Programacíon de Seguridad en Java

Sesión 6

Seguridad en Bases de Datos

© 2003-2004 Depto. Ciencia Computación e IA

Índice

- Encriptación de tarjetas de crédito
- Configuración de la base de datos
- Clase `CreditCardFactory`
- Modificando y consultando la BD

© 2003-2004 Depto. Ciencia Computación e IA

Encriptación de tarjetas de crédito

- **Encriptación de tarjetas de crédito**
- **Configuración de la base de datos**
- **Clase `CreditCardFactory`**
- **Modificando y consultando la BD**

© 2003-2004 Depto. Ciencia Computación e IA

Encriptación de tarjetas de crédito

- **Supuesto:**
  - Tenemos un servidor que acepta datos de tarjetas y los almacena en una BD encriptada con una clave pública.
- **Clases y objetos java:**
  - `CreditCard()`: Objeto para codificar tarjetas
    - `mAccountID`: con `getAccountID()`
    - `mCreditCardNumber`: con `getCreditCardNumber()`
  - `CreditCardDBO()`: Tarjetas encriptadas y "session-key"
    - `mAccountID: getAccountID()`
    - `mEncryptedCCNumber: getEncryptedCCNumber()`
    - `mEncryptedSessionKey: getEncryptedSessionKey()`

Encriptación de tarjetas de crédito

- **Clases y objetos java:**
  - `DatabaseOperation`: Maneja acceso a JDBC mediante las siguientes operaciones definidas y **sin encriptación**:
    - `getAllCreditCardAccountIDs()`
    - `loadCreditCardDBO()`
    - `getCreditCardNumber()`
  - `CreditCardFactory`: Maneja la encriptación y desecriptación de tarjetas utilizando una `mPublicKey` para encriptar. Dada la **clave privada** permite:
    - `createCreditCard()`
    - `findAllCreditCards()`
    - `findCreditCard()`

Configuración de la BD

- Encriptación de tarjetas de crédito
- Configuración de la base de datos
- Clase `CreditCardFactory`
- Modificando y consultando la BD

© 2003-2004 Depto. Ciencia Computación e IA

Configuración de la BD

- **mySQL: Acceso: $ mysql –u admin –p**

```
CREATE DATABASE projava;
USE projava;
CREATE TABLE account (
    account_id INT8 PRIMARY KEY,
    customer_name VARCHAR(40),
    balance FLOAT,
    cert_serial_number VARCHAR(255)
);
CREATE TABLE credit_card (
  account_id INT8 PRIMARY KEY,
  session_key VARCHAR(255),
  cc_number VARCHAR(100) );
```

Configuración de la BD

- **mySQL: Añadir: usuario, "clave"**

```
USE projava;
GRANT ALL PRIVILEGES ON * TO usuario@localhost
   IDENTIFIED BY "clave";
GRANT ALL PRIVILEGES ON * TO
   usuario@localhost.localdomain
   IDENTIFIED BY "clave";
```

- **Crear certificado con la clave pública:**

```
$keytool –genkey –keyalg RSA –keystore tarjetas.ks
$keytool –export –file publica.cer –keystore tarjetas.ks
```

Configuración de la BD

- **Crear configuración: config.properties como objeto de la clase Java.util.Properties:**
  - **Path al certificado que contendrá la clave pública**
  - **Usuario de la BD y password de dicho usuario**
  - **URL de la BD y driver (JDBC connector)**

```
PublicKeyFilename:publica.cer
DBUsername:usuario DBPassword:clave
DBUrl:jdbc:mysql://localhost/projava
DBDriver:org.gjt.mm.mysql.Driver
```

En 3.0.11: com.mysql.jdbc.Driver

Clase `CreditCardFactory`

- Encriptación de tarjetas de crédito
- Configuración de la base de datos
- Clase `CreditCardFactory`
- Modificando y consultando la BD

Clase `CreditCardFactory`

- **Package:**
  - `com.projavasecurity.ecommerce.creditcard`
- **Secuencia de "creación de una tarjeta":**

  Método: `CreditCardFactory.createCreditCard()`

  Argumentos: accountID (1,2,,..) y creditCardNumber (String)

