public static String getOptions() {
        // checked by SecurityManager
        return System.getProperty(OPTIONS);
    }
}

package yy.app;

class AppClass {
    public static void main(String[] args) {
        System.out.println(
            xx.lib.LibClass.getOptions()
        );
    }
}
```



Access Controller

Java Access Control Model

Application and library have permission to read a file

```
package xx.lib;

public class LibClass {
    private static final String OPTIONS = "xx.lib.options";

    public static String getOptions() {
        // checked by SecurityManager
        return System.getProperty(OPTIONS);
    }
}

package yy.app;

class AppClass {
    public static void main(String[] args) {
        System.out.println(
            xx.lib.LibClass.getOptions()
        );
    }
}
```

Permission to read a file

Permission to read a file

```
+-------------------------------+
| java.security.AccessController |
|   .checkPermission(Permission) |
+-------------------------------+
| java.lang.SecurityManager      |
|   .checkPermission(Permission) |
+-------------------------------+
| java.lang.SecurityManager      |
|   .checkPropertyAccess(String) |
+-------------------------------+
| java.lang.System               |
|   .getProperty(String)         |
+-------------------------------+
| xx.lib.LibClass                |
|   .getOptions()                |
+-------------------------------+
| yy.app.AppClass                |
|   .main(String[])              |
+-------------------------------+
```

Access Controller

Java Access Control Model

Library has permissions to read the file but applications don't

```
package xx.lib;

public class LibClass {
    private static final String OPTIONS = "xx.lib.options";

    public static String getOptions() {
        // checked by SecurityManager
        return System.getProperty(OPTIONS);
    }
}

package yy.app;

class AppClass {
    public static void main(String[] args) {
        System.out.println(
            xx.lib.LibClass.getOptions()
        );
    }
}
```

Permission to read a file

No permission to read a file

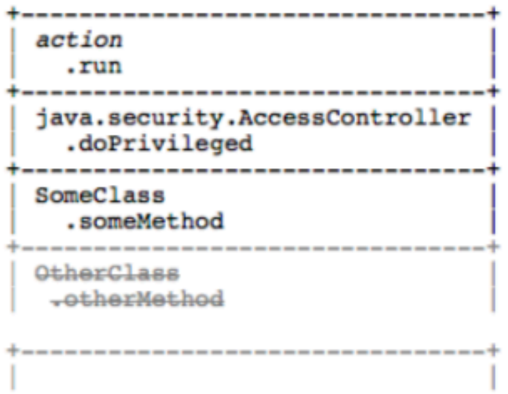
```
+-------------------------------+
| java.security.AccessController |
|   .checkPermission(Permission) |
+-------------------------------+
| java.lang.SecurityManager      |
|   .checkPermission(Permission) |
+-------------------------------+
| java.lang.SecurityManager      |
|   .checkPropertyAccess(String) |
+-------------------------------+
| java.lang.System               |
|   .getProperty(String)         |
+-------------------------------+
| xx.lib.LibClass                |
|   .getOptions()                |
+-------------------------------+
| yy.app.AppClass                |
|   .main(String[])              |
+-------------------------------+
```

Access Controller

Stack Checks Are Modified In Some Cases

AccessController.doPrivileged

- Truncates the SecurityManager checks to that of the immediate caller of doPrivileged

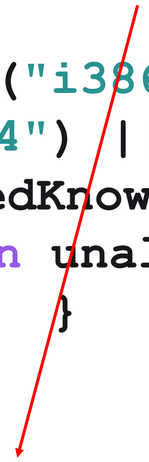


Access Controller

```
static boolean unaligned() {
    if (unalignedKnown)
      return unaligned;

  String arch =
new sun.security.action.GetPropertyAction("os.arch" );

unaligned = arch.equals("i386") || arch.equals("x86")
   || arch.equals("amd64") || arch.equals("x86_64");
            unalignedKnown = true;
            return unaligned;
                }
```



get_property permission

Access Controller

```
static boolean unaligned() {
    if (unalignedKnown)
      return unaligned;

  String arch = AccessController.doPrivileged(
new sun.security.action.GetPropertyAction("os.arch"));

