Fundamentos para el desarrollo de aplicaciones en la red

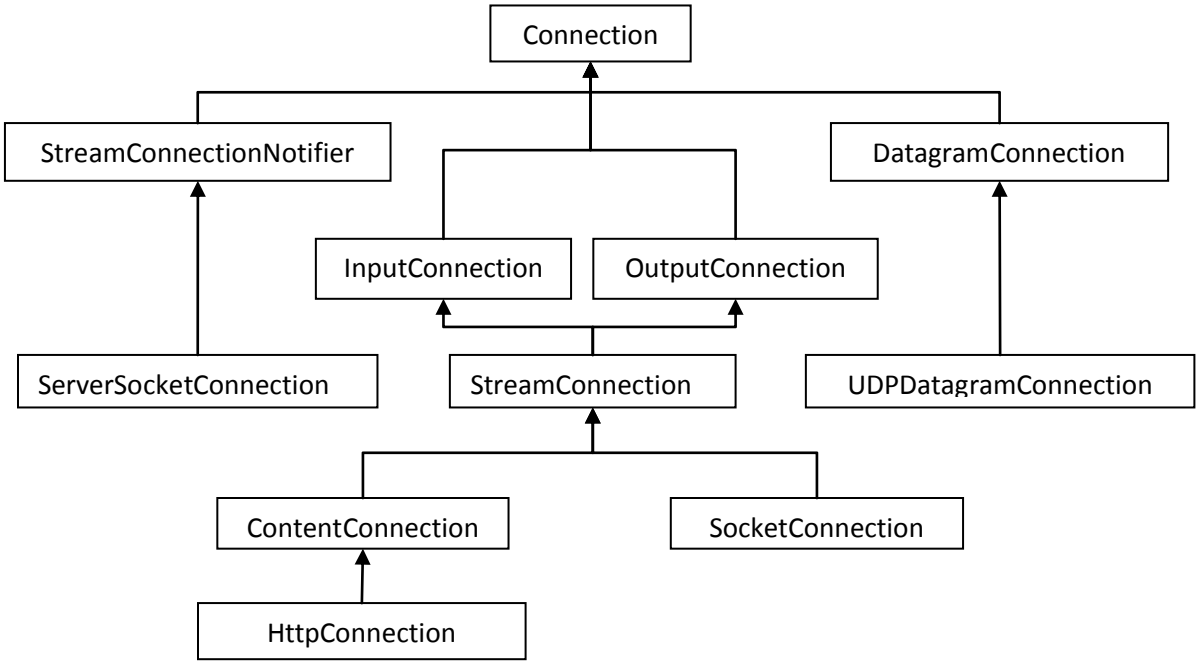
Tema: Arquitectura CLDC

Autor: Elías Niño

Interface Connection

Es el más básico de la clases genéricas de tipo Connection, solo tiene un método definido, el método c*lose(),* se preguntaran entonces y como abro las conexiones, de eso se encarga otra clase y es la clase Connector que contiene el método *open*().

Entonces entre la interface Connection y la Clase Connector, se crean las clases que realizan las conexión a los diferentes servicios como ejemplo las clases SocketConnection, DatagramConnection, HttpConnection, FileConnection, SSLConnection, HttpsConnection y SocketConnection.

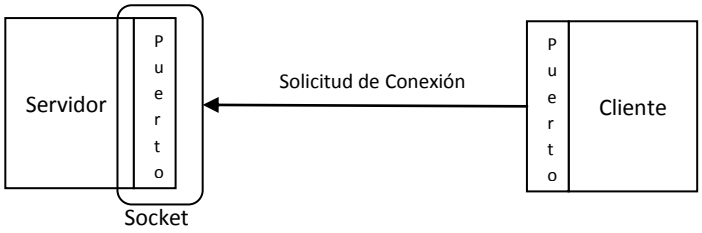


Socket

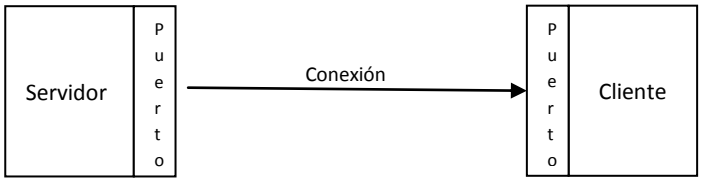
Es el punto final de la comunicación entre 2 programas que se ejecutan sobre una red, entonces los Socket se utilizan para representar un conexión entre un programa servidor y un programa cliente.

Ejemplo:

Usualmente se utilizan de esta forma, un programa servidor coloca un socket a que se enlace o escuche un determinado puerto del computador, esperando a que un cliente haga una solicitud de conexión. El cliente por su parte debe conocer la dirección de la maquina donde se está ejecutando el programa servidor y el puerto a que este se encuentre escuchando.



Si todo sale bien, el servidor acepta la conexión. Inmediatamente el servidor recibe otro Socket que está ligado al mismo puerto pero tiene establecido su extremo remoto en la dirección y puerto del cliente. La razón de ello, es para que el servidor pueda responderle a las peticiones del cliente con quien realizo la conexión.



En el lado del cliente, si la conexión es aceptada, el socket se crea y entonces el cliente puede usar el socket para comunicarse con el servidor, escribiendo o leyendo en el socket creado.

SocketConnection

Esta interface implementa la interface StreamConnection que le provee de unos métodos.

