Les réseaux

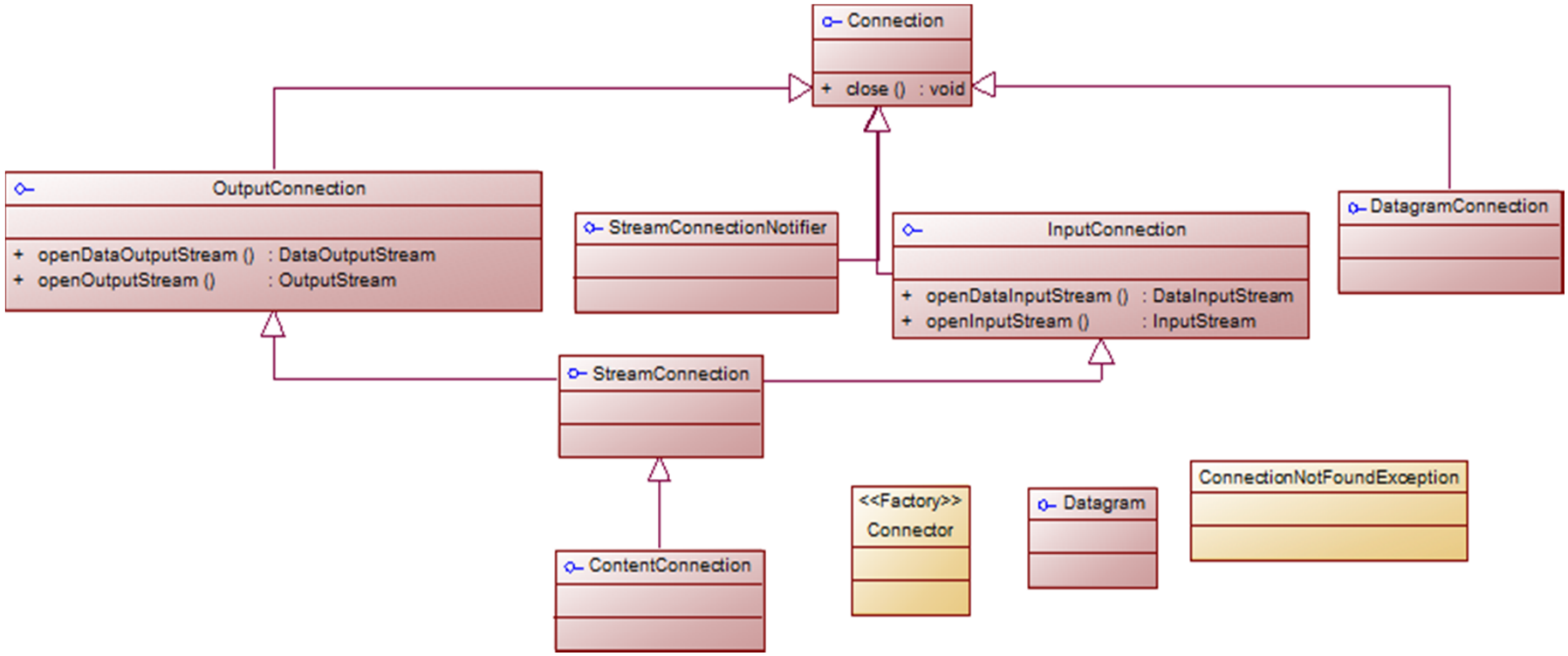
Le Generic Connection Framework
GCF

Connexion réseau

- Le profil MIDP fournit quelques classes du package java.io mais aucune classe du package java.net
- les terminaux mobiles peuvent supporter un ou plusieurs protocoles et mécanismes de réseaux
- ➔ le GCF (*Generic Connection Framework*) est un framework générique et extensible, implémenté au niveau de la configuration CLDC. La généricité est assurée par l'utilisation de:
  - une collection d'interfaces et de classes extensibles pour créer des connexions et gérer les opérations E/S.
  - Une fabrique de connexions
  - Des URLs pour indiquer les types de connexions à créer.
- Les profils et packages optionnels peuvent étendre le GCF de base défini au niveau de la CLDC.

Implémentation du framework GCF au niveau de la CLDC (1.0)

javax.microedition.io



L'URL

protocole://user:password@host:port/url-path;parametres.