unaligned = arch.equals("i386") || arch.equals("x86")
   || arch.equals("amd64") || arch.equals("x86_64");
            unalignedKnown = true;
            return unaligned;
            }
```

get_property permission

Java Security

Security Scenarios?

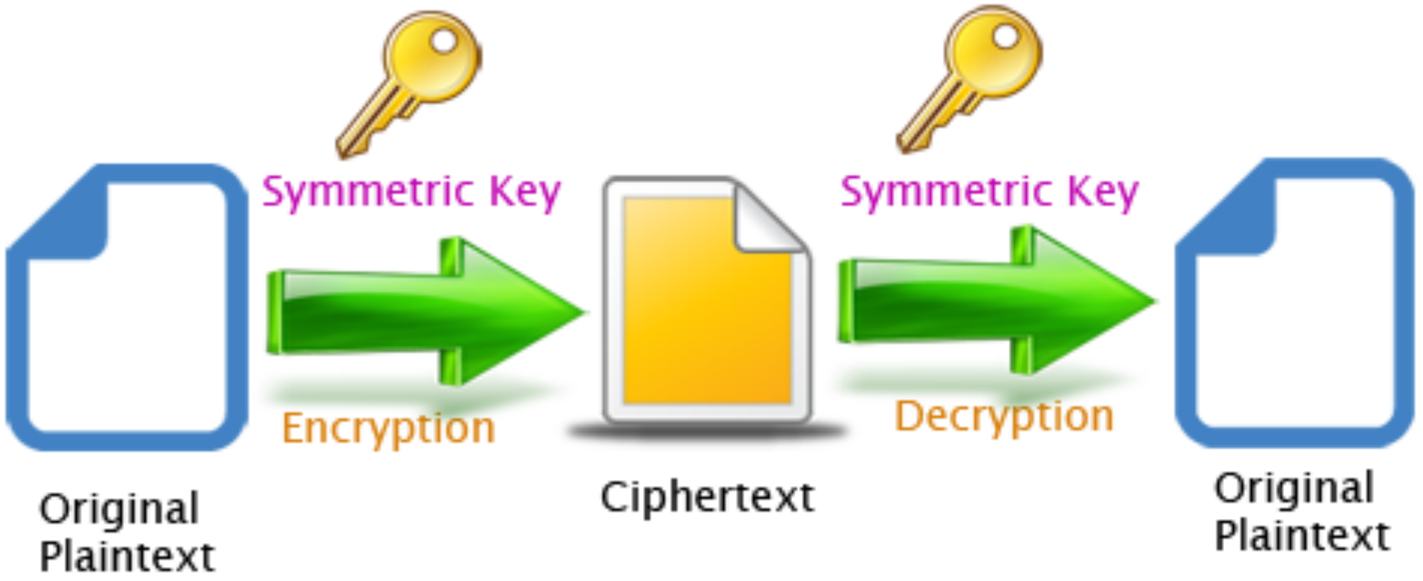
Java Cryptography

Symmetric Key Encryption

Public/Private Key Cryptography

Keytool & KeyStore

Symmetric Key Encryption



Symmetric Key Encryption

Generating Key

```
KeyGenerator keyGenerator =
KeyGenerator.getInstance("AES");
keyGenerator.init(128);
SecretKey key = keyGenerator.generateKey();
```

AES: Advanced Encryption Standard
DES: Data Encryption Standard
BlowFish
RC4

Symmetric Key Encryption

Generating Key

```
KeyGenerator keyGenerator =
KeyGenerator.getInstance("AES");
keyGenerator.init(128);
SecretKey key = keyGenerator.generateKey();
```

Encrypt

```
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, key);
byte[] buf = cipher.doFinal("xyz".getBytes());
System.out.println(new String(buf));
```

Symmetric Key Encryption

Generating Key

```
KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
keyGenerator.init(128);
SecretKey key = keyGenerator.generateKey();
```

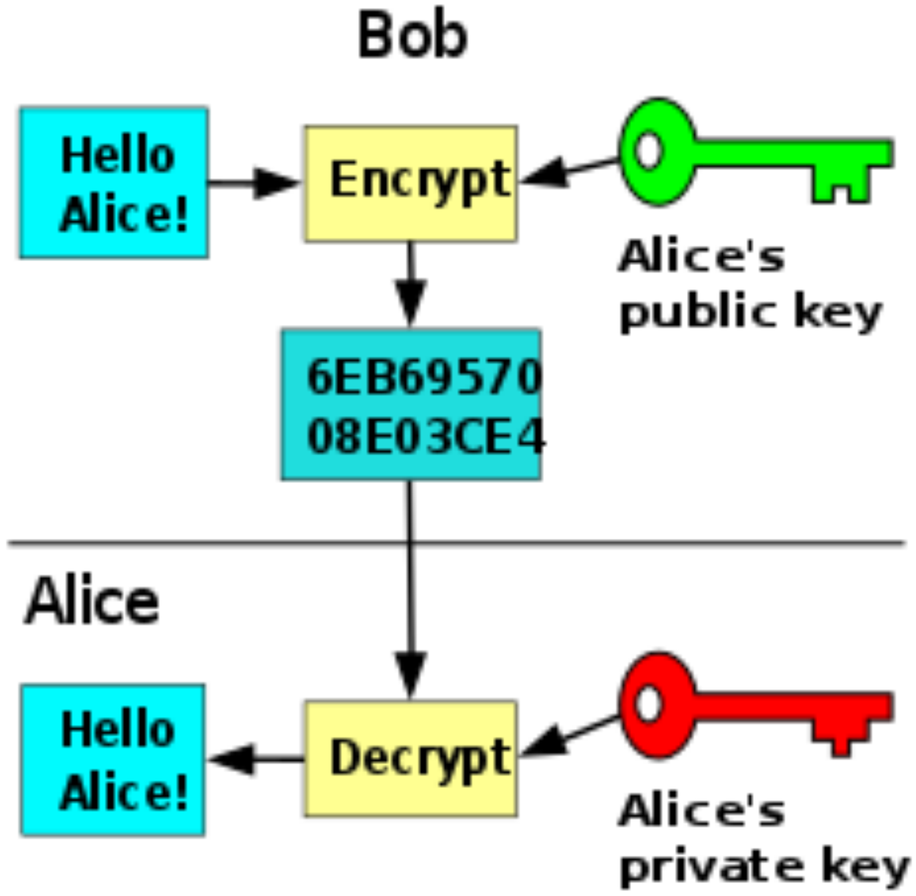
Encrypt