Un SocketConnection es asesado mediante una conexión genérica de flujo de datos, donde se especifica la dirección del host y puerto. El host se puede especificar con un nombre de host o una IP versión 4, ejemplo: socket://host.com:79, donde el nombre del host es “host.com” y el número del puerto es “79”.

<socket_connection_string>	::= "socket://"<hostport>
<hostport>	::= host ":" port
<host>	::= Nombre del host o la dirección IP
<port>	::= El numero del puerto

Las clases que son StreamConnection, proveen de un objeto que esta un ligado y provee los métodos de un InputStream y un OutputStream que son los que se encargan de leer y escribir en el flujo de datos, donde cada una de ellas tiene su propio método close(); Las comunicaciones dúplex en este tipo de socket se debe hacer en un solo sentido a la vez por ejemplo si quiero hacer un InputStream debo asegurarme de cerrar (*close()*) el OutputStream, así mismo en sentido inverso si quiero abrir un OutputStream. Cada vez que cierro un InputStream ó OutputStream este solo puede ser reabierto con el objeto Connetor.open().

Los atributos y propiedades del SocketConnection son:

- Opciones del Socket:
  - static byte DELAY: Es una opción para el socket que especifica si debe utilizar el algoritmo de Nagle que evita la congestión de enviar pequeños paquetes constantemente. Para desactivarlo asigne cero o asigne un número diferente cero para activarlo.

- static byte KEEPALIVE: Es una opción para el socket, para que mantenga activa o viva la conexión, asigne cero si quiere desactivarla u otro número la activara.
- static byte LINGER: Es una opción para el socket que deja unos segundos en espera la conexión antes de cerrarla para la espera de la salida de datos pendientes.
- static byte RCVBUF: Indica el tamaño del buffer que recibe los datos.
- static byte SNDBUF: Indica el tamaño del buffer que envía los datos.

- Métodos:
  - String getAddress(): Obtiene o regresa la dirección remota al que se le enlazo el socket.
  - String getLocalAddress(): Obtiene o regresa la dirección local a la que está vinculado el socket.
  - Int getLocalPort(): Obtiene o regresa el puerto local al que está enlazado este socket.
  - Int getPort(): Obtiene o regresa el puerto remoto al que está enlazado este conector.
  - Int getSocketOption(byte option): Recibe como parámetro la opción del socket y devuelve u obtiene el valor de la opción seleccionada.
  - void setSocketOption(byte option, int value): Recibe como parámetro la opción del socket y el valor de la opción.

Ejemplo:

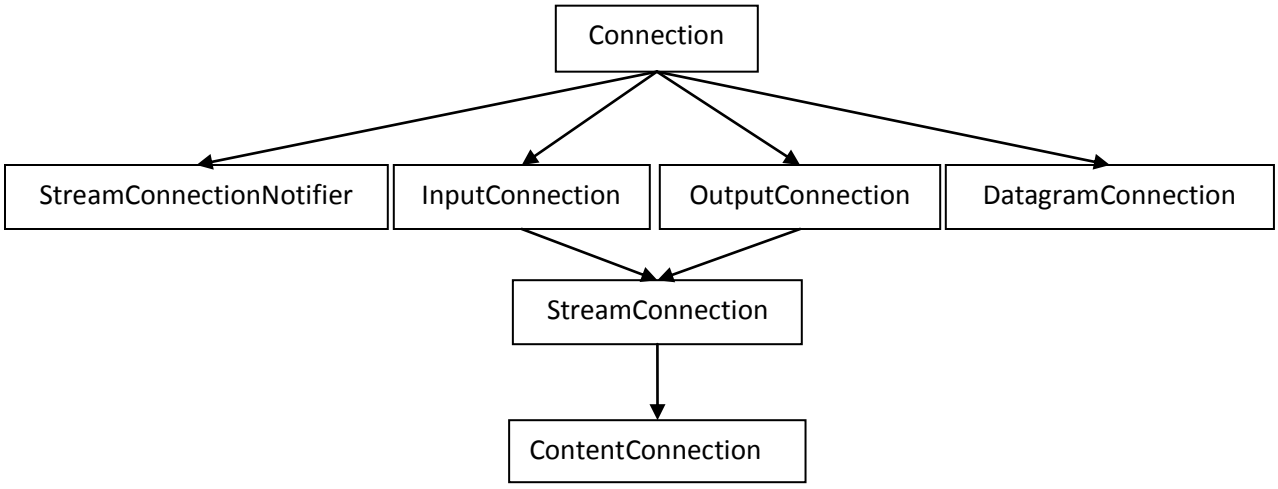
```
SocketConnection sc = (SocketConnection)
Connector.open("socket://localhost:79");
//Se abrió una conexión con el objeto Connector a la dirección (localhost),
indicando el puerto ‘79’.
```

```
sc.setSocketOption(SocketConnection.LINGER, 5);
//Se le agrega la opción LINGER al socket.

InputStream is  = sc.openInputStream();
//Se crea y abre la conexión para la toma de los datos en el socket.

OutputStream os = sc.openOutputStream();
//Se crea y abre la conexión para el envío de los datos en el socket.

os.write("\r\n".getBytes());
//Se envía los datos al host.

int ch = 0;
while(ch != -1) {
        ch = is.read();
//Se leen los datos que se recibió el socket.
}

is.close();
os.close();
sc.close();
//Por ultimo cierro las conexiones abiertas.
```

Datagrama

Es un fragmento de un paquete de datos que es enviado con la suficiente información para que pueda ser encaminado en una red, hacia un equipo terminal de datos (ETD). Cada fragmento de un paquete de datos, es enviado de manera independiente, donde cada fragmento puede perderse o llegar duplicado, por lo que no se garantiza que los paquetes lleguen en el orden adecuado. La estructura de un datagrama está conformada por una cabecera que contiene la dirección IP del esquipo de destino y el fragmento de datos.