- Protocole: définit le protocole ou la méthode d'accès (HTTP,HTTPS, FTP) .
- User: optionnel
- Password: optionnel
- host: adresse IP ou bien le nom complétement qualité de l'hôte où réside la ressource
- port: optionnel
- url-path: chemin d'accès à la ressource. Son format et son interprétation dépendent du type de la ressource.
- parametres: optionnel

Types de connexion

Protocole	Connectivité	Type de connexions	Profil ou package optionnel
btl2cap	Bluetooth	L2CAPConnection	JSR 82.
datagram	Datagram	DatagramConnection	Tous les profils basés sur CDC ou CLDC, comme MIDP, ou FP (Foundation Profil) et dans la JSR 197, J2SE le support est optionnel.
file	File Access	FileConnection, InputConnection	JSR 75. Support optionnel.
http	HTTP	HttpConnection	MIDP 1.0, MIDP 2.0, Foundation Profile, J2SE (JSR 197). Support obligatoire
https	Secure HTTP	HttpsConnection	MIDP 2.0 support obligatoire
comm	Serial I/O	CommConnection	MIDP 2.0. Support optionnel
sms mms cbs	Short Messaging Service Multimedia Messaging Service Cell Broadcast SMS	MessageConnection	JSR 120, JSR 205. Support optionnel.
apdu jcrmi	Security Element	APDUConnection, JavaCardRMIConnection	JSR 177. Support optionnel.
socket serversocket	Socket	SocketConnection, ServerSocketConnection	JSR118 (MIDP 2.0). Support optionnel.
datagram	UDP Datagram	UDPDatagramConnection	JSR118 (MIDP 2.0). Support optionnel.

Créer des connexion

- Connection Connector.open (String url [, int mode, boolean timeouts])
  - mode: READ, WRITE, READ_WRITE (par défaut)
  - timeouts: indique si l'appelant souhaite être notifié des exceptions de type timeouts (InterruptedException) (par défaut false)
- HttpConnection

```
String url = "http://www.serveur.com/SSOSevlet";
try { HttpConnection c = (HttpConnection) Connector.open(url); }
catch (IOException ex) {}
```

- FileConnection (JSR 75)

```
import javax.microedition.io.file.FileConnection;
String url = "file:/donnees.txt";
try { FileConnection c = (FileConnection) Connector.open(url); } catch
(IOException ex) {}
```

Créer des connexion

- DataGramConnection

  String url = "datagram://www.server.com:7001"; try { UDPDatagramConnection c = (UDPDatagramConnection) Connector.open(url); }

  catch (IOException ex) {}

- SocketConnection

  String url = "socket://www.server.com:80"; try { SocketConnection c = (SocketConnection) Connector.open(url); } catch (IOException ex) {}

- CommConnection

  Connector.open("comm:0;baudrate=9600');

Les sockets: Le Serveur

```
 try {ServerSocketConnection scs;
//Création et ouverture de la connexion
scs = (ServerSocketConnection) Connector.open("socket://:5000");
//En attente d'une demande de connexion, la méthode AcceptAndOpen est une méthode synchrone
 SocketConnection  sc = (SocketConnection) scs.acceptAndOpen();
//Ouverture du flux en lecture
DataInputStream is = sc.openDataInputStream();
//Ouverture du flux en écriture
DataOutputStream os = sc.openDataOutputStream();
os.writeUTF("Bonjour");
while (true) {
… } catch (Exception ex) {}
```