```
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, key);
byte[] buf = cipher.doFinal("xyz".getBytes());
System.out.println(new String(buf));
```

Decrypt

```
Cipher cipher2 = Cipher.getInstance("AES");
cipher2.init(Cipher.DECRYPT_MODE, key);
byte[] buf2 = cipher2.doFinal(buf);
System.out.println(new String(buf2));
```

Public Key Encryption



Public Key Encryption

Generating Key

```
KeyPairGenerator keyGen =
KeyPairGenerator.getInstance("RSA");
keyGen.initialize(512);
KeyPair pair = keyGen.generateKeyPair();
PublicKey publicKey = pair.getPublic();
PrivateKey privateKey = pair.getPrivate();
```

RSA: most common public key, based on factoring large primes

Diffie-Hellman
Elliptic Curve

Public Key Encryption

```
KeyPairGenerator keyGen =
KeyPairGenerator.getInstance("RSA");
keyGen.initialize(512);
KeyPair pair = keyGen.generateKeyPair();
PublicKey publicKey = pair.getPublic();
PrivateKey privateKey = pair.getPrivate();
```

Generating Key

```
Cipher cipher = Cipher.getInstance(“RSA");
cipher.init(Cipher.ENCRYPT_MODE, publickey);
byte[] buf = cipher.doFinal("xyz".getBytes());
System.out.println(new String(buf));
```

publicKey

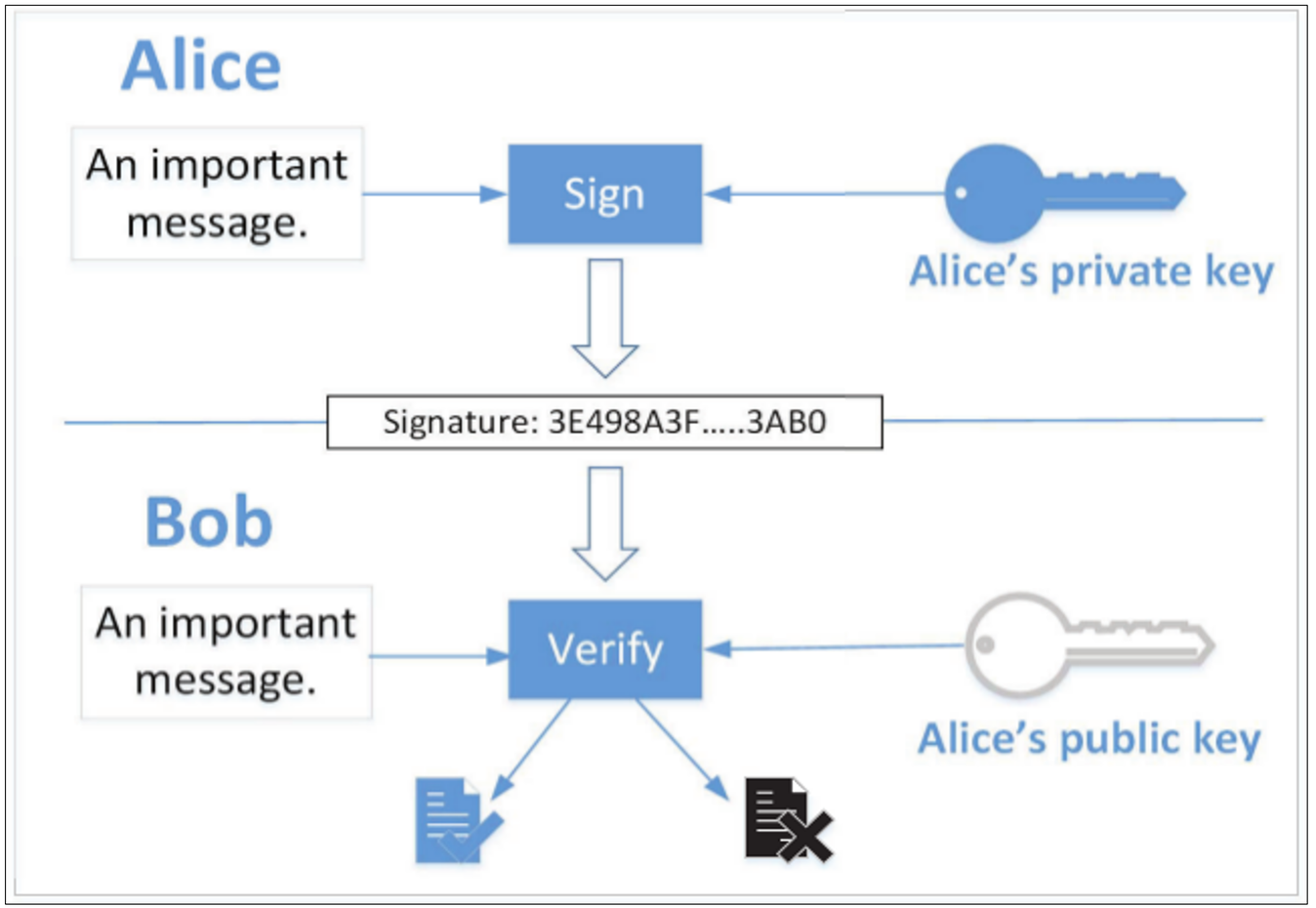
Encrypt