El protocolo UDP, hace uso de los datagramas, que se utilizan en servicios no orientados a la conexión.



DatagramConnection

La interface DatagramConnection representa el destino de un datagrama. Un DatagramConnection es asesado mediante una conexión genérica de flujo de datos, donde se especifica la dirección del host y puerto. El host se puede especificar con un nombre de host o una IP versión 4, ejemplo: socket://host.com:79, donde el nombre del host es “host.com” y el número del puerto es “79”.

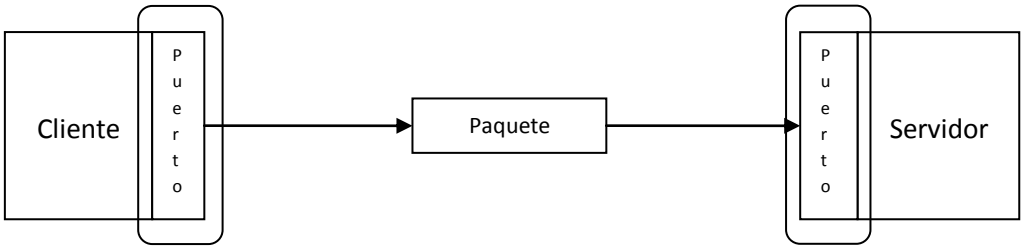
{protocolo}	[{host}]:	[{puerto}]	Modo

Un DatagramConnection puede ser abierto como modo cliente o servidor, si se necesita abrir como modo servidor (Es el que escucha o acepta los datagramas) en el protocolo se coloca “//”y en modo cliente (Es el que escribe o envía los datagramas a un servidor) en el protocolo se coloca “//{host}”

Ejemplo:

Modo servidor:
datagram://:1234

Modo Cliente:
datagram://123.456.789.12:1234



- Métodos:

- Int getMaximumLength(): Obtiene la longitud máxima que un datagrama puede tener al ser enviado o recibido.
- Int getNominalLength(): Obtiene la longitud que tiene el datagrama que es guardado dentro del buffer, esta longitud puede ser igual o menor que el getMaximumLength.
- Datagram newDatagram(byte[] buf, int size): Crea un objeto tipo datagrama. El parámetro *buf*, es el buffer que es usado para el datagrama y el *size* el tamaño del datagrama.
- Datagram newDatagram(byte[] buf, int size, String addr): Crea un objeto tipo datagrama. El parámetro *buf*, es el buffer que es usado para el datagrama, el *size* el tamaño del datagrama y el *addr* que indica la dirección donde será enviado el datagrama.
- Datagram newDatagram(int size): Crea un objeto tipo datagrama. El parámetro *size* es el tamaño del datagrama.
- Datagram newDatagram(int size, String addr): Crea un objeto tipo datagrama. El parámetro *size* es el tamaño del datagrama y el *addr* que indica la dirección donde será enviado el datagrama.
- Void receive(Datagram dgram): Este metodo recibe un datagrama. El método estará bloqueado hasta que reciba un datagrama, si recibe un datagrama mayor al buffer, este es automáticamente truncando.
- Void send(Datagram dgram): Este método envía un datagrama. El datagrama contiene la información que indica a quien será enviado los datos y su longitud.

Ejemplo:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.util.*;

public class PruebaDatagrama extends MIDlet {
```

```
//Se declara el puerto.
static final int receiveport  = 9001;
Receive receiveThread = new Receive();

 public PruebaDatagrama() {}

 public void startApp() {
    //Se inicial el hilo que escucha
    receiveThread.start();
    //Se envia el mensaje de prueba
    enviarMensaje("Mensaje de Prueba");
 }

 public void pauseApp() {}
 public void destroyApp(boolean unconditional)  {}

 //Funcion que envia el Mensaje
 void enviarMensaje(String msg)  {
    String destAddr = "datagram://localhost:" + receiveport;
    DatagramConnection dc = null;
    Datagram dgram;
    byte[] bMsg = msg.getBytes();
    try {
           dc = (DatagramConnection)Connector.open(destAddr);
           // Crea el datagram socket y envia el mensaje
           dgram= dc.newDatagram(bMsg,bMsg.length,destAddr);
           dc.send(dgram);
           System.out.println("Enviando el paquete:" +  msg);
           dc.close();
        }catch (Exception e)  {
```

```
                System.out.println("Se genero un excepcion: " + e.getMessage());
            }
        finally {
            if (dc != null) {
                try {
                    dc.close();
                }catch (Exception e) {
                    System.out.println("Se genero un excepcion al cerrar: " +
e.getMessage());
                }
            }
        }
    }

    //Esta es la funcion que está escuchando el puerto.
    class Receive extends Thread  {
        public void run() {
        doReceive();
        }

        void doReceive() {
            DatagramConnection dc = null;
            Datagram dgram;
          try {
                // Abre un datagrama en modo servidor
                dc = (DatagramConnection)Connector.open("datagram://:"+receiveport);
                String receivedMsg;
                while (true) {
                    dgram = dc.newDatagram(dc.getMaximumLength());
                    try {
                        dc.receive(dgram);
```