  1. **Encriptar creditCardNumber:**
     - **Crear una clave sesión simétrica con Blowfish ("clave de sesión"),**
     - **Inicializar un cifrador.**
     - **Aplicarlo sobre el número de la tarjeta para tenerlo encriptado.**
  2. **Usar la clave pública del certificado (asimétrica) llamada mPublicKey para encriptar la clave sesión.**

Clase CreditCardFactory

```
// 1a. Crear clave sesión (simétrica).

KeyGenerator kg = KeyGenerator.getInstance("Blowfish");
kg.init(128); Key sessionKey = kg.generateKey();

// 1b. Inicializar cifrador y encriptar número de tarjeta.
Cipher symmetricCipher = Cipher.getInstance
("Blowfish/ECB/PKCS5Padding");
symmetricCipher.init(Cipher.ENCRYPT_MODE, sessionKey);
encryptedCCNumber = symmetricCipher.doFinal
(creditCardNumber.getBytes("UTF8"));

// 2. Usar clave pública para encriptar clave sesión.

Cipher asymmetricCipher = Cipher.getInstance
("RSA/ECB/PKCS1Padding");
asymmetricCipher.init(Cipher.ENCRYPT_MODE, mPublicKey);
encryptedSessionKey = asymmetricCipher.doFinal
(sessionKey.getEncoded());
```

Clase `CreditCardFactory`

- **Secuencia de "creación de una tarjeta":**
  3. **Crear objeto CreditCardDBO con**
     - **Número de cuenta: accountID**
     - **Clave sesión encriptada: encryptedSessionKey**
     - **Número de tarjeta encriptado: encryptedCCNumber**
  4. **Crear un objeto DatabaseOperations:**
     - **Llamar al método `store()` pasándole dicho objeto para que lo almacene en la BD.**
     - **El almacenamiento se realiza a través de JDBC.**
  5. **Devolver el objeto CreditCard con los datos sin encriptar.**

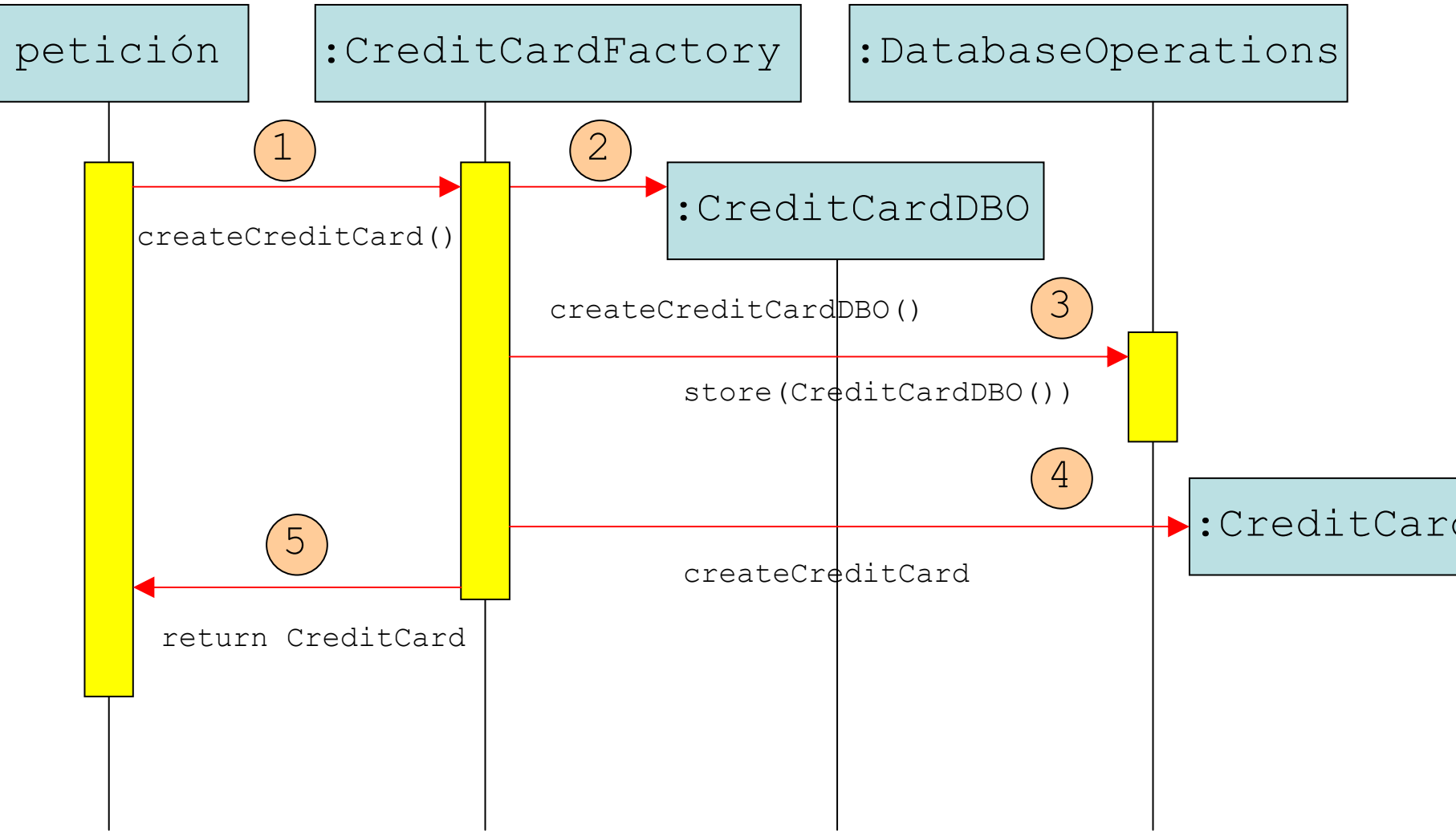
Clase CreditCardFactory

```
// 3. Crear objeto con la información encriptada.
creditCardDBO = new CreditCardDBO(accountID,
encryptedSessionKey, encryptedCCNumber);


// 4. Almancenar objeto con la información encriptada.
mDBOperations.store(creditCardDBO);
CreditCard creditCard = new CreditCard(accountID,
creditCardNumber);


// 5. Devolver objeto con la información original.
return creditCard;
```