```
Cipher cipher2 = Cipher.getInstance(“RSA");
cipher2.init(Cipher.DECRYPT_MODE, privatekey);
byte[] buf2 = cipher2.doFinal(buf);
System.out.println(new String(buf2));
```

privateKey

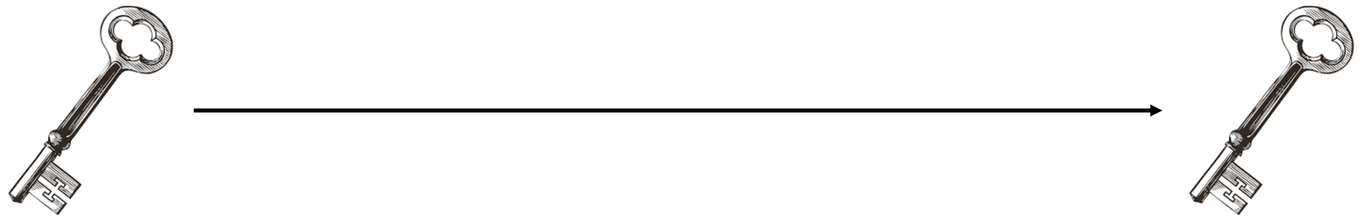
Decrypt

Digital Signature



Public Key as Certificate