```
                } catch (Exception e) {
                    System.out.println("Se genero un excepcion recibiendo el mensaje:"
+ e.getMessage());
                }
                    receivedMsg = new String(dgram.getData(), 0,dgram.getLength());
                    System.out.println("Mensaje recibido: " + receivedMsg);
                    try {
                       Thread.sleep(500);
                    }
                    catch (Exception e) {
                        System.out.println("Se genero un excepcion al doReceive(): " +
e.getMessage());
                    }
              }
           }
            catch (Exception e) {
            System.out.println("Se genero un excepcion al doReceive(): " +
e.getMessage());
          }
          finally {
            if (dc != null) {
               try {
                   dc.close();
                }
                catch (Exception e) {
                  System.out.println("Se genero un excepcion al cerrar " +
e.getMessage());
                }
             }
          }
      }
```

```
    }
}
```

Interface HttpConnection

La interface HttpConnection se creó para el soporte a conexiones HTTP. Este interface extiende del interface ContentConnection.

El protocolo HTTP es un protocolo a nivel de aplicación del tipo petición/respuesta, en el que los parámetros de una petición deben establecerse antes de que se envíe la petición. La conexión puede estar en uno de los siguientes tres posibles estados:

"Setup": No se ha establecido todavía la conexión.

"Connected": Se ha establecido la conexión, la petición se ha enviado y se está esperando por una respuesta.

"Closed": La conexión se ha cerrado.

- Métodos estado “setup”:
    - setRequestMethod(String method): Establece el método de la petición, que puede ser POST, GET o HEAD. El método por defecto es GET.
    - setRequestProperty(String key, String value): Permite indicar el valor de algunas propiedades HTTP antes de enviar la petición, la propiedad que se quiere establecer se indica en el parámetro key y su valor se establece en el parámetro value.

Ejemplo “setup”.

Se crea una conexión HTTP a la URL http://www.google.com.co y se indica que el método de la petición es POST, y que la propiedad HTTP User-Agent tiene el valor Profile/MIDP-2.0 Configuration/CLDC-1.0:

```
HttpConnection c = (HttpConnection)Connector.open("http://www.google.com.co");
c.setRequestMethod(HttpConnection.POST);
c.setRequestProperty("User-Agent", "Profile/MIDP-2.0 Configuration/CLDC-1.0");
```

- Métodos que hacen pasar de estado “setup” a “connected” una conexión tipo HttpConnection:
  - openInputStream ()
  - openOutputStream()
  - openDataInputStream()
  - openDataOutputStream()
  - getLength()
  - getType()
  - getDate()
  - getExpiration()

- Métodos estado “connected”:
  - getURL()
  - getProtocol()
  - getHost()
  - getPort()
  - close()

Ejemplo:

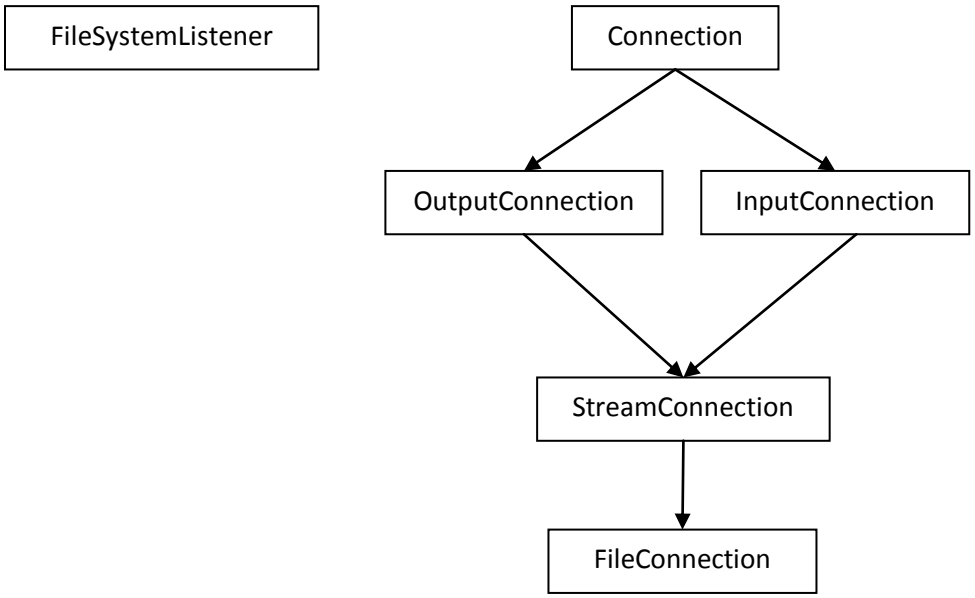
```
void getViaHttpConnection(String url) throws IOException {
    HttpConnection c = null;
    InputStream is = null;
    int rc;
    try {
```