Les sockets: Le Client

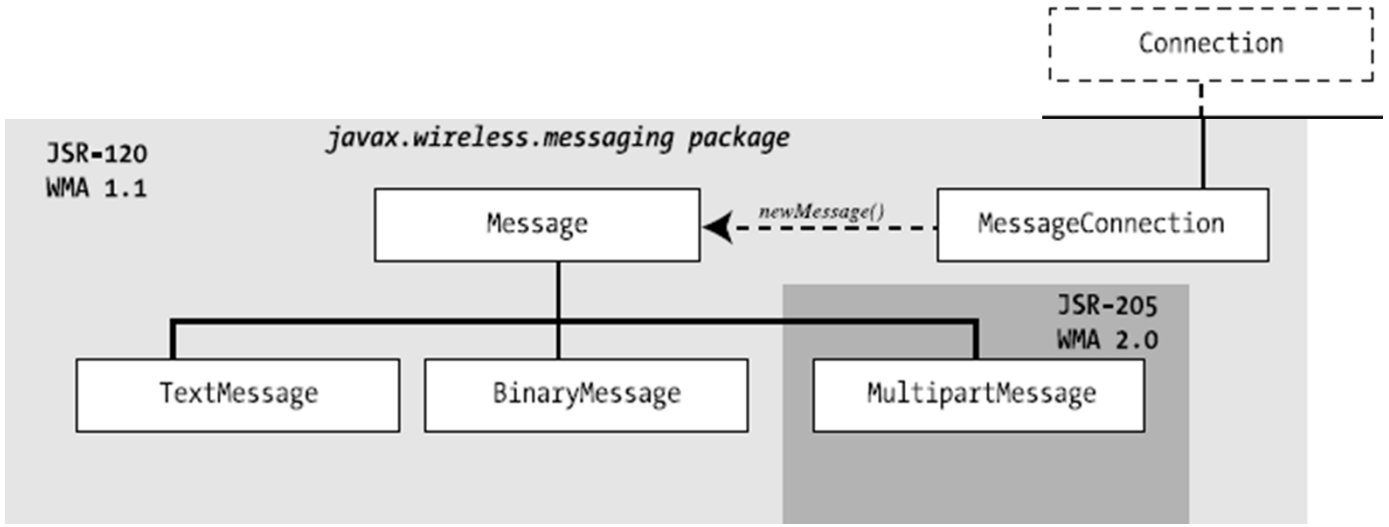
```
// Création et ouverture de la connexion
try  {
sc = (SocketConnection) Connector.open("socket://localhost:5000");

// Ouverture des flux en lecture et en écriture
is = sc.openDataInputStream();
os = sc.openDataOutputStream();
 while (true) {
… } } catch (Exception ex) { }
```

WMA (Wireless Messaging API)

- **L'API WMA**
  - **WMA encode un message binaire et l'envoie par SMS**
  - **Un message peut être découpé en segments et envoyé par envoyé en plusieurs SMS**
- **WMA**
  - **JSR 120 WMA 1.1**
  - **JSR 205 WMA 2.0 (support de l'envoi de messages binaires plus larges)**

WMA


Connection
javax.wireless.messaging package
JSR-120
WMA 1.1
Message
newMessage()
MessageConnection
JSR-205
WMA 2.0
TextMessage
BinaryMessage
MultipartMessage

Deux modes de connexion

- Mode Client: Permet uniquement d'envoyer des messages.
  - MessageConnection cnx= (MessageConnection) Connector.open("sms://0610123040:1234");
- Mode Serveur: pour la réception et l'envoi des messages
  - MessageConnection msgConn = (MessageConnection) Connector.open("sms:// :1234");

Création d'un message

- Message texte:
  - TextMessage txtMessage = (TextMessage) cnx.newMessage(MessageConnection.TEXT_MESSAGE);
- Message binaire
  - BinaryMessage binMessage = (BinaryMessage) cnx.newMessage(MessageConnection.BINARY_MESSAGE);
- Message en plusieurs parties:
  - MultipartMessage mpmMessage = (MultipartMessage) cnx.newMessage(MessageConnection.MULTIPART_MESSAGE);

Envoi du message

- Message texte:
  - txtMessage.setPayloadText(String message);
  - cnx.send(txtMessage);
- Message binaire
  - binMessage. setPayloadData(byte [] data);
  - cnx.send(binMessage);

Réception (Méthode bloquante)

- Message cnx.receive() est une méthode bloquante, en attente de la réception d'un message.

```
Message msg = cnx.receive();
String adresse= msg.getAddress();
if (msg instanceof TextMessage) {
String message= ((TextMessage)msg).getPayloadText();
// Traitement du message
} else if (msg instanceof BinaryMessage) {
byte [] msgReceived = ((BinaryMessage)msg).getPlayloadData();
// traitement
}
}
```

Réception (Utiliser un écouteur)

- Implémenter l'interface MessageListener
- Définir l'écouteur setMessageListener(MessageListener)
- Traiter le message reçu dans la méthode:
  - public void notifyIncomingMessage(MessageConnection mc) { }