How to make sure the public keys are real?

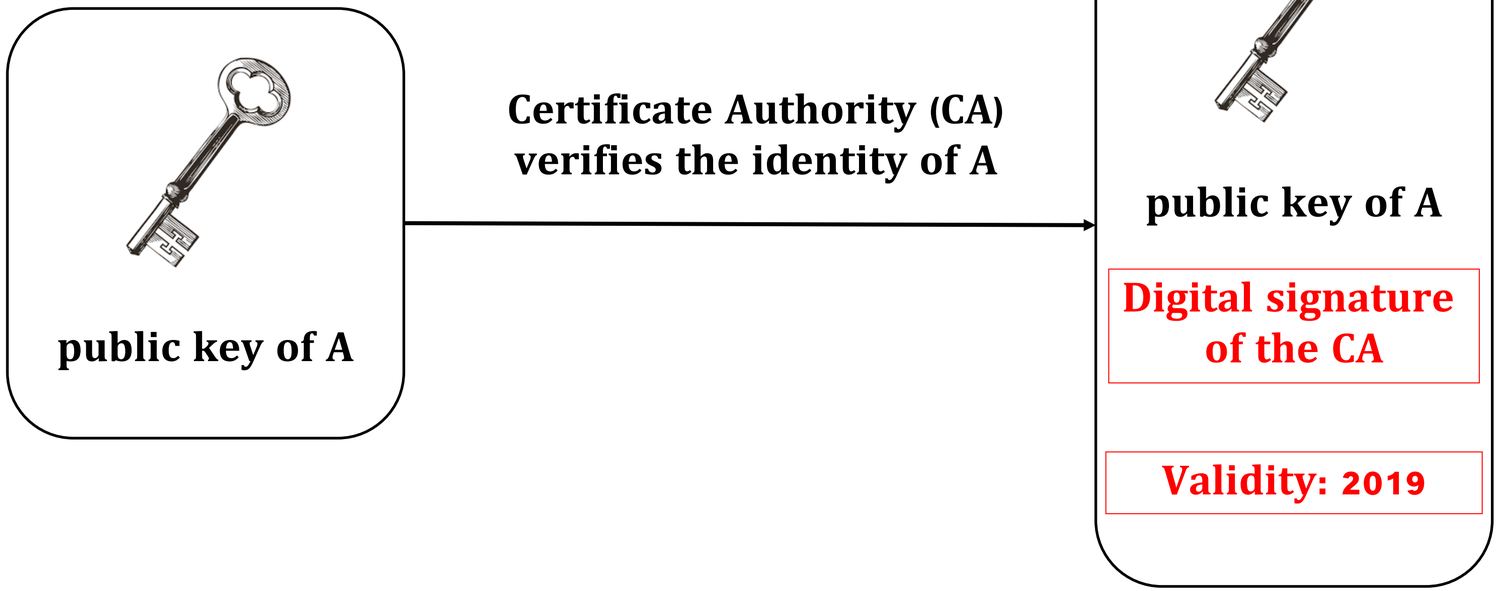


public key of A **B**

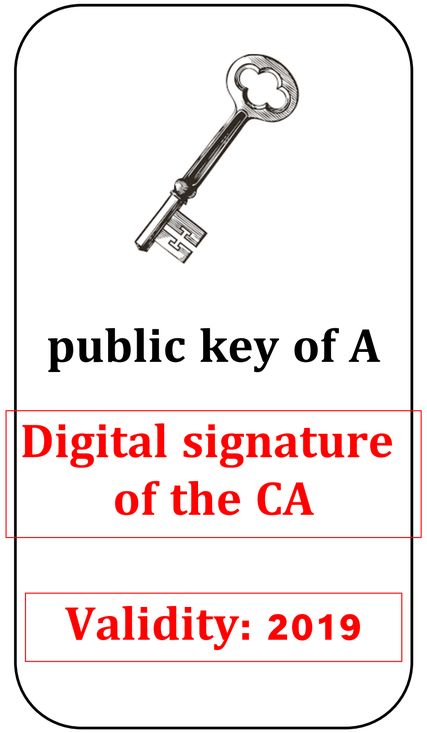
Public Key as Certificate

How to make sure the public keys are real?

Certificate



Public Key as Certificate

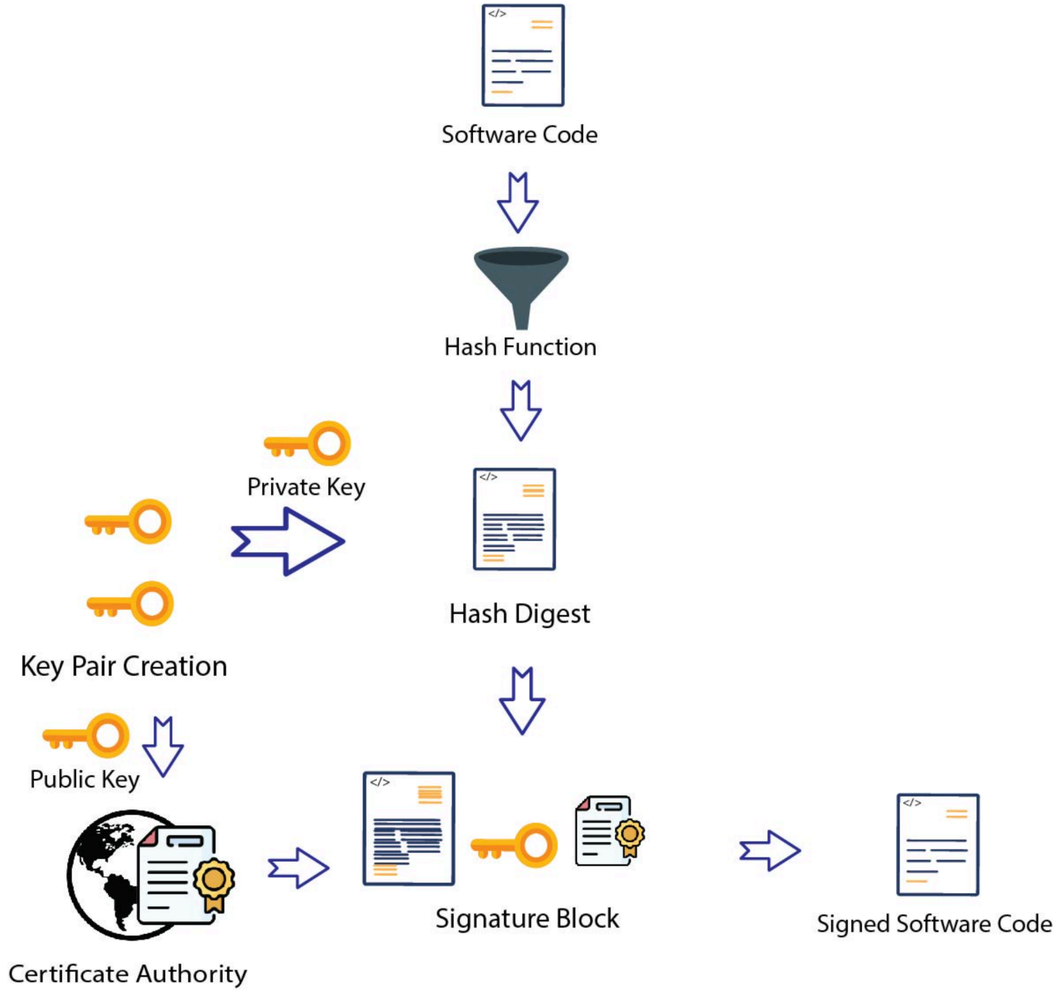