Clase `CreditCardFactory`



© 2003-2004 Depto. Ciencia Computación e IA

Clase `CreditCardFactory`

- **Secuencia de "consulta de una tarjeta":**

  Método: `CreditCardFactory.findCreditCard()`

  Argumentos: accountID (1,2,,..) y privateKey (PrivateKey)

  1. **Cargar información encriptada de la tarjeta:**
     - **Crear un objeto DatabaseOperations.**
     - **Llamar al método `loadCreditCardDBO` con accountID.**
     - **Entonces tenemos un objeto CreditCardDBO que contiene:**
       - **Número de cuenta: mAccountID**
       - **Clave de sesión encriptada: mEncryptedSessionKey**
       - **Número de tarjeta encriptado: mEncryptedCCNumber**

```
CreditCardDBO creditCardDBO =
mDBOperations.loadCreditCardDBO(accountID);
```

Clase `CreditCardFactory`

- **Secuencia de "consulta de una tarjeta":**
  2. **Utilizar la clave privada para desencriptar clave sesión**

```
Cipher asymmetricCipher =
   Cipher.getInstance ("RSA/ECB/PKCS1Padding");
asymmetricCipher.init(Cipher.DECRYPT_MODE,
privateKey);

byte[] sessionKeyBytes = asymmetricCipher.doFinal
(creditCardDBO.getEncryptedSessionKey());
```

Clase `CreditCardFactory`

- **Secuencia de "consulta de una tarjeta":**
  3. **Utilizar la clave sesión para desencriptar a su vez el número de tarjeta.**

```
SecretKey symmetricKey = new SecretKeySpec
(sessionKeyBytes, "Blowfish");
Cipher symmetricCipher = Cipher.getInstance
("Blowfish/ECB/PKCS5Padding");
symmetricCipher.init(Cipher.DECRYPT_MODE,
symmetricKey);
byte[] ccNumberBytes = symmetricCipher.doFinal
(creditCardDBO.getEncryptedCCNumber());
creditCardNumber = new String(ccNumberBytes, "UTF8");
```