```
        c = (HttpConnection)Connector.open(url);

        rc = c.getResponseCode();
        if (rc != HttpConnection.HTTP_OK) {
            throw new IOException("Código HTTP de respuesta: " + rc);
        }
        is = c.openInputStream();
        String type = c.getType();
        int len = (int)c.getLength();
        if (len > 0) {
            int actual = 0;
            int bytesread = 0 ;
            byte[] data = new byte[len];
            while ((bytesread != len) && (actual != -1)) {
                actual = is.read(data, bytesread, len - bytesread);
                bytesread += actual;
            }
        } else {
            int ch;
            while ((ch = is.read()) != -1) {
                ...
            }
        }
    } catch (ClassCastException e) {
        throw new IllegalArgumentException("No se encontro la URL");
    } finally {
        if (is != null)
            is.close();
        if (c != null)
            c.close();
    }
```

```
}
```

FileConnection

Es una interface que permite a las clases que la implementen crear, leer, escribir, consultar y eliminar información relacionada a los directorios y a los archivos.



Para abrir una conexión tipo FileConnection, solo debes utilizar el objeto connector.open(), enviando como parámetro la ruta del archivo o directorio. Cada vez que se crea una conexión tipo FileConnection, esta solo esta referenciando a un archivo o directorio en un momento de tiempo, si se quiere referenciar a otro

archivo es mejor cerrar la conexión y abrir una nueva para referenciar otro archivo o directorio.

Ejemplo:

CFCard: Es un root del sistema de archivos.

```
try {
    FileConnection fconn =
(FileConnection)Connector.open("file:///CFCard/newfile.txt");
    if (!fconn.exists())
        fconn.create();  // Sino existe se crea.

    fconn.close();/ Se cierra la conexion.
}
 catch (IOException ioe) {
}
```

CFCard/	FileConnection fc = (FileConnection) Connector.open("file:///CFCard/");
SDCard/	FileConnection fc = (FileConnection) Connector.open("file:///SDCard/");
MemoryStick/	FileConnection fc = (FileConnection) Connector.open("file:///MemoryStick/");
C:/	FileConnection fc = (FileConnection) Connector.open("file:///C:/");
/	FileConnection fc = (FileConnection) Connector.open("file:////");

Siempre se debe verificar si el archivo existe, porque si se trata de crear el archivo cuando el archivo existe, se genera una excepción. Si se quiere crear el archivo siempre sin importar si existe o no, se puede utilizar el método delete().

Para evitar manipulación desautorizada, se puede utilizar los siguientes permisos:

- READ_WRITE: Permiso escritura y Lectura.
- WRITE_ONLY: Permiso de escritura.
- READ_ONLY: Permiso de lectura.

Por defecto al crear la conexión se crea con el permiso READ_WRITE.

Métodos:

- Long availableSize(): Regresa la memoria libre que tiene el sistema de archivos o el directorio.
- Boolean canRead(): Chequea si el archivo o el directorio puede ser leído.
- Boolean canWrite(): Chequea si al archivo o el directorio se le puede escribir.
- Void create(): Permite la creación de archivos.
- Void delete(): Permite eliminar archivos o directorios.
- Long directorySize(boolean includeSubDirs): Regresa el tamaño que ocupa los archivos en el directorio si se le envió como parámetro “false” y si se envía “true” en el parámetro incluye el tamaño de los subdirectorios.
- Boolean exists(): Cheque si el archivo o el directorio existe.
- Long fileSize(): Regresa el tamaño del archivo.
- java.lang.String getName(): Regresa la ruta completa del archivo en el dispositivo incluyendo al nombre del archivo.

  <Directorio>/ ó <nombrearchivo.extension>
- java.lang.String getPath():Regresa la ruta del archivo pero sin incluir el nombre del archivo.

  /<root>/<Directorio>/
- java.lang.String getURL(): Regresa los mismo que el getName(), incluyendo la ruta del dispositivo.

  file://<host>/<root>/<Directorio>/< nombrearchivo.extension >

  Ó file://<host>/<root>/<Directorio>/<Directorio>/
- Boolean isDirectory(): Chequea si lo seleccionado es un directorio.
- Boolean isHidden(): Chequea si el archivo está oculto.

- Boolean isOpen(): Chequea si el archivo está abierto por otra conexión.
- Long lastModified(): Regresa la fecha de la última vez que editaron el archivo.
- java.util.Enumeration list(): Regresa o enumera las lista de todos los archivos que no están ocultos que contiene un directorios.
- java.util.Enumeration list(java.lang.String filter, boolean includeHidden): Regresa o enumera la lista de todos los archivos no ocultos, si se le envía "true" en el segundo parámetro incluye los ocultos y en el primer parámetro se le envió como se va filtrar los archivos, que no están ocultos que contiene un directorios.
- Void mkdir(): Permite la creación de carpetas.
- java.io.DataInputStream openDataInputStream()
- java.io.DataOutputStream openDataOutputStream()
- java.io.InputStreamopenInputStream():
- java.io.OutputStreamopenOutputStream()
- java.io.OutputStream openOutputStream(long byteOffset): El método abre y regresa la salida del Stream y se le manda por parámetro, la posición en el archivo.
- Void rename(java.lang.String newName): Renombra el archive o la carpeta.
- Void setFileConnection(java.lang.String fileName): Resetea la conexión para asignarle otro archivo o directorio.
- Void setHidden(boolean hidden): Asigna o retira el atributo oculto.
- Void setReadable(boolean readable): Asigna o retira el atributo de lectura
- Void setWritable(boolean writable) : Asigna o retira el atributo de escritura
- Long totalSize(): Regresa el tamaño de la memoria, del todo el sistema.
- Long usedSize(): Regresa el tamaño usado por en la memoria.

Ejemplo:

```
import java.io.*;
import java.util.*;
```

```
import javax.microedition.io.*;
import javax.microedition.midlet.*;
import javax.microedition.io.file.*;

public class FileConnectionDemo extends MIDlet {

  public void startApp() {
    System.out.println("MIDlet Started....");
    getRoots();
    GetSDcardContent();
    //showFile("readme.txt");
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean condition) {
    notifyDestroyed();
  }

  private void getRoots() {
    Enumeration drives = FileSystemRegistry.listRoots();
    System.out.println("Los roots validos encontrados sone: ");
    while(drives.hasMoreElements()) {
      String root = (String) drives.nextElement();
      System.out.println("\t"+root);
    }
  }

  private void GetSDcardContent() {
    try {
```

```
      FileConnection fc = (FileConnection)
        Connector.open("file:///CFCard/");
      // Lista todos los archivos y directorios, incluyendo a los ocultos.
      System.out.println("Lista de archivos y directories en CFCard:");
      Enumeration filelist = fc.list("*", true);
      while(filelist.hasMoreElements()) {
        String fileName = (String) filelist.nextElement();
        fc = (FileConnection)
          Connector.open("file:///CFCard/" + fileName);
        if(fc.isDirectory()) {
          System.out.println("\tNombre del directorio: " + fileName);
        } else {
          System.out.println
            ("\tNombre del ardhivo: " + fileName +
             "\tSize: "+fc.fileSize());
        }

      }
      fc.close();
    } catch (IOException ioe) {
      System.out.println(ioe.getMessage());
    }
}

public void showFile(String fileName) {
    try {
      FileConnection fc = (FileConnection)
        Connector.open("file:///CFCard/" + fileName);
      if(!fc.exists()) {
        throw new IOException("El archivo no existe");
      }
```