```
private static boolean test(
        Certificate target,
        Certificate signer) {

PublicKey pubKey =signer.getPublicKey();
        target.verify(pubKey);
        }
```

Code Signing

- A one-way hash of the document is produced.
- The hash is encrypted with the private key, thereby signing the document.
- The document, the signed hash and the certificate are transmitted.
- The recipient checks the authenticity of the code signing certificate.
- The recipient produces a one-way hash of the document.
- Using the digital signature algorithm, the recipient decrypts the signed hash with the sender's public key
- If the signed hash matches the recipient's hash, the signature is valid and the document is intact.



Keytool & KeyStore

Most Java programs use existing keys rather than create keys themselves.

password protected private keys and public keys as certificates.

Keytool & KeyStore

keytool -help

```
Commands:

 -certreq            Generates a certificate request
 -changealias        Changes an entry's alias
 -delete             Deletes an entry
 -exportcert         Exports certificate
 -genkeypair         Generates a key pair
 -genseckey          Generates a secret key
 -gencert            Generates certificate from a certificate request
 -importcert         Imports a certificate or a certificate chain
 -importpass         Imports a password
 -importkeystore     Imports one or all entries from another keystore
 -keypasswd          Changes the key password of an entry
 -list               Lists entries in a keystore
 -printcert          Prints the content of a certificate
 -printcertreq       Prints the content of a certificate request
 -printcrl           Prints the content of a CRL file
 -storepasswd        Changes the store password of a keystore
```

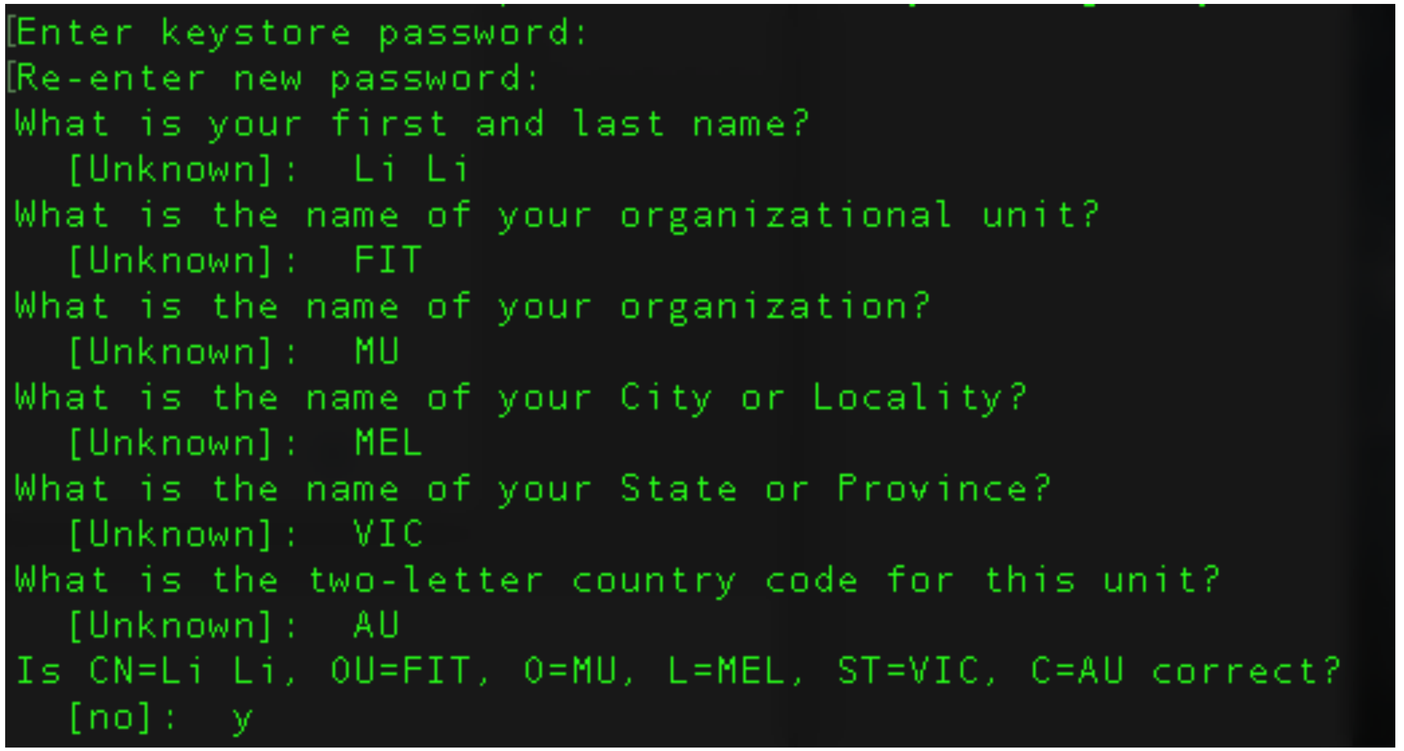
Keytool & KeyStore

keytool -list -v

```
MU00106169X:JavaExamples clii0040$ keytool -list -v
keytool error: java.lang.Exception: Keystore file does not exist: /Users/clii0040/.keystore
java.lang.Exception: Keystore file does not exist: /Users/clii0040/.keystore
        at sun.security.tools.keytool.Main.doCommands(Main.java:783)
        at sun.security.tools.keytool.Main.run(Main.java:366)
        at sun.security.tools.keytool.Main.main(Main.java:359)
```

Keytool & KeyStore

```
keytool -genkey -alias li -keyalg RSA -keypass 123456
```



By default: the generated key will be stored at ~/.keystore

Keytool & KeyStore

keytool -list -v

```
MU00106169X:JavaExamples clii0040$ keytool -list -v
Enter keystore password:
Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: li
Creation date: 07/04/2018
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Li Li, OU=FIT, O=MU, L=MEL, ST=VIC, C=AU
Issuer: CN=Li Li, OU=FIT, O=MU, L=MEL, ST=VIC, C=AU
Serial number: 25eb95b3
Valid from: Sat Apr 07 10:35:34 AEST 2018 until: Fri Jul 06 10:35:34 AEST 2018
Certificate fingerprints:
         MD5:  A5:36:FF:B6:73:CA:1A:49:8F:03:D9:60:6B:D3:87:35
         SHA1: C9:99:E7:96:31:95:06:07:C6:40:CD:E8:EF:66:6F:F4:0B:F3:18:6A
         SHA256: 82:D5:C0:C5:D1:23:5C:51:51:41:E4:38:83:03:E4:4E:BB:77:EC:9D:C1:
4D:48:29:E8:AA:95:66:52:CD:B5:5B
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
```

Keytool & KeyStore

keytool -export -alias li -file joey.crt

keytool -printcert -file joey.crt

```
MU00106169X:JavaExamples clii0040$ keytool -printcert -file joey.crt
Owner: CN=Li Li, OU=FIT, O=MU, L=MEL, ST=VIC, C=AU
Issuer: CN=Li Li, OU=FIT, O=MU, L=MEL, ST=VIC, C=AU
Serial number: 25eb95b3
Valid from: Sat Apr 07 10:35:34 AEST 2018 until: Fri Jul 06 10:35:34 AEST 2018
Certificate fingerprints:
         MD5:  A5:36:FF:B6:73:CA:1A:49:8F:03:D9:60:6B:D3:87:35
         SHA1: C9:99:E7:96:31:95:06:07:C6:40:CD:E8:EF:66:6F:F4:0B:F3:18:6A
         SHA256: 82:D5:C0:C5:D1:23:5C:51:51:41:E4:38:83:03:E4:4E:BB:77:EC:9D:C1:
4D:48:29:E8:AA:95:66:52:CD:B5:5B
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
```

Keytool & KeyStore

Default Keystore location: JAVA_HOME/jre/lib/security/cacerts.

Create KeyStore

```
KeyStore ks =
KeyStore.getInstance(KeyStore.getDefau
ltType());
```

Load KeyStore

```
FileInputStream fis = new
FileInputStream("/Users/clii0040/Projects/JavaExamples/ca_test.keystore");

KeyStore keystore = KeyStore.getInstance(KeyStore.getDefaultType());
keystore.load(fis, "storepass".toCharArray());
```

Keytool & KeyStore

Get Private Key

```
String alias = "ca2";
PrivateKey privateKey = (PrivateKey)
keystore.getKey(alias,
"keypass".toCharArray());
```

Because it needs to have the key password to retrieve the private key, only the owner of the private key can obtain it and use it.

Keytool & KeyStore

Get Public Key

```
Certificate cert =
keystore.getCertificate(alias);
PublicKey publicKey =
 cert.getPublicKey();
```

Password is not needed for obtaining the public key.
Anyone can retrieve it and use it.

- Java Security Issues with the Reflection

Java Basics: Reminder

Access Control Modifier	Class or Interface Accessibility	Member (Field or Method) Accessibility
Public	All	All if class or interface is accessible; interface members always public
Protected	N/A	Same package OR subclass
“default” (Package private)	Same package	Same package
Private	N/A	Only same class (not subclass)

Java Basics: Reminder

```
public class CreditCard {
    public String acctNo = "1234";
}
public class Test {
    public static void main(String[] args) {
        CreditCard cc = new CreditCard();
        System.out.println("account is " + cc.acctNo);
    }
}
```

```
javac CreditCard.java
```

CreditCard.class

```
javac Test.java
```

Test.class

```
java -pc . Test
```

```
MU00106169X:JavaExamples clii0040$ java -cp . Test
account is 1234
```

Java Basics

```
public class CreditCard {
    public String acctNo = "1234";
}
```

private

```
javac CreditCard.java  CreditCard.class

java –pc . Test
```

```
MU00106169X:JavaExamples cliii0040$ java -cp . Test
Exception in thread "main" java.lang.IllegalAccessError:
tried to access field CreditCard.acctNo from class Test
    at Test.main(Test.java:4)
```


```
MU00106169X:JavaExamples cliii0040$ java -cp . Test
account is 1234
```

Reflection

- Reflection is a Java feature that allows developers to manipulate things that already exist and, normally, are set.
- It also has the ability to dynamically change what things are, regardless of how they were written!
- It can even modify objects at runtime.

Reflection

- Class
- Method
- Constructor
- Field
- Modifier
- Others

Reflection

```
Method[] methods = MyObject.class.getMethods();


        for(Method method : methods){
System.out.println("method = " + method.getName());
                        }
```

This example obtains the Class object from the class called MyObject. Using the class object the example gets a list of the methods in that class, iterates the methods and print out their names.

Reflection

How to get complete information about a class ?

```
public class Example{
  public static void main (String[] args){

   //1 - By using Class.forname() method
   Class c1 = Class.forName("Example");

   //2- By using getClass() method
   Example exObj= new Example();
   Class c2 = exObj.getClass();

   //3- By using .class
   Class c3= Example.class;
  }
}
```

Reflection

How to get Metadata of Class?

```
import java.io.Serializable;
public abstract class fit5003base implements Serializable,Cloneable {
}
```

2 1 3

4

1→ `cls.getName();`

2 → `cls.getModifiers();`

3 → `cls.getInterfaces();`

4 → `cls.getSuperclass().getName();`

Reflection

```
import java.lang.reflect.Modifier;
public class fit5003classMetaData {

    public static void main (String [] args) throws ClassNotFoundException {          // Create Class object for fit5003base.class
            Class<fit5003base> fit5003classObj = fit5003base.class;

            System.out.println("Name of the class is : " +fit5003classObj.getName());       // Print name of the class

            System.out.println("Name of the super class is : " + fit5003classObj.getSuperclass().getName());       // Print Super class name

 // Get the list of implemented interfaces in the form of Class array using getInterface() method
    Class[] fit5003InterfaceList = fit5003classObj.getInterfaces();

            System.out.print("Implemented interfaces are : ");                          // Print the implemented interfaces using foreach loop
    for (Class fit5003class1 : fit5003InterfaceList) {
                System.out.println(fit5003class1.getName() + " ");
     }
     System.out.println();

 //Access modifiers: get Modifiers() method and toString() of java.lang.reflect.Modifier class
      int fit5003AccessModifier= fit5003classObj.getModifiers();                                  // Print the access modifiers
System.out.println("Access modifiers of the class are : " + Modifier.toString(fit5003AccessModifier));

        }
}
```

Reflection

How to get Metadata of Variable?

```
from the specified class as well as from its super class
Field[]  field1 = cls.getFields();

//from the specified class only
Field[]  fiel2 = cls.getDeclaredFields();
```

Reflection

```
1  import java.lang.reflect.Field;
2  import java.lang.reflect.Modifier;
3
4  public class ReflectionExample {
5      public static void main(String[] args) throws Exception {
6          Class<?> cls = ReflectionExample.class;
7
8          // Get the metadata of all the fields
9          Field[] field1= cls.getDeclaredFields();
10
11         for(Field field : field1) {
12             System.out.println("Variable name : "+field.getName());
13             System.out.println("Datatypes of the variable :"+field.getType());
14
15             int modifiers = field.getModifiers();
16             System.out.println("Access Modifiers of the variable : " +
17                     Modifier.toString(modifiers));
18             System.out.println("Value of the variable : "+field.get(field));
19         }
20     }
21 }
```

Reflection

How to get Metadata of Method?

```
from the specified class as well as from its super class
Method[]  methods1 = cls.getMethods();

//from the specified class only
Method[]  methods2 = cls.getDeclaredMethods();
```

Reflection

```java
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;

public class ReflectionExample {
    @SuppressWarnings("rawtypes")
    public static void main(String[] args) throws Exception {
        Class<?> cls = ReflectionExample.class;

        Method[] methods = cls.getDeclaredMethods();

        for(Method method : methods) {
            System.out.println("Name of the method : "+method.getName());
            System.out.println("Return type of the method : "+
                    method.getReturnType());

            int modifierList = method.getModifiers();
            System.out.println ("Method access modifiers : "+
                    Modifier.toString(modifierList));

            Class[] paramList= method.getParameterTypes();
            System.out.print ("Method parameter types : ");
            for (Class class1 : paramList){
                System.out.println(class1.getName()+" ");
            }

            Class[] exceptionList = method.getExceptionTypes();
            System.out.print("Excpetion thrown by method :");
            for (Class class1 : exceptionList) {
                System.out.println (class1.getName() +" ");
            }
        }
    }
}
```

Reflection

Access private field

```
Class<?> clazz = Child.class;
Object cc = clazz.newInstance();


Field f1 = cc.getClass().getSuperclass().
        getDeclaredField("a_field");

f1.setAccessible(true);
f1.set(cc, "reflecting on life");

String str1 = (String) f1.get(cc);
System.out.println("field: " + str1);
```

Reflection

Access private method

```
Method method =
object.getClass().getDeclaredMethod(methodName);

method.setAccessible(true);

Object r = method.invoke(object);
```

Realistic attack example:

https://cwe.mitre.org/data/definitions/470.html

Reflection vulnerability security issues

Security Manager:

- ✓ **enabled**
- ✓ **Restrict permissions accordingly in security policy:**
  - ✓ **Watch out for the `ReflectPermission` permission**