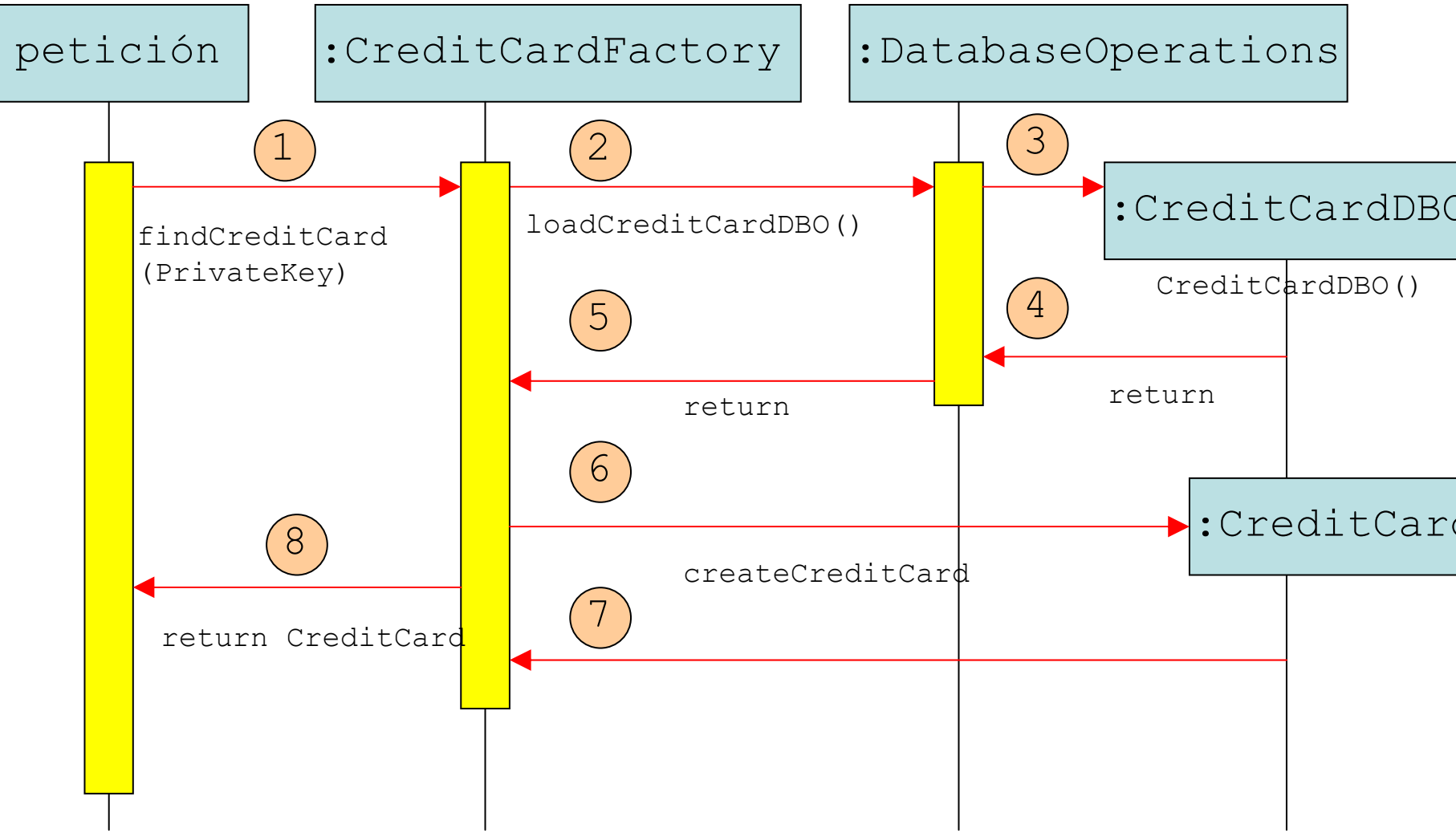
Clase `CreditCardFactory`

- **Secuencia de "consulta de una tarjeta":**
  4. **Crear y devolver un CreditCard con los datos desencriptados:**

```
CreditCard creditCard =
    new CreditCard(accountID, creditCardNumber);
return creditCard;
```


© 2003-2004 Depto. Ciencia Computación e IA

Clase CreditCardFactory



© 2003-2004 Depto. Ciencia Computación e IA

Clase `CreditCardFactory`

- Constructor "para consultas o modificaciones":

  Método: `CreditCardFactory()`

  Argumentos: properties (de "config.properties")

  1. Acceso al fichero que contiene el certificado y que contiene la clave pública (p.e. "publica.cer")
  2. Obtención de la clave pública
     - Instanciar una CertificateFactory
     - Obtener un Certificate
     - Llamar a `getPublicKey`
  3. Creación de un objeto DatabaseOperations para llevar a cabo la consulta.

Clase CreditCardFactory

```
public CreditCardFactory (Properties properties) throws IOException {
  String certFilename = properties.getProperty("PublicKeyFilename");
  try {
      // Acceder a la clave pública
       FileInputStream fis = new FileInputStream(certFilename);
       java.security.cert.CertificateFactory cf =
           java.security.cert.CertificateFactory.getInstance("X.509");
       java.security.cert.Certificate cert =
            cf.generateCertificate(fis);
       fis.close();
       mPublicKey = cert.getPublicKey();
  } catch (Exception e) {
       e.printStackTrace();
       throw new IOException(e.getMessage());
  }
  // Crear objeto de operacion de base de datos
  mDBOperations = new DatabaseOperations(properties);
```

Modificando y consultando la BD

- Encriptación de tarjetas de crédito
- Configuración de la base de datos
- Clase `CreditCardFactory`
- **Modificando y consultando la BD**

Modificando y consultando la BD

- **Aplicación: `CreateTest.java`**
  - **Parámetros:**
    - **Número de cuenta, número de tarjeta de crédito**

```
// Cargar propiedades.
Properties properties = new Properties();
FileInputStream fis = new
FileInputStream(PROPERTIES_FILE);
properties.load(fis); fis.close();

// Crear la tarjeta.
CreditCardFactory factory = new
CreditCardFactory(properties);
CreditCard creditCard =
factory.createCreditCard(id,ccNumber);
```

Modificando y consulando la BD

- **Aplicación: `CreateTest.java`**
  - **cd a directorio con los ficheros:**
    - **config.properties, publica.cer tarjetas.ks**
    - **Ficheros *.java**
    - **Fichero *.jar del JDBC instalado o accesible**
  - **Compilar y probar:**
    - **$javac –d . *.java y sale com.projavasecurity.ecommerce**
    - **$ java com.projavasecurity.ecommerce.CreateTest**

      1 “1234 5678 9012 3456”
  - **Consultar desde mysql:**
    - **USE projava;**
    - **SELECT * FROM credit_card;**

Modificando y consultando la BD

- **Aplicación: `ViewTest.java`**
  1. **Consultar el keystore tarjetas.ks (saber password) y consultar la "clave privada" (para desencriptar)**

```
String ksType = KeyStore.getDefaultType();
KeyStore ks = KeyStore.getInstance(ksType);
FileInputStream fis = new
FileInputStream(KEYSTORE);
ks.load(fis,PASSWORD);
fis.close();

PrivateKey privateKey =
(PrivateKey)ks.getKey("mykey",PASSWORD);
```

Modificando y consultando la BD

- Aplicación: `ViewTest.java`
  2. `CreditCardFactory` y `findAllCreditCards`:

```
// Cargar las properties
Properties properties = new Properties();
fis =new FileInputStream(PROPERTIES_FILE);
properties.load(fis);
fis.close();
// Crear un CreditCardFactory.
CreditCardFactory factory = new
CreditCardFactory(properties);
// Coger todas las tarjetas
Iterator iterator =
factory.findAllCreditCards(privateKey);
```

Ejercicios…

- **Ejercicio8.java…**
  - **Encriptar también el nombre del cliente:**
    - **Aplicar mecanismos de encriptación**
    - **Cambiar este dato desde la tabla account a la tabla credit_card en donde se guardan datos encriptados.**
  - **Modificar todas las clases al incluirse un nuevo campo a encriptar.**
  - **Redefinir la BD: projava2**
  - **Sugerencia: aplicar los mismos pasos que para tratar el código de cuenta del cliente.**