```
            InputStream is = fc.openInputStream();
            byte b[] = new byte[1024];
            int length = is.read(b, 0, 1024);
            System.out.println
              ("Contenido del archivo "+fileName + ": "+ new String(b, 0, length));
        } catch (Exception e) {
        }
    }
}
```

SSLConnection

El SSL (Secure Sockets Layer -Protocolo de Capa de Conexión Segura), es un protocolo criptográfico que proporciona comunicaciones seguras por una red.

Pasos:

1. El protocolo SSL intercambia registros; donde cada registro puede ser comprimido, cifrado y empaquetado con un código de autentificación del mensaje (MAC). Cada registro tiene un campo de content_type que especifica el protocolo de nivel superior que se está usando.
2. Cuando la conexión se establece, el nivel de registro encapsula otro protocolo, el protocolo handshake.
3. El cliente envía y recibe varias estructuras handshake:
4. Se envía un mensaje tipo ClientHello, donde se especifica una lista de conjunto de cifrados, métodos de compresión y la versión del protocolo SSL más alta permitida. Aquí mismo se envían bytes aleatorios llamados Challenge de Cliente o Reto. Además se puede incluir el identificador de la sesión.
5. Después se recibe un registro tipo ServerHello, donde el servidor elige los parámetros de conexión a partir de las opciones ofertadas con anterioridad por el cliente.
6. Cuando los parámetros de la conexión son conocidos, cliente y servidor intercambian certificados (dependiendo de las claves públicas de cifrado seleccionadas).
7. El servidor puede pedir un certificado al cliente, para que la conexión sea mutuamente autentificada.
8. Cliente y servidor negocian una clave secreta (simétrica) común llamada master secret, que es el resultado del cifrado de una clave secreta con una clave pública que es descifrada con la clave privada de cada uno. Todos los datos de claves restantes son derivados a partir de este master secret (y los

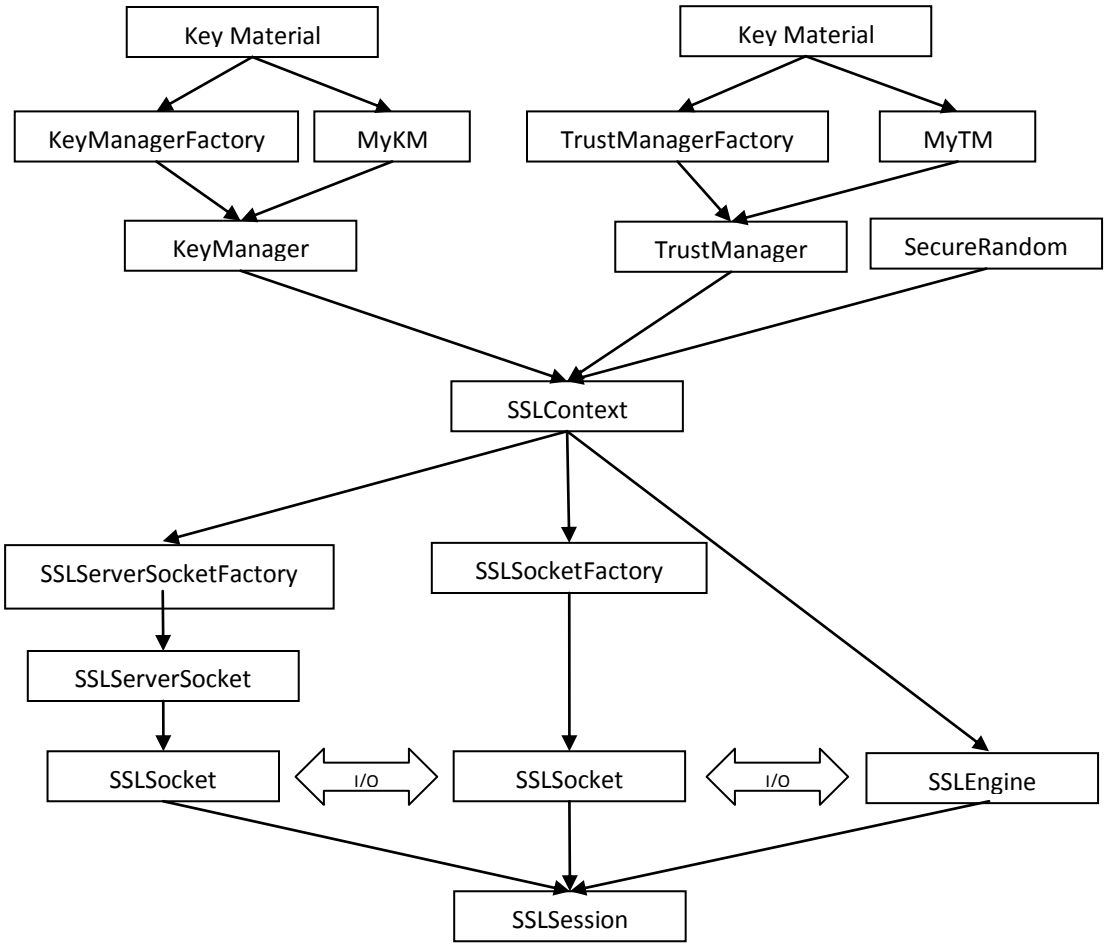
valores aleatorios generados en el cliente y el servidor), que son pasados a través una función pseudoaleatoria cuidadosamente elegida.

Cliente	Servidor
1. ClientHello	
	2. ServerHello
	3. Certificado *opcional*
	4. Petición del certificado *opcional*
	5. Servidor cambio la llave (key Exchange) *opcional*
	6. ServerHelloDone
7. Certificado *opcional*	
8. Cliente cambio la llave (key Exchange)	
9. Certificado cipher spec	
10. ChangeCipherSpec	
11. Finished	
	12. ChangeCipherSpec
	13. Finished
14. Datos Encriptados	
15. Cierre de mensajes	

Funcionamiento del SSL

client	SSL/TLS message	HSStatus
wrap()	ClientHello	NEED_UNWRAP
unwrap()	ServerHello/Cert/ServerHelloDone	NEED_WRAP
wrap()	ClientKeyExchange	NEED_WRAP

wrap()	ChangeCipherSpec	NEED_WRAP
wrap()	Finished	NEED_UNWRAP
unwrap()	ChangeCipherSpec	NEED_UNWRAP
unwrap()	Finished	FINISHED



Para más información puedes ir a link:

http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html

HttpsConnection

Primero debemos hablar de SecurityInfo.

La interface SecurityInfo implementa, una serie de métodos que permiten acceder a la información asociada a conexiones seguras.

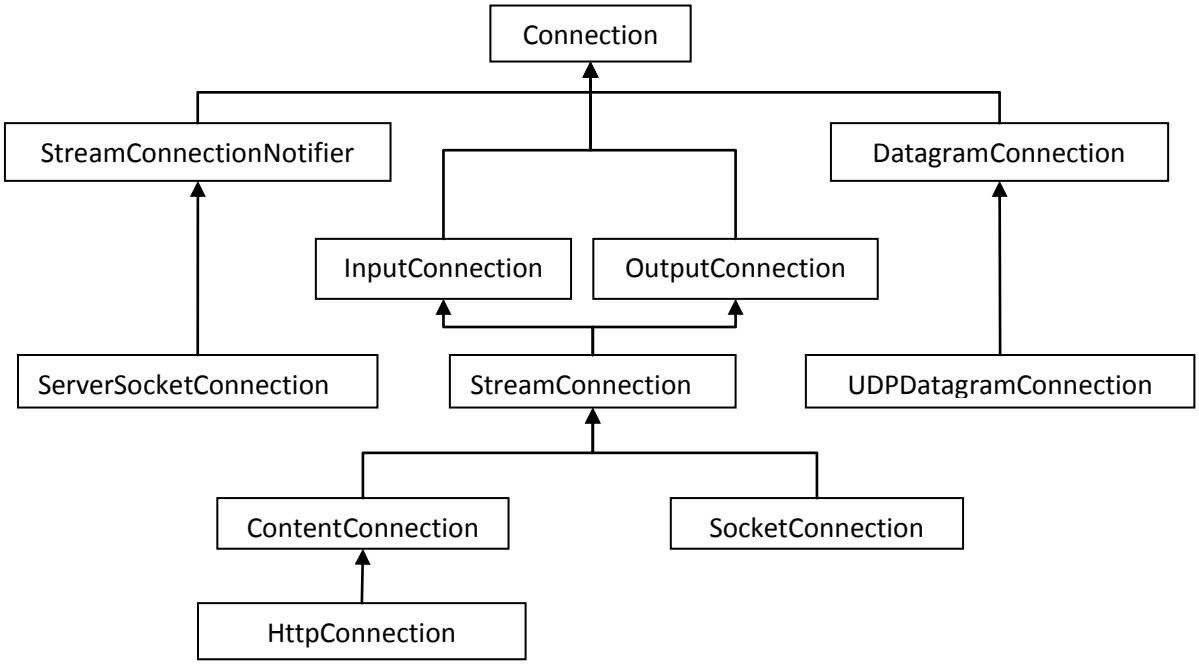
Los métodos que tiene son:

- getCipherSuite(): Regresa o devuelve un String con el nombre del sistema de cifrado que se está utilizando.
- getProtocolName(): Regresa o devuelve el protocolo seguro al que pertenece la conexión.
- getProtocolVersion(): Regresa o devuelve la versión utilizada del protocolo seguro.
- getServerCertificate(): Regresa o devuelve el certificado (objeto Certificate) utilizado para establecer la conexión segura.

Ejemplo:

```
//obtener el objeto SecurityInfo asociado a una conexión segura SecurityInfo
secInfo = conexionSegura.getSecurityInfo();

// Extraer informacion de seguridad
String cipherSuite = secInfo.getCipherSuite();

String protocol = secInfo.getProtocolName();

String version = secInfo.getProtocolVersion();
```

Certificate certificado = secInfo.getServerCertificate();



El HTTPS es la versión segura del protocolo HTTP se basa en establecer conexiones HTTP sobre SSL (Secure Sockets Layer). Para poder realizar este tipo de conexiones se implementa la interface HttpsConnection,. Que implementa los métodos y constantes de HttpsConnection, además de algunas nuevas, que permiten establecer conexiones de red seguras.

Métodos adicionales:

- getPort(): Devuelve el puerto utilizado por una HttpsConnection
- getSecurityInfo(): Devuelve la información de seguridad (objeto javax.microedition.io.SecurityInfo) asociada a una conexión establecida correctamente.

Si el estado de la conexión se encuentra en el estado “setup”, esta será inicializada para que se realice o establezca una conexión segura con el servidor. El método devolverá el resultado cuando la conexión haya sido establecida y este haya validado el certificado dado por el servidor.

Otro de habilidades que introducen las conexiones HTTP seguras son las excepciones CertificateExceptio, que son un subtipo de la IOException y ocurren en la transición al estado "connected". Están relacionadas específicamente con errores ocurridos en el establecimiento de conexiones seguras.

Ejemplo:

```
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.pki.*;
import javax.microedition.lcdui.*;
import java.util.*;
import java.io.*;
import java.lang.*;

public class EjemploHttpSeguro extends MIDlet implements CommandListener{
  Form form = null;
  Display display = null;
  StringItem string_item_form = null;
  StringItem string_item_form_msg = null;
  HttpsConnection httpSecure = null;
  Command command_salir = null;

   public void startApp() {
   // Creamos el GameCanvas eliminando el mecanismo habitual
   // de eventos de teclado
     display = Display.getDisplay(this);
     form = new Form(" Conectando de forma segura con el servidor");
```

```
string_item_form = new StringItem("","");
string_item_form_msg = new StringItem("","");
form.append(string_item_form);
form.append(string_item_form_msg);
 command_salir = new Command("Salir", Command.OK, 3);
 form.addCommand(command_salir);
 form.setCommandListener(this);
display.setCurrent(form);

int ch=0;
StringBuffer b = new StringBuffer();

try{
   httpSecure = (HttpsConnection)Connector.open("https://www.cert.org/");
   string_item_form.setText("Conexion segura establecida con el servidor");

                        SecurityInfo secInfo = httpSecure.getSecurityInfo();
                        String cipherSuite = secInfo.getCipherSuite();
                        String protocol = secInfo.getProtocolName();
                        String version = secInfo.getProtocolVersion();
                        Certificate certificado = secInfo.getServerCertificate();
                        String mensaje = "Datos de la conexion segura:";
                        mensaje+="\n   *Cifrado = "+cipherSuite;
                        mensaje+="\n   *Protocolo = "+protocol;
                        mensaje+="\n   *Version = "+version;
                        mensaje+="\n\nDatos del certificado:";
                        mensaje+="\n  *Tipo = "+certificado.getType();
                        mensaje+="\n  *Version = "+certificado.getVersion();
                        mensaje+="\n  *Emisor = "+certificado.getIssuer();
        string_item_form_msg.setText(mensaje);
```

```
            }
            catch(Exception e){
                        e.printStackTrace();
            }
        }
    public void commandAction(Command c, Displayable s){
        if (c == command_salir) {
         try{
                httpSecure.close();
                System.out.println("Cerrado");
                destroyApp(true);
            notifyDestroyed();
         }
         catch(Exception e){
         }
         }
}
      public void pauseApp() {
     }

  // Implementa el metodo commandApp()
      public void destroyApp(boolean unconditional)throws
MIDletStateChangeException {
        display = null;
     }
  }
```

Bibliografía

http://java.sun.com/docs/books/tutorial/networking/sockets/index.html

http://developers.sun.com/mobility/midp/articles/midp2network/

http://www.java2s.com/Code/Java/J2ME/DemonstratesthefunctionalityofDatagramConnectionframework.htm

http://www.it.uc3m.es/celeste/docencia/j2me/tutoriales/midp2_0/PracticaIO/

http://jcp.org/aboutJava/communityprocess/final/jsr075/index.html

http://developers.sun.com/mobility/apis/articles/fileconnection/

http://java.sun.com/javame/reference/apis/jsr218/

